

Syd Manual



Ali Polatel

2025-12-06

*Change return success
Going and coming without error
Action brings good fortune*

— Pink Floyd, *Chapter 24*



Hexagram 24 is named fù, “Returning”. Other variations include “return (the turning point)”. Its inner trigram is ☳ (zhèn) shake = thunder, and its outer trigram is ☷ (kūn) field = earth.

http://en.wikipedia.org/wiki/I_Ching_hexagram_24#Hexagram_24

COPYLEFT

Written by Ali Polatel. Distributed under the terms of the GNU General Public License v3. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

METADATA

Author: Ali Polatel

Email: alip@hexsys.org

Date: 2025-12-06 15:03:09 +0100

Commit: bf2fa56f28610e9a3ffbd9749ef47306c724528a

Contents

sydtutorial(7)	1
NAME	1
SYNOPSIS	1
DESCRIPTION	1
INTRODUCTION	1
SYD QUICK INSTALL	2
SYD 101	2
AUTHORS	5
 syd(7)	 7
NAME	7
SANDBOXING	7
SANDBOX CATEGORY SETS	13
SANDBOX RULE SHORTCUTS	13
SegvGuard	14
Force Sandboxing	14
TPE sandboxing	15
Lock Sandboxing	15
Crypt Sandboxing	16
Proxy Sandboxing	18
PTY Sandboxing	18
Memory Sandboxing	18
PID sandboxing	19
SafeSetID	19
Ghost mode	19
SECURITY	20
Threat Model	20
Accessing remote process memory	21
Enhanced Handling of PTRACE_TRACEME	21
Hardened procfs and devfs	22
Hardened proc_pid_status(5)	23
Hardened uname(2)	23
Denying TIOCLINUX ioctl	23
Denying TIOCSTI ioctl	24
Denying FS_IOC_SETFLAGS ioctl	24
Denying PR_SET_MM prctl	24
Restricting prctl option space and trace/allow_unsafe_prctl	24
Restricting io_uring interface and trace/allow_unsafe_uring	25
Restricting creation of device special files	25
Sharing Pid namespace with signal protections	25
Process Priority and Resource Management	25
Streamlining File Synchronization Calls	26
Restricting Resource Limits, Core Dumps, and trace/allow_unsafe_prlimit	26
Enhancing Sandbox Security with Landlock	26
Namespace Isolation in Syd	27
Restricting environment and trace/allow_unsafe_env	27
Managing Linux Capabilities for Enhanced Security	28
Path Resolution Restriction For Chdir and Open Calls	28

Enhanced Symbolic Link Validation	28
Trusted Symbolic Links	29
Trusted Hardlinks	29
Trusted File Creation	29
Memory-Deny-Write-Execute Protections	30
Advanced Memory Protection Mechanisms	31
Null Address Mapping Prevention	31
Default Memory Allocator Security Enhancement	32
Enhanced Security for Memory File Descriptors	32
Path Masking	32
Refined Socket System Call Enforcement	33
Enhanced Execution Control (EEC)	33
Enhanced execve and execveat Syscall Validation	34
Securebits and Kernel-Assisted Executability	34
Enhanced Path Integrity Measures	35
Device Sidechannel Mitigations	36
Restricting CPU Emulation System Calls	36
Kernel Keyring Access Restriction	36
Restricting Memory Protection Keys System Calls	36
Restricting vmsplce System Call	36
Enforcing Position-Independent Executables (PIE)	37
Enforcing Non-Executable Stack	37
Mitigation Against Heap Spraying	37
Mitigation against Page Cache Attacks	38
Enforcing AT_SECURE and UID/GID Verification	38
Process Name Modification Restriction	39
Mitigation against Sigreturn Oriented Programming (SROP)	39
Speculative Execution Mitigation	40
Cryptographically Randomized Sysinfo	41
Memory Sealing of Sandbox Policy Regions on Lock	41
Force Close-on-Exec File Descriptors	41
Force Randomized File Descriptors	41
Syscall Argument Cookies	42
Shared Memory Permissions Hardening	44
Denying Restartable Sequences	44
Personality Syscall Restrictions	44
Thread-Level Filesystem and File-Descriptor Namespace Isolation	45
Denying MSG_OOB Flag in send/recv System Calls	45
Denying O_NOTIFICATION_PIPE Flag in pipe2	45
madvise(2) Hardening	46
HISTORY & DESIGN	46
EXHERBO	47
SEE ALSO	47
AUTHORS	47

syd(5)	49
NAME	49
API	49
CONFIGURATION	49
NAMING	49
SYNTAX	50
PROFILES	50
Stacking Profiles	51
Login shell and the User Profile	51
SECURITY	52
EXAMPLE	52

SEE ALSO	53
AUTHORS	53

syd(2)	55
NAME	55
SYNOPSIS	55
DESCRIPTION	55
COMMANDS	56
stat	56
reset	56
panic	56
ghost	56
config/expand	57
ipc	57
ipc/uid	58
ipc/gid	59
lock	59
log/level	59
log/lock/same_exec_off	60
log/lock/new_exec_on	60
log/lock/subdomains_off	60
log/verbose	61
pty/row	61
pty/col	61
setenv	61
unsetenv	62
clearenv	62
sandbox/walk	62
sandbox/stat	62
sandbox/read	63
sandbox/write	63
sandbox/exec	63
sandbox/ioctl	63
sandbox/create	63
sandbox/delete	64
sandbox/rename	64
sandbox/symlink	64
sandbox/truncate	64
sandbox/chdir	64
sandbox/readdir	65
sandbox/mkdir	65
sandbox/rmdir	65
sandbox/chown	65
sandbox/chgrp	65
sandbox/chmod	66
sandbox/chattr	66
sandbox/chroot	66
sandbox/utime	66
sandbox/mkdev	66
sandbox/mkfifo	67
sandbox/mktemp	67
sandbox/net	67
sandbox/lock	67
sandbox/force	67
sandbox/tpe	67
sandbox/crypt	68

sandbox/proxy	68
sandbox/pty	68
sandbox/mem	69
sandbox/pid	69
default/walk	69
default/stat	69
default/read	69
default/write	70
default/exec	70
default/ioctl	70
default/create	70
default/delete	70
default/rename	71
default/symlink	71
default/truncate	71
default/chdir	71
default/readdir	71
default/mkdir	72
default/rmdir	72
default/chown	72
default/chgrp	72
default/chmod	72
default/chattr	73
default/chroot	73
default/utime	73
default/mkdev	73
default/mkfifo	73
default/mktemp	74
default/net	74
default/block	74
default/force	74
default/segvguard	74
default/tpe	75
default/mem	75
default/pid	75
default/lock	75
unshare/mount	76
unshare/uts	76
unshare/ipc	76
unshare/user	76
unshare/pid	77
unshare/net	77
unshare/cgroup	77
unshare/time	77
root	78
root/map	78
root/fake	78
time	79
time/boot	79
time/mono	79
uts/host	79
uts/domain	79
uts/version	80
ioctl/allow	80
ioctl/deny	80

mem/max	81
mem/vm_max	81
pid/max	82
bind	82
crypt	83
crypt/key	83
crypt/key/enc	83
crypt/key/mac	83
crypt/tmp	84
force	84
proxy/addr	84
proxy/port	85
proxy/ext/host	85
proxy/ext/port	85
proxy/ext/unix	85
segvguard/expiry	86
segvguard/suspension	86
segvguard/maxcrashes	86
tpe/gid	86
tpe/negate	86
tpe/root_owned	86
tpe/user_owned	87
tpe/root_mount	87
allow/walk	87
allow/stat	87
allow/read	87
allow/write	87
allow/exec	87
allow/ioctl	88
allow/create	88
allow/delete	88
allow/rename	88
allow/symlink	88
allow/truncate	88
allow/chdir	89
allow/readdir	89
allow/mkdir	89
allow/rmdir	89
allow/chown	89
allow/chgrp	89
allow/chmod	89
allow/chattr	90
allow/chroot	90
allow/utime	90
allow/mkdev	90
allow/mkfifo	90
allow/mktemp	90
allow/net/bind	90
allow/net/accept	91
allow/net/connect	91
allow/net/sendfd	91
allow/net/link	91
allow/lock/read	91
allow/lock/write	92
allow/lock/exec	92

allow/lock/ioctl	92
allow/lock/create	92
allow/lock/delete	93
allow/lock/rename	93
allow/lock/symlink	93
allow/lock/truncate	93
allow/lock/readdir	94
allow/lock/mkdir	94
allow/lock/rmdir	94
allow/lock/mkdev	94
allow/lock/mkcddev	95
allow/lock/mkfifo	95
allow/lock/bind	95
allow/lock/connect	96
warn/walk	96
warn/stat	96
warn/read	96
warn/write	96
warn/exec	96
warn/ioctl	96
warn/create	97
warn/delete	97
warn/rename	97
warn/symlink	97
warn/truncate	97
warn/chdir	97
warn/readdir	98
warn/mkdir	98
warn/rmdir	98
warn/chown	98
warn/chgrp	98
warn/chmod	98
warn/chattr	98
warn/chroot	99
warn/utime	99
warn/mkdev	99
warn/mkfifo	99
warn/mktemp	99
warn/net/bind	99
warn/net/accept	99
warn/net/connect	100
warn/net/sendfd	100
deny/walk	100
deny/stat	100
deny/read	100
deny/write	100
deny/exec	100
deny/ioctl	101
deny/create	101
deny/delete	101
deny/rename	101
deny/symlink	101
deny/truncate	101
deny/chdir	101
deny/readdir	102

deny/mkdir	102
deny/rmdir	102
deny/chown	102
deny/chgrp	102
deny/chmod	102
deny/chattr	102
deny/chroot	103
deny/utime	103
deny/mkdev	103
deny/mkfifo	103
deny/mktemp	103
deny/net/bind	103
deny/net/accept	103
deny/net/connect	104
deny/net/sendfd	104
panic/walk	104
panic/stat	104
panic/read	104
panic/write	104
panic/exec	104
panic/ioctl	105
panic/create	105
panic/delete	105
panic/rename	105
panic/symlink	105
panic/truncate	105
panic/chdir	105
panic/readdir	106
panic/mkdir	106
panic/rmdir	106
panic/chown	106
panic/chgrp	106
panic/chmod	106
panic/chattr	106
panic/chroot	107
panic/utime	107
panic/mkdev	107
panic/mkfifo	107
panic/mktemp	107
panic/net/bind	107
panic/net/accept	107
panic/net/connect	108
panic/net/sendfd	108
stop/walk	108
stop/stat	108
stop/read	108
stop/write	108
stop/exec	108
stop/ioctl	109
stop/create	109
stop/delete	109
stop/rename	109
stop/symlink	109
stop/truncate	109
stop/chdir	109

stop/readdir	110
stop/mkdir	110
stop/rmdir	110
stop/chown	110
stop/chgrp	110
stop/chmod	110
stop/chattr	110
stop/chroot	111
stop/utime	111
stop/mkdev	111
stop/mkfifo	111
stop/mktemp	111
stop/net/bind	111
stop/net/accept	111
stop/net/connect	112
stop/net/sendfd	112
abort/walk	112
abort/stat	112
abort/read	112
abort/write	112
abort/exec	112
abort/ioctl	113
abort/create	113
abort/delete	113
abort/rename	113
abort/symlink	113
abort/truncate	113
abort/chdir	113
abort/readdir	114
abort/mkdir	114
abort/rmdir	114
abort/chown	114
abort/chgrp	114
abort/chmod	114
abort/chattr	114
abort/chroot	115
abort/utime	115
abort/mkdev	115
abort/mkfifo	115
abort/mktemp	115
abort/net/bind	115
abort/net/accept	115
abort/net/connect	116
abort/net/sendfd	116
kill/walk	116
kill/stat	116
kill/read	116
kill/write	116
kill/exec	116
kill/ioctl	117
kill/create	117
kill/delete	117
kill/rename	117
kill/symlink	117
kill/truncate	117

kill/chdir	117
kill/readdir	118
kill/mkdir	118
kill/rmdir	118
kill/chown	118
kill/chgrp	118
kill/chmod	118
kill/chattr	118
kill/chroot	119
kill/utime	119
kill/mkdev	119
kill/mkfifo	119
kill/mktemp	119
kill/net/bind	119
kill/net/accept	119
kill/net/connect	120
kill/net/sendfd	120
exit/walk	120
exit/stat	120
exit/read	120
exit/write	120
exit/exec	120
exit/ioctl	121
exit/create	121
exit/delete	121
exit/rename	121
exit/symlink	121
exit/truncate	121
exit/chdir	121
exit/readdir	122
exit/mkdir	122
exit/rmdir	122
exit/chown	122
exit/chgrp	122
exit/chmod	122
exit/chattr	122
exit/chroot	123
exit/utime	123
exit/mkdev	123
exit/mkfifo	123
exit/mktemp	123
exit/net/bind	123
exit/net/accept	123
exit/net/connect	124
exit/net/sendfd	124
append	124
mask	124
block	124
cmd/exec	125
load	125
trace/allow_safe_setuid	126
trace/allow_safe_setgid	126
setuid	126
setgid	127
trace/allow_unsafe_cbpf	127

trace/allow_unsafe_ebpf	127
trace/allow_unsafe_dumpable	127
trace/allow_unsafe_exec_ldso	128
trace/allow_unsafe_exec_libc	128
trace/allow_unsafe_exec_memory	128
trace/allow_unsafe_exec_nopie	128
trace/allow_unsafe_exec_null	128
trace/allow_unsafe_exec_stack	129
trace/allow_unsafe_exec_script	129
trace/allow_unsafe_exec_interactive	129
trace/allow_unsafe_exec_speculative	129
trace/allow_unsafe_ptrace	130
trace/allow_unsafe_perf	130
trace/allow_unsafe_create	130
trace/allow_unsafe_filename	130
trace/allow_unsafe_hardlinks	130
trace/allow_unsafe_machine_id	131
trace/allow_unsafe_proc_files	131
trace/allow_unsafe_proc_pid_status	131
trace/allow_unsafe_magiclinks	131
trace/allow_unsafe_symlinks	132
trace/allow_unsafe_namespace	132
trace/allow_unsafe_nice	132
trace/allow_unsafe_nocookie	132
trace/allow_unsafe_nomseal	132
trace/allow_unsafe_sigreturn	133
trace/allow_unsafe_chown	133
trace/allow_unsafe_chroot	133
trace/allow_unsafe_pivot_root	133
trace/allow_unsafe_oob	134
trace/allow_unsafe_open_kfd	134
trace/allow_unsafe_open_path	134
trace/allow_unsafe_mkbdev	135
trace/allow_unsafe_mkcddev	135
trace/allow_unsafe_stat_bdev	135
trace/allow_unsafe_stat_cdev	135
trace/allow_unsafe_notify_bdev	135
trace/allow_unsafe_notify_cdev	136
trace/allow_unsafe_cpu	136
trace/allow_unsafe_deprecated	136
trace/allow_unsafe_keyring	136
trace/allow_unsafe_pipe	136
trace/allow_unsafe_pkey	137
trace/allow_unsafe_madvise	137
trace/allow_unsafe_mbind	137
trace/allow_unsafe_msgsnd	137
trace/allow_unsafe_page_cache	137
trace/allow_unsafe_time	138
trace/allow_unsafe_uring	138
trace/allow_unsafe_xattr	138
trace/allow_unsafe_caps	138
trace/allow_unsafe_cap_fixup	138
trace/allow_unsafe_env	139
trace/allow_safe_kcapi	139
trace/allow_safe_syslog	139

trace/allow_safe_bind	139
trace/allow_unsafe_bind	140
trace/allow_unsafe_socket	140
trace/allow_unsupp_socket	140
trace/allow_unsafe_personality	140
trace/allow_unsafe_prctl	140
trace/allow_unsafe_prlimit	141
trace/allow_unsafe_mqueue	141
trace/allow_unsafe_rseq	141
trace/allow_unsafe_shm	141
trace/allow_unsafe_sysinfo	142
trace/allow_unsafe_syslog	142
trace/allow_unsafe_sync	142
trace/allow_unsafe_memfd	142
trace/allow_unsafe_uname	142
trace/allow_unsafe_vmsplice	143
trace/deny_dotdot	143
trace/deny_exec_elf32	143
trace/deny_exec_elf_dynamic	143
trace/deny_exec_elf_static	143
trace/deny_exec_script	144
trace/deny_tsc	144
trace/exit_wait_all	144
trace/force_cloexec	144
trace/force_rand_fd	145
trace/force_ro_open	145
trace/force_no_symlinks	145
trace/force_no_magiclinks	145
trace/force_no_xdev	146
trace/force_umask	146
trace/memory_access	146
trace/sync_seccomp	147
PATTERN MATCHING	147
ADDRESS MATCHING	148
SECURITY	148
RETURN VALUE	148
ERRORS	149
SEE ALSO	149
AUTHORS	150

syd(1)	151
NAME	151
SYNOPSIS	151
DESCRIPTION	151
OPTIONS	152
INVOCATION	153
ENVIRONMENT	153
LOGGING	154
EXIT CODES	155
BENCHMARKS	155
SEE ALSO	155
AUTHORS	156

syd-aes(1)	157
NAME	157
SYNOPSIS	157

DESCRIPTION	157
OPTIONS	157
SEE ALSO	157
AUTHORS	158
syd-asm(1)	159
NAME	159
SYNOPSIS	159
DESCRIPTION	159
OPTIONS	159
SEE ALSO	159
AUTHORS	159
syd-aux(1)	161
NAME	161
SYNOPSIS	161
DESCRIPTION	161
OPTIONS	161
SEE ALSO	161
AUTHORS	161
syd-bit(1)	163
NAME	163
SYNOPSIS	163
DESCRIPTION	163
OPTIONS	163
SEE ALSO	163
AUTHORS	163
syd-cap(1)	165
NAME	165
SYNOPSIS	165
DESCRIPTION	165
OPTIONS	165
SEE ALSO	165
AUTHORS	165
syd-cat(1)	167
NAME	167
SYNOPSIS	167
DESCRIPTION	167
OPTIONS	167
SEE ALSO	167
AUTHORS	167
syd-cpu(1)	169
NAME	169
SYNOPSIS	169
DESCRIPTION	169
OPTIONS	169
SEE ALSO	169
AUTHORS	169
syd-dns(1)	171
NAME	171
SYNOPSIS	171
DESCRIPTION	171

OPTIONS	171
SEE ALSO	171
AUTHORS	172
syd-elf(1)	173
NAME	173
SYNOPSIS	173
DESCRIPTION	173
OPTIONS	173
SEE ALSO	174
AUTHORS	174
syd-emacs(1)	175
NAME	175
SYNOPSIS	175
DESCRIPTION	175
FILES	175
SEE ALSO	175
AUTHORS	176
syd-env(1)	177
NAME	177
SYNOPSIS	177
DESCRIPTION	177
SECURITY	177
SEE ALSO	177
AUTHORS	177
syd-exec(1)	179
NAME	179
SYNOPSIS	179
DESCRIPTION	179
SEE ALSO	179
AUTHORS	179
syd-fd(1)	181
NAME	181
SYNOPSIS	181
DESCRIPTION	181
OPTIONS	181
EXIT CODES	181
SEE ALSO	181
AUTHORS	182
syd-fork(1)	183
NAME	183
SYNOPSIS	183
DESCRIPTION	183
OPTIONS	183
EXAMPLES	183
NOTES	183
SEE ALSO	184
AUTHORS	184
syd-hex(1)	185
NAME	185
SYNOPSIS	185

DESCRIPTION	185
OPTIONS	185
SEE ALSO	185
AUTHORS	186
syd-info(1)	187
NAME	187
SYNOPSIS	187
DESCRIPTION	187
OPTIONS	187
SEE ALSO	187
AUTHORS	187
syd-key(1)	189
NAME	189
SYNOPSIS	189
DESCRIPTION	189
OPTIONS	189
CAVEATS	190
SEE ALSO	190
AUTHORS	190
syd-ldd(1)	191
NAME	191
SYNOPSIS	191
DESCRIPTION	191
INVOCATION	191
SEE ALSO	191
AUTHORS	192
syd-lock(1)	193
NAME	193
SYNOPSIS	193
DESCRIPTION	193
OPTIONS	193
CONFIGURATION	194
ABI	194
SETS	194
COMPATIBILITY LEVELS	195
FLAGS	195
SECURITY	195
HISTORY	196
EXIT STATUS	196
EXAMPLES	197
SEE ALSO	197
AUTHORS	197
syd-ls(1)	199
NAME	199
SYNOPSIS	199
DESCRIPTION	199
EXAMPLES	199
SEE ALSO	200
AUTHORS	200

syd-mdwe(1)	201
NAME	201
SYNOPSIS	201
DESCRIPTION	201
OPTIONS	201
EXAMPLES	201
EXIT STATUS	202
CAVEATS	202
SEE ALSO	202
AUTHORS	202
 syd-net(1)	 203
NAME	203
SYNOPSIS	203
DESCRIPTION	203
OPTIONS	203
SEE ALSO	203
AUTHORS	203
 syd-mem(1)	 205
NAME	205
SYNOPSIS	205
DESCRIPTION	205
OPTIONS	205
SEE ALSO	205
AUTHORS	205
 syd-oci(1)	 207
NAME	207
SYNOPSIS	207
DESCRIPTION	207
INTEGRATION	207
CONFIGURATION	207
SEE ALSO	208
AUTHORS	208
 syd-ofd(1)	 209
NAME	209
SYNOPSIS	209
DESCRIPTION	209
OPTIONS	209
EXIT STATUS	209
SECURITY	210
SEE ALSO	210
AUTHORS	210
 syd-path(1)	 211
NAME	211
SYNOPSIS	211
DESCRIPTION	211
OPTIONS	211
BUGS	212
SEE ALSO	212
AUTHORS	212

syd-pause(1)	213
NAME	213
SYNOPSIS	213
DESCRIPTION	213
OPTIONS	213
EXIT STATUS	213
SEE ALSO	213
AUTHORS	214
 syd-pds(1)	 215
NAME	215
SYNOPSIS	215
DESCRIPTION	215
OPTIONS	215
EXIT STATUS	215
SEE ALSO	215
AUTHORS	215
 syd-poc(1)	 217
NAME	217
SYNOPSIS	217
DESCRIPTION	217
OPTIONS	217
BUGS	217
SEE ALSO	217
AUTHORS	217
 syd-pty(1)	 219
NAME	219
SYNOPSIS	219
DESCRIPTION	219
OPTIONS	219
USAGE	219
IMPLEMENTATION	220
SECURITY	220
ENVIRONMENT	220
BUGS	220
SEE ALSO	220
AUTHORS	221
 syd-read(1)	 223
NAME	223
SYNOPSIS	223
DESCRIPTION	223
OPTIONS	223
SEE ALSO	223
AUTHORS	224
 syd-rnd(1)	 225
NAME	225
SYNOPSIS	225
DESCRIPTION	225
OPTIONS	225
SEE ALSO	225
AUTHORS	225

syd-run(1)	227
NAME	227
SYNOPSIS	227
DESCRIPTION	227
OPTIONS	227
EXIT STATUS	227
SEE ALSO	228
AUTHORS	228
 syd-sec(1)	 229
NAME	229
SYNOPSIS	229
DESCRIPTION	229
OPTIONS	229
SECURE BITS	230
EXIT STATUS	231
SEE ALSO	231
AUTHORS	231
 syd-sh(1)	 233
NAME	233
SYNOPSIS	233
DESCRIPTION	233
OPTIONS	233
SEE ALSO	233
AUTHORS	233
 syd-sha(1)	 235
NAME	235
SYNOPSIS	235
DESCRIPTION	235
OPTIONS	235
SEE ALSO	235
AUTHORS	235
 syd-size(1)	 237
NAME	237
SYNOPSIS	237
DESCRIPTION	237
SEE ALSO	237
AUTHORS	237
 syd-stat(1)	 239
NAME	239
SYNOPSIS	239
DESCRIPTION	239
SEE ALSO	239
AUTHORS	239
 syd-sys(1)	 241
NAME	241
SYNOPSIS	241
DESCRIPTION	241
OPTIONS	241
SEE ALSO	242
AUTHORS	242

syd-test(1)	243
NAME	243
SYNOPSIS	243
DESCRIPTION	243
SEE ALSO	243
AUTHORS	243
 syd-tck(1)	 245
NAME	245
SYNOPSIS	245
DESCRIPTION	245
OUTPUT	245
PORTABILITY	245
SEE ALSO	246
AUTHORS	246
 syd-tor(1)	 247
NAME	247
SYNOPSIS	247
DESCRIPTION	247
OPTIONS	247
USAGE	247
IMPLEMENTATION	248
SECURITY	248
ENVIRONMENT	248
CAVEATS	248
SEE ALSO	248
AUTHORS	249
 syd-tty(1)	 251
NAME	251
SYNOPSIS	251
DESCRIPTION	251
SEE ALSO	251
AUTHORS	251
 syd-utc(1)	 253
NAME	253
SYNOPSIS	253
DESCRIPTION	253
SEE ALSO	253
AUTHORS	253
 syd-uts(1)	 255
NAME	255
SYNOPSIS	255
DESCRIPTION	255
OPTIONS	255
SEE ALSO	255
AUTHORS	255
 syd-x(1)	 257
NAME	257
SYNOPSIS	257
DESCRIPTION	257
OPTIONS	257

EXIT STATUS 257

SEE ALSO 257

AUTHORS 258

sydtutorial(7)

NAME

sydtutorial - Tutorial introduction to Syd

SYNOPSIS

syd *

DESCRIPTION

This tutorial explains how to sandbox applications using Syd, write sandbox profiles, and configure Syd at runtime from within the sandbox. If you are instead primarily interested in using Syd as a package build sandbox, like we do at Exherbo Linux, you may prefer to start with *syd(2)* and the "paludis" profile whose rules you may list using "syd-cat -p paludis".

INTRODUCTION

Syd is secure by default and highly configurable for your application's usecase. As we go towards the steps you are going to learn how to restrict an application in various ways and at the same time keep the sandbox flexible for cases where restriction is not possible and/or needed. To make the most out of this tutorial, you are recommended to pick an application whose systemic functionality is known to you and try and sandbox this application similar to the instructions in the respective chapter. This functionality, above all, includes the system calls the process calls to interact with the Linux kernel and which parts of the filesystem/network the application needs to access to fulfill its functionality correctly. *bpftrace(1)* and *strace(1)* are your friends. In a further chapter we'll also get to know *pandora(1)* which is a tool to generate Syd profiles automatically for a given application, stay tuned!

In its simplest sense, you can think Syd as a proxy between the Linux kernel and the sandbox process: Syd checks system call arguments for access and if access is granted Syd will execute the system call *on behalf of the sandbox process* and return the result to the sandbox process. Going forward this is important to keep in mind: from the point of view of the Linux kernel, it's one of Syd's syscall handler threads that's running the syscall *not* the sandbox process. This is necessary to achieve a Time-of-check-to-Time-of-use free sandbox. Syd does their best to reduce the side-effects, e.g. with `ls /proc/self`, the sandbox process will still see their own process ID, not Syd's.

1. **Learn by doing:** Trace your applications, learn the ins-and-outs!
2. **Experiment:** Tweak Syd in various different ways and observe the effects!
3. **Make it a game:** Try and break the own sandbox profile you configured, then make it stricter and retry!

SYD QUICK INSTALL

You have the following alternatives:

1. Use the latest release binary located @ <https://distfiles.exherbo.org/#sydbox/>
2. `cave resolve sys-apps/sydbox:3` # if Exherbo (unmask with testing keyword)
3. `emerge sys-apps/syd` # if Gentoo
4. `cargo install syd` # You will not get the manual pages, check: <https://man.exherbo.org>
5. **Take the time to package Syd for your Linux distribution and spread the love!**

Note, releases are signed with this PGP key https://keybase.io/alip/pgp_keys.asc, so take the time to verify the tarball you downloaded. If using cargo to install, you need to install the "libseccomp" library manually. This is a relatively common library and it's packaged by almost all Linux distributions these days. Two things to keep in mind:

1. Install libseccomp development headers (usually included or comes with e.g. the package libseccomp-devel).
2. Install libseccomp static libraries if you want to link Syd statically (usually included or comes with e.g. the package libseccomp-static).

One final note, at the time of writing with libseccomp version 2.5.5, a patched libseccomp is required to make interrupts work correctly under Syd (libseccomp.git has support for the new Linux kernel flag already, we also add a patch to set it by default). The binary release is built with a patched libseccomp and Exherbo source build patches the libseccomp package during preparation phase. Note, in our experience, this bug is mostly noticeable when you sandbox applications written in the Go language. Otherwise, you'll rarely notice it with the latest libseccomp release version. For reference, the patchset resides here: <https://gitlab.exherbo.org/exherbo/arbor/-/tree/master/packages/sys-libs/libseccomp/files>

SYD 101

First, if you run Syd without arguments, you'll silently drop into a new shell. This is because Syd is designed to act as a login shell and in this case it uses the "user" sandbox profile. We'll get to profiles at a later chapter but if you're curious do `syd-cat -p user | less` and read through the rules.

Second, Syd is secure by default and allows you to construct a sandbox to your applications' needs. Here is how the state of the sandbox looks before we pass any options to Syd:

```
$ syd -mstat
syd:
Process ID: 0
Lock: None
Capabilities: Read, Stat, Write, Execute, Connect, Bind
Options:
Memory Max: 134217728
Virtual Memory Max: 4294967296
Pid Max: 128
SegvGuard Max Crashes: 5
SegvGuard Expiry: 120 seconds
SegvGuard Suspension: 600 seconds
Allowed UID Transitions: (total: 0, source -> target)
Allowed GID Transitions: (total: 0, source -> target)
Cidr Rules: (total 0, highest precedence first)
Glob Rules: (total 0, highest precedence first)
Mask Rules: (total 1)
1. Pattern: /proc/cmdline
Force Rules: (total 0, default action: Kill)
$
```

For now let's just take into attention the "Capabilities" line. These are the sandboxing types that are enabled at startup by default.

Initially, we'll do the bare minimum and try to execute a statically linked binary under Syd. *busybox*(1) is a handy tool for our experiment:

```
$ file $(which busybox)
/usr/host/bin/busybox: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, stripped
$ syd busybox true
syd: exec error: No such file or directory
$ echo $?
2
$ syd-sys -e 2
2      ENOENT  No such file or directory
$
```

We get an error that the path does not exist. This is because "Stat Sandboxing" is on by default and the path to the *busybox*(1) binary is hidden. We can see that Syd makes clear by its exit value which error caused the execution to fail. We use the utility *syd-sys*(1), one of the many utilities that come with *syd*(1), to look up the error definition by the exit code.

Let's try to allow and retry:

```
$ syd -m'allow/stat+/usr/host/bin/busybox' busybox true
syd: exec error: No such file or directory
$
```

No luck, we get the same error. This is because the path we specified to "allow/stat" is not a canonicalised path. A canonicalised path is a path which begins with "/" and has neither "." nor ".." nor repeating slashes nor any symbolic links in any of its path components. Let's find out the canonicalised path to our *busybox*(1) binary and retry with it.

```
$ readlink -f /usr/host/bin/busybox
/usr/x86_64-pc-linux-musl/bin/busybox
$ syd -m'allow/stat+/usr/x86_64-pc-linux-musl/bin/busybox' busybox true
{"act": "Deny", "cap": "x", "ctx": "access", "id": "nostalgic_black", "l": 2, "path": "/usr/x86_64-pc-linux-musl/bin/busybox", "pid": 2602591, "sys": "execve(2)"}
syd: exec error: Permission denied
$ echo $?
13
$ syd-sys -e 13
13      EACCES  Permission denied
$
```

We get an error again, but this time we have context. Since Stat Sandboxing is about hiding paths, reporting access violations about it on standard error would beat its purpose so Syd was quiet. However, this time we see "Exec Sandboxing" at play and Syd gives us details about the access violation. The format is JSON lines. It may be hard to read at first but the fact that it's easily parseable allows you to easily search for Syd access violation logs in your system log and filter using tools such as *jq*(1).

Back to the task, for now let's briefly observe that this was an access violation ("ctx": "access") about the *execve*(2) system call ("sys": "execve"). The access violation is of category Exec ("cap": "x") and the target path is "/usr/x86_64-pc-linux-musl/bin/busybox". The decision was to deny the system call ("act": "Deny"). We also have useful metadata such as the process ID ("pid") and the user ID ("uid") executing the offending system call. The "id" field is a human-readable name generated from the "pid" field to make logs easier to follow. There are more information in the omitted fields, it's recommended that you take a look at a complete access violation log entry on your own and make note of the fields that are of value to you. Let's this time allow our *busybox*(1) binary for exec and retry:

```
$ syd -m'allow/exec,stat+/usr/x86_64-pc-linux-musl/bin/busybox' busybox true
$ echo $?
$ 0
```

Task accomplished! Note, how we used the short notation "allow/exec,stat+/path" which is a convenient way to pass -m "allow/exec/path" -m "allow/stat+/path" as a single rule.

Now let's try again with a dynamically linked executable and figure out what we have to add to make it work. This time we will use the *gtrue*(1) utility from the GNU coreutils project which is dynamically linked on this system:

```
$ file $(which gtrue)
/usr/host/bin/gtrue: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /usr/x86_64-pc-linux-musl/
$ lddtree $(which gtrue)
/usr/x86_64-pc-linux-musl/lib/ld-musl-x86_64.so.1 => /usr/x86_64-pc-linux-musl/lib/libc.so
libc.so => /usr/x86_64-pc-linux-musl/lib/libc.so
$ readlink -f $(which gtrue)
/usr/x86_64-pc-linux-musl/bin/gtrue
$ syd -m'allow/exec,stat+/usr/x86_64-pc-linux-musl/bin/gtrue' gtrue
{"act": "Kill", "cap": "x", "ctx": "access", "id": "compassionate_spence", "l": 2, "path": "/usr/x86_64-pc-linux-musl/lib/libc.so", "pid": 2601331, "s": 1}
$ echo $?
137
$
```

Observing the offending path of the new access violation, we understand libc.so is denied execution access. We can also observe, this time Syd has terminated the process ("act": "Kill") rather than denying access to the system call ("act": "Deny"). This is also evident from the exit code which is 137 = 128 + 9 where 9 is the value of the signal "SIGKILL". The deny/kill distinction stems from Syd internals and is not significant for us at this point. Suffice it to say in both cases the execution has been stopped before any code of the target binary had a chance to run.

During access check for Exec Sandboxing, Syd treats dynamically linked executables and their tree of dynamic library dependencies as a single unit. In that sense "allow/read+/path/to/libc.so" and "allow/exec+/path/to/libc.so" serves two different purposes: the former allows you to literally read the contents of the file whilst the latter allows you to load the file into memory as part of an executable.

Having clarified that, let's allow libc.so and retry:

```
$ syd -m'allow/exec,stat+/usr/x86_64-pc-linux-musl/bin/gtrue' -m 'allow/exec+/usr/x86_64-pc-linux-musl/lib/libc.so' gtrue
$ echo $?
0
$
```

Task accomplished! Curious reader will recognise we did not have to add an "allow/stat" clause for "libc.so". This is because the concepts of Stat Sandboxing and Path Hiding pertain specifically to direct access to file paths. Loading libraries into memory is part of the execution process and is therefore only subject to Exec Sandboxing (and Force Sandboxing, aka Binary Verification, which we'll talk more about later).

Now at the third step, let's generalise our small sandbox such that it will allow whichever version of the *true*(1) binary we execute, moreover it will also allow the execution of any other coreutils utility prefixed with "g*". We also do not want to worry if "libc.so" has a version suffix and want to allow all libraries under the common library paths without having to list them one by one. To achieve all this we're going to use *glob*(3) patterns:

```
$ eclectic coreutils list
Available providers for coreutils:
[1]   gnu
[2]   busybox *
$ readlink -f /bin/true
/usr/x86_64-pc-linux-musl/bin/busybox
$ syd -m'allow/stat,exec+/usr/**/bin/{busybox,g*}' -m 'allow/exec+/usr/**/lib/*.so*' true
$ echo $?
0
$ doas eclectic coreutils set -1
$ readlink -f /bin/true
/usr/x86_64-pc-linux-musl/bin/gtrue
$ syd -m'allow/stat,exec+/usr/**/bin/{busybox,g*}' -m 'allow/exec+/usr/**/lib/*.so*' true
$ echo $?
0
$
```

We have seen how *glob*(3) patterns make life easy for us in configuring our sandbox. We have seen using "***" is possible to match recursively and alternates of the form "{foo,bar}" are supported. Syd also supports **empty** alternates of the form "foo/{bar/,}baz" and the **triple star extension**, ie "foo/****" is equivalent to the combination of the two patterns "foo" and "foo/**". Finally we can see we managed to allow a lot more using the same number of rules. Syd has many more powerful features that makes rule editing simple and efficient such as:

- You may specify denylisted paths with "deny/" in addition to "allow/".

- You may specify filtered paths with `"filter/"`, similar to `"deny/"` and `"allow/"` to quiet access violations but still deny access.
- If more than one rule matches the target path, **the last matching rule wins**.
- Many rules may be assembled into a configuration file and passed to Syd with `-P<path>`.
- Files having common rulesets can be included from other configuration files using the `"include <path>"` clause.
- Relative paths in `"include"` clauses are canonicalised based on the parent directory of the current configuration file (*not* the current working directory!).
- Environment variables are expanded in configuration files. Unset environment variables will cause an error.
- Configuration can be locked at any point with the `"lock:on"` clause preventing further edits to the sandbox.

At this point you're highly recommended to experiment with configuring Syd. Do not be afraid to add as many rules as you like. Internally, Syd keeps *glob(3)* patterns as **globsets** and compiles them into a single **regular expression** for efficient matching. This offers acceptable performance up to roughly 10k rules on my system, your mileage may vary.

We have taken a sneak peek at how to configure Syd path allowlists. This is similar for other sandboxing types. Let's leave those for later and explore another way of configuring Syd. This time we'll do it at runtime, from within the sandbox. It may come as a shock from a security perspective to allow access to the sandbox policy from within the sandbox but Syd has a fair set of restrictions to provide this usecase securely and as we'll see later this gives the chance to restrict the sandbox process even further. Another alternative is to make Syd load a dynamic library at startup rather than running a command which is another advanced topic for later. The idea of runtime configuration depends on the **Sandbox Lock** and the lock can have three states: `"on"`, `"off"`, and `"exec"`. The first two are self-explanatory while `"exec"`, allows access to the sandbox policy only for the initial sandbox process. Once the sandbox lock is set to `"on"`, there is no turning back so subsequent edits to the sandbox will no longer be possible. Now let's execute a shell under Syd. This time we will not submit any configuration at startup and run Syd without arguments. This is going to put Syd into login mode when Syd will use the builtin, dynamic `"user"` profile and spawn a shell. We will not delve into details of the user profile for now, check out `"syd-cat -p user"` if you're curious. Suffice it to say it provides a relatively safe set of access rules to system paths and read+write access to your HOME directory and user `"/run"`time paths. In addition, Syd comes with a shell library, called `"esyd"`, that makes Syd interaction easier:

TODO

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in `#sydbox` on Libera Chat or in `#sydbox:mailstation.de` on Matrix.

syd(7)

NAME

Overview of sandboxing with Syd

SANDBOXING

The list of available sandboxing categories is given below:

stat	Confine file metadata accesses. This sandboxing category may be used to effectively <i>hide files and directories</i> from the sandbox process. List of filtered system calls are <i>access(2)</i> , <i>faccessat(2)</i> , <i>faccessat2(2)</i> , <i>getdents64(2)</i> , <i>readlink(2)</i> , <i>readlinkat(2)</i> <i>stat(2)</i> , <i>fstat(2)</i> , <i>lstat(2)</i> , <i>statx(2)</i> , <i>newfstatat(2)</i> , <i>getxattr(2)</i> , <i>getxattrat(2)</i> , <i>lgetxattr(2)</i> , <i>fgetxattr(2)</i> , <i>listxattr(2)</i> , <i>listxattrat(2)</i> , <i>flistxattr(2)</i> , <i>llistxattr(2)</i> , <i>statfs(2)</i> , <i>statfs64(2)</i> , <i>fstatfs(2)</i> , <i>fstatfs64(2)</i> , <i>fanotify_mark(2)</i> , and <i>inotify_add_watch(2)</i> . In addition, paths may be masked using the <i>mask</i> command. In this case, all filtered system calls on the path will be executed on the character device <i>/dev/null</i> instead. See the description of the <i>mask</i> command in <i>syd(2)</i> manual page for more information.
walk	Confine path traversals. This sandboxing category is used during path canonicalization to confine path traversals. As such, its arguments are not necessarily fully canonicalized paths but they're guaranteed to be absolute paths without any <i>.</i> (dot) or <i>..</i> (dotdot) components. It has been split from the <i>stat</i> category as of version 3.39.0. Together with the <i>stat</i> category, path hiding provides a full implementation resilient against attempts to unhide otherwise hidden paths by passing through them during path canonicalization. Notably, OpenBSD's <i>unveil(2)</i> pioneered similar capabilities and remains a widely respected, mature reference implementation.
read	Confine file reads. List of filtered system calls are <i>open(2)</i> , <i>openat(2)</i> and <i>openat2(2)</i> with the O_RDONLY or O_RDWR flags.
write	Confine file writes. List of filtered system calls are <i>open(2)</i> , <i>openat(2)</i> and <i>openat2(2)</i> with the O_WRONLY or O_RDWR flags.
exec	Confine binary execution and dynamic library loading. The list of filtered system calls are <i>execve(2)</i> , <i>execveat(2)</i> , <i>mmap(2)</i> , <i>mmap2(2)</i> , and <i>memfd_create(2)</i> . Note, for scripts access check is done for both the script and the interpreter binary. As of version 3.16.3, Syd checks the paths of the dynamic libraries an executable is linked against for exec access as well. This only works for ELF binaries. As of version 3.21.2, Syd seals memory file descriptors as non-executable by default, therefore memory file descriptors are not checked for exec access unless the option <i>trace/allow_unsafe_memfd:1</i> is set to lift this restriction. As of version 3.21.3, Syd hooks into <i>mmap(2)</i> and <i>mmap2(2)</i> system calls and checks the file descriptor for exec access when the memory protection mode includes PROT_EXEC and flags does not include MAP_ANONYMOUS which typically indicates a <i>dlopen(3)</i> . Therefore, libraries dynamically loaded at runtime are checked for exec access as well. In addition, SegvGuard is used to deny execution if binary is crashing repeatedly which is similar to the implementation of Grsecurity & HardenedBSD. See the SegvGuard section for more information.
ioctl	Confine <i>ioctl(2)</i> system call for filesystem access. This sandboxing type may be used to effectively access GPU, PTY, DRM, and KVM etc. safely. In addition, <i>ioctl(2)</i> requests may be allowed or denied by adding them to the respective list using the options <i>ioctl/allow+</i> and <i>ioctl/deny+</i> . As of version 3.38.0, architecture-agnostic <i>ioctl(2)</i> decoding was introduced, allowing ioctls to be specified by name in addition to numeric values. See the <i>syd(2)</i> manual page for more information.

create	Confine creation of regular files and memory file descriptors. List of filtered system calls are <i>creat(2)</i> , <i>mknod(2)</i> , <i>mknodat(2)</i> , and <i>memfd_create(2)</i> . In addition, open system calls <i>open(2)</i> , <i>openat(2)</i> , and <i>openat2(2)</i> are filtered if the flag O_CREAT is set and the flag O_TMPFILE is not set in arguments. <i>memfd_create(2)</i> name argument is prepended with <i>!memfd:</i> before access check. Use e.g. <i>deny/create+!memfd:**</i> to deny access to memory file descriptors regardless of name. As of version 3.37.0, <i>memfd_create(2)</i> name argument is prepended with <i>!memfd-hugetlb:</i> before access check in case flags include MFD_HUGETLB .
delete	Confine file deletions. List of filtered system calls are <i>unlink(2)</i> and <i>unlinkat(2)</i> . As of version 3.33.0, <i>unlinkat(2)</i> is confined by this category if and only if AT_REMOVEDIR is not set in flags, otherwise it's confined by the <i>rmdir</i> category.
rename	Confine file renames and hard links. List of filtered system calls are <i>rename(2)</i> , <i>renameat(2)</i> , <i>renameat2(2)</i> , <i>link(2)</i> , and <i>linkat(2)</i> .
symlink	Confine creation of symbolic links. List of filtered system calls are <i>symlink(2)</i> and <i>symlinkat(2)</i> .
truncate	Confine file truncations. List of filtered system calls are <i>truncate(2)</i> , <i>truncate64(2)</i> , <i>ftruncate(2)</i> , <i>ftruncate64(2)</i> , and <i>fallocate(2)</i> . In addition, open system calls <i>open(2)</i> , <i>openat(2)</i> , and <i>openat2(2)</i> are filtered if the flag O_TRUNC is set in arguments and the flags O_TMPFILE or O_CREAT are not set in arguments.
chdir	Confine directory changes. List of filtered system calls are <i>chdir(2)</i> and <i>fchdir(2)</i> . Additional hardening may be achieved using the <i>trace/deny_dotdot:1</i> option to deny parent directory traversals. It is possible to set this option at runtime before sandbox is locked. This allows for incremental confinement. See the Path Resolution Restriction For Chdir and Open Calls section for more information.
readdir	Confine directory listings. List of filtered system calls are <i>open(2)</i> , <i>openat(2)</i> , and <i>openat2(2)</i> when they're called on an existing directory regardless of the O_DIRECTORY flag.
mkdir	Confine creation of directories. List of filtered system calls are <i>mkdir(2)</i> , <i>mkdirat(2)</i> , <i>mknod(2)</i> and <i>mknodat(2)</i> .
rmdir	Confine deletion of directories. List of filtered system calls are <i>rmdir(2)</i> and <i>unlinkat(2)</i> . Note <i>unlinkat(2)</i> is confined by this category if and only if AT_REMOVEDIR is set in flags, otherwise it's confined by the <i>delete</i> category. This category was split from the <i>delete</i> category as of version 3.33.0.
chown, chgrp	Confine owner and group changes on files. List of filtered system calls are <i>chown(2)</i> , <i>chown32(2)</i> , <i>fchown(2)</i> , <i>fchown32(2)</i> , <i>lchown(2)</i> , <i>lchown32(2)</i> , and <i>fchownat(2)</i> .
chmod	Confine mode changes on files. List of filtered system calls are <i>chmod(2)</i> , <i>fchmod(2)</i> , <i>fchmodat(2)</i> , and <i>fchmodat2(2)</i> . In addition, a <i>umask(2)</i> value may be set using the <i>trace/force_umask</i> option which is enforced at <i>chmod(2)</i> boundary as well as during regular file creation, e.g. setting <i>trace/force_umask:7177</i> effectively disallows setting s{u,g}id bits, all group+other bits and execute bit for the current user. This feature is useful in setting up W^X (Write XOR Execute) configuration for the sandbox.
chattr	Confine extended attribute changes on files. List of filtered system calls are <i>setxattr(2)</i> , <i>setxattrat(2)</i> , <i>fsetxattr(2)</i> , <i>lsetxattr(2)</i> , <i>removexattr(2)</i> , <i>removexattrat(2)</i> , <i>fremovexattr(2)</i> , and <i>lremovexattr(2)</i> . In addition, Syd ensures extended attributes whose name start with the one of the prefixes <i>security.</i> , <i>trusted.</i> and <i>user.syd.</i> can not be listed or tampered by the sandbox process unless the sandbox lock is <i>off</i> for the respective process. This access can be permitted to the initial sandbox process with <i>lock:exec</i> or to all sandbox processes with <i>lock:off</i> . As of version 3.37.0, this restriction may be lifted with <i>trace/allow_unsafe_xattr:1</i> .

chroot	Confine change of the root directory using the <i>chroot(2)</i> system call. This sandboxing category can be disabled with <i>trace/allow_unsafe_chroot:1</i> at startup, when the <i>chroot(2)</i> system call becomes a no-op. Similarly the <i>pivot_root(2)</i> system call is denied with the <i>errno(3)</i> EPERM by default unless <i>trace/allow_unsafe_pivot_root:1</i> is set at startup in which case it becomes a no-op like <i>chroot(2)</i> . Note, though, no actual change of root directory takes place either way. Syd must share the root directory with the sandbox process to work correctly. Instead, Syd will prevent all filesystem access after the first allowed <i>chroot(2)</i> attempt regardless of the root directory argument. The only exception to the prevention of filesystem access is the <i>chdir(2)</i> system call with the specific argument <i>/</i> , aka the root directory, is allowed. This ensures a TOCTOU-free way to support the common use-case of cutting all filesystem access by means of a <i>chroot(2)</i> call to <i>/var/empty</i> which is common case among unix daemons. This sandboxing category does not depend on the Linux capability CAP_SYS_CHROOT , therefore can be used in unprivileged context. Syd drops the CAP_SYS_CHROOT Linux capability by default unless <i>trace/allow_unsafe_caps:1</i> is passed at startup.
utime	Confine last access and modification time changes on files. List of filtered system calls are <i>utime(2)</i> , <i>utimes(2)</i> , <i>futimesat(2)</i> , <i>utimensat(2)</i> , and <i>utimensat_time64(2)</i> .
mkbdev	Confine block device creation. List of filtered system calls are <i>mknod(2)</i> and <i>mknodat(2)</i> . Block device creation is disabled by default to adhere to the principle of secure defaults with a kernel level seccomp-bpf filter which terminates the process on violation. This filter includes the Syd process, so a compromised Syd process will not be able to create block devices either. Therefore, the user must opt-in at startup using the <i>trace/allow_unsafe_mkbdev:1</i> option to use this category for path-based access checks on block devices.
mkcdev	Confine character device creation. List of filtered system calls are <i>mknod(2)</i> and <i>mknodat(2)</i> . Character device creation is disabled by default to adhere to the principle of secure defaults with a kernel level seccomp-bpf filter which terminates the process on violation. This filter includes the Syd process, so a compromised Syd process will not be able to create character devices either. Therefore, the user must opt-in at startup using the <i>trace/allow_unsafe_mkcddev:1</i> option to use this category for path-based access checks on character devices.
mkfifo	Confine named pipe (FIFO) creation. List of filtered system calls are <i>mknod(2)</i> and <i>mknodat(2)</i> .
mktemp	Confine temporary file creation. List of filtered system calls are <i>open(2)</i> , <i>openat(2)</i> , and <i>openat2(2)</i> with the O_TMPFILE flag. A rule such as <i>allow/mktemp+/tmp</i> permits the sandbox process to create <i>anonymous</i> temporary files under the directory <i>/tmp</i> . Note, the creation of regular files of temporary nature are confined by the create category instead.
net	Confine network access. Socket types UNIX, IPv4, IPv6, NetLink and KCAPI are supported, use the option <i>trace/allow_unsupp_socket:1</i> to pass-through sockets of unsupported types. Note, UNIX domain sockets are always matched on absolute path, therefore always start with the character <i>/</i> . UNIX abstract sockets are prefixed with the <i>***</i> character before access check. Similarly unnamed UNIX sockets use the dummy path <i>!unnamed</i> for access check. Finally, network sandboxing concentrates on confining the initial connection action and leaves out the system calls <i>recvfrom(2)</i> , <i>recvmsg(2)</i> and <i>recvmsg(2)</i> as out of scope for sandbox confinement for performance reasons and due to a lack of security implications noting the fact that <i>recv*</i> system calls cannot specify target addresses.
net/bind	Confine binding network access. This category confines the <i>bind(2)</i> system call, UNIX domain socket file creation using the <i>mknod(2)</i> and <i>mknodat(2)</i> system calls, and UNIX socket-pair creation using the <i>socketpair(2)</i> system call. <i>socketpair(2)</i> system call uses the dummy path <i>!unnamed</i> for access check. Unnamed UNIX sockets use the same dummy path.
net/connect	Confine connecting network access. List of filtered system calls are <i>connect(2)</i> , <i>sendto(2)</i> , <i>sendmsg(2)</i> , and <i>sendmmsg(2)</i> . For IPv4 and IPv6 sockets, the target address of these system calls are also checked against the IP blocklist, see the description of the <i>block</i> command in <i>syd(2)</i> manual page for more information.
net/sendfd	Confine sending of file descriptors. The list of filtered system calls are <i>sendmsg(2)</i> and <i>sendmmsg(2)</i> . As of version 3.31.0, file descriptors referring to block devices, directories and symbolic links may not be passed. The restriction on block devices can be lifted with <i>trace/allow_unsafe_mkbdev:1</i> . UNIX domain sockets are always matched on absolute path, therefore always start with the character <i>/</i> . UNIX abstract sockets are prefixed with the <i>(at sign) character before access check</i> . Similarly unnamed UNIX sockets use the dummy path <i>!unnamed*</i> for access check.

net/link	Confine <i>netlink</i> (7) sockets used in communication between kernel and user space. This sandboxing category may be used to specify a list of <i>netlink</i> (7) families to allow for the sandbox process. Use e.g. <i>allow/net/link+route</i> to allow the NETLINK_ROUTE family. See the <i>syd</i> (2) manual page for more information.
lock/read	Use <i>landlock</i> (7) to confine file read access. This category corresponds to the <i>landlock</i> (7) access right LANDLOCK_ACCESS_FS_READ_FILE and only applies to the content of the directory not the directory itself. As of version 3.33.0, <i>lock/exec</i> and <i>lock/readdir</i> access rights are confined in their respective categories. Previously, this category included the access rights LANDLOCK_ACCESS_FS_EXECUTE and LANDLOCK_ACCESS_FS_READ_DIR as well. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/write	Use <i>landlock</i> (7) to confine file write access. This category corresponds to the <i>landlock</i> (7) access right LANDLOCK_ACCESS_FS_WRITE_FILE and only applies to the content of the directory not the directory itself. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/exec	Use <i>landlock</i> (7) to confine file execution. This category corresponds to the <i>landlock</i> (7) access right LANDLOCK_ACCESS_FS_EXECUTE and only applies to the content of the directory not the directory itself. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/ioctl	Use <i>landlock</i> (7) to confine <i>ioctl</i> (2) operations. This category corresponds to the <i>landlock</i> (7) access right LANDLOCK_ACCESS_FS_IOCTL_DEV and only applies to the content of the directory not the directory itself. This access right is supported as of Landlock ABI version 4 which was introduced with Linux-6.7. This command has no effect when running on older Linux kernels. Use <i>syd-lock</i> (1) to check the latest Landlock ABI supported by the running Linux kernel. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/create	Use <i>landlock</i> (7) to confine file creation, renames and links. This category corresponds to the <i>landlock</i> (7) access right LANDLOCK_ACCESS_FS_MAKE_REG and only applies to the content of the directory not the directory itself. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/delete	Use <i>landlock</i> (7) to confine file unlinking, renames and links. This category corresponds to the <i>landlock</i> (7) access right LANDLOCK_ACCESS_FS_REMOVE_FILE and only applies to the content of the directory not the directory itself. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/rename	Use <i>landlock</i> (7) to confine link or rename a file from or to a different directory (i.e. reparent a file hierarchy). This category corresponds to the <i>landlock</i> (7) access right LANDLOCK_ACCESS_FS_REFER and only applies to the content of the directory not the directory itself. This access right is supported as of Landlock ABI version 2 which was introduced with Linux-5.19. This command has no effect when running on older Linux kernels. Use <i>syd-lock</i> (1) to check the latest Landlock ABI supported by the running Linux kernel. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/symlink	Use Landlock LSM to confine symbolic link creation, renames and links. This category corresponds to the <i>landlock</i> (7) access right LANDLOCK_ACCESS_FS_MAKE_SYM and only applies to the content of the directory not the directory itself. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.

lock/truncate	Use Landlock LSM to confine file truncation with <i>truncate(2)</i> , <i>ftruncate(2)</i> , <i>creat(2)</i> , or <i>open(2)</i> with O_TRUNC . This category corresponds to the <i>landlock(7)</i> access right LANDLOCK_ACCESS_FS_TRUNCATE and only applies to the content of the directory not the directory itself. This access right is supported as of Landlock ABI version 3 which was introduced with Linux-6.2. This command has no effect when running on older Linux kernels. Use <i>syd-lock(1)</i> to check the latest Landlock ABI supported by the running Linux kernel. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/readdir	Use Landlock LSM to confine directory listings. This category corresponds to the <i>landlock(7)</i> access right LANDLOCK_ACCESS_FS_READ_DIR and applies to the given directory and the directories beneath it. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/mkdir	Use Landlock LSM to confine directory creation and renames. This category corresponds to the <i>landlock(7)</i> access right LANDLOCK_ACCESS_FS_MAKE_DIR and only applies to the content of the directory not the directory itself. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/rmdir	Use Landlock LSM to confine directory deletion and renames. This category corresponds to the <i>landlock(7)</i> access right LANDLOCK_ACCESS_FS_REMOVE_DIR and only applies to the content of the directory not the directory itself. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/mkblockdev	Use Landlock LSM to confine block device creation, renames and links. This category corresponds to the <i>landlock(7)</i> access right LANDLOCK_ACCESS_FS_MAKE_BLOCK . This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/mkchardev	Use Landlock LSM to confine character device creation, renames and links. This category corresponds to the <i>landlock(7)</i> access right LANDLOCK_ACCESS_FS_MAKE_CHAR . This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/mkfifo	Use Landlock LSM to confine named pipe (FIFO) creation, renames and links. This category corresponds to the <i>landlock(7)</i> access right LANDLOCK_ACCESS_FS_MAKE_FIFO . This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/bind	Use Landlock LSM to confine network ports for <i>bind(2)</i> and UNIX domain socket creation, renames and links. This category corresponds to the Landlock access right LANDLOCK_ACCESS_NET_BIND_TCP for network ports, and LANDLOCK_ACCESS_FS_MAKE SOCK for UNIX domain sockets. The latter access right only applies to the content of the directory not the directory itself. The access right LANDLOCK_ACCESS_NET_BIND_TCP is supported as of Landlock ABI version 4 which was introduced with Linux-6.7. This command has no effect when running on older Linux kernels. Use <i>syd_lock(1)</i> to check the latest Landlock ABI supported by the running Linux kernel. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
lock/connect	Use Landlock LSM to confine network ports for <i>connect(2)</i> . This category corresponds to the Landlock access right LANDLOCK_ACCESS_NET_CONNECT_TCP . This access right is supported as of Landlock ABI version 4 which was introduced with Linux-6.7. This command has no effect when running on older Linux kernels. Use <i>syd_lock(1)</i> to check the latest Landlock ABI supported by the running Linux kernel. This category is enforced completely in kernel-space so it can be used to construct a multi-layered sandbox. See the Lock Sandboxing section for more information.
block	Application firewall with capability to include <i>ipset</i> and <i>netset</i> files. List of filtered system calls are <i>accept(2)</i> , <i>accept4(2)</i> , <i>connect(2)</i> , <i>sendto(2)</i> , <i>sendmsg(2)</i> , <i>sendmmsg(2)</i> . IPv4 and IPv6 family sockets are supported. Source and target addresses are checked against the IP blocklist. Refer to the description of the block command in <i>syd(2)</i> manual page for more information.
force	Verified Execution: Verify binary/library integrity at <i>exec(3)/mmap(2)</i> time which is similar to Veriexec (NetBSD) & IntegriForce (HardenedBSD). See the Force Sandboxing section for more information.

tpe	Trusted Path Execution: Execution only allowed from Trusted directories for Trusted files which are not writable by group or others and are optionally owned by root or current user. This feature is similar to the implementation of Grsecurity & HardenedBSD. See the TPE Sandboxing section for more information.
crypt	Transparent File Encryption with AES-CTR and HMAC-SHA256, see the Crypt Sandboxing section for more information.
proxy	SOCKS5 proxy forwarding with network namespace isolation. Defaults to TOR. See the Proxy Sandboxing section for more information.
pty	Run sandbox process inside a new pseudoterminal. See the PTY Sandboxing section for more information.
mem, pid	Memory and PID sandboxing: Simple, unprivileged alternatives to Control Groups. See the Memory Sandboxing and PID Sandboxing sections for more information.
SafeSetID	Safe user/group switching with predefined UID/GID transitions like SafeSetID of the Linux kernel. See the SafeSetID section for more information.
Ghost mode	Detach Syd from the sandbox process, similar to <i>seccomp(2)</i> Level 1, aka "Strict Mode". See the Ghost mode section for more information.

Sandboxing for a category may be *on* or *off*: If sandboxing is off, none of the relevant system calls are checked and all access is granted. If, however, sandboxing is on, the action defaults to *deny* and allowlists and denylists can be used to refine access rights, e.g. *allow/read+/etc/passwd*. The default action for a sandboxing category may be changed with the respective option, e.g. *default/force:kill*. See the *syd(2)* manual page for more information on how to configure Syd sandbox policies. If the sandbox process invokes a system call that violates access, this attempt is reported in system log and the system call is denied from execution. There are two ways to customise this behaviour. Syd may be configured to *allow* some *glob(3p)* patterns. If the path argument of the system call which is subject to be modified matches a pattern in the list of allowed *glob(3p)* patterns, this attempt is not denied. If, however it matches a pattern in the list of *deny glob(3p)* patterns the attempt is denied. **If many rules match** the same path or address, the last matching pattern wins. It is also possible to use the actions *exit*, *kill*, *abort*, *stop*, *panic*, and *warn* instead of the *allow* and *deny* actions. The list of available sandboxing actions is given below:

allow	Allow system call.
warn	Allow system call and warn.
filter	Deny system call silently.
deny	Deny system call and warn. This is the default.
panic	Deny system call, warn and panic the current Syd thread.
stop	Deny system call, warn and stop offending process.
abort	Deny system call, warn and abort offending process.
kill	Deny system call, warn and kill offending process.
exit	Warn, and exit Syd immediately with deny <i>errno(3)</i> as exit value.

deny is default unless another default action is set using one of the *default/<category>:<action>* options. See *syd(2)* manual page for more information. *exit* causes Syd to exit immediately with all the sandbox processes running under it. *kill* makes Syd send the offending process a **SIGKILL** signal and deny the system call. *stop* makes Syd send the offending process a **SIGSTOP** signal and deny the system call. *abort* makes Syd send the offending process a **SIGABRT** signal and deny the system call. Unlike *kill* and *stop* actions sandbox processes are able to catch the **SIGABRT** signal, therefore *abort* action should only be used for debugging in trusted environments where a *core(5)* dump file may provide invaluable information. *panic* causes the respective Syd emulator thread to panic in which case the system call is denied by an RAII guard. This behaviour of *panic* action is currently functionally equivalent to the *deny* action, however it may be further extended in the future where Syd emulator processes are fork+exec'ed and address space is rerandomized by ASLR on each access violation. *warn* makes Syd allow the system call and print a warning about it which is used by *pandora(1)* for learning mode. Additionally, Syd may be configured to *filter* some *glob(3p)* patterns. In this case a match will prevent Syd from reporting a warning about the access violation, the system call is still denied though. For *lock/** categories the only available action is *allow*, and these categories accept path names rather than *glob(3p)* patterns as arguments. Relative paths are permitted for all *lock/** categories except *lock/bind* which requires either an absolute UNIX domain socket path or a port-range as argument.

SANDBOX CATEGORY SETS

As of v3.38.0, multiple categories may be specified split by commas and the following sets are defined to streamline sandbox profile composition. Names are intentionally chosen to be consistent with OpenBSD's *pledge(2)* and FreeBSD's capsicum *rights(4freebsd)*:

all	All categories
all-x	All categories except exec
lock/all	All <i>landlock(7)</i> access rights
lpath	walk, stat, chdir
rpath	read, readdir
lock/rpath	lock/read, lock/readdir
wpath	write, truncate
lock/wpath	lock/write, lock/truncate
cpath	create, delete, rename
lock/cpath	lock/create, lock/delete, lock/rename
dpath	mkbdev, mkcdev
lock/dpath	lock/mkbdev, lock/mkcdev
spath	mkfifo, symlink
lock/spath	lock/mkfifo, lock/symlink
tpath	mkdir, rmdir
lock/tpath	lock/mkdir, lock/rmdir
fown	chown, chgrp
fattr	chmod, chattr, utime
net	net/bind, net/connect, net/sendfd
lock/net	lock/bind, lock/connect
inet	net/bind, net/connect
lock/inet	lock/bind, lock/connect
bnet	net/bind
lock/bnet	lock/bind
cnet	net/connect
lock/cnet	lock/connect
snet	net/sendfd

Some examples are given below:

```
default/all:kill
sandbox/inet:off
deny/cpath,rpath,wpath+${HOME}/.ssh/**
kill/spath+/tmp/**
allow/inet+loopback!1024-65535
kill/unix+/dev/log
```

SANDBOX RULE SHORTCUTS

Sandbox capabilities may be passed to sandbox actions either as a single unit or as a comma-delimited list, e.g:

```
allow/read,write,stat,exec+/**
allow/read,write,stat-/**
deny/read,write,stat+/**
deny/read,write-/**
filter/read,write,stat+/dev/mem
filter/read,write-/dev/mem
```

As of version 3.18.14, sandboxing modes may be specified as a single unit or as a comma-delimited list, e.g:

```
sandbox/read,write,stat,exec:on
sandbox/net,lock:off
```

As of version 3.19.0, namespace types may be specified as a single unit or as a comma-delimited list, e.g.:

```
unshare/user,pid,mount:on
unshare/net,cgroup:off
```

As of version 3.35.0, default modes may be specified as a single unit or as a comma-delimited list, e.g.:

```
default/write,truncate:kill
default/read,stat:allow
```

SegvGuard

As of version 3.16.3, Syd has a simple implementation of SegvGuard. The implementation is inspired by that of HardenedBSD with identical defaults: If a sandbox process receives a signal that may produce a *core(5)* dump file for *segvguard/maxcrashes* times (defaults to 5), in a period of *segvguard/expiry* seconds (defaults to 2 minutes), subsequent attempts to execute the same executable is denied for *segvguard/suspension* seconds (defaults to 10 minutes). SegvGuard can be disabled by setting *segvguard/expiry:0*. SegvGuard support depends on *ptrace(2)*, therefore it may also be disabled by setting *trace/allow_unsafe_ptrace:1* at startup. The trigger signals for SegvGuard are **SIGABRT**, **SIGBUS**, **SIGFPE**, **SIGILL**, **SIGIOT**, **SIGKILL**, **SIGQUIT**, **SIGSEGV**, **SIGSYS**, **SIGTRAP**, **SIGXCPU**, and **SIGXFSZ**. The signal **SIGKILL** is intentionally included into the list even though it is not a *core(5)* dump file generating signal to make *kill* rules trigger SegvGuard, a design later mirrored in HardenedBSD's work on PaX SEGVGUARD and Capsicum integration.

Check out the following links for further information on SegvGuard:

- http://en.wikibooks.org/wiki/Grsecurity/Appendix/Grsecurity_and_PaX_Configuration_Options#Deter_exploit_brut
- http://en.wikibooks.org/wiki/Grsecurity/Appendix/Grsecurity_and_PaX_Configuration_Options#Active_kernel_explo
- <http://phrack.org/archives/issues/59/9.txt>
- <http://phrack.org/archives/issues/58/4.txt>
- <https://github.com/HardenedBSD/hardenedBSD/wiki/segvguard2-ideas---brainstorm>
- <https://hardenedbsd.org/article/shawn-webb/2025-03-01/hardenedbsd-february-2025-status-report>

Force Sandboxing

Force Sandboxing enhances system security by scrutinizing the path provided to *execve(2)* and *execveat(2)* system calls, comparing them against a predefined Integrity Force map -- a registry of path-to-checksum correlations. Upon invocation of these calls, the sandbox computes the checksum of the target binary and cross-references it with the map. Discrepancies trigger rule-defined actions: execution might proceed with a logged warning, or culminate in the termination of the process in violation. This mechanism allows for rigorous enforcement of binary integrity, echoing the preventative ethos of HardenedBSD's Integriforce and NetBSD's Veriexec by proactively mitigating unauthorised code execution, albeit with a unique emphasis on flexible, user-defined consequence management ranging from permissive alerts to stringent execution blocks.

Distinguishing itself through user-centric customization, Force Sandboxing offers a versatile approach to execution integrity. Administrators can tailor the sandbox's response to checksum mismatches -- kill, deny, or warn -- thereby balancing security needs with operational flexibility. This adaptability, combined with tools like *syd-sha(1)* for checksum calculation and *syd-path(1)* for rule creation, positions Force Sandboxing as a powerful ally in the preservation of system integrity. See *force* command in *syd(2)* manual page on how to add/remove entries to/from the Integrity Force map.

As of version 3.16.3, Syd checks the paths of the dynamic libraries an executable is linked against for force access as well. This only works for ELF files.

As of version 3.21.3, Syd hooks into *mmap(2)*, and *mmap2(2)* system calls and checks the file descriptor for Force access when the memory protection mode includes **PROT_EXEC** and flags does not include **MAP_ANONYMOUS** which typically indicates a *dlopen(3)*. Therefore libraries dynamically loaded at runtime are checked for Force access as well.

TPE sandboxing

As of version 3.21.0, Syd introduces Trusted Path Execution (TPE) sandboxing, which restricts the execution of binaries to ensure they come from *trusted directories*. As of version 3.37.2, the binary file must be *trusted* as well as its parent directory. The intention is to make privilege escalation harder when an account restricted by TPE is compromised as the attacker won't be able to execute custom binaries which are not in the trusted path. A binary is *trusted* if the file and its parent directory meet the following criteria:

- Not writable by group or others.
- Optionally owned by root, controlled by the *tpe/root_owned* option.
- Optionally owned by the current user or root, controlled by the *tpe/user_owned* option.
- Optionally part of the root filesystem, controlled by the *tpe/root_mount* option.

If these criteria are not met, the execution is denied with an **EACCES** *errno(3)*, and optionally, the offending process can be terminated with the **SIGKILL** signal using the *default/tpe:kill* option. This mechanism ensures that only binaries from secure, trusted paths can be executed, enhancing security by preventing unauthorized code execution. TPE sandboxing operates by checking the executables at three stages:

- During the system call entry of *execve(2)* and *execveat(2)* to check scripts.
- On *ptrace(2)* exec event to check the ELF executable and dynamic loader.
- On *mmap(2)* when dynamic libraries are mapped to memory, typically with *dlopen(3)*.

TPE can be configured to apply to a specific user group. By default, TPE applies to all users. However, administrators can specify an untrusted GID with the *tpe/gid* setting, restricting TPE only to users in that group. Additionally, TPE can negate GID logic with the *tpe/negate* setting, making the specified group trusted and exempt from TPE.

Syd's TPE implementation is based on HardenedBSD's which is inspired by GrSecurity's TPE. Check out the following links for more information:

- <http://phrack.org/issues/52/6.html#article>
- <http://phrack.org/issues/53/8.html#article>
- https://wiki.gentoo.org/wiki/Hardened/Grsecurity_Trusted_Path_Execution

Lock Sandboxing

Lock sandboxing utilises the **Landlock Linux Security Module** for simple unprivileged access control. It is enforced completely in kernel-space and the policy is also applied to the Syd process, such that a compromised Syd process is still stuck inside the *landlock(7)* sandbox, therefore Lock sandboxing can be used to construct a multi-layered sandbox for added security. Lock sandboxing may be turned on with the *sandbox/lock:on* sandbox command at startup. Paths to files and file hierarchies should be populated using the *lock/** categories either specifying them one at a time, e.g. *allow/lock/read+/usr*, *allow/lock/write+/dev/null* or by specifying them as a comma delimited list, e.g. *allow/lock/read,write,ioclt+/dev/null*. The shorthand *lock/all* is provided to ease configuration and it stands for the union of categories *lock/read*, *lock/write*, *lock/exec*, *lock/ioclt*, *lock/create*, *lock/delete*, *lock/rename*, *lock/symlink*, *lock/truncate*, *lock/readdir*, *lock/mkdir*, *lock/rmdir*, *lock/mkdev*, *lock/mkfifo*, and *lock/bind*. As of version 3.29.0, network confinement is supported and allowlisted *bind(2)* and *connect(2)* ports can be specified using the commands *allow/lock/bind+port* and *allow/lock/connect+port*. A closed range in format *port1-port2* may also be specified instead of a single port number. Use the *lock/bind* category with an absolute path to confine UNIX domain socket creation,

renames and links, e.g *allow/lock/bind+/run/user/\${SYD_UID}*. As of version 3.35.0, the default compatibility level has been changed to *Hard Requirement*. Compared to the old default *Best Effort*, this level ensures the sandbox is fully enforced. Moreover, **ENOENT** ("No such file or directory"), errors are made fatal in this level. The compatibility level may be changed at startup using the command *default/lock*. See the *syd(2)* manual page for more information.

Crypt Sandboxing

This sandboxing category provides transparent file encryption using AES-CTR, with HMAC-SHA256 ensuring secure data handling without manual encryption steps. When *sandbox/crypt:on* is set, files matching the *glob(3)* patterns specified by *crypt+* are encrypted on write and decrypted on read. Configuration includes specifying a 32-bit decimal encryption key serial ID for the *keyrings(7)* interface using *crypt/key/main*, and specifying a 32-bit decimal authentication key serial ID for the *keyrings(7)* interface using *crypt/key/auth*. Specifying the same key serial ID for both options is permitted and the option *crypt/key* may be used as a shorthand to set both key serial IDs. The specified key serial IDs are used with the **ALG_SET_KEY_BY_KEY_SERIAL** *setsockopt(2)* operation which is new in Linux-6.2, therefore *Crypt sandboxing requires Linux-6.2 or newer*. The keys must have *search* permission -- i.e. have the **KEY_(POS|USR|GRP|OTH)_SEARCH** permission bit(s) set so the kernel can locate and copy the key data into the crypto API; otherwise the operation will be denied (**EPERM**: "Operation not permitted"). Refer to the following link for more information <https://lkml.org/lkml/2022/10/4/1014>.

The utility *syd-key(1)* may be used to generate encryption keys and save them to *keyrings(7)* for use with Crypt sandboxing. To avoid including the key serial IDs into the configuration file, the user may set the key serial IDs using an environment variable and then specify this environment variable, e.g: *crypt/key:\${SYD_KEY_ID}*. The user *must* use an environment variable name that starts with the prefix **SYD_** but does not start with the prefix **SYD_TEST_** as such environment variables don't leak into the sandbox process. Similarly the user *must* refrain from using any environment variable specified under the ENVIRONMENT section of the *syd(1)* manual page.

Encryption operates via Linux kernel cryptography API sockets, using zero-copy techniques with *splice(2)* and *tee(2)* to avoid unencrypted data in memory. To assert we use zero-copy exclusively and respect user's privacy by avoiding to read plain-text into memory at all costs, *syd_aes* threads who are responsible for encryption are confined with a *seccomp(2)* filter to deny the *read(2)*, *open(2)*, and *socket(2)* system calls (and many more) and allow the *write(2)* system call only up to 32 bytes which is required to write the HMAC tag and the random IV to the file. The setup sockets are created on startup, the key is selected using the *keyrings(7)* interface without copying the key material into userspace. IV uniqueness is ensured by generating a random IV using *getrandom(2)* per file. In case of an error retrieving entropy via *getrandom(2)* the random bytes in **AT_RANDOM** are used instead. Per-file IV is prepended to encrypted files. This ensures security by preventing IV reuse. Syd ensures that per-file IVs are securely zeroized on drop.

A 32-byte HMAC (SHA256) message authentication tag is included between the file magic header and the IV, and is authenticated on decrypt, following the Encrypt-then-MAC approach. This provides integrity checking and resistance against bit-flip attacks. By default, decryption occurs in a memory file descriptor to prevent tampering, which limits practicality for large files due to memory constraints. User may specify a secure temporary backing directory with *crypt/tmp* to workaround this. Ideally this directory should be on encrypted storage as Syd is going to write plaintext here. File locks are set before attempting to encrypt files to ensure security and safe concurrent access. Linux OFD locks are used for locking. Encrypted data is flushed to disk only after all file descriptors that point to the encrypted open file description are closed enabling safe and performant concurrent access. File appends are handled efficiently with last block reencryption. Only regular files will be encrypted. The file format header **\x7fSYD3** identifies encrypted files and the version in the header must match the current Syd API which at the moment is **3**. Compared to GSWTK's *dbfencrypt*, Crypt sandboxing avoids TOCTOU vulnerabilities and encryption weaknesses by utilizing AES-CTR with HMAC-SHA256 and robust setup steps, providing a more secure and streamlined encryption process.

Crypt sandboxing employs the AES-CTR algorithm, a secure and efficient symmetric key encryption method suitable for various applications. It operates as a stream cipher (skcipher) with a block size of 1 byte, allowing data to be encrypted in a byte-by-byte manner. The algorithm uses a fixed key size of 32 bytes (256 bits) by default, providing robust security, and a fixed initialization vector (IV) size of 16 bytes to ensure randomness and uniqueness in each encryption operation. Processing data in byte-sized chunks, the algorithm maintains a consistent walk size of 16 bytes for traversal and operations, ensuring seamless encryption and decryption processes. This configuration, with its secure default key size,

significantly enhances security, preventing common encryption weaknesses and supporting efficient, transparent file encryption within the sandbox environment. The inclusion of HMAC-SHA256 for integrity checking further enhances security by detecting any unauthorized modifications or corruption of data. CTR is infinitely parallelizable because each block in the stream can be encrypted independently. This allows for encryption and decryption processes to be split across multiple processors, significantly increasing throughput. With hardware support such as AES-NI CPU instructions, speeds can easily exceed a gigabyte per second.

As of version 3.21.2, Syd opens memory file descriptors with the flag **MFD_NOEXEC_SEAL** during transparent decryption to ensure the memfds are non-executable and can't ever be marked executable. This ensures security as otherwise transparent decryption can be used to bypass Exec, Force and TPE sandboxing. Notably, this flag requires Linux-6.3 or newer. On older kernels, a backing directory must be specified with *crypt/tmp* for transparent decryption to work. Attempt to use transparent decryption without a backing directory on older kernels will fail with the *errno(3)* **EOPNOTSUPP** ("Operation not supported on transport endpoint"). As of version 3.28.0, Syd allows this restriction to be lifted with the option *trace/allow_unsafe_memfd:1*.

As of version 3.39.0, *keyrings(7)* interface is used for key management and specifying keys as raw payload is no longer permitted. Moving key material into the kernel *keyrings(7)* interface substantially reduces the exposure of raw keys to userland, narrowing the attack surface for memory-disclosure, core-dump, and accidental-persistence vulnerabilities while enabling cryptographic operations to be performed without copying key bytes into process memory. Because *keyrings(7)* enforce kernel-side permissions and lifecycle semantics (search/view/revoke, expiries, etc.), they provide a principled provenance and access-control model that simplifies secure rotation, auditing, and least-privilege enforcement. Together, these properties both harden the runtime security posture and facilitate integration with hardware-backed or sealed key types, improving operational compliance and reducing the likelihood of application-level key-management errors.

File Format: Each file encrypted within the Crypt sandboxing framework follows a structured format to ensure consistency, secure handling, and clear identification. Each encrypted file starts with a five-byte magic header, **\x7fSYD3**, where **\x7fSYD** indicates that the file is encrypted by Syd, and **3** denotes the current API version. This header is followed by a 32-byte HMAC (SHA256) message authentication tag, providing integrity checking by authenticating the encrypted content. Next is followed by a 16-byte initialization vector (IV), which is unique per file, ensuring strong cryptographic security. The AES-CTR-encrypted ciphertext follows the IV, providing the file's protected content. Syd will only process files that match this format and have a compatible version; if a file does not have the correct file format header or API version, or if it exists unencrypted, Syd will leave it untouched. This approach prevents unintended operations on incompatible or unencrypted files.

+-----+-----+-----+-----+			
Magic Header	HMAC Tag	Initialization Vector	Encrypted Content
"\x7fSYD3"	32 bytes (SHA256 HMAC)	16 bytes	AES-CTR Ciphertext
+-----+-----+-----+-----+			

Limitations:

- **Large files** are not handled efficiently during decryption by default due to usage of in-memory files, specify a secure temporary backing directory with *crypt/tmp:/path* to workaround this. Ideally this directory should be on encrypted storage as Syd is going to write plaintext here.
- **Concurrent Access:** Encrypted file access utilises Linux OFD locks, which are now standardized in POSIX 2024. Ensure that the underlying filesystem fully supports OFD locks to enable effective advisory file locking. Modern filesystems and NFS implementations compliant with POSIX 2024 typically provide this support, mitigating issues present in older versions. The multithreaded architecture of Syd relies on OFD locks to ensure safe and efficient concurrent access, eliminating the need for alternative locking mechanisms such as POSIX advisory locks. For further details, refer to the *fcntl_locking(2)* manual page.
- **Crash Safety:** Currently, encrypted data is flushed to disk only after all file descriptors are closed. In the event of a system or sandbox crash, this may result in incomplete writes or potential data loss, as in-flight data might not be persisted. Future enhancements will focus on implementing transactional flush mechanisms and crash recovery procedures to ensure atomicity and integrity of encrypted data, thereby improving resilience against unexpected terminations.

Utilities:

- *syd-aes(1)*: Encrypt/decrypt files akin to *openssl-enc(1ssl)*.
- *syd-key(1)* - Generate random AES-CTR keys using *getrandom(2)*, and save to *keyrings(7)*. - Read passphrases from TTY or STDIN, hash with SHA3-256, and save to *keyrings(7)*.

Proxy Sandboxing

As of version 3.22.0, Proxy sandboxing in Syd confines network communication exclusively through a designated SOCKS proxy, enforced by the helper utility *syd-tor(1)*. Configured at startup with *sandbox/proxy:on*, this type implies the use of *unshare/net:1*, isolating network namespaces to prevent direct network access. Traffic is forwarded from a specified local port (proxy/port:9050) to an external address and port (proxy/ext/host:127.0.0.1, proxy/ext/port:9050). As of version 3.34.1, you may also specify an external UNIX domain socket using e.g. proxy/ext/unix:/path/socks5.sock. This setup ensures all network interactions route through the proxy, leveraging zero-copy data transfers and edge-triggered *epoll(7)* for efficient event handling. The implementation enhances security by employing *seccomp* and *Landlock* for additional confinement, preventing unauthorized network access and ensuring strict adherence to the defined network path. This approach minimizes the risk of proxy bypasses and maintains the integrity of the network isolation.

PTY Sandboxing

As of version 3.36.0, PTY Sandboxing runs the target process inside a dedicated pseudoterminal managed by the *syd-pty(1)* helper, isolating all terminal I/O from the host TTY and preventing direct *ioctl(2)* or control-sequence escapes. The PTY main is proxied via an edge-triggered *epoll(7)* loop with non-blocking zero-copy *splice(2)*, ensuring no unencrypted data ever traverses user space. A minimal *seccomp(2)* filter confines only the essential PTY syscalls (e.g. **TIOCGWINSZ**, **TIOCSWINSZ**) and denies all others -- including injection via **TIOCSTI** -- while *Landlock* locks down access to the PTY device, filesystem, and network. Combined with no-exec memory seals and namespace isolation, this approach hardens against terminal-based attacks and preserves the confidentiality and integrity of the sandboxed session.

Memory Sandboxing

This sandboxing category handles the system calls *brk(2)*, *mmap(2)*, *mmap2(2)*, and *mremap(2)* and checks the per-process memory usage on each memory allocation request. If the memory usage reaches the maximum value defined by *mem/max*, the system call is denied with **ENOMEM**. Moreover the virtual memory size can be limited using *mem/vm_max*. If the limit is reached on the entry of any of the respective system calls, the system call is denied with **ENOMEM** and the signal **SIGKILL** is delivered to the offending process. Subsequent to the delivery of the signal, the *process_mrelease(2)* system call is called on the process to immediately release memory. The default action may be changed using the *default/mem* option. The per-process memory usage is a fair estimate calculated using the file *proc_pid_smaps(5)* summing the following fields together:

- *Pss (Proportional Set Size)* is similar to *Rss*, but accounts for shared memory more accurately by dividing it among the processes that share it. *Rss (Resident Set Size)* is the portion of memory occupied by a process that is held in RAM.
- *Private_Dirty* represents the private memory that has been modified (dirty).
- *Shared_Dirty* represents the shared memory that has been modified.

As of version 3.43.1, the memory sandboxing system has been updated to improve memory usage tracking. Syd now enforces a strict memory limit based on allocation granularity, meaning that programs cannot exceed the defined memory limits, even by the amount they allocate at once. This change aligns the limit with the allocation size rather than allowing any overflow beyond the limit. Additionally, memory tracking has been optimized by switching from iterating over *proc_pid_smaps(5)* to using the more efficient */proc/pid/smmaps_rollup*, which consolidates memory usage information for better performance and more accurate enforcement of memory constraints.

Memory sandboxing is not an alternative to cgroups(7)! You should use cgroups(7) when you can instead. This sandboxing category is meant for more constrained environments where cgroups(7) is not supported or not available due to missing permissions or other similar restrictions.

PID sandboxing

This sandboxing category handles the system calls *fork(2)*, *vfork(2)*, *clone(2)*, and *clone3(2)* and checks the total number of tasks running on the system on each process creation request. If the count reaches the maximum value defined by *pid/max*, the system call is denied with **EAGAIN**. If *pid/kill* is set to true, the signal **SIGKILL** is delivered to the offending process. This sandboxing category is best coupled with a pid namespace using *unshare/pid*. In this mode, Syd will check the number of running tasks in the current namespace only.

As of version 3.40.0, with *unshare/pid:1* the limit and accounting apply per PID namespace; on Linux 6.14 and newer the namespaced *kernel.pid_max sysctl(8)* is set to *max(pid/max, 301)* so the kernel's 300 reserved PIDs do not reduce the configured headroom, and on older kernels *kernel.pid_max sysctl(8)* is not modified.

PID sandboxing is not an alternative to cgroups(7)! You should use cgroups(7) when you can instead. This is meant for more constrained environments where cgroups(7) is not supported or not available due to missing permissions or other similar restrictions.

SafeSetID

SafeSetID, introduced in version 3.16.8, enhancing the management of UID/GID transitions. This feature enables finer-grained control by allowing administrators to explicitly specify permissible transitions for UID and GID changes, thus tightening security constraints around process privilege management. It works by allowing predefined UID and GID transitions that are explicitly configured using the *setuid+<source_uid>:<target_uid>* and *setgid+<source_gid>:<target_gid>* commands in the Syd configuration. This ensures that transitions can only occur between specified user and group IDs, and unauthorised privilege escalations are blocked. For instance, a transition might be allowed from a higher-privileged user to a less-privileged user but not vice versa, thereby preventing any escalation of privileges through these system calls.

As of version 3.24.5, Syd applies a kernel-level *seccomp(2)* filter by default to deny all *set*uid* system calls with UID less than or equal to 11 which is typically the operator user, and all *set*gid* system calls with GID less than or equal to 14 which is typically the uucp group. This means even a compromised Syd process cannot elevate privileges using these system calls. Refer to the output of the command *syd-ls setid* to see the full list of system calls in this group.

When a UID or GID transition is defined Syd will keep the **CAP_SETUID** and **CAP_SETGID** capabilities respectively and sandbox process will inherit these capabilities from Syd. Since version 3.24.6, Syd drops the **CAP_SETUID** capability after the first successful UID transition and similarly the **CAP_SETGID** capability after the first successful GID transition. This means Syd can only ever change its UID and GID once in its lifetime. However, this does not completely lock the *setid* system calls in the sandbox process: Transitions to Syd's current UID and GID are continued in the sandbox process which means the first successful UID and GID transition will continue to function as long as the sandbox process keeps the respective **CAP_SETUID**, and **CAP_SETGID** capabilities. This allows containing daemons, such as *nginx(1)*, which spawn multiple unprivileged worker processes out of a single main privileged process.

Ghost mode

Ghost Mode, introduced in Syd version 3.20.0, is a one-way sandboxing mode, closely resembling *seccomp(2)* Level 1, also known as **Strict Mode**. This mode enhances security by allowing a process to transition to a highly restrictive state after completing its initial setup. When a sandboxed process is ready for this higher level of confinement, it invokes Ghost Mode by executing the *stat(2)* system call with the virtual path */dev/syd/ghost*. Upon receiving this command, Syd closes the *seccomp_unotify(2)* file descriptor. This action elevates all previously hooked system calls to a kernel-level deny with the **ENOSYS** ("Function not implemented") *errno(3)*, effectively making them unavailable. The transition to Ghost Mode is irreversible; once the file descriptor is closed, the process is locked into this restricted

state. This mechanism ensures that the sandboxed process can only perform a very limited set of operations, akin to those allowed in `Seccomp Level 1`, thus significantly reducing its potential attack surface. Ghost Mode provides a robust security measure by denying all but the most essential system calls, which is crucial for applications that require maximum isolation and security after their initial configuration phase.

The mode is aptly named ghost because, upon closing the `seccomp_unotify(2)` file descriptor, the sandboxed process effectively detaches from Syd and becomes independent, much like a ghost. Entering ghost mode subsequently causes the `syd_mon` monitor thread and all `syd_emu` emulator threads to exit, and the remaining `syd_main` thread merely waits for the sandbox process to exit without any further intervention. This detachment underscores the finality and isolation of the Ghost Mode, ensuring that the process operates in a secure, tightly confined environment without further interaction from Syd. This mechanism is particularly useful for processes that require maximum security and minimal system call exposure after their initial configuration phase, providing a robust layer of protection against various exploits and vulnerabilities.

A process cannot enter Ghost mode once the sandbox lock is set. Alternatively, though, a process can set its process dumpable attribute to zero using the `PR_SET_DUMPABLE prctl(2)`. Under Syd, this achieves almost the same effect as Syd will not be able to emulate system calls with the per-process directory inaccessible. This provides an unprivileged way to enter Ghost mode.

SECURITY

Syd stands out for its ability to operate without requiring elevated privileges, eliminating the need for root access. This feature significantly simplifies setup and usage. Users benefit from the capability to dynamically configure the sandbox from within, with options to secure it further as needed. Tip: To take a quick peek at the `seccomp` filters applied by Syd under various different configurations, use `syd <flags...> -Epfc` where PFC stands for Pseudo Filter Code which yields a human-readable textual dump of Syd's `seccomp(2)` filters. Syd further enriches the output of this textual dump with `#` comments.

Threat Model

Syd strictly adheres to the current threat model of `seccomp(2)`. The goal is to restrict how untrusted userspace applications interact with the shared OS kernel through system calls to protect the kernel from userspace exploits (e.g., shellcode or ROP payload). The kernel is trusted. Syd's threat model delineates the sandbox as the trusted interceptor of system calls, while all user applications running within the sandbox are considered untrusted. These untrusted applications can manipulate their execution environment through syscalls, and attackers are assumed to have the capability to execute arbitrary code within these applications. Syd uses several mechanisms, including `seccomp(2)` and `ptrace(2)` for syscall filtering, `landlock(7)` for filesystem access restrictions, and `namespaces(7)` for process and device isolation, to limit the impact of these potential attacks. The threat model assumes that attackers have control over the untrusted user space and may attempt reads, writes, or arbitrary code execution that could influence the behavior of the trusted sandbox or exploit syscall handling. The security of Syd relies on the correctness of its implementation and the underlying Linux kernel features it utilises. It is assumed that there are no vulnerabilities in Syd's interception and handling of syscalls, nor in the enforcement mechanisms provided by `landlock(7)` and `namespaces(7)`. External attacks via network vectors or physical access to hardware are considered out of scope for this threat model.

”The sandbox lock” is an integral component of Syd's security architecture, which governs the configurability and integrity of the sandbox environment. By default, the sandbox lock is set to *on*, effectively preventing any further sandbox commands after the initial setup, thereby ensuring that once the sandbox is configured and the primary process is executed, the security policies remain unaltered by any untrusted processes within the sandbox. When the lock is set to *exec*, only the initial sandbox process retains the authority to access and modify the sandbox configuration, enabling a trusted process to securely establish the sandbox parameters while maintaining a *pidfd* (process ID file descriptor) to the initial process to safeguard against PID recycling attacks. Conversely, if the lock is set to *off*, all sandbox processes are permitted to access and modify the sandbox configuration, allowing for broader configurability during the setup phase. However, this state persists only until the sandbox is explicitly locked, after which the lock becomes immutable and the sandbox policies are fixed, preventing any subsequent processes from altering the configuration. This layered

locking mechanism, reinforced by the use of *pidfd* in *exec* mode, effectively safeguards against untrusted processes attempting to modify sandbox settings to escalate privileges or circumvent restrictions, thereby maintaining a robust and secure execution environment within Syd's framework. In *ipc* mode, the sandbox configuration is accessible through a UNIX socket which may or may not be accessible from within the sandbox depending on sandbox ACL rules. In *read* mode, the sandbox configuration is accessible only to reads, but NOT edits. Transition from lock modes *off*, *exec*, and *ipc* into one of *read* and *on* is one-way and idempotent: It results in the sandbox policy getting sealed in memory using the *mseal(2)* system call either immediately or simultaneously with sandbox process startup. Transitions between lock modes *read* and *on* are not permitted.

”Crypt Sandboxing” in Syd ensures the confidentiality and integrity of specified files by transparently encrypting them using AES-CTR with HMAC-SHA256, even when adversaries fully control processes within the sandbox (i.e., attackers can execute arbitrary code and perform any allowed system calls). In this extended threat model, it is acknowledged that while attackers may access plaintext data within the sandbox's memory during process execution, they cannot extract encryption keys or plaintext data from outside the controlled environment, nor can they interfere with the encryption process to leak keys or plaintext to persistent storage or external channels. Cryptographic operations are performed via kernel-level cryptography API sockets using zero-copy techniques to prevent plaintext from residing in user-space memory buffers accessible to attackers. The *syd_aes* threads responsible for encryption are confined with strict *seccomp(2)* filters, denying them critical system calls like *read(2)*, *open(2)*, and *socket(2)*, and allowing only minimal *write(2)* operations required for encryption metadata (e.g., writing the HMAC tag and random IV to the file). This confinement prevents exploitation that could leak sensitive data. Encryption keys are handled using kernel *keyrings(7)* interface and the **ALG_SET_KEY_BY_KEY_SERIAL** *setsockopt(2)* option. The threat model trusts the kernel and Syd's implementation, assuming attackers cannot exploit kernel vulnerabilities to access keys or plaintext within kernel memory or cryptographic operations. Additionally, file locks are employed before attempting to encrypt files to ensure safe concurrent access. In contrast to the general threat model, Crypt Sandboxing acknowledges that untrusted processes within the sandbox have access to plaintext data in memory during normal operation, as they need to read or write the plaintext files. However, the goal is to prevent attackers from accessing the plaintext outside the controlled environment or tampering with the encryption process to compromise confidentiality and integrity. This is achieved by ensuring that the encryption keys remain secure and that the encryption and decryption processes are tightly controlled and isolated from untrusted code.

Accessing remote process memory

Syd denies various system calls which can access remote process memory such as *ptrace(2)* and *process_vm_writev(2)* and common sandboxing profiles such as *paludis* and *user* disallow write access to the */proc/pid/mem* file. This makes TOCTOU attack vectors harder to realise. Refer to the the output of the command *syd-ls deny* to see the full list of denied system calls.

Enhanced Handling of PTRACE_TRACEME

As of version 3.16.3, Syd introduced a new feature for managing the **PTRACE_TRACEME** operation, aimed at improving stealth against detection. Traditionally, **PTRACE_TRACEME** is the only *ptrace(2)* operation allowed by a tracee, which makes it a common target for detection of ptracers. By converting **PTRACE_TRACEME** into a no-operation (no-op) that always succeeds, Syd aims to subtly prevent simple detection methods that rely on this operation. Additionally, other *ptrace(2)* operations are modified to return an **EPERM** (“Operation not permitted”) *errno(3)* instead of **ENOSYS** (“Function not implemented”), which helps reduce the likelihood of the sandbox being detected through these errors. This approach enhances the discreetness of Syd's operation by mitigating straightforward detection tactics used by monitored processes.

As of version 3.19.0, Syd extends this mitigation and turns the system call *ptrace(2)* into a no-op. Again, this provides a best-effort mitigation against using requests such as **PTRACE_ATTACH** or **PTRACE_SEIZE** to detect a ptracer.

Note, this mitigation is simple and zero-cost, however a clever *ptrace(2)* detector can bypass it with e.g. a double *ptrace(2)* as exemplified here: <https://arxiv.org/pdf/2109.06127>

```
if (ptrace(PTRACE_TRACEME)==0 && ptrace(PTRACE_TRACEME)==-1){
    evade();
}
```

Since this example relies on internal function states and side-effects, it bypasses Syd's mitigation. In such cases, user may opt for the option *trace/allow_unsafe_ptrace:1*, when Syd will not use *ptrace(2)* at all, hence there is going to be no ptracer to detect for the malware with the logic bomb.

Hardened procfs and devfs

To enhance system security and mitigate potential attack vectors, Syd enforces restrictions on *procfs(5)* and *devfs* file systems by implementing several key measures: denying both the listing and opening of block devices and files of unknown types by omitting entries corresponding to these file types (identified by **DT_BLK** and **DT_UNKNOWN**) from directory listings and rejecting *open(2)* operations on them. This prevents unauthorized enumeration and access to system storage devices, thereby mitigating information disclosure and potential tampering.

Syd also restricts visibility within the */proc* directory so that processes can only see their own process IDs, effectively preventing discovery and potential interaction with other running processes, which reduces risks of information leakage, privilege escalation, and process manipulation. Access to the */proc* entries of the Syd process itself is explicitly denied, safeguarding the sandbox manager from inspection or interference and preventing access to sensitive information about the sandboxing mechanism that could be exploited to bypass security controls or escape the sandbox.

Additionally, Syd addresses risks associated with magic symbolic links in */proc* -- such as */proc/[pid]/exe* and */proc/[pid]/fd/** -- by denying access to these links when they refer to processes other than the calling process, thus preventing exposure of sensitive file descriptors or executable paths of other processes and mitigating unauthorized access or container escape scenarios; this mitigation can be disabled with the *trace/allow_unsafe_magiclinks:1* option if necessary, though doing so is not recommended.

Collectively, these hardened controls over *procfs* and *devfs* significantly reduce the attack surface by preventing information disclosure, unauthorized access, and potential privilege escalations, ensuring that sandboxed applications operate within a tightly controlled and secure environment that adheres to the principle of least privilege and maintains system integrity. Refer to the following links for more information:

- <https://forums.whonix.org/t/proc-pid-sched-spy-on-keystrokes-proof-of-concept-spy-gksu/8225>
- <https://homes.luddy.indiana.edu/xw7/papers/zhou2013identity.pdf>
- https://petsymposium.org/2016/files/papers/Don%E2%80%99t_Interrupt_Me_While_I_Type__Inferring_Text_Enter
- <https://staff.ie.cuhk.edu.hk/~khzhang/my-papers/2016-oakland-interrupt.pdf>
- https://www.cs.ucr.edu/~zhiyunq/pub/sec14_android_activity_inference.pdf
- <https://www.gruss.cc/files/procharvester.pdf>
- https://www.kicksecure.com/wiki/Dev/Strong_Linux_User_Account_Isolation#/proc/pid/sched_spy_on_keystrokes
- <https://www.openwall.com/lists/oss-security/2011/11/05/3>
- https://www.usenix.org/legacy/event/sec09/tech/full_papers/zhang.pdf
- <https://www.openwall.com/lists/oss-security/2025/11/05/3>

Hardened `proc_pid_status(5)`

As of version 3.38.0, Syd filters `proc_pid_status(5)` at `open(2)` boundary to defeat common sandbox-fingerprinting heuristics while preserving compatibility with ordinary tooling. When a process (or its threads) reads `/proc/<pid>/status` or `/proc/<pid>/task/<tid>/status`, Syd normalizes only the security-critical fields -- zeroing `TracerPid`, `NoNewPrivs`, `Seccomp`, and `Seccomp_filters`, and rewriting the sandbox-revealing phrases in `Speculation_Store_Bypass` and `SpeculationIndirectBranch`. This targeted normalization breaks trivial anti-analysis checks (ptracer presence, seccomp/no_new_privs probes, speculative mitigation fingerprints) without altering process state.

The security impact is twofold: untrusted code loses a low-cost oracle for environment discovery, reducing the likelihood of logic bombs or capability gating based on sandbox detection, and defenders retain observability because the kernel's real enforcement still applies -- only the user-space view of these select fields is masked. For forensic and debugging workflows that explicitly need the unfiltered view, this mitigation can be temporarily relaxed per trace with `trace/allow_unsafe_proc_pid_status:1`, after which toggling back to `:0` restores the hardened, stealth-preserving default.

Hardened `uname(2)`

As of version 3.15.1, Syd mediates `uname(2)` and returns a policy governed `utsname` that suppresses host identification and constrains kernel disclosure. The release string is synthesized to expose only the Linux major and minor as observed on the host or, as of 3.36.1, as supplied via **SYD_ASSUME_KERNEL** for controlled feature detection, while the micro component is randomized per Syd run to limit patch level fingerprinting; reads of `/proc/version` and `/proc/sys/kernel/osrelease` are hardened to present the same masked view. As of 3.40.0, the nodename, domainname, and version fields are sourced from the options `uts/host`, `uts/domain`, and `uts/version` with defaults `localhost`, `(none)`, and a startup random value. As of 3.44.2, this restriction may be relaxed at startup with the option `trace/allow_unsafe_uname:1`. Practical effects include disrupting exploit and loader selection that depend on exact release matching, reducing cross host correlation via stable node and domain labels, neutralizing sandbox and VM fingerprinting heuristics that key off `uname(2)` and the corresponding `proc(5)` paths, and keeping build and compatibility probes functional by retaining `major.minor` semantics while allowing explicit control through **SYD_ASSUME_KERNEL**. Workloads that tie licensing, clustering, telemetry, or feature gates to the precise host release or to the original nodename should use the `uts` options to supply the required identity or opt out with the relaxation flag.

Denying **TIOCLINUX** `ioctl`

The limitation on the use of the **TIOCLINUX** `ioctl(2)` within secure environments, similar to the Syd sandbox, is an essential security measure addressing vulnerabilities specific to Linux terminal operations. The **TIOCLINUX** `ioctl(2)` command offers various functionalities, including but not limited to manipulating console settings, changing keyboard modes, and controlling screen output. While these capabilities can be leveraged for legitimate system management tasks, they also introduce potential security risks, particularly in multi-user environments or in the context of sandboxed applications.

The security concerns surrounding **TIOCLINUX** stem from its ability to alter terminal behaviors and settings in ways that could be exploited for unauthorised information disclosure, terminal hijacking, or privilege escalation. For instance, manipulating the console display could mislead users about the true nature of the operations being executed, or altering keyboard settings could capture or inject keystrokes.

In summary, the restriction on **TIOCLINUX** within secure environments is a vital security strategy, addressing the complex risks associated with direct terminal manipulation capabilities. This precaution is in keeping with the broader security community's efforts to mitigate known vulnerabilities and enhance the security posture of systems handling sensitive processes and data.

Denying TIOCSTI ioctl

The restriction on the use of the **TIOCSTI** *ioctl*(2) within the Syd sandbox addresses a significant security vulnerability associated with terminal input injection. The **TIOCSTI** *ioctl*(2) allows a byte to be inserted into the terminal input queue, effectively simulating keyboard input. This capability, while potentially useful for legitimate purposes, poses a *substantial security risk*, especially in scenarios where a process might retain access to a terminal beyond its intended lifespan. Malicious use of this *ioctl*(2) can lead to the injection of commands that execute with the privileges of the terminal's owning process, thereby breaching the security boundaries intended by user permissions and process isolation mechanisms. The concern over **TIOCSTI** is well-documented in the security community. For example, OpenBSD has taken measures to mitigate the risk by disabling the **TIOCSTI** *ioctl*(2), reflecting its stance on the *ioctl*(2) as *one of the most* dangerous due to its potential for abuse in command injection attacks. The decision to disable or restrict **TIOCSTI** in various Unix-like operating systems underscores the *ioctl*(2)'s inherent security implications, particularly in the context of privilege escalation and the execution of unauthorised commands within a secured environment.

In summary, the restriction on **TIOCSTI** within Syd is a critical security measure that prevents a class of vulnerabilities centered around terminal input injection, safeguarding against unauthorised command execution and privilege escalation. This precaution aligns with broader security best practices and mitigations adopted by the security community to address known risks associated with terminal handling and process isolation.

Denying FS_IOC_SETFLAGS ioctl

As of version 3.24.2, Syd denies the **FS_IOC_SETFLAGS** *ioctl*(2) request by default, a critical security measure to ensure that once file flags are set, they remain unchanged throughout the runtime of the sandbox. This policy is particularly focused on the *immutable* and *append-only* flags, which need to be configured by an administrator at the start of the Syd process. Once these attributes are set on crucial system and log files -- marking them either as immutable to prevent any modification, or append-only to ensure that existing data cannot be erased -- they are frozen. This means that no subsequent modifications can be made to these attributes, effectively locking down the security settings of the files against any changes. This approach prevents scenarios where, even after a potential security breach, malicious entities are unable to alter or delete important files, thus maintaining the integrity and reliability of the system against tampering and ensuring that audit trails are preserved.

Denying PR_SET_MM prctl

The **PR_SET_MM** *prctl*(2) call allows processes with the **CAP_SYS_RESOURCE** capability to adjust their memory map descriptors, facilitating operations like self-modifying code by enabling dynamic changes to the process's memory layout. For enhanced security, especially in constrained environments like Syd, this capability is restricted to prevent unauthorised memory manipulations that could lead to vulnerabilities such as code injection or unauthorised code execution. Notably, Syd proactively drops **CAP_SYS_RESOURCE** among other capabilities at startup to minimise security risks. This action is part of Syd's broader security strategy to limit potential attack vectors by restricting process capabilities.

Restricting prctl option space and trace/allow_unsafe_prctl

Syd meticulously confines the scope of permissible *prctl*(2) operations to enhance security within its sandbox environment. By limiting available *prctl*(2) options to a specific set, including but not limited to **PR_SET_PDEATHSIG**, **PR_GET_DUMPABLE**, **PR_SET_NO_NEW_PRIVS**, and **PR_SET_SECCOMP**, Syd ensures that only necessary process control functionalities are accessible, thereby reducing the risk of exploitation through less scrutinised *prctl*(2) calls. This constraint is pivotal in preventing potential security vulnerabilities associated with broader *prctl*(2) access, such as unauthorised privilege escalations or manipulations of process execution states. However, recognizing the need for flexibility in certain scenarios, Syd offers the option to lift these restrictions through the *trace/allow_unsafe_prctl:1* setting. This capability allows for a tailored security posture, where users can opt for a more permissive *prctl*(2) environment if required by their specific use case, while still maintaining awareness of the increased security risks involved.

Restricting `io_uring` interface and `trace/allow_unsafe_uring`

The `io_uring(7)` interface can be used to *bypass path sandboxing*. By default, Syd restricts `io_uring(7)` operations due to their ability to perform system calls that could undermine the sandbox's security controls, particularly those designed to limit file access and modify file permissions. The setting, `trace/allow_unsafe_uring`, when enabled, relaxes these restrictions, allowing `io_uring(7)` operations to proceed unimpeded. While this can significantly enhance I/O performance for applications that rely on `io_uring(7)` for efficient asynchronous operations, it requires careful consideration of the security implications, ensuring that its use does not inadvertently compromise the sandboxed application's security posture. Refer to the output of the command `syd-ls uring` to see the full list of system calls that belong to the `io_uring(7)` interface.

Restricting creation of device special files

Since version 3.1.12, Syd has enhanced its security model by disallowing the creation of device special files through the `mknod(2)` and `mknodat(2)` system calls. This decision is rooted in mitigating potential security vulnerabilities, as device special files could be exploited to circumvent established path-based access controls within the sandbox environment. These files, which include character and block devices, can provide direct access to hardware components or facilitate interactions with kernel modules that could lead to unauthorised actions or data exposure. By restricting their creation, Syd significantly reduces the risk of such exploit paths, reinforcing the integrity and security of the sandboxed applications. This measure ensures that only predefined types of files -- such as FIFOs, regular files, and sockets -- are permissible, aligning with the principle of least privilege by limiting file system operations to those deemed safe within the sandbox's context.

Sharing Pid namespace with signal protections

Since version 3.6.7, Syd has introduced a crucial security feature that prevents sandboxed processes from sending signals to the Syd process or any of its threads. This protection is implemented by hooking and monitoring system calls related to signal operations, including `kill(2)`, `tkill(2)`, `tgkill(2)`, and `pidfd_open(2)`. When a sandboxed process attempts to send a signal to Syd or its threads, these system calls are intercepted, and the operation is denied at the seccomp level with an **EACCES** ("Permission denied") `errno(3)`. This measure ensures that Syd maintains control over the execution and management of sandboxed processes, safeguarding against interruptions or unauthorised interactions that could compromise the security or stability of the sandbox environment. This security mechanism is part of Syd's broader strategy to share the same root, private proc, and mount namespaces with the sandboxed process, facilitating secure and simple system call emulation. By making Syd and its threads immune to signals from sandboxed processes, the integrity and isolation of the sandboxed environment are significantly enhanced, preventing potential exploitation scenarios where sandboxed processes could disrupt the operation of the sandbox manager or interfere with other sandboxed processes.

As of version 3.35.2, Syd puts itself in a new process group using `setpgid(2)` and releases the controlling terminal using the **TIOCNOTTY** `ioctl(2)` request. Moreover a scope-only Landlock sandbox is installed unconditionally to further isolate the sandbox process from the Syd process. This ensures that terminal-generated signals and I/O remain confined to the sandbox's process group and cannot affect Syd or any other processes, further strengthening the sandbox's isolation guarantees alongside the existing seccomp-based PID namespace protections.

Process Priority and Resource Management

Since version 3.8.1, Syd has been implementing strategies to ensure the smooth operation of the host system while managing security through its sandboxing mechanism. It sets the `nice(2)` value of its system call handler threads to `19`, ensuring these threads operate at *the lowest* priority to minimise CPU starvation for other critical processes. This approach prioritises system stability and fair CPU resource distribution, enabling Syd to handle numerous system calls without compromising the host's performance and responsiveness.

Enhancing this strategy, Syd introduced further adjustments in versions 3.8.6 and 3.9.7 to address I/O and CPU resource management more comprehensively. From version 3.8.6, it sets the I/O priority of the system call handler threads to *idle*, ensuring that I/O operations do not monopolise resources and lead to I/O starvation for other processes. Similarly, from version 3.9.7, it adjusts the CPU scheduling priority of these threads to *idle*, further safeguarding against CPU starvation. These measures collectively ensure that Syd maintains optimal performance and system responsiveness while securely sandboxing applications, striking a balance between security enforcement and efficient system resource utilization.

As of version 3.30.0, changes in process and I/O priorities are inherited by sandbox processes as well and sandbox processes are prevented from making any further changes. Moreover, the option *trace/allow_unsafe_nice* may be set at startup to prevent Syd from making any changes and allow sandbox processes access to the system calls that are used to make process and I/O priority changes.

Streamlining File Synchronization Calls

As of version 3.8.8, Syd has rendered the *sync(2)* and *syncfs(2)* system calls as no-operations (no-ops), ensuring they report success without executing any underlying functionality. This adjustment is designed to streamline operations within the sandboxed environment, bypassing the need for these file synchronization actions that could otherwise impact performance or complicate the sandbox's control over file system interactions. By adopting this approach, Syd enhances its compatibility with applications that issue these calls, without altering the sandboxed process's behavior or the integrity of file system management. As of version 3.28.0, this restriction can be disabled at startup with the option *trace/allow_unsafe_sync:1*. This is useful in scenarios where *sync* is actually expected to work such as when sandboxing databases.

Restricting Resource Limits, Core Dumps, and *trace/allow_unsafe_prlimit*

Since version 3.9.6, Syd has implemented restrictions on setting process resource limits and generating core dumps for the sandboxed process, enhancing the sandbox's security posture. This measure prevents the sandboxed process from altering its own resource consumption boundaries or producing core dumps, which could potentially leak sensitive information or be exploited for bypassing sandbox restrictions. However, recognizing the need for flexibility in certain use cases, Syd provides the option to disable these restrictions at startup through the *trace/allow_unsafe_prlimit:1* setting. This allows administrators to tailor the sandbox's behavior to specific requirements, balancing security considerations with functional needs.

Enhancing Sandbox Security with Landlock

Since version 3.0.1, Syd leverages *landlock(7)* to enforce advanced filesystem sandboxing, significantly bolstering the security framework within which sandboxed processes operate. By integrating Landlock, Syd empowers even unprivileged processes to create secure sandboxes, enabling fine-grained access control over filesystem operations without requiring elevated permissions. This approach is instrumental in mitigating the risk of security breaches stemming from bugs or malicious behaviors in applications, offering a robust layer of protection by restricting ambient rights, such as global filesystem or network access. Landlock operates by allowing processes to self-impose restrictions on their access to system resources, effectively creating a secure environment that limits their operation to a specified set of files and directories. This mechanism is particularly useful for running legacy daemons or applications that require specific environmental setups, as it allows for the precise tailoring of access rights, ensuring processes can only interact with designated parts of the filesystem. For instance, by setting Landlock rules, Syd can confine a process's filesystem interactions to read-only or read-write operations on explicitly allowed paths, thus preventing unauthorised access to sensitive areas of the system.

Furthermore, the inclusion of the Syd process itself within the Landlock-enforced sandbox adds an additional layer of security. This design choice ensures that even if the Syd process were compromised, the attacker's ability to manipulate the sandboxed environment or access unauthorised resources would be significantly constrained. This self-sandboxing feature underscores Syd's commitment to maintaining a high security standard, offering peace of mind to users by ensuring comprehensive containment of sandboxed processes.

Namespace Isolation in Syd

Syd enhances sandbox isolation through meticulous namespace use, starting from version 3.0.2. Version 3.9.10 marks a pivotal enhancement by restricting user subnamespace creation, addressing a key path sandboxing bypass vulnerability. This strategic limitation thwarts sandboxed processes from altering their namespace environment to access restricted filesystem areas. Furthermore, since version 3.11.2, Syd maintains process capabilities within user namespaces, mirroring the *unshare*(1) command's *--keep-caps* behavior. This ensures sandboxed processes retain necessary operational capabilities, enhancing security without compromising functionality. Additionally, Syd utilises the powerful *bind* command within the mount namespace to create secure, isolated environments by allowing specific filesystem locations to be remounted with custom attributes, such as *ro*, *noexec*, *nosuid*, *nodev*, or *nosymfollow*, providing a flexible tool for further restricting sandboxed processes' access to the filesystem.

Syd also introduces enhanced isolation within the mount namespace by offering options to bind mount temporary directories over */dev/shm* and */tmp*, ensuring that sandboxed processes have private instances of these directories. This prevents inter-process communication through shared memory and mitigates the risk of temporary file-based attacks, further solidifying the sandbox's defence mechanisms. As of version 3.35.2, an empty mount namespace may be built from scratch starting with the *root:tmpfs* command. As of version 3.11.2, Syd mounts the *procfs*(5) filesystem privately with the *hidepid=2* option, enhancing privacy by concealing process information from unauthorised users. As of version 3.37.2, this option is changed to *hidepid=4* which is new in Linux ≥ 5.8 for added hardening. As of version 3.39.0 the option *subset=pid* is also supplied to private *procfs*(5) mount for added hardening. This option is also new in Linux ≥ 5.8 .

Syd's *container* and *immutable* profiles exemplify its adaptability, offering from isolated to highly restrictive environments. The container profile provides a general-purpose sandbox, while the immutable profile enforces stricter controls, such as making essential system directories read-only, to prevent tampering. This comprehensive approach underlines Syd's adept use of kernel features for robust sandbox security, ensuring a secure and controlled execution environment for sandboxed applications. See *syd-cat -pcontainer*, and *syd-cat -pimmutable* to list the rules in these sandboxing profiles.

As of version 3.23.0, Syd has further strengthened its security with the introduction of a time namespace, represented by the *unshare/time:1* option, allows Syd to reset the boot-time clock, ensuring that the *uptime*(1) command reports container uptime instead of host uptime. Moreover, the creation of namespaces, including mount, UTS, IPC, user, PID, net, cgroup, and time is denied by default to prevent unauthorized namespace manipulation that could undermine path sandboxing security. To allow specific namespace types, administrators must explicitly enable them via the *trace/allow_unsafe_namespace* setting. Another restriction to note is that the system calls *mount*(2), *mount_setattr*(2), *umount*(2), and *umount2*(2) are denied by default unless *mount* namespace is allowed. This change ensures tighter control over process capabilities and isolation, reinforcing the defense mechanisms against potential security breaches.

Restricting environment and trace/allow_unsafe_env

As of version 3.11.1, Syd has implemented measures to clear unsafe environment variables, such as **LD_PRELOAD**, enhancing security by preventing the manipulation of dynamic linker behavior by sandboxed processes. This action mitigates risks associated with dynamic linker hijacking, where adversaries may load malicious shared libraries to execute unauthorised code, potentially leading to privilege escalation, persistence, or defence evasion. Variables like **LD_PRELOAD** allow specifying additional shared objects to be loaded before any others, which could be exploited to override legitimate functions with malicious ones, thus hijacking the execution flow of a program. To accommodate scenarios where developers might need to use these variables for legitimate purposes, Syd allows this security feature to be disabled at startup with *trace/allow_unsafe_env:1*, offering flexibility while maintaining a strong security posture. This careful balance ensures that sandboxed applications operate within a tightly controlled environment, significantly reducing the attack surface and enhancing the overall security framework within which these applications run. Refer to the output of the command *syd-ls env* to see the full list of environment variables that Syd clears from the environment of the sandbox process. As of version 3.39.0, Syd additionally clears **LANG** and the full set of **LC_*** locale variables (e.g. **LC_CTYPE**, **LC_TIME**, **LC_ALL**, etc.) to avoid leaking locale settings into the sandboxed process -- preventing subtle behavior differences or information disclosure that could be abused. Similarly, the **TZ** variable is cleared to prevent leaking timezone settings to the sandbox process. The builtin *linux* profile masks the file */etc/localtime* and the *glob*(3p) pattern */usr/share/zoneinfo/*** with the file */usr/share/zoneinfo/UTC* preventing another vector of

timezone settings leaking into the environment of the sandbox process. For controlled exceptions, the CLI `-e` flag provides fine-grained control: `-e var=val` injects `var=val` into the child environment, `-e var` removes `var` from the child environment, and `-e var=` explicitly passes through an otherwise unsafe variable; any of these forms may be repeated as needed.

Managing Linux Capabilities for Enhanced Security

Since its 3.0.17 release, Syd strategically curtails specific Linux *capabilities(7)* for sandboxed processes to bolster security. By revoking privileges such as **CAP_SYS_ADMIN** among others, Syd significantly reduces the risk of privilege escalation and system compromise. This proactive measure ensures that even if a sandboxed process is compromised, its ability to perform sensitive operations is severely limited. The comprehensive list of dropped capabilities, including but not limited to **CAP_NET_ADMIN**, **CAP_SYS_MODULE**, and **CAP_SYS_RAWIO**, reflects a meticulous approach to minimizing the attack surface. Refer to the output of the command `syd-ls drop` to see the full list of *capabilities(7)* that Syd drops at startup.

Exceptions to this stringent policy, introduced in version 3.11.1, such as retaining **CAP_NET_BIND_SERVICE** with `trace/allow_unsafe_bind:1`, **CAP_NET_RAW** with `trace/allow_unsafe_socket:1`, **CAP_SYSLOG** with `trace/allow_unsafe_syslog:1` and **CAP_SYS_TIME** with `trace/allow_unsafe_time:1`, offer a nuanced security model. These exceptions allow for necessary network, syslog and time adjustments within the sandbox, providing flexibility without significantly compromising security.

Since version 3.12.5, Syd allows the user to prevent dropping capabilities at startup using the command `trace/allow_unsafe_caps:1`. This command may be used to construct privileged containers with Syd.

This balanced strategy of restricting *capabilities(7)*, coupled with selective permissions, exemplifies Syd's commitment to crafting a secure yet functional sandbox environment. By leveraging the granularity of Linux *capabilities(7)*, Syd offers a robust framework for safeguarding applications against a variety of threats, underscoring its role as a pivotal tool in the security arsenal of Linux environments.

Path Resolution Restriction For Chdir and Open Calls

In Syd version 3.15.1, a configurable security feature is available to address the risk of directory traversal attacks by restricting the use of `..` components in path arguments for `chdir(2)`, `open(2)`, `openat(2)`, `openat2(2)`, and `creat(2)` system calls. This feature is off by default, ensuring broad compatibility and operational flexibility for a range of applications. When enabled with the `trace/deny_dotdot:1` command, Syd strengthens its defence mechanisms against unauthorised directory access, echoing the flexibility seen in FreeBSD's `vfs.lookup_cap_dotdot` sysctl. This allows for a nuanced approach to filesystem security, where administrators can tailor the sandbox's behavior to match specific security requirements or operational contexts. By drawing on the security insights of FreeBSD and HardenedBSD, Syd provides a versatile toolset for managing path traversal security, adaptable to the unique demands of various application environments. See the following links for more information:

- [https://man.freebsd.org/cgi/man.cgi?open\(2\)](https://man.freebsd.org/cgi/man.cgi?open(2))
- https://cgит.freebsd.org/src/tree/sys/kern/vfs_lookup.c#n351

Enhanced Symbolic Link Validation

As of version 3.13.0, Syd enhances security by enforcing stricter validation on symbolic links within `/proc/pid/fd`, `/proc/pid/cwd`, `/proc/pid/exe`, and `/proc/pid/root`, addressing potential misuse in container escape scenarios. Specifically, Syd returns an **EACCES** ("Permission denied") `errno(3)` for attempts to resolve these symbolic links if they do not pertain to the *current process*, akin to implementing **RESOLVE_NO_MAGICLINKS** behavior of the `openat2(2)` system call. This measure effectively hardens the sandbox against attacks exploiting these links to access resources outside the intended confinement, bolstering the isolation provided by Syd and mitigating common vectors for privilege escalation and sandbox escape. As of version 3.14.5, Syd keeps intercepting path system calls even if sandboxing is off making this protection unconditional.

Trusted Symbolic Links

As of version 3.37.2, Syd implements a robust symbolic-link hardening mechanism that intercepts every *symlink(7)* resolution within untrusted directories -- those marked world-writable, group-writable, or carrying the sticky bit -- and denies any follow operation, returning **EACCES** ("Permission denied"); this behavior mirrors GrSecurity's **CONFIG_GRKERNSEC_LINK** and guarantees that symlink chains in shared or temporary locations cannot be weaponized for TOCTOU or link-trick exploits. Under the default policy, neither direct nor nested symlinks in untrusted paths will be traversed, and the check is applied at the *seccomp(2)* interception layer prior to any mutable state changes -- ensuring an early, fail-close enforcement. Administrators may relax this restriction at startup or runtime by enabling the *trace/allow_unsafe_symlinks:1* option, which restores legacy symlink behavior for compatibility at the cost of re-exposing potential link-based race vulnerabilities. Refer to the following links for more information:

- https://wiki.gentoo.org/wiki/Hardened/Grsecurity2_Quickstart
- https://en.wikibooks.org/wiki/Grsecurity/Appendix/Grsecurity_and_PaX_Configuration_Options#Linking_restrictions
- https://xorl.wordpress.com/2010/11/11/grkernsec_link-linking-restrictions/
- https://man7.org/linux/man-pages/man5/proc_sys_fs.5.html

Trusted Hardlinks

As of version 3.37.4, Syd introduces a comprehensive *Trusted Hardlinks* policy to mitigate a class of vulnerabilities stemming from unsafe hardlink creation, particularly those enabling time-of-check-to-time-of-use (TOCTOU) exploitation and privilege escalation in shared filesystem environments. This mitigation enforces strict constraints on which files may be linked, based on their visibility, mutability, and privilege-related attributes. A file is permitted as a hardlink target only if it is accessible for both reading and writing by the caller, ensuring that immutable or opaque targets cannot be leveraged in multi-stage attack chains. Furthermore, the file must be a regular file and must not possess privilege-escalation enablers such as the set-user-ID bit or a combination of set-group-ID and group-executable permissions. These checks are performed preemptively and unconditionally during syscall handling to eliminate reliance on ambient filesystem state and to maintain integrity under adversarial conditions. Administrators may relax this policy for compatibility purposes using the *trace/allow_unsafe_hardlinks:1* option, though doing so reintroduces well-documented attack surfaces and undermines the guarantees provided by Syd's secure execution model. Refer to the following links for more information:

- https://wiki.gentoo.org/wiki/Hardened/Grsecurity2_Quickstart
- https://en.wikibooks.org/wiki/Grsecurity/Appendix/Grsecurity_and_PaX_Configuration_Options#Linking_restrictions
- https://xorl.wordpress.com/2010/11/11/grkernsec_link-linking-restrictions/
- https://man7.org/linux/man-pages/man5/proc_sys_fs.5.html

Trusted File Creation

As of version 3.37.4, Syd enforces a strict *Trusted File Creation* policy designed to mitigate longstanding race-condition vulnerabilities associated with unprivileged use of **O_CREAT** in shared or adversarial environments. Building upon the Linux kernel's *protected_fifos* and *protected_regular* sysctls -- as well as the stricter semantics of grsecurity's **CONFIG_GRKERNSEC_FIFO** -- this mitigation blocks all **O_CREAT** operations targeting pre-existing FIFOs or regular files unless the calling process is the file's owner and the file is neither group-writable nor world-writable, irrespective of the parent directory's ownership or permissions. Unlike upstream Linux, which allows certain accesses if the file resides in a directory owned by the caller, Syd eliminates this dependency to close subtle privilege boundary gaps and ensure consistent, capability-centric enforcement even in nested namespace or idmapped mount scenarios. This policy guarantees that users cannot preempt or hijack file-based IPC or partial writes via shared directories, while maintaining usability through precise capability trimming. For compatibility with legacy workloads or permissive setups, this restriction may be selectively disabled by setting the *trace/allow_unsafe_create:1* option, though doing so reintroduces exposure to well-documented filesystem race attacks.

As of version 3.45.0, Syd extends this policy to deny file creation through dangling symbolic links as part of its filesystem race hardening. At the *open(2)* boundary, the presence of **O_CREAT** implicitly adds **O_NOFOLLOW** unless **O_EXCL** is also specified, so attempts to create or truncate a path whose final component is a symlink will fail rather than resolving the link target. This behaviour directly addresses classes of vulnerabilities where privileged components are tricked into creating or modifying files behind attacker-controlled symlinks, such as CVE-2021-28153 in GLib (file creation via dangling symlink replacement) and repeated symlink- or mount-race attacks in container runtimes: CVE-2018-15664 (docker cp path traversal via symlink and mount races), CVE-2019-16884 (runc bind-mount escape through user-controlled symlinked host paths), CVE-2021-30465 (runc container escape via crafted /proc and mount races), CVE-2025-31133 (runc maskedPath abuse to obtain writable procfs bindings), CVE-2025-52565 (runc /dev/console bind-mount symlink races leading to writable procfs targets), and CVE-2025-52881 (runc redirected writes bypassing LSM enforcement to arbitrary procfs files). By enforcing fail-closed semantics for all **O_CREAT** operations that encounter symlinks, Syd reduces the attack surface for these patterns even when higher-level code assumes symbolic links cannot influence file creation. Refer to the following links for more information:

- https://wiki.gentoo.org/wiki/Hardened/Grsecurity2_Quickstart
- https://en.wikibooks.org/wiki/Grsecurity/Appendix/Grsecurity_and_PaX_Configuration_Options#FIFO_restrictions
- https://xorl.wordpress.com/2010/11/24/grkernsec_fifo-named-pipe-restrictions/
- https://man7.org/linux/man-pages/man5/proc_sys_fs.5.html
- <https://nvd.nist.gov/vuln/detail/CVE-2021-28153>
- <https://github.com/advisories/GHSA-9hh6-p5c5-mmmf>
- <https://nvd.nist.gov/vuln/detail/CVE-2018-15664>
- <https://nvd.nist.gov/vuln/detail/CVE-2019-16884>
- <https://nvd.nist.gov/vuln/detail/CVE-2021-30465>
- <https://nvd.nist.gov/vuln/detail/CVE-2025-31133>
- <https://nvd.nist.gov/vuln/detail/CVE-2025-52565>
- <https://nvd.nist.gov/vuln/detail/CVE-2025-52881>
- <https://www.openwall.com/lists/oss-security/2025/11/05/3>
- <https://github.com/opencontainers/runc/security>
- <https://www.starlab.io/blog/linux-symbolic-links-convenient-useful-and-a-whole-lot-of-trouble>

Memory-Deny-Write-Execute Protections

Syd version 3.14.1 enhances its security framework by implementing Memory-Deny-Write-Execute (MDWE) protections, aligning with the **PR_SET_MDWE** and **PR_MDWE_REFUSE_EXEC_GAIN** functionality introduced in Linux kernel 6.3. This feature establishes a stringent policy against creating memory mappings that are *simultaneously writable and executable*, closely adhering to the executable space protection mechanisms inspired by PaX project. In addition, Syd fortifies these MDWE protections by employing kernel-level seccomp filters on critical system calls, including *mmap(2)*, *mmap2(2)*, *mprotect(2)*, *pkey_mprotect(2)*, and *shmat(2)*. These filters are designed to intercept and restrict operations that could potentially contravene MDWE policies, such as attempts to make non-executable memory mappings executable or to map shared memory segments with executable permissions. By integrating **PR_SET_MDWE** for preemptive kernel enforcement and utilizing seccomp filters for granular, kernel-level control over system call execution, Syd provides a robust defence mechanism against exploitation techniques that exploit memory vulnerabilities, thereby ensuring a securely hardened execution environment. This restriction may be relaxed using the *trace/allow_unsafe_exec_memory:1* sandbox command at startup. Even with this restriction relaxed, Syd is going to call **PR_SET_MDWE**, but it will use the **PR_MDWE_NO_INHERIT** flag to prevent propagation of the MDWE protection to child processes on *fork(2)*.

As of version 3.25.0, Syd kills the process on memory errors rather than denying these system calls with **EACCES** ("Permission denied"). This ensures the system administrator gets a notification via *dmesg*(1), and has a higher chance to react soon to investigate potentially malicious activity. In addition, repeated failures are going to trigger SegvGuard.

As of version 3.37.0, Syd addresses a fundamental architectural vulnerability in the Linux kernel's Memory-Deny-Write-Execute (MDWE) implementation through proactive file descriptor writability assessment during memory mapping operations. This enhancement directly mitigates Linux kernel bug 219227, which exposes a critical W^X enforcement bypass wherein adversaries can circumvent memory protection mechanisms by exploiting the semantic disconnect between file-backed memory mappings and their underlying file descriptors. The vulnerability manifests when executable memory regions are mapped with **PROT_READ|PROT_EXEC** permissions from file descriptors that retain *write* access, enabling post-mapping modification of executable memory content through standard file I/O operations -- effectively transforming read-only executable mappings into mutable code regions that violate fundamental W^X invariants. By implementing mandatory writability validation prior to permitting any file-backed executable memory mapping, Syd enforces strict temporal isolation between memory mapping permissions and underlying file descriptor capabilities, thereby preventing the exploitation of this kernel-level abstraction leakage that would otherwise enable arbitrary code injection through seemingly benign file operations. This defense mechanism operates at the syscall interception layer, providing comprehensive protection against sophisticated memory corruption attacks that leverage the incongruity between virtual memory management and file system semantics to achieve unauthorized code execution within ostensibly hardened environments. This restriction may be relaxed using the *trace/allow_unsafe_exec_memory:1* sandbox command at startup.

Advanced Memory Protection Mechanisms

Syd version 3.15.1 enhances its security framework by integrating sophisticated a seccomp BPF hook to meticulously block *executable+shared* memory mappings, targeting a critical vulnerability exploitation pathway. As of version 3.21.3, Syd also blocks *executable+anonymous* memory. These updates refine the sandbox's defence against unauthorised memory access and arbitrary code execution by inspecting and filtering system calls, notably *mmap*(2), and *mmap2*(2), to enforce stringent policies against dangerous memory mapping combinations. While this bolstered security measure significantly reduces the attack surface for exploits like buffer overflows and code injections, it acknowledges potential legitimate use cases, such as Just-In-Time (JIT) compilation and plugin architectures, that may require exceptions. To accommodate necessary exceptions without compromising overall security, Syd allows these restrictions to be relaxed with explicit configuration through the *trace/allow_unsafe_exec_memory:1* command, ensuring that users can fine-tune the balance between security and functionality according to specific requirements, with a keen eye on preventing the propagation of relaxed security settings to child processes.

Null Address Mapping Prevention

In our ongoing effort to enhance the security features of Syd, as of version 3.15.1 we introduced a crucial update inspired by the practices of HardenedBSD, specifically aimed at bolstering our sandbox's defences against null pointer dereference vulnerabilities. Following the model set by HardenedBSD, Syd now includes a new security measure that completely prohibits the mapping of memory at the NULL address using the *mmap*(2) and *mmap2*(2) system calls with the **MAP_FIXED** and **MAP_FIXED_NOREPLACE** flags. This addition is implemented through meticulous seccomp filter rules that block these specific mapping requests when the first argument (*addr*) is zero, effectively rendering attempts to exploit null pointer dereferences as non-viable by ensuring such memory allocations result in respective system call getting denied with **EACCES** ("Permission denied"). By disallowing the execution of arbitrary code at the NULL address, Syd significantly reduces the attack surface associated with such vulnerabilities, reinforcing the sandbox's commitment to providing a robust security framework for Linux systems. This technical enhancement reflects our dedication to leveraging advanced security insights from the broader community, embodying our proactive stance on safeguarding against evolving threats.

Linux has *vm/mmap_min_addr* which guards against this already. Hence, this acts as a second layer of defense. Note, though, unlike Syd, Linux allows processes with the **CAP_SYS_RAWIO** capability to edit/override this value. As of version 3.37.0, Syd caps this value at page size like OpenBSD does for added hardening against such edits.

As of version 3.25.0, all addresses lower than the value of `vm/mmap_min_addr` at Syd startup are included into the seccomp filter the action of the filter is set to kill process rather than deny with EACCES. This ensures the system administrator gets a notification via `dmesg(1)`, and has a higher chance to react soon to investigate potentially malicious activity. In addition, repeated failures are going to trigger SegvGuard.

Default Memory Allocator Security Enhancement

As of version 3.46.0, Syd has transitioned to using the GrapheneOS allocator as its default memory allocator. This new allocator leverages modern hardware capabilities to provide substantial defenses against common vulnerabilities like heap memory corruption, while reducing the lifetime of sensitive data in memory. While the previously used `mimalloc` with the `secure` option offered notable security improvements, the GrapheneOS allocator goes further with features like out-of-line metadata protection, fine-grained randomization, and aggressive consistency checks. It incorporates advanced techniques such as hardware memory tagging for probabilistic detection of use-after-free errors, zero-on-free with write-after-free detection, and randomized quarantines to mitigate use-after-free vulnerabilities. The allocator is designed to prevent traditional exploitation methods by introducing high entropy, random base allocations across multiple memory regions, and offers a portable solution being adopted by other security-focused operating systems like Secureblue. It also heavily influenced the next-generation `musl malloc` implementation, improving security with minimal memory usage. Refer to the following links for more information:

- <https://grapheneos.org/features#exploit-mitigations>
- https://github.com/GrapheneOS/hardened_malloc

Enhanced Security for Memory File Descriptors

In version 3.21.1, Syd significantly enhanced its security posture by introducing restrictions on memory file descriptors (`memfds`). The `memfd_create(2)` system call is now sandboxed under Create sandboxing, with the name argument prepended with `!memfd:` before access checks. This allows administrators to globally deny access to `memfds` using rules like `deny/create+!memfd:*`. Additionally, the `memfd_secret(2)` system call, which requires the `secretmem.enable=1` boot option and is seldom used, was denied to prevent potential exploits. Despite file I/O being restricted on secret `memfds`, they could be abused by attackers to write payloads and map them as executable, thus bypassing denylisted code execution controls.

Building on these changes, version 3.21.2 further fortifies security by making `memfds` non-executable by default. This is achieved by removing the `MFD_EXEC` flag and adding the `MFD_NOEXEC_SEAL` flag to `memfd_create(2)`, ensuring `memfds` cannot be made executable. Notably, the `MFD_NOEXEC_SEAL` flag requires Linux-6.3 or newer to function. These measures collectively mitigate the risk of `memfd` abuse, which can involve executing malicious code within a sandbox, circumventing security mechanisms like Exec, Force, and TPE sandboxing. For scenarios where executable or secret `memfds` are genuinely required, the `trace/allow_unsafe_memfd:1` option allows for relaxing these restrictions, though it introduces increased security risks. By default, these enhancements enforce a robust security posture, preventing attackers from leveraging `memfds` as a vector for unauthorized code execution.

Path Masking

Introduced in version 3.16.7, the *Path Masking* feature in Syd enhances security by enabling the obfuscation of file contents without denying access to the file itself. This functionality is critical in scenarios where compatibility requires file presence, but not file readability. Path Masking works by redirecting any attempt to `open(2)` a specified file to the character device `/dev/null`, effectively presenting an empty file to the sandboxed process. The original file metadata remains unchanged, which is essential for applications that perform operations based on this data. Moreover, masked files can still be executed, providing a seamless integration where executability is required but content confidentiality must be preserved.

This feature leverages `glob(3p)` patterns to specify which files to mask, allowing for flexible configuration tailored to diverse security needs. By default, Syd masks sensitive paths such as `/proc/cmdline` to prevent the leakage of potentially

sensitive boot parameters, aligning with Syd's security-first design philosophy. Path Masking is a robust security enhancement that minimises the risk of sensitive data exposure while maintaining necessary system functionality and compliance with expected application behaviors.

Refined Socket System Call Enforcement

In Syd version 3.16.12, we have strengthened the enforcement of socket system call restrictions within the sandbox using kernel-level BPF filters. This enhancement builds upon existing features by embedding these controls directly into the Syd process, ensuring that even if Syd is compromised, it cannot utilise or manipulate denied socket domains. This proactive measure restricts socket creation strictly to permitted domains such as UNIX (**AF_UNIX**), IPv4 (**AF_INET**), and IPv6 (**AF_INET6**), significantly reducing the network attack surface. The *trace/allow_unsupp_socket:1* option allows for the extension of permissible socket domains, catering to specific needs but potentially increasing exposure risks. Additionally, *trace/allow_safe_kcapi:1* enables access to the Kernel Crypto API, facilitating necessary cryptographic operations directly at the kernel level. These enhancements provide a more secure and configurable environment, allowing administrators precise control over network interactions and improving the overall security posture of the sandbox.

Enhanced Execution Control (EEC)

The Enhanced Execution Control (EEC) feature, introduced in Syd version 3.17.0, represents a significant advancement in the sandbox's defence mechanisms. This feature strategically disables the *execve(2)* and *execveat(2)* system calls for the Syd process after they are no longer required for executing the sandbox process, thus safeguarding against their potential abuse by a compromised Syd process. The prohibition of these critical system calls adds a robust layer to the existing Memory-Deny-Write-Execute (MDWE) protections, intensifying the system's defences against exploit techniques such as code injection or return-oriented programming (ROP). Concurrently, EEC ensures that the *ptrace(2)* syscall is limited following the initial use of the **PTRACE_SEIZE** call for execution-related mitigations. This action effectively prevents subsequent system trace operations, barring unauthorised process attachments and further securing the system against manipulation. Together, these measures enhance Syd's security architecture, reflecting an ongoing commitment to implement rigorous, state-of-the-art safeguards within the execution environment.

As of version 3.17.1, the Enhanced Execution Control (EEC) has been further strengthened by integrating *mprotect(2)* hardening mechanisms specifically targeting the prevention of the *ret2mprotect* exploitation technique. This enhancement blocks attempts to alter memory protections to executable (using the **PROT_EXEC** flag) via the *mprotect(2)* and *pkey_mprotect(2)* system calls. By adding these checks, EEC mitigates the risk associated with compromised Syd processes by enforcing stringent memory operation policies that prevent unauthorised memory from becoming executable, thereby countering sophisticated memory corruption attacks such as return-oriented programming (ROP) and other code injection strategies. This proactive security measure is crucial for maintaining the integrity of the sandbox environment, ensuring that Syd continues to offer robust protection against evolving exploit techniques.

As of version 3.23.9, the Enhanced Execution Control (EEC) feature has been expanded to mitigate Sigreturn Oriented Programming (SROP) attacks by denying access to the system calls *sigreturn(2)* and *rt_sigreturn(2)* for *syd(1)*, *syd-oci(1)*, and *syd-tor(1)*. Given the lack of signal handlers, these system calls have no legitimate use. By preventing these calls, the system is better protected against SROP attacks, which involve manipulating signal handler frames to control program state, thus significantly enhancing the security of the execution environment. For further reading, refer to section 2.4.4 Sigreturn-oriented programming in the Low-Level Software Security book (URL: <https://llsoftsec.github.io/llsoftsecbook/#sigreturn-oriented-programming>). SROP (Bosman and Bos 2014) is a special case of ROP where the attacker creates a fake signal handler frame and calls *sigreturn(2)*, a system call on many UNIX-type systems normally called upon return from a signal handler, which restores the state of the process based on the state saved on the signal handler's stack by the kernel previously. The ability to fake a signal handler frame and call *sigreturn* gives an attacker a simple way to control the state of the program.

Enhanced `execve` and `execveat` Syscall Validation

As of version 3.24.2, security enhancements to `execve(2)` and `execveat(2)` syscalls have been introduced to thwart simple Return-Oriented Programming (ROP) attacks. Per the Linux `execve(2)` manpage: "On Linux, `argv` and `envp` can be specified as `NULL`. In both cases, this has the same effect as specifying the argument as a pointer to a list containing a single null pointer. *Do not take advantage of* this nonstandard and nonportable misfeature! On many other UNIX systems, specifying `argv` as `NULL` will result in an error (**EFAULT**: "Bad address"). Some other UNIX systems treat the `envp==NULL` case the same as Linux." Based on this guidance, Syd now rejects `execve(2)` and `execveat(2)` with **EFAULT** when one of the `pathname`, `argv` and `envp` arguments is `NULL`. This mitigation targets basic ROP chains where `NULL` pointers are used as placeholders to bypass argument validation checks, a common tactic in exploiting buffer overflow vulnerabilities. For example, a typical ROP chain trying to execute `execve(2)` with `argv` and `envp` set to `NULL` would be intercepted and denied under these rules:

```
0x0000:      0x40ee2b pop rdx; ret
0x0008:              0x0 [arg2] rdx = 0
0x0010:      0x402885 pop rsi; ret
0x0018:              0x0 [arg1] rsi = 0
0x0020:      0x4013cc pop rdi; ret
0x0028:      0x460000 [arg0] rdi = 4587520
0x0030:      0x438780 execve
```

An attacker might circumvent this mitigation by ensuring that none of the critical syscall arguments are `NULL`. This requires a more sophisticated setup in the ROP chain, potentially increasing the complexity of the exploit and reducing the number of vulnerable targets. This focused security measure enhances system resilience against simple ROP exploits while maintaining compliance with POSIX standards, promoting robustness and cross-platform security.

As of version 3.25.0, Syd terminates the process upon entering these system calls with `NULL` arguments rather than denying them with **EFAULT**. This ensures the system administrator gets a notification via kernel audit log, ie. `dmesg(1)`, about potentially malicious activity. In addition, repeated failures are going to trigger `SegvGuard`.

We have verified the same issue is also present on HardenedBSD and notified upstream:

- Issue: <https://git.hardenedsbsd.org/hardenedsbsd/HardenedBSD/-/issues/106>
- Fix: <https://git.hardenedsbsd.org/hardenedsbsd/HardenedBSD/-/commit/cd93be7afbcfd134b45b52961fc9c6907984c85>

Securebits and Kernel-Assisted Executability

As of version 3.41.0, Syd initializes the per-thread securebits in a kernel-cooperative manner: on Linux 6.14 and newer, which provide the executability-check interface (`execveat(2)` with **AT_EXECVE_CHECK**) and the corresponding interpreter self-restriction securebits, Syd first attempts to install a comprehensive securebits configuration (with locks) that hardens capability semantics and execution constraints; if the kernel refuses changes due to privilege (e.g., **CAP_SETPCAP** not present) and returns **EPERM** ("Operation not permitted"), Syd deterministically degrades to the unprivileged, interpreter-facing policy only, thereby enabling and locking a file-descriptor-based executability check and prohibiting interactive snippet execution unless the same kernel probe passes, while on older kernels the secure-exec policy setup is treated as a no-op and startup proceeds without altering executability behavior; this initialization is inherited across forks and execs (with the kernel rule that the *keep capabilities* base flag is cleared on exec), is orthogonal to the `no_new_privs` attribute, and is designed to be monotonic and predictable under mixed-privilege and mixed-kernel deployments: unsupported features are ignored, permission failures do not abort startup, and the resulting state is the strongest policy the kernel will accept; Users may opt out of these defaults per deployment by setting `trace/allow_unsafe_exec_script:1` to skip the script/file vetting policy, `trace/allow_unsafe_exec_interactive:1` to allow interactive interpreter inputs again, `trace/allow_unsafe_exec_null:1` to permit legacy exec with `NULL` `argv/envp` as described in the previous subsection, or `trace/allow_unsafe_cap_fixup:1` to preserve traditional UID/capability-fixup semantics. Refer to the following links for more information:

- https://docs.kernel.org/userspace-api/check_exec.html
- <https://man7.org/linux/man-pages/man2/execveat.2.html>

- <https://man7.org/linux/man-pages/man7/capabilities.7.html>
- <https://man7.org/linux/man-pages/man2/prctl.2.html>
- https://man7.org/linux/man-pages/man2/pr_set_securebits.2const.html
- https://www.man7.org/linux/man-pages/man2/PR_SET_KEEPCAPS.2const.html

Enhanced Path Integrity Measures

As of version 3.17.4, Syd incorporates crucial enhancements to maintain the integrity of file system paths by systematically denying and masking paths that contain control characters. These modifications are essential for preventing the exploitation of terminal-based vulnerabilities and for maintaining robustness in logging activities. Paths identified with control characters are not only denied during sandbox access check but are also sanitized when logged to ensure that potentially harmful data does not compromise log integrity or facilitate inadvertent security breaches. Such measures underscore Syd's ongoing commitment to fortifying security by adhering to rigorous, up-to-date standards for handling untrusted input efficiently.

As of version 3.18.6, this restriction can be relaxed by using the setting *trace/allow_unsafe_filename:1*. This setting may be toggled from within the sandbox during runtime prior to locking the sandbox.

As of version 3.28.0, Syd has enhanced its path integrity measures by incorporating an implementation based on David A. Wheeler's Safename Linux Security Module (LSM) patches. This update not only prevents the creation of filenames containing potentially harmful characters but also hides existing files with such names. Invalid filenames are now denied with an **EINVAL** ("Invalid argument") *errno*(3) when necessary. In alignment with Wheeler's recommendations on restricting dangerous filenames, the validation now enforces stricter rules:

- **Control Characters:** Filenames containing control characters (bytes 0x00–0x1F and 0x7F) are denied.
- **UTF-8 Encoding:** Filenames must be valid UTF-8 sequences.
- **Forbidden Characters:** The following characters are disallowed in filenames as they may interfere with shell operations or be misinterpreted by programs: *, ?, [,], ", <, >, |, (,), &, ', !, \, ;, \$, and `.
- **Leading Characters:** Filenames cannot start with a space (), dash (-), or tilde (~).
- **Trailing Characters:** Filenames cannot end with a space ().

As of version 3.37.9, space checks have been extended to cover UTF-8 whitespace, thanks to an idea by Jacob Bachmeyer, see <https://seclists.org/oss-sec/2025/q3/123> for more information.

As of version 3.38.0, the characters :, {, and } have been removed from the forbidden set to improve usability and reduce false positives. : is used commonly across /dev and /proc. {} are used by *firefox*(1) for filenames under the profile directory.

These measures mitigate security risks associated with malicious filenames by ensuring that both new and existing filenames adhere to stringent validation rules. This enhancement strengthens overall system robustness by preventing potential exploitation through untrusted input in file operations. For more information, refer to the following links:

- <https://dwheeler.com/essays/fixing-unix-linux-filenames.html>
- <https://lwn.net/Articles/686021/>
- <https://lwn.net/Articles/686789/>
- <https://lwn.net/Articles/686792/>

Device Sidechannel Mitigations

As of Syd version 3.21.0, Syd's device sidechannel mitigations align closely with **GRKERNSEC_DEVICE_SIDECHANNEL** in Grsecurity, aiming to prevent timing analyses on block or character devices via *stat(2)* or *inotify(7)/fanotify(7)*. For *stat*-family system calls, Syd, like Grsecurity, matches the last access and modification times to the creation time for devices, thwarting unprivileged user timing attacks. Instead of dropping events, Syd strips access and modify *fanotify(7)/inotify(7)* flags at syscall entry, preventing unsafe *fanotify(7)/inotify(7)* event generation. This approach ensures unauthorized users cannot determine sensitive information, such as the length of the administrator password. Syd's solution offers robust security by dynamically stripping flags, enhancing protection against these sidechannel attacks without compromising functionality. As of version 3.40.0, these mitigations can be disabled using the options *trace/allow_unsafe_stat_bdev*, *trace/allow_unsafe_stat_cdev*, *trace/allow_unsafe_notify_bdev*, *trace/allow_unsafe_notify_cdev* respectively. Refer to the following links for more information:

- https://web.archive.org/web/20130111093624/http://vladz.devzero.fr/013_ptmx-timing.php
- https://en.wikibooks.org/wiki/Grsecurity/Appendix/Grsecurity_and_PaX_Configuration_Options#Eliminate_stat/notify

Restricting CPU Emulation System Calls

As of version 3.22.1, Syd denies the *modify_ldt(2)*, *subpage_prot(2)*, *switch_endian(2)*, *vm86(2)*, and *vm86old(2)* system calls by default, which are associated with CPU emulation functionalities. These calls can only be allowed if the *trace/allow_unsafe_cpu* option is explicitly set. This restriction helps mitigate potential vulnerabilities and unauthorized access that can arise from modifying CPU state or memory protections, thus strengthening the overall security posture of the sandbox environment.

Kernel Keyring Access Restriction

To enhance system security, access to the kernel's key management facility via the *add_key(2)*, *keyctl(2)*, and *request_key(2)* system calls is restricted by default as of version 3.22.1. These calls are crucial for managing keys within the kernel, enabling operations such as adding keys, manipulating keyrings, and requesting keys. The restriction aims to prevent unauthorized or potentially harmful modifications to keyrings, ensuring that only safe, controlled access is permitted. However, administrators can relax this restriction by enabling the "trace/allow_unsafe_keyring" option, allowing these system calls to be executed when necessary for legitimate purposes.

Note, because of this restriction, Syd is not affected by CVE-2024-42318 although we use Landlock. See here for more information: <https://www.openwall.com/lists/oss-security/2024/08/17/2>

Restricting Memory Protection Keys System Calls

As of version 3.22.1, Syd denies the system calls *pkey_alloc(2)*, *pkey_free(2)*, and *pkey_mprotect(2)* by default. These system calls are associated with managing memory protection keys, a feature that can be leveraged to control memory access permissions dynamically. To allow these system calls, administrators can enable the *trace/allow_unsafe_pkey* option. This restriction enhances security by preventing unauthorized or potentially harmful manipulations of memory access permissions within the sandbox environment, ensuring stricter control over memory protection mechanisms.

Restricting vmsplice System Call

As of version 3.23.5, Syd disables the *vmsplice(2)* system call by default to enhance security. This syscall, identified as a potential vector for memory corruption and privilege escalation, poses significant risks in sandboxed environments. By default, disabling *vmsplice(2)* reduces the attack surface, aligning with security practices in other systems like Podman. Refer to the following links for more information:

- <https://lore.kernel.org/linux-mm/X+PoXCizo392PBX7@redhat.com/>

- <https://lwn.net/Articles/268783/>

As of version 3.41.3, *vmsplice(2)* call may be permitted at startup using the *trace/allow_unsafe_vmsplice:1* option.

Enforcing Position-Independent Executables (PIE)

As of version 3.23.9, Syd mandates that all executables must be Position-Independent Executables (PIE) to leverage Address Space Layout Randomization (ASLR). PIE allows executables to be loaded at random memory addresses, significantly enhancing security by making it more difficult for attackers to predict the location of executable code. This randomization thwarts various types of exploits, such as buffer overflow attacks, which rely on predictable memory addresses to execute malicious code. To accommodate scenarios where PIE is not feasible, users can relax this restriction using the *trace/allow_unsafe_exec_nopie:1* option. This ensures compatibility while maintaining a robust security posture by default, aligning with Syd's overarching strategy of employing advanced security measures to mitigate potential attack vectors.

Enforcing Non-Executable Stack

As of version 3.23.16, Syd mandates that all executables must have a non-executable stack to enhance security. A non-executable stack helps to prevent various types of exploits, such as stack-based buffer overflow attacks, by making it more difficult for attackers to execute malicious code from the stack. This security measure is similar to the enforcement of Position-Independent Executables (PIE) and is a crucial part of Syd's comprehensive security strategy. To accommodate scenarios where a non-executable stack is not feasible, administrators can relax this restriction using the *trace/allow_unsafe_exec_stack:1* option. This ensures compatibility while maintaining a robust security posture by default, aligning with Syd's overarching strategy of employing advanced security measures to mitigate potential attack vectors.

As of version 3.23.19, Syd enforces this restriction at *mmap(2)* boundary as well so it is no longer possible to *dlopen(3)* a library with executable stack to change the stack permissions of the process to executable. This is useful in mitigating attacks such as CVE-2023-38408. Refer to the URL <https://www.qualys.com/2023/07/19/cve-2023-38408/rce-openssh-forward> for more information. As of version 3.25.0, Syd kills the process in this case rather than denying the system call to be consistent with other memory related seccomp filters. This ensures the system administrator gets a notification via the audit log, and has a higher chance to react soon to investigate potentially malicious activity. In addition, repeated failures are going to trigger SegvGuard.

Mitigation Against Heap Spraying

As of version 3.23.18, Syd introduces a critical security enhancement to mitigate kernel heap-spraying attacks by restricting the *msgsnd(2)* system call. This call, integral to System V message queues, is essential for inter-process communication (IPC) in Unix-like operating systems. System V message queues allow processes to send and receive messages asynchronously, facilitating robust communication between processes. However, it is also frequently exploited for heap spraying, a technique that increases the predictability of memory allocations to facilitate arbitrary code execution. Notably, exploits such as CVE-2016-6187, CVE-2021-22555, and CVE-2021-26708 have leveraged this system call for kernel heap-spraying to achieve privilege escalation and kernel code execution. Heap spraying aims to introduce a high degree of predictability to heap allocations, facilitating arbitrary code execution by placing specific byte sequences at predictable memory locations. This method is particularly dangerous because it increases the reliability of exploiting vulnerabilities by aligning memory in a way that malicious code execution becomes feasible. To counter this, Syd now disables the *msgsnd(2)* system call by default, which is commonly used for heap spraying due to its ability to allocate large, contiguous blocks of memory in the kernel heap. This preemptive measure significantly reduces the attack surface, preventing attackers from leveraging this system call to bypass security mitigations and achieve kernel code execution. Administrators can re-enable this call using the *trace/allow_unsafe_msgsnd:1* option if required for legitimate inter-process communication needs, ensuring that the default configuration prioritizes security against such advanced exploitation techniques.

For more information refer to the following links:

- https://en.wikipedia.org/wiki/Heap_spraying
- https://grsecurity.net/how_autoslab_changes_the_memory_unsafety_game
- <https://duasynt.com/blog/cve-2016-6187-heap-off-by-one-exploit>
- <https://google.github.io/security-research/pocs/linux/cve-2021-22555/writeup.html>
- <https://a13xp0p0v.github.io/2021/02/09/CVE-2021-26708.html>

Mitigation against Page Cache Attacks

As of version 3.25.0, Syd denies the *mincore(2)* system call by default, which is typically not needed during normal run and has been successfully (ab)used for page cache attacks: <https://arxiv.org/pdf/1901.01161>

To quote the **Countermeasures** section of the article:

Our side-channel attack targets the operating system page cache via operating system interfaces and behavior. Hence, it clearly can be mitigated by modifying the operating system implementation. **Privileged Access.** The *QueryWorkingSetEx* and *mincore* system calls are the core of our side-channel attack. Requiring a higher privilege level for these system calls stops our attack. The downside of restricting access to these system calls is that existing programs which currently make use of these system calls might break. Hence, we analyzed how frequently *mincore* is called by any of the software running on a typical Linux installation. We used the Linux *perf* tools to measure over a 5 hour period whenever the *sys_enter_mincore* system call is called by any application. During these 5 hours a user performed regular operations on the system, i.e., running various work-related tools like Libre Office, gcc, Clion, Thunderbird, Firefox, Nautilus, and Evince, but also non-work-related tools like Spotify. The system was also running regular background tasks during this time frame. Surprisingly, the *sys_enter_mincore* system call was not called a single time. This indicates that making the *mincore* system call privileged is feasible and would mitigate our attack at a very low implementation cost.

As of version 3.35.2, the new system call *cachestat(2)* is also denied for the same reason as it is a scalable version of the *mincore(2)* system call. Again, as of version 3.35.2, the option *trace/allow_unsafe_page_cache* has been added to relax this restriction at startup. This may be needed to make direct rendering work with Firefox family browsers.

Enforcing AT_SECURE and UID/GID Verification

As of version 3.27.0, Syd enhances security by enforcing the **AT_SECURE** flag in the auxiliary vector of executables at *ptrace(2)* boundary upon receiving the **PTRACE_EVENT_EXEC** event to enforce secure-execution mode. This event happens after the executable binary is loaded into memory but before it starts executing. This enforcement ensures that the C library operates in a secure mode, disabling unsafe behaviors like loading untrusted dynamic libraries or accessing insecure environment variables. Additionally, Syd performs strict UID and GID verification to confirm that the process's user and group IDs match the expected values, preventing unauthorized privilege escalation. If the verification fails or the **AT_SECURE** flag cannot be set, Syd terminates the process to prevent potential security breaches. This mitigation can be relaxed at startup with the option *trace/allow_unsafe_exec_libc:1*, though doing so is not recommended as it reduces the effectiveness of the sandbox. Notably, secure-execution mode is enforced by *apparmor(7)* too and it may also be enforced by other LSMs and eBPF. You may find some implications of the secure-execution mode below. Refer to the *ld.so(8)* and *getauxval(3)* manual pages for implications of secure-execution mode on your system.

glibc dynamic linker strips/ignores dangerous **LD_*** variables in secure-execution mode, including **LD_LIBRARY_PATH**, **LD_PRELOAD** (only standard dirs; paths with slashes ignored), **LD_AUDIT**, **LD_DEBUG**, **LD_DEBUG_OUTPUT**, **LD_DYNAMIC_WEAK**, **LD_HWCAP_MASK**, **LD_ORIGIN_PATH**, **LD_PROFILE**, **LD_SHOW_AUXV**, **LD_USE_LOAD_BIAS**, etc. glibc also treats some non-**LD_*** variables as unsafe in secure-execution mode: **GCONV_PATH**, **GETCONF_DIR**, **HOSTALIASES**, **LOCALDOMAIN**, **LOCPATH**, **MALLOC_TRACE**, **NIS_PATH**, **NLSPATH**, **RESOLV_HOST_CONF**, **RES_OPTIONS**, **TMPDIR**, **TZDIR** (stripped/ignored). Refer to the *ld.so(8)* manual page for more information. Note, as of version 3.11.1, Syd also strips unsafe environment variables before executing the sandbox process by default and this can be disabled altogether with *trace/allow_unsafe_env:1* or unsafe environment variables can be selectively allowed using the *-e var=* format, e.g. *-eLD_PRELOAD=* See the **Restricting environment** and *trace/allow_unsafe_env* section of this manual page for more information.

glibc's **LD_PREFER_MAP_32BIT_EXEC** is always disabled in secure-execution mode (mitigates ASLR-weakening). Historical bugs (e.g., CVE-2019-19126) fixed cases where this wasn't ignored after a security transition. Refer to the *ld.so(8)* manual page and the following links for more information:

- <https://lists.gnu.org/archive/html/info-gnu/2020-02/msg00001.html>
- <https://alas.aws.amazon.com/ALAS-2021-1511.html>

glibc **GLIBC_TUNABLES** environment variable handling under **AT_SECURE**: tunables carry security levels (**SEXID_ERASE**, **SEXID_IGNORE**) so they're ignored/erased for secure-execution mode; post-CVE-2023-4911 hardening ensures secure-execution mode invocations with hostile **GLIBC_TUNABLES** are blocked/terminated. Refer to the following links for more information:

- <https://lwn.net/Articles/947736/>
- <https://access.redhat.com/security/cve/cve-2023-4911>
- <https://nvd.nist.gov/vuln/detail/CVE-2023-4911>

glibc *secure_getenv(3)* returns NULL when **AT_SECURE** is set; any glibc subsystem that uses *secure_getenv(3)* (e.g., timezone, locale, iconv, resolver paths) will ignore environment overrides in secure-execution mode. Similarly calling *getauxval(3)* with the flag **AT_SECURE** returns true in secure-execution mode.

musl libc honors **AT_SECURE** and likewise ignores preload/library/locale environment knobs in secure-execution mode; examples include **LD_PRELOAD**, **LD_LIBRARY_PATH**, and **MUSL_LOCPATH**. Refer to the following links for more information:

- <https://musl.libc.org/manual.html>
- <https://wiki.musl-libc.org/environment-variables>

Because the Linux host kernel is not aware of Syd setting the **AT_SECURE** bit, the *proc_pid_auxv(5)* file will report the bit as unset. On the contrary, when verbose logging is turned on using the *log/verbose:1* option, Syd will correctly log this bit as set after parsing the *proc_pid_auxv(5)* file of the sandbox process.

Process Name Modification Restriction

As of version 3.28.0, Syd introduces a critical security enhancement that logs and denies attempts to set a process's name using the **PR_SET_NAME** *prctl(2)* request. This mitigation is essential as it prevents malicious software from disguising itself under legitimate process names such as *apache* or other system daemons, thereby thwarting attempts to evade detection and maintain stealth within the system. By default, any invocation of **PR_SET_NAME** within the sandboxed environment is intercepted; the action is logged for audit purposes if verbose logging is on, and the system call is denied with success return, essentially turning it into a no-op. If there is a legitimate need to permit process name changes within the sandbox, this restriction can be overridden by enabling the *trace/allow_unsafe_prctl:1* option, which allows **PR_SET_NAME** requests to succeed without logging.

Mitigation against Sigreturn Oriented Programming (SROP)

As of version 3.30.0, Syd employs a robust, multi-layered mitigation strategy against Sigreturn Oriented Programming (SROP), a sophisticated exploit technique that manipulates the state restoration behavior of the *sigreturn(2)* system call to hijack process execution. This approach addresses SROP's ability to bypass critical memory protections such as ASLR, NX, and partial RELRO by setting up a fake stack frame to redirect control flow upon signal return. Inspired by Erik Bosman's proposal in May 2014 (LKML PATCH 3/4), Syd incorporates a signal counting mechanism to track the number of signals delivered to a thread group, ensuring that each *sigreturn(2)* invocation corresponds to an actual, in-progress signal handler. A stray *sigreturn(2)* call violating this rule causes the process to be terminated with the signal **SIGKILL**. This method provides more precise protection than *sigreturn(2)* frame canaries, which are susceptible to circumvention under certain conditions and significantly enhances the integrity of sandboxed environments, effectively blocking a critical class of attacks. Administrators can disable these mitigations via the *trace/allow_unsafe_sigreturn:1*

option, though doing so exposes systems to exploitation and undermines security. For more information, refer to the following links:

- http://www.cs.vu.nl/~herbertb/papers/srop_sp14.pdf
- <https://web.archive.org/web/20221002135950/https://lkml.org/lkml/2014/5/15/660>
- <https://web.archive.org/web/20221002123657/https://lkml.org/lkml/2014/5/15/661>
- <https://web.archive.org/web/20221002130349/https://lkml.org/lkml/2014/5/15/657>
- <https://web.archive.org/web/20221002135459/https://lkml.org/lkml/2014/5/15/858>
- <https://lwn.net/Articles/674861>
- <https://lore.kernel.org/all/1454801964-50385-1-git-send-email-sbauer@eng.utah.edu/>
- <https://lore.kernel.org/all/1454801964-50385-2-git-send-email-sbauer@eng.utah.edu/>
- <https://lore.kernel.org/all/1454801964-50385-3-git-send-email-sbauer@eng.utah.edu/>
- <https://marc.info/?l=openbsd-tech&m=146281531025185>
- <https://isopenbsdsecu.re/mitigations/srop/>

Speculative Execution Mitigation

As of version 3.30.0, Syd integrates a robust mitigation mechanism leveraging the *prctl(2)* system call to enforce speculative execution controls to fortify the sandbox against advanced speculative execution vulnerabilities, such as **Spectre** and related side-channel attacks. Upon initialization, Syd attempts to apply the **PR_SPEC_FORCE_DISABLE** setting for critical speculative execution features -- namely **PR_SPEC_STORE_BYPASS**, **PR_SPEC_INDIRECT_BRANCH**, and **PR_SPEC_L1D_FLUSH** -- thereby irrevocably disabling these CPU-level misfeatures when permissible. This proactive stance ensures that, where supported by the underlying kernel and hardware, speculative execution is constrained to eliminate potential avenues for data leakage and privilege escalation across privilege domains. The mitigation is conditionally enforced based on the availability of per-task control via *prctl(2)*, and any inability to apply these settings due to architectural constraints or insufficient permissions results in logged informational messages without disrupting sandbox operations. Furthermore, administrators retain the capability to override this stringent security posture through the *trace/allow_unsafe_exec_speculative:1* configuration option, permitting flexibility in environments where speculative execution controls may need to be relaxed for compatibility or performance reasons. This dual approach balances rigorous security enforcement with operational adaptability, ensuring that Syd maintains a hardened execution environment while providing mechanisms for controlled exceptions. By systematically disabling speculative execution vulnerabilities at the kernel interface level, Syd significantly mitigates the risk of sophisticated side-channel exploits, thereby enhancing the overall integrity and confidentiality of sandboxed applications. Refer to the links below for more information:

- <https://docs.kernel.org/admin-guide/hw-vuln/spectre.html>
- https://docs.kernel.org/userspace-api/spec_ctrl.html

As of version 3.35.2, Syd disables Speculative Store Bypass mitigations for *seccomp(2)* filters when *trace/allow_unsafe_exec* is set at startup.

Cryptographically Randomized Sysinfo

Since Syd 3.28.0, the *sysinfo(2)* system call has been cryptographically obfuscated by applying high-entropy offsets to memory fields (e.g., total RAM, free RAM) and constraining them to plausible power-of-two boundaries, frustrating trivial attempts at system fingerprinting. Specifically, uptime and idle counters each incorporate a distinct offset up to 0xFF_FFFF (~194 days), while load averages are randomized in fixed-point format and clamped to realistic upper limits. Administrators seeking genuine system metrics may disable these transformations via *trace/allow_unsafe_sysinfo:1*, albeit at the cost of enabling straightforward correlation and potential data leakage.

Memory Sealing of Sandbox Policy Regions on Lock

Beginning with version 3.33.1, Syd applies Linux's *mseal(2)* syscall to enforce immutability of policy-critical memory regions at the moment the sandbox is locked with *lock:on*. At this point, all mutable structures influencing access control -- such as ACLs, action filters, and syscall mediation rules -- are sealed at the virtual memory level. Unlike traditional permission schemes (e.g., W^X or *mprotect(2)*), *mseal(2)* protects against structural manipulation of memory mappings themselves, preventing *mmap(2)*, *mremap(2)*, *mprotect(2)*, *munmap(2)*, and destructive *madvise(2)* operations from altering sealed VMAs. This eliminates attacker primitives that rely on reclaiming, remapping, or changing permissions on enforcement data, thereby closing off advanced data-oriented exploitation paths such as policy subversion through remapped ACLs or revocation of constraints via memory permission resets. Syd permits legitimate late-stage policy configuration during startup and defers sealing until *lock:on* is called, after which mutation of enforcement state is structurally frozen. The process is one-way and idempotent; sealed memory cannot be unsealed, ensuring strong guarantees once lockdown is complete. For diagnostic or non-hardened environments, this mechanism may be disabled explicitly via the startup toggle *trace/allow_unsafe_nomseal:1*, which should only be used with full awareness of the resulting relaxation in protection. When enabled, sealing substantially raises the integrity threshold of the sandbox, ensuring that post-lock policy enforcement is immune to both direct and indirect memory-level tampering.

Force Close-on-Exec File Descriptors

The *trace/force_cloexec* option, introduced in Syd version 3.35.2, ensures that all *creat(2)*, *open(2)*, *openat(2)*, *openat2(2)*, *memfd_create(2)*, *socket(2)*, *accept(2)*, and *accept4(2)* system calls made by the sandbox process include the **O_CLOEXEC** flag. This feature can be toggled at runtime via Syd's virtual stat API, enabling dynamic adjustment of confinement levels as needed. The **O_CLOEXEC** flag, when set on file descriptors, ensures they are automatically closed when executing a new program via *execve(2)* or similar system calls. This automatic closure of file descriptors is critical for enhancing security and safety, as it prevents file descriptors from being unintentionally inherited by newly executed programs, which could otherwise lead to unauthorized access to sensitive files or resources. By enforcing the **O_CLOEXEC** flag across all *open(2)* calls, Syd mitigates the risk of file descriptor leakage, effectively isolating the sandboxed environment and ensuring a clean execution context for newly spawned processes.

Force Randomized File Descriptors

The *trace/force_rand_fd* option, introduced in Syd version 3.35.2, ensures that all *creat(2)*, *open(2)*, *openat(2)*, *openat2(2)*, *memfd_create(2)*, *socket(2)*, *accept(2)*, and *accept4(2)* system calls made by the sandbox process allocate file descriptors at random available slots rather than the lowest-numbered one. When this feature is enabled, Syd specifies a random available slot (rather than the lowest-numbered one) to the **SECCOMP_IOCTL_NOTIF_ADDFD** operation which is used to install a file descriptor to the sandbox process. Randomizing file descriptor numbers makes it significantly harder for an attacker to predict or deliberately reuse critical descriptors, thereby raising the bar against file-descriptor reuse and collision attacks. Note that enabling this may break programs which rely on the POSIX guarantee that *open(2)* returns the lowest available descriptor. This behavior can be toggled at runtime via Syd's virtual stat API, allowing operators to enable or disable descriptor randomization without restarting or recompiling the sandboxed process. We're also cooperating with the HardenedBSD project to implement a similar feature in the BSD kernel. Refer to the following link for more information: <https://git.hardenedsbsd.org/hardenedsbsd/HardenedBSD/-/issues/117>

Syscall Argument Cookies

To further harden the *seccomp(2)* boundary, as of version 3.35.2 Syd embeds cryptographically-strong, per-instance “cookies” into unused architecture-defined syscall argument slots (e.g., the 5th and 6th arguments of *openat2(2)*). These cookies are generated at startup via the OS random number generator using *getrandom(2)*, and are checked in the BPF filter so that only calls bearing the correct 32- or 64-bit values will be allowed. By requiring this unpredictable token, Syd raises the bar against arbitrary or forged syscalls: Attackers must first discover or leak the randomized cookies despite Address Space Layout Randomization (ASLR) before mounting a successful path or network operation. This approach effectively transforms unused syscall parameters into an application-level authorization mechanism, preventing trivial reuse of legitimate code paths and mitigating time-of-check-to-time-of-use (TOCTTOU) and ROP payloads that rely on guessing or omitting optional arguments. In combination with absolute path enforcement and the denial of relative descriptors (e.g. *AT_FDCWD*), syscall argument cookies form a lightweight, zero-cost integrity check that elevates syscall hardening without kernel modifications or performance penalties. As an example, here is how the filters look in pseudo filter code for the system calls *openat2(2)* and *socket(2)* on x86-64. *openat2(2)* uses two unused arguments as cookies and *socket(2)* uses three. In addition, *openat2(2)* denies negative file descriptor arguments such as *AT_FDCWD*:

```
# filter for syscall "openat2" (437) [priority: 65528]
if ($syscall == 437)
    if ($a0.hi32 > 0)
    else
        if ($a0.hi32 == 0)
            if ($a0.lo32 > 2147483647)
            else
                if ($a4.hi32 == 2047080271)
                    if ($a4.lo32 == 419766579)
                        if ($a5.hi32 == 2863373132)
                            if ($a5.lo32 == 396738706)
                                action ALLOW;
                else
                    if ($a4.hi32 == 2047080271)
                        if ($a4.lo32 == 419766579)
                            if ($a5.hi32 == 2863373132)
                                if ($a5.lo32 == 396738706)
                                    action ALLOW;

# filter for syscall "socket" (41) [priority: 65529]
if ($syscall == 41)
    if ($a3.hi32 == 3378530982)
        if ($a3.lo32 == 4160747949)
            if ($a4.hi32 == 2899982880)
                if ($a4.lo32 == 990920938)
                    if ($a5.hi32 == 3611760485)
                        if ($a5.lo32 == 1163305215)
                            action ALLOW;
```

Another example is how the critical *seccomp(2)* notify *ioctl(2)* requests **SECCOMP_IOCTL_NOTIF_SEND** and **SECCOMP_IOCTL_NOTIF_ADDFD** are confined for the Syd emulator threads. **SECCOMP_IOCTL_NOTIF_SEND** is critical because it allows pass-through of system calls to the host Linux kernel with the **SECCOMP_USER_NOTIF_FLAG** flag in the *seccomp(2)* response data structure. This flag must be used with utmost care and in the hands of an attacker it can be a tool for further exploitation. **SECCOMP_IOCTL_NOTIF_ADDFD** is critical because it allows file descriptor transfer between the Syd process and the sandbox process and in the hands of an attacker it can be a tool for file descriptor stealing. As part of this mitigation three syscall cookies are enforced for *ioctl(2)* system calls with the **SECCOMP_IOCTL_NOTIF_SEND** and **SECCOMP_IOCTL_NOTIF_ADDFD** requests. Coupled with the startup randomization of the *seccomp(2)* notify file descriptor, this mitigation raises the bar for an attacker trying to call arbitrary or forged syscalls within a compromised Syd emulator thread. Excerpt from the *seccomp* filter in pseudo filter code is given below:

```
# Syd monitor rules with seccomp fd 626
#
# pseudo filter code start
#
# filter for arch x86_64 (3221225534)
...
# filter for syscall "ioctl" (16) [priority: 65497]
if ($syscall == 16)
    if ($a0.hi32 == 0)
        if ($a0.lo32 == 626)
```

```

if ($a1.hi32 == 4294967295)
  if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_RECV)
    action ALLOW;
  if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_SEND)
    if ($a3.hi32 == 4195042482)
      if ($a3.lo32 == 329284685)
        if ($a4.hi32 == 3163914537)
          if ($a4.lo32 == 2000745976)
            if ($a5.hi32 == 3932715328)
              if ($a5.lo32 == 2409429749)
                action ALLOW;
          if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_ADDFD)
            if ($a3.hi32 == 2387882717)
              if ($a3.lo32 == 529632567)
                if ($a4.hi32 == 2017338540)
                  if ($a4.lo32 == 3732042218)
                    if ($a5.hi32 == 4202049614)
                      if ($a5.lo32 == 546113052)
                        action ALLOW;
            if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_SET_FLAGS)
              action ALLOW;
            if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_ID_VALID)
              action ALLOW;
if ($a1.hi32 == 0)
  if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_RECV)
    action ALLOW;
  if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_SEND)
    if ($a3.hi32 == 4195042482)
      if ($a3.lo32 == 329284685)
        if ($a4.hi32 == 3163914537)
          if ($a4.lo32 == 2000745976)
            if ($a5.hi32 == 3932715328)
              if ($a5.lo32 == 2409429749)
                action ALLOW;
          if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_ADDFD)
            if ($a3.hi32 == 2387882717)
              if ($a3.lo32 == 529632567)
                if ($a4.hi32 == 2017338540)
                  if ($a4.lo32 == 3732042218)
                    if ($a5.hi32 == 4202049614)
                      if ($a5.lo32 == 546113052)
                        action ALLOW;
            if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_SET_FLAGS)
              action ALLOW;
            if ($a1.lo32 == SECCOMP_IOCTL_NOTIF_ID_VALID)
              action ALLOW;
...
# default action
action KILL_PROCESS;
# invalid architecture action
action KILL_PROCESS;

```

List of system calls protected by cookies is given below. The list may be further extended in the future to cover more system calls used by Syd:

- *ioctl(2)* - **PROCMP_QUERY - SECCOMP_IOCTL_NOTIF_SEND - SECCOMP_IOCTL_NOTIF_ADDFD**
- *linkat(2)*, *renameat2(2)*, *unlinkat(2)*
- *memfd_create(2)*
- *openat2(2)*
- *pipe2(2)*
- *socket(2)*, *bind(2)*, *connect(2)*, *accept4(2)* (**64-bit only**)
- *truncate(2)*, *truncate64(2)*, *ftruncate(2)*
- *uname(2)*

As of version 3.36.0, this mitigation may be disabled at startup using the *trace/allow_unsafe_nocookie:1* option.

Shared Memory Permissions Hardening

As of version 3.37.0, Syd introduces a kernel-enforced mitigation against System V shared memory squatting by conditioning allow rules on strict permission masks. By inspecting the mode bits passed to *shmget(2)*, *msgget(2)*, *semget(2)* and *mq_open(2)* system calls, the sandbox admits creates only when user-, group-, and other-permission fields exclude unsafe write or execute flags (i.e., no bits set in mask 0o177). This measure prevents untrusted processes from elevating permissions after creation or exploiting legacy IPC segments with permissive ACLs, which could lead to disclosure or corruption of shared pages. Based on the attack taxonomy described in **Memory Squatting: Attacks on System V Shared Memory** (Portcullis, 2013), mode checks take place within the *seccomp(2)* BPF filter before any mapping. The **IPC_SET** operations of the *shmctl(2)*, *msgctl(2)*, and *semctl(2)* system calls are also denied, preventing permission changes after creation. Additionally, any attempt to attach a shared memory segment with the **SHM_EXEC** flag via *shmat(2)* is denied to enforce W^X policies, blocking executable mappings through shared memory. The *seccomp(2)* filter also blocks the **MSG_STAT_ANY**, **SEM_STAT_ANY**, and **SHM_STAT_ANY** operations (Linux 4.17+), which would otherwise return segment metadata without verifying its mode, mitigating unintended information leaks. This mitigation is applied in the parent *seccomp(2)* filter, ensuring that the Syd process itself is subject to these restrictions. Administrators may relax this policy at startup using the *trace/allow_unsafe_mqueue:1* and *trace/allow_unsafe_shm:1* options, but doing so reintroduces the classic squatting vulnerabilities documented in CVE-2013-0254 and related research. For more information refer to the following links:

- <https://labs.portcullis.co.uk/whitepapers/memory-squatting-attacks-on-system-v-shared-memory/>
- <https://labs.portcullis.co.uk/presentations/i-miss-bsd/>
- <https://www.cve.org/CVERecord?id=CVE-2013-0254>

Denying Restartable Sequences

As of version 3.37.0, Syd denies access to the restartable sequences with the *rseq(2)* system call by default, substantially elevating the security baseline of the sandbox. The restartable sequences interface enables user space to register per-thread critical regions with kernel-enforced atomicity guarantees, but critically, also exposes a user-controlled abort handler address. In adversarial scenarios, this facility can be abused: attackers with the ability to manipulate process memory or *rseq(2)* registration can redirect execution to arbitrary, attacker-chosen code locations on preemption or CPU migration, bypassing intra-process isolation boundaries and subverting mechanisms such as memory protection keys or control-flow integrity. By prohibiting *rseq(2)*, Syd eliminates this kernel-facilitated control-flow transfer primitive, foreclosing a sophisticated class of attacks that leverage restartable sequence state for privilege escalation, sandbox escape, or bypass of compartmentalization. This mitigation exemplifies a least-privilege syscall surface and strong adherence to modern threat models, allowing only strictly necessary system calls and neutralizing emergent attack vectors rooted in nuanced kernel-user collaboration. Administrators may explicitly re-enable this system call if required for compatibility using the *trace/allow_unsafe_rseq:1* startup option, with the understanding that doing so weakens this critical security boundary. For more information, refer to the following links:

- <https://arxiv.org/abs/2108.03705>
- <https://arxiv.org/abs/2406.07429>
- <https://www.usenix.org/system/files/usenixsecurity24-yang-fangfei.pdf>

Personality Syscall Restrictions

As of version 3.37.0, Syd implements comprehensive restrictions on the *personality(2)* system call to mitigate security vulnerabilities associated with unsafe *personality(2)* flags, particularly the **ADDR_NO_RANDOMIZE** flag which can disable Address Space Layout Randomization (ASLR) -- a fundamental memory protection mechanism that prevents reliable exploitation of memory corruption vulnerabilities by randomizing memory layout or the **READ_IMPLIES_EXEC** flag which can bypass memory protections provided by Memory-Deny-Write-Execute, aka W^X. This security enhancement aligns Syd with industry-standard container runtimes including Docker and Podman, which employ identical restrictions to balance security with application compatibility by maintaining an

allowlist of safe personality values: **PER_LINUX** for standard Linux execution domain, **PER_LINUX32** for 32-bit compatibility, **UNAME26** for legacy kernel version reporting, **PER_LINUX32|UNAME26** for combined 32-bit and legacy compatibility, and **GET_PERSONALITY** for querying current *personality*(2) without modification. The implementation follows the principle of least privilege by denying all potentially dangerous *personality*(2) modifications while permitting only essential compatibility requirements, thereby preventing malicious actors from leveraging *personality*(2) flags to make exploits more predictable and reliable -- a behavior specifically monitored by security detection systems. Administrators requiring unrestricted personality system call access can disable these restrictions using *trace/allow_unsafe_personality:1*, though this should be undertaken with careful consideration of the security implications as it potentially exposes the sandbox to personality-based security bypasses that could compromise the isolation guarantees provided by Syd's broader security hardening strategy encompassing comprehensive system call filtering, capability restrictions, and resource access controls.

Thread-Level Filesystem and File-Descriptor Namespace Isolation

As of version 3.37.2, Syd's interrupt, IPC and emulator worker threads are each placed into their own filesystem and file-descriptor namespace by *unshare*(2)'ing both **CLONE_FS** and **CLONE_FILES**. This per-thread isolation ensures that working directory, *umask*(2) and open-file table changes in one thread cannot leak into -- or be influenced by -- any other, closing subtle attack vectors such as TOCTOU races on shared *procfs*(5) or fd entries, descriptor reuse across threads, and cwd-based side channels. By scoping thread-local filesystem state and descriptor tables, this enhancement hardens Syd's sandbox manager against advanced multithreading exploits and preserves strict separation between the monitoring and emulation components.

Denying MSG_OOB Flag in send/recv System Calls

As of version 3.37.5, Syd unconditionally denies the use of the **MSG_OOB** flag in all *send*(2), *sendto*(2), *sendmsg*(2), and *sendmmsg*(2) calls -- regardless of socket family -- by returning the **EOPNOTSUPP** ("Operation not supported on transport endpoint") *errno*(3). As of version 3.41.1, the restriction includes the system calls *recv*(2), *recvfrom*(2), *recvmsg*(2), and *recvmmsg*(2). This measure addresses long-standing security concerns with out-of-band messaging semantics in stream sockets, where urgent data bypasses normal in-order delivery rules and is handled via separate kernel paths. Such semantics are rarely required by modern software but introduce complexity and subtle state transitions inside the kernel's networking stack, which have historically led to memory safety bugs and race conditions exploitable from unprivileged code. By default, removing **MSG_OOB** support reduces the kernel attack surface for sandboxed processes without impacting typical application behavior. For controlled environments where **MSG_OOB** is explicitly required, Syd provides the opt-in *trace/allow_unsafe_oob:1* flag to restore legacy behavior, though enabling it reintroduces the inherent risks associated with out-of-band data handling. For more information refer to the following links:

- <https://googleprojectzero.blogspot.com/2025/08/from-chrome-renderer-code-exec-to-kernel.html>
- <https://chromium-review.googlesource.com/c/chromium/src/+6711812>
- <https://u1f383.github.io/linux/2025/10/03/analyze-linux-kernel-1-day-0aeb54ac.html>

Denying O_NOTIFICATION_PIPE Flag in pipe2

As of version 3.37.5, Syd unconditionally denies the use of the **O_NOTIFICATION_PIPE** flag in *pipe2*(2) by returning the **ENOPKG** ("Package not installed") *errno*(3), unless the *trace/allow_unsafe_pipe:1* option is provided at startup. This restriction addresses the security risks associated with notification pipes -- a specialized and seldom-used mechanism designed for delivering kernel event notifications (currently only from the keys subsystem) to userspace when the kernel is built with **CONFIG_WATCH_QUEUE**. Unlike normal pipes, notification pipes operate with distinct semantics and are tightly integrated with kernel internals, creating a more complex and less widely audited code path. Historically, vulnerabilities in notification pipe handling have demonstrated that exposing this functionality to unprivileged, sandboxed code can create exploitable kernel attack surface. Because typical sandboxed applications, including high-risk workloads such as browser renderers, have no legitimate need for notification pipes, Syd disables this flag by default, thereby eliminating an entire class of low-value yet high-risk kernel interfaces. The *trace/allow_unsafe_pipe:1* flag can be used

to re-enable this capability for controlled testing or compatibility purposes, but doing so reintroduces the underlying security concerns. Refer to the following links for more information:

- <https://chromium-review.googlesource.com/c/chromium/src/+4128252>
- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/log/?qt=grep&q=watch_queue

madvise(2) Hardening

As of version 3.41.3, Syd tightens its *seccomp(2)* BPF policy by argument-filtering *madvise(2)* to an allow-list that is safe for untrusted workloads and has well-understood locality: **MADV_SEQUENTIAL**, **MADV_DONTNEED**, **MADV_REMOVE**, **MADV_HUGEPAGE**, **MADV_NOHUGEPAGE**, **MADV_DONTDUMP**, **MADV_COLLAPSE**, **MADV_POPULATE_READ**, **MADV_POPULATE_WRITE**, and (since Linux 6.13) the lightweight guard operations **MADV_GUARD_INSTALL/MADV_GUARD_REMOVE** (page-table-level red zones that fault on access without VMA churn). The advice **MADV_HWPOISON** is denied and all other advice are treated as no-op because they enable cross-domain information leaks or system-wide pressure channels with no isolation benefit, e.g., **MADV_MERGEABLE** drives KSM deduplication which has been repeatedly shown to enable cross-VM/process side channels and targeted bit-flip exploitation (Flip Feng Shui) as well as newer remote and timing channels. **MADV_WILLNEED/MADV_RANDOM** manipulate page-cache residency and prefetch behavior that underpin page-cache side-channel attacks; and reclaim steering like **MADV_FREE/MADV_COLD/MADV_PAGEOUT** introduces externally observable memory-pressure/timing signals and accounting ambiguity that sandboxes should not expose; privileged page state changes **MADV_SOFT_OFFLINE/MADV_HWPOISON** are unnecessary in least-authority contexts and remain outside the sandbox contract even if capability checks would reject them. This design follows the strict syscall-and-argument allow-listing discipline also employed by Google's Sandbox2/Sandboxed-API while remaining specific to Syd's threat model. To temporarily relax this mitigation for tracing/compatibility, set *trace/allow_unsafe_madvise:1* at startup, otherwise unsafe advice remain blocked by default. Refer to the following links for more information:

- https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_razavi.pdf
- <https://www.ndss-symposium.org/wp-content/uploads/2022-81-paper.pdf>
- https://svs.informatik.uni-hamburg.de/publications/2024/Lindemann_ACSAC2024_FakeDD.pdf
- <https://arxiv.org/pdf/1901.01161>
- <https://lwn.net/Articles/790123/>
- <https://lwn.net/Articles/1011366/>
- <https://developers.google.com/code-sandboxing/sandbox2/explained>
- <https://developers.google.com/code-sandboxing/sandboxed-api/explained>

HISTORY & DESIGN

- **sydbox-0** <https://git.sr.ht/~alip/syd/tree/sydbox-0> is a *ptrace(2)* based sandbox.
- **sydbox-1** <https://git.sr.ht/~alip/syd/tree/sydbox-1> is a *ptrace(2)* and *seccomp(2)* based sandbox.
- **sydbox-2** <https://git.sr.ht/~alip/syd/tree/sydbox-1> is a *seccomp(2)* and *seccomp-notify* based sandbox.
- **sydbox-3** is a rewrite of **sydbox-2** in Rust and it's what you are looking at.

This codebase has a history of a bit over 15 years and up to this point we have used C11 as our implementation language for various reasons. With **sydbox-3** we are moving forwards one step and writing the sandbox from scratch using the Rust programming language with the only non-Rust dependency being libseccomp. Although we inherit many ideas and design decisions from the old codebase, we also don't shy away from radically changing the internal implementation making it much simpler, idiomatic, and less prone to bugs. We have *proper multiarch support* since release 3.0.11, e.g on x86-64, you can run your x32 or x86 binaries just fine under Syd.

This version takes advantage of multithreading and handles system calls using a thread pool whose size is equal to the number of CPUs on the running machine and utilises globsets to match a list of patterns at once, thus continues to perform reasonably well even with very long rulesets. This version also comes with four new sandboxing categories called **Lock Sandboxing**, **Memory Sandboxing**, **PID sandboxing**, **Stat Sandboxing**, **Force Sandboxing**: **Lock Sandboxing** utilises the Landlock Linux Security Module (LSM), **Memory Sandboxing** allows the user to define a per-process memory limit, **PID sandboxing** allows the user to define a limit on the maximum number of running tasks under the sandbox, **Stat Sandboxing** can be used to effectively *hide files and* directories from the sandboxed process whereas **Force Sandboxing** can be used to verify file checksums prior to exec, similar to HardenedBSD's Integriforce and NetBSD's Veriexec.

Finally, the new Syd has support for namespaces. Use e.g. *syd -munshare/user:1* to create a user namespace. You may use *mount*, *uts*, *ipc*, *pid*, *net*, and *cgroup* instead of *user* to create various namespaces. You may use the *container* profile as a shorthand to create namespaces with *syd -pcontainer*.

You may use Syd as your login shell because it is very practical to have a restricted user. To do this simply add */path/to/syd* to the file */etc/shells* and do *chsh -s /path/to/syd username* as root. In this mode the sandbox may be configured using the files */etc/user.syd-3* and *~/.user.syd-3*. If you want to restrict user configuration of the sandbox, lock the sandbox using *lock:on* at the end of the site-wide configuration file.

EXHERBO

Syd is the default sandbox of **Exherbo Linux**. We use it to provide a restricted environment under which package builds run with controlled access to file system and network resources. *exheres-0* has a function called *esandbox* to interact with Syd.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *seccomp(2)*, *pidfd_getfd(2)*, *pidfd_send signal(2)*, *ioctl(2)*, *ioctl_tty(2)*, *prctl(2)*, *namespaces(7)*, *cgroup_namespaces(7)*, *ipc_namespaces(7)*, *mount_namespaces(7)*, *network_namespaces(7)*, *pid_namespaces(7)*, *user_namespaces(7)*, *uts_namespaces(7)*

<https://exherbo.org/docs/eapi/exheres-for-smarties.html#sandboxing>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd(5)

NAME

syd - Document format for writing Syd profiles

API

Current version of the Syd command API is **3**. This version is **stable**.

CONFIGURATION

Syd is configured through sandbox commands. For multiple matching rules (e.g. two rules matching the same path), the last matching rule wins. There are two ways to supply sandbox commands. First, Syd may be configured using a configuration file. The path to the configuration file is specified using the **-P** command line switch. More than one configuration file may be specified this way. Single commands may also be passed via **-m** command line switch. Configuration profiles may be applied using the **-p** command line switch. See the **PROFILES** section for more information. Second, Syd may be configured using magic *stat(2)* calls during runtime. This is achieved by calling *stat(2)* system call on the special path `/dev/syd` followed by the sandbox command. Runtime configuration is only possible if the sandbox lock is **off**. The system call *stat(2)* was chosen because it is practical to invoke using builtin shell commands like:

```
; test -c /dev/syd/sandbox/read:on
```

which enables **Read Sandboxing** for a shell running under Syd. It is also possible to query certain values using the return value of the *stat(2)* call:

```
test -c /dev/syd/sandbox/read? && echo read sandboxing on || echo read sandboxing off
```

Some of these shell builtins may actually call other system calls such as *fstat(2)*, *lstat(2)*, *newfstatat(2)*, or *statx(2)*. Syd supports the same interface through all these system calls transparently. Check the manual page *syd(2)* for a description of the *stat(2)* interface.

NAMING

Configuration file naming of Syd follows a naming scheme which makes it possible to extract command API version from the file name. A Syd configuration file must have the extension **syd-** followed by the API version (e.g. **"syd-3"** for API version **3**).

SYNTAX

Input files must use the UTF-8 encoding. Config format is line oriented. Comments start with `"#"`. Inline comments are *not* supported. Blank lines are ignored. All the other lines are treated as if they were supplied to Syd via the `-m` command line switch. For a list of available sandbox commands, consult `syd(2)`. For a VIM syntax file for Syd profiles check here: <https://gitlab.exherbo.org/sydbox/sydbox/-/tree/main/vim>

As of version 3.15.1, Syd adds two additional features to configuration file parsing:

- Environment variable expansion is performed on arguments. By default **shellexpand** crate is used to perform expansion and a timeout may be set using **config/expand** to perform expansion using *wordexp(3)* instead. Notably, unset environment variables are not expanded to empty strings. On environment variable lookup errors and UTF-8 decoding errors Syd stops parsing and exits with error. This is done for safety as an unintended empty-string expansion can potentially cause the resulting sandboxing rule to allowlist unintended paths without the user easily noticing it. The user is recommended to set default values for environment variables using the familiar **`\${HOME}:/var/empty`** notation. If you really want empty-string expansion on unset environment variables, you can get this effect using the notation **`\${HOME}:-`** but this is not recommended and should be used with care.
- **include** directives can be used to request the inclusion of another configuration file. Upon reading an include line, Syd stops parsing the current file, validates the given include path and starts to parse the new configuration file. The file must not be writable by group or others for safety. For include files with relative paths, Syd searches the file under the directory of the previous configuration file rather than the current working directory for safety and ease of configuration. Loops in include directives are detected by caching the device id and inode of the configuration files. Note, this directive is not permitted when loading configuration from a file descriptor using the **load** command.

As of version 3.17.6, Syd adds the **include_profile** directive which may be used to include a Syd profile. See **syd-cat -plist** for the list of profiles.

PROFILES

Syd has a number of predefined profiles to make configuration easier. These profiles may be used standalone or stacked with other profiles and custom configuration to create various levels of isolation and confinement. To see the complete list of profiles, use **syd-cat -plist**. To list the rules of a profile, use **syd-cat -p<profile-name>**. Below you may find a brief list of common profiles and their functionality:

container	Enables Linux namespaces. You may refer to this profile shortly as just c .
immutable	Enables Linux namespaces and remounts the following directories <i>read only</i> in the new mount namespace: /etc , /home , /media , /mnt , /opt , /srv , and /usr . Further mount options such as nodev , noexec , nosuid , and noatime are also applied as necessary. In addition, /dev/shm and /tmp are mounted private and kernel filesystems are masked. See syd-cat -p immutable for the full list of mount options. You may refer to this profile shortly as just i .
privileged	Do not drop Linux capabilities at startup. Used to construct privileged containers. You may refer to this profile shortly as just p .
readonly	Deny all write sandbox capabilities to the entire root filesystem. You may refer to this profile shortly as just ro .
landlock	Enables LandLock and allows system directories for Lock Sandboxing . You may refer to this profile shortly as just l .
linux	Common Linux system profile, used by oci , paludis and user profiles.
oci	Used by <i>syd-oci(1)</i> as the default container profile.
paludis	Used by the Paludis package mangler.
noipv4	Disables IPv4 connectivity. You may refer to this profile shortly as just 6 .

noipv6	Disables IPv6 connectivity. You may refer to this profile shortly as just 4 .
core	Allows generation of coredumps. You may refer to this profile shortly as just C .
debug	Allows debuggers inside the sandbox. Syd does not use <i>ptrace</i> (1) with this profile, so tracers may attach. You may refer to this profile shortly as just D .
nomem	Allows unsafe memory (no W^X , no Memory-Deny-Write-Execute, allows e.g. JITs). You may refer to this profile shortly as just M .
nopie	Relaxes PIE (Position Independent Executable) restriction. You may refer to this profile shortly as just P .
quiet	Silences all access violations. You may refer to this profile shortly as just q .
rand	Enables randomized file descriptors. See Force Randomized File Descriptors section of the <i>syd</i> (7) manual page for more information. You may refer to this profile shortly as just r .
off	Turns all sandboxing off.
lib	libsyd helper profile. Turns all sandboxing off and sets sandbox lock to exec . Useful to configure Syd in the application using libsyd .
user	Allows user-specific directories, and connections, and parses the files /etc/user.syd-3 , and ~/.user.syd-3 if they exist. Syd sets the environment variables SYD_UID , SYD_GID , SYD_USER , SYD_HOME before parsing this profile. To enforce system-wide settings, set lock:on at the end of /etc/user.syd-3 . You may refer to this profile shortly as just u .
kvm	Profile to allowlist KVM <i>ioctl</i> (2) requests without path check. Read: https://www.kernel.org/doc/Documentation/virtual/kvm/api.txt
tty	Profile to allow TTY access, used by oci , paludis , and user profiles. Syd sets the environment variable SYD_TTY before parsing this profile. If the process has no controlling terminal, SYD_TTY variable is set to /dev/null .
firefox	Profile to relax restrictions to enable running Firefox family browsers. You may refer to this profile shortly as just ff .

Stacking Profiles

It is possible to stack multiple profiles to configure a more restricted sandbox. Remember the order you stack the profiles matter, *the last matching* rule wins. Below are some examples:

- `syd -puser -pimmutable -mroot:/mnt/gnu ...`
- `syd -ppaludis -plandlock -mallow/lock/write+/var/tmp ...`

It is also possible to combine the one character shortcuts of helper profiles, in order to stack them together. Below are some examples:

- `syd -pMPX ...` # Disable MDWE, PIE and exec restrictions.
- `syd -puiq ...` # Parse user profile, create an immutable container, and silence access violations.

Login shell and the User Profile

When invoked without arguments, **/bin/sh** is executed under Syd with the **user** profile as a login shell, use **SYD_SH** environment variable to override the shell to execute.

SECURITY

As of version 3.30.0, Syd aborts with error if path to a specified configuration file has a symbolic link in *any* of its path components. Therefore, the user *must* supply canonicalized paths as configuration file arguments. As of version 3.46.0, parent (".") components are not permitted in configuration file path and configuration files must be regular files.

EXAMPLE

```
# Syd profile for OpenNTPD

# Seccomp sandbox
sandbox/read,stat,write,exec,net:on

# Landlock
sandbox/lock:on

# Provide isolation using namespaces.
unshare/mount,uts,pid,ipc,cgroup:1

# Allow adjtimex and keep CAP_SYS_TIME.
trace/allow_unsafe_time:1

# Mount everything ro except /var
bind+tmpfs:/dev/shm:nodev,nosuid,noexec
bind+tmpfs:/tmp:nodev,nosuid
bind+/etc:/etc:ro,nodev,noexec,nosuid,noatime
bind+/home:/home:ro,nodev,noexec,nosuid,noatime
bind+/media:/media:ro,nodev,noexec,nosuid,noatime
bind+/mnt:/mnt:ro,nodev,noexec,nosuid,noatime
bind+/opt:/opt:ro,nodev,nosuid,noatime
bind+/srv:/srv:ro,nodev,noexec,nosuid,noatime
bind+/usr:/usr:ro,nodev,noatime

# Hide Syd
deny/read,stat,write+/proc/1/**

# Allow listen to the ntp port on loopback.
allow/net/bind+loopback!123

# Allow connections to NTP servers.
allow/net/connect+any!53
allow/net/connect+any!123
allow/net/connect+any!65535

# Allow logging to syslog.
allow/net/connect+/dev/log

# Allow `listen wildcard`
allow/net/bind+0.0.0.0!0
allow/net/connect+0.0.0.0!0

# Allow listen to the ntpd socket.
allow/net/bind+/run/ntpd.sock
allow/net/bind+/var/run/ntpd.sock
allow/write+/run/ntpd.sock
allow/write+/var/run/ntpd.sock

# Allow access to system paths
allow/read,stat+/dev/urandom
allow/lock/read+/dev/urandom
allow/read,stat+/etc/hosts
allow/lock/read+/etc/hosts
allow/read,stat+/etc/ntpd.conf
allow/lock/read+/etc/ntpd.conf
allow/read,stat+/etc/passwd
allow/lock/read+/etc/passwd
allow/read,stat+/etc/resolv.conf
allow/lock/read+/etc/resolv.conf
allow/read,stat+/etc/services
allow/lock/read+/etc/services
allow/read,stat+/usr/share/zoneinfo-posix/UTC

# chroot /var/empty && cd /
allow/stat+/
allow/stat+/var/empty
allow/write+/dev/null
allow/lock/write+/dev/null
```

```
# Allow executing the ntp binary.
allow/lock/read+/proc
allow/lock/read+/usr
allow/lock/write+/run
allow/lock/write+/var/run
allow/exec+/usr/**/bin/openntpd*

# Allow writing the drift file.
allow/write+/var/db/ntpd.drift
allow/lock/write+/var/db/ntpd.drift

# Lock configuration
lock:on
```

SEE ALSO

syd(1), *syd(2)*, *syd(7)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd(2)

NAME

/dev/syd virtual system call interface

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct stat stat;

/* Execute sandbox commands */
int stat("/dev/syd/[command]", &stat); // $ syd -m command
int stat("/dev/syd/[config]?", &stat);
int stat("/dev/syd/[config]:[value]", &stat); // $ syd -m config:value
int stat("/dev/syd/[list]+[value]", &stat); // $ syd -m list+value
int stat("/dev/syd/[list]-[value]", &stat); // $ syd -m list-value
int stat("/dev/syd/[list]^[value]", &stat); // $ syd -m list^value
int stat("/dev/syd/[command]![value]", &stat);

/* Read sandbox state as JSON */
int open("/dev/syd", O_RDONLY);

/* Read syd.el which is the Emacs Lisp implementation of the API */
int open("/dev/syd.el", O_RDONLY);

/* Read syd.sh which exports esyd shell function */
int open("/dev/syd.sh", O_RDONLY);
```

DESCRIPTION

The **/dev/syd** virtual system call interface is a unique mechanism designed for runtime configuration of the Syd sandbox environment. It enables sandboxed processes to interact with the Syd process to dynamically adjust sandbox settings or query its state. This interaction is facilitated through the use of virtual system calls, specifically via the *stat(2)* system call, applied to specially constructed paths under **/dev/syd**. This interface allows for a range of operations, including enabling or disabling sandbox features, appending or removing elements from lists, querying the sandbox state, and executing special Syd commands. Operations are specified through paths constructed with **/dev/syd** as the prefix, followed by a sandbox command and an operation character that denotes the desired action:

- **:** for setting a value (boolean, integer, string),
- **?** for querying a value,
- **+** for appending to a string vector,
- **-** for removing an element from a string vector,
- **^** for removing all matching elements from a string vector, and
- **!** for executing a special Syd command.

The type **string-map** is similar to **string-vec**, except the operator `^` does not accept an argument and removes all elements from the string map. The `-` operator of a string map is functionally equivalent to the `^` operator of a string vector in that both remove all matching elements from the respective set.

This interface supports a flexible and powerful method for managing sandbox policies dynamically, allowing for real-time adjustments to the security and operational behavior of sandboxed processes. **libsyd** is a comprehensive C library designed for interfacing with the Syd stat interface. It offers functionalities for managing sandbox states, and facilitating runtime configuration and interaction with the Syd sandboxing environment. **gosyd** is a Go module that uses *cgo* to use **libsyd**. **plsyd** is a Perl module that uses *FFI::Platypus* to use **libsyd**. **pysyd** is a Python module that uses *ctypes* to use **libsyd**. **rbsyd** is a Ruby module that uses *ffi* gem to use **libsyd**. **syd.el** is an *Emacs Lisp* implementation of the Syd stat interface.

COMMANDS

The **/dev/syd** interface supports the following commands for runtime configuration of the sandbox. Each command can be invoked through the *stat(2)* system call on special paths under **/dev/syd**. Note, Syd provides similar interfaces for the *stat(2)* interface, **-m** command-line option and the configuration file. Some sandbox commands only take affect when they're submitted on startup, such as **unshare/user**, and **sandbox/lock**. Such commands are noted as **static** in the descriptions below.

stat

This command causes Syd to output sandbox state on standard error.

reset

This command causes Syd to reset sandboxing to the default state. Allowlists, denylists and filters are going to be cleared. The state of the sandbox lock is not affected by reset. This ensures an unintended reset cannot open window for a sandbox bypass. In addition, the state of Crypt sandboxing is not affected by reset too. This ensures concurrent or near-concurrent encryption operations continue uninterrupted.

panic

This command causes Syd to exit immediately with code 127.

Due to security reasons, this command is only available via the virtual *stat(2)* call, it may not be used with the **-m** command line switch or in a configuration file.

Due to safety reasons, panic may not be called when Crypt sandboxing is on. In this case the virtual *stat(2)* returns -1 and sets *errno(3)* to **EBUSY**. This ensures concurrent or near-concurrent encryption operations continue uninterrupted.

ghost

This command initiates Ghost mode. Ghost mode is irreversible so you can call this command only once during Syd runtime. Refer to the **Ghost** mode section of the *syd(7)* manual page for more information. This command implies **reset**, ie. the sandbox state is reset before Ghost mode initiation to ensure there're no run-away exec processes after the invocation of the **ghost** command. Ghost mode is only available via the virtual *stat(2)* call, it can not be used with the **-m** command line switch or in a configuration file.

config/expand

type	integer (u64)
default	0
static	yes

Given zero as timeout in seconds, which is the default, enables environment variable and tilde expansion using the **shellexpand** crate. This runs much faster as it does not require confinement, however it does not support command substitution and recursive environment variable expansion like *wordexp(3)* does. Unset environment variables are not expanded to empty strings. On environment variable lookup errors and UTF-8 decoding errors Syd stops parsing and exits with error. This is done for safety as an unintended empty-string expansion can potentially cause the resulting sandboxing rule to allowlist unintended paths without the user easily noticing it. The user is recommended to set default values for environment variables using the familiar **\${HOME:-/var/empty}** notation. Empty-string expansion on unset environment variables can still be done using the notation **\${HOME:-}** but this is not recommended and should be used with care.

Given a positive integer as timeout in seconds, enables environment variable expansion and command substitution for configuration using *wordexp(3)*. The fork process which calls **/bin/sh** for expansion is executed in a confined environment and it is terminated if its runtime exceeds the given timeout. Confinement is done using *landlock(7)*, *namespaces(7)* and *seccomp(2)*.

This is a startup-only setting. For safety reasons, no expansion is performed for runtime configuration.

ipc

type	string
static	yes

Configure sandbox during runtime using the given UNIX socket address with kernel-validated peer authentication. Authentication leverages **SCM_CREDENTIALS** and **SO_PASSCRED** mechanisms to verify that connecting processes share identical UID and GID with the IPC worker process. Authentication UID and GID may be overridden by **ipc/uid** and **ipc/gid** options at startup. This kernel-enforced authentication prevents privilege escalation and unauthorized access by validating credentials on every message, ensuring only the specified user and group or the system administrator can execute IPC commands.

If the argument starts with the character **,** *the address is taken to be an abstract* UNIX socket. Use the keywords none** or off* to unset a previously set IPC address. The IPC implementation is inspired by HAProxy's stats socket implementation. All responses except the **stats** command are in compact JSON. User is recommended to use the **version** command to check the API version prior to use. As a safety measure, the IPC service is provided as long as the sandbox is unlocked. When the sandbox is locked, the *syd_ipc* thread exits. This thread makes no attempt to *unlink(2)* the UNIX domain socket path at startup or exit. The user should perform the cleanup or use abstract sockets which is recommended. To access the socket, an external utility such as *socat(1)* is required. Socat is a swiss-army knife to connect anything to anything. We use it to connect terminals to the socket, or a couple of stdin/stdout pipes to it for scripts. The two main syntaxes we'll use are the following:

```
# socat ~/.syd/sandbox.sock stdio
# socat ~/.syd/sandbox.sock readline
```

The first one is used with scripts. It is possible to send the output of a script to Syd, and pass Syd's output to another script. That's useful for retrieving sandbox configuration as JSON for example. The second one is only useful for issuing commands by hand. It has the benefit that the terminal is handled by the readline library which supports line editing and history, which is very convenient when issuing repeated commands (eg: watch a counter).

The socket supports three operation modes:

- non-interactive, silent
- interactive, silent
- interactive with prompt

The non-interactive mode is the default when *socat*(1) connects to the socket. In this mode, a single line may be sent. It is processed as a whole, responses are sent back, and the connection closes after the end of the response. This is the mode that scripts and monitoring tools use. A single command may be sent at a time only. The interactive mode allows new commands to be sent after the ones from the previous lines finish. It exists in two variants, one silent, which works like the non-interactive mode except that the socket waits for a new command instead of closing, and one where a prompt is displayed (':') at the beginning of the line. The interactive mode is preferred for advanced tools while the prompt mode is preferred for humans.

The mode can be changed using the **prompt** command. By default, it toggles the interactive+prompt modes. Entering **prompt** in interactive mode will switch to prompt mode. The command optionally takes a specific mode among the following:

- **n**: non-interactive mode (single command and quits)
- **i**: interactive mode (multiple commands, no prompt)
- **p**: prompt mode (multiple commands with a prompt)

Since the default mode is non-interactive, **prompt** must be used as the first command in order to switch it, otherwise the previous command will cause the connection to be closed. Switching to non-interactive mode will result in the connection to be closed after all the commands of the same line complete.

For this reason, when debugging by hand, it's quite common to start with the **prompt** command:

```
# socat ~/.syd/sandbox.sock readline
prompt
; stats
...
;
```

Interactive tools might prefer starting with **prompt i** to switch to interactive mode without the prompt.

The following commands are supported in addition to the *syd*(2) API:

- **stat**: Prints sandbox state in compact JSON.
- **stats**: Prints sandbox state in human-readable format.
- **version**: Prints IPC api version in compact JSON.

The commands **quit** and **exit** may be used to close a socket connection. The command **ping** is supported for aliveness checks.

ipc/uid

type	uid
default	Uid::current
static	yes

User ID override for IPC authentication. Specifies the UID that connecting processes must possess to authenticate with the IPC worker. Accepts either numeric user IDs or user names. When specified as a user name, the system resolves it to the corresponding UID using *getpwnam*(3). Defaults to the current process UID obtained via *getuid*(2). When set, the IPC worker validates that all connecting clients have this exact UID via **SCM_CREDENTIALS** authentication. This setting allows privilege delegation scenarios where the IPC worker runs as one user but accepts connections from processes running as a different specific UID. Set the option to **none** or **off** to disable UID authentication for IPC.

ipc/gid

type	gid
default	Gid::current
static	yes

Group ID override for IPC authentication. Specifies the GID that connecting processes must possess to authenticate with the IPC worker. Accepts either numeric group IDs or group names. When specified as a group name, the system resolves it to the corresponding GID using *getgrnam(3)*. Defaults to the current process GID obtained via *getgid(2)*. When set, the IPC worker validates that all connecting clients have this exact GID via **SCM_CREDENTIALS** authentication. This setting enables group-based access control where multiple users belonging to the same group can access the IPC interface. Set the option to **none** or **off** to disable GID authentication for IPC.

lock

type	string
------	---------------

Set the state of the sandbox lock. Possible values are **on**, **off**, **exec**, **ipc**, and **read** or shortly just **1**, **0**, **x**, **i**, and **r**. The values are case-sensitive. The values **ro**, **readonly** and **read-only** are also permitted for **read** mode which was added as of version 3.39.0. Specifying just **lock** without value or shortly **l** is permitted as a short-hand for **lock:on**.

If the sandbox lock is **on** no sandbox commands are allowed. If sandbox lock is **read**, only reads are allowed but NOT edits. A read locked sandbox makes available only the read-only *open(2)* hooks of the *syd(2)* virtual system call API to the sandbox process. *stat(2)* hooks for edits are NOT permitted in a read locked sandbox.

If **exec** is specified, the sandbox lock is set to **on** for all processes except the initial process, aka Syd exec child. If the sandbox lock is **ipc**, sandbox commands may only be specified using the IPC socket. The IPC socket is a UNIX socket which may or may not be accessible from within the sandbox depending on sandbox ACL rules.

Transition from lock modes **off**, **exec**, and **ipc** into one of **read** and **on** is one-way and idempotent: It results in the sandbox policy getting sealed in memory using the *mseal(2)* system call either immediately or simultaneously with sandbox process startup. Transitions between lock modes **read** and **on** are not permitted.

The sandbox lock used to default to **exec** but as a hardening measure and to ensure security by default, as of version 3.17.0, this has been changed such that the default is **unset** and if no lock clause has been specified by the time Syd executes the initial sandbox process, then the sandbox lock is automatically set to **on**. This means if no **lock** clause is specified in any of the profiles, configuration files or **-m** CLI arguments, the lock will be **on** by default. As of version 3.35.2, this default is set to **ipc** if the **ipc** command was specified but lock was not set explicitly. Setting lock to **on** at any point during configuration parsing prevents further commands from being emitted by the sandbox. This feature may be used to lock site-wide defaults for a Syd login shell by adding a **lock:on** clause at the end of the site-wide configuration file which prevents Syd from subsequently parsing the user configuration file, practically enforcing the site-wide defaults.

Setting lock to **off**, **exec**, or **ipc** at startup makes Syd skip preventing *execve(2)* and *execveat(2)* system calls as part of the **Execution Control (EEC)** feature. This is done to allow **cmd/exec** command to execute commands outside the sandbox. This filter to prevent *exec(3)* is only applied when the sandbox is locked.

log/level

type	string
default	warn

Set the log level. Available log levels are **emerg**, **alert**, **crit**, **error**, **warn**, **notice**, **info**, and **debug**. Defaults to **warn** unless **SYD_LOG** environment variable is set at startup. An integer in the closed range of [0,7] can also be used as an argument to set the log level, where **0** corresponds to **emerg** and **7** corresponds to **debug**. All access violations except the **stat** and **walk** categories are logged with the **warn** level. Stat and Walk categories are logged with the **notice** level. Startup messages are logged with the **info** level.

log/lock/same_exec_off

type	boolean
default	off

Disables logging of denied accesses originating from the thread creating the *landlock(7)* domain, as well as its children, as long as they continue running the same executable code (i.e., without an intervening *execve(2)* call). This is intended for programs that execute unknown code without invoking *execve(2)*, such as script interpreters. Programs that only sandbox themselves should not set this flag, so users can be notified of unauthorized access attempts via system logs.

This option requires *landlock(7)* ABI 7 support which is new in Linux-6.15. Setting this option is a NO-OP otherwise. Setting this option is also a NO-OP when **sandbox/lock** is off. Multiple options may be set or unset at once by passing them as a comma-delimited list. Environment variables in the value are expanded.

log/lock/new_exec_on

type	boolean
default	off

Enables logging of denied accesses after an *execve(2)* call, providing visibility into unauthorized access attempts by newly executed programs within the created *landlock(7)* domain. This flag is recommended only when all potential executables in the domain are expected to comply with the access restrictions, as excessive audit log entries could make it more difficult to identify critical events.

This option requires *landlock(7)* ABI 7 support which is new in Linux-6.15. Setting this option is a NO-OP otherwise. Setting this option is also a NO-OP when **sandbox/lock** is off. Multiple options may be set or unset at once by passing them as a comma-delimited list. Environment variables in the value are expanded. A sandboxer should not log denied access requests to avoid spamming logs, therefore this option is off by default. Use this option to test audit logging.

log/lock/subdomains_off

type	boolean
default	off

Disables logging of denied accesses originating from nested *landlock(7)* domains created by the caller or its descendants. This flag should be set according to runtime configuration, not hardcoded, to avoid suppressing important security events. It is useful for container runtimes or sandboxing tools that may launch programs which themselves create *landlock(7)* domains and could otherwise generate excessive logs. Unlike **log/lock/same_exec_off**, this flag only affects future nested domains, not the one being created.

This option requires *landlock(7)* ABI 7 support which is new in Linux-6.15. Setting this option is a NO-OP otherwise. Setting this option is also a NO-OP when **sandbox/lock** is off. Multiple options may be set or unset at once by passing them as a comma-delimited list. Environment variables in the value are expanded.

log/verbose

type	u8
default	0

Set verbose logging level. Syd acquires and logs various additional information depending on the level of verbosity. Supported verbosity levels are given below:

0: Raw logs only, this is the default.

1: Log *ioctl* names(2) under the *ctl* key. *pandora*(1) uses this.

2: Log process name change attempts with the **PR_SET_NAME** *prctl*(2).

3: Enrich *seccomp*(2) requests under the *req* key.

Levels above 2 are intended for malware analysis. Setting the log level to a value above the highest supported level is equivalent to setting verbosity to the highest supported level.

pty/row

type	ushort
default	<inherit>
static	yes

Set row size for PTY sandboxing. Default is to inherit the window-size. Use the keyword **none** to unset a previously set value. You may shortly refer to this option as **pty/x**.

pty/col

type	ushort
default	<inherit>
static	yes

Set column size for PTY sandboxing. Default is to inherit the window-size. Use the keyword **none** to unset a previously set value. You may shortly refer to this option as **pty/y**.

setenv

type	command
static	yes

Set an environment variable from within a Syd profile. Environment variables in the value are expanded before calling *setenv*(3). Setting internal Syd environment variables, i.e those that start with "SYD_", isn't permitted. Illustrative examples are given below:

```
setenv!HOME=/tmp
setenv!HOME=${HOME}/.syd
setenv!HOME=${HOME::-/tmp}/.syd
```

unsetenv

type	command
static	yes

Unset an environment variable from within a Syd profile. Unsetting internal Syd environment variables, i.e those that start with "SYD_", isn't permitted. Illustrative examples are given below:

```
unsetenv!PWD
unsetenv!TZ
```

clearenv

type	command
static	yes

Clear all environment variables from within a Syd profile. This command does not clear internal Syd environment variables, i.e those that start with "SYD_".

sandbox/walk

type	boolean
default	off
query	yes

Turn Walk sandboxing **on** or **off**.

For performance reasons, this sandboxing is off by default.

sandbox/stat

type	boolean
default	off
query	yes
static	yes

Turn Stat sandboxing **on** or **off**.

For performance reasons, this sandboxing is off by default and setting it on only works at startup. If not given at startup, Syd will just allow *access(2)*, *faccessat(2)*, *faccessat2(2)*, *getdents64(2)*, *readlink(2)*, *readlinkat(2)*, *stat(2)*, *stat64(2)*, *statx(2)*, *lstat(2)*, *lstat64(2)*, *fstatat64(2)*, *newfstatat(2)*, *fstat(2)*, *fstat64(2)*, *statfs(2)*, *statfs64(2)*, *fstatfs(2)*, *fstatfs64(2)*, *getxattr(2)*, *fgetxattr(2)*, *lgetxattr(2)*, *getxattrat(2)*, *listxattr(2)*, *flistxattr(2)*, *llistxattr(2)*, *llistxattrat(2)*, *fanotify_mark(2)*, and *inotify_add_watch(2)* system calls at seccomp-bpf level. Turning this sandboxing off during runtime is still possible, in this case the respective system calls handlers will skip the access checks. As an exception, if Stat sandboxing is off but sandbox lock is one of *off* or *exec*, the system calls *stat(2)*, *stat64(2)*, *statx(2)*, *lstat(2)*, *lstat64(2)*, *fstatat64(2)*, and *newfstatat(2)* are handled at userspace to support the *syd(2)* API.

sandbox/read

type	boolean
default	on
query	yes

Turn Read sandboxing **on** or **off**.

sandbox/write

type	boolean
default	on
query	yes

Turn Write sandboxing **on** or **off**.

sandbox/exec

type	boolean
default	on
query	yes

Turn Exec sandboxing **on** or **off**.

sandbox/ioctl

type	boolean
default	on
query	yes
static	yes

Turn Ioctl sandboxing **on** or **off**.

For performance reasons, this only works at startup. If not given at startup, Syd will just allow the *ioctl(2)* system call at seccomp-bpf level. Turning this sandboxing off during runtime is still possible, in this case the respective system calls handlers will skip the access checks.

As of version 3.36.0, *ioctl(2)* requests to block devices are always denied, and *ioctl(2)* requests to magic links are denied unless **trace/allow_unsafe_magiclinks:true** is set.

sandbox/create

type	boolean
default	on
query	yes

Turn Create sandboxing **on** or **off**.

sandbox/delete

type	boolean
default	on
query	yes

Turn Delete sandboxing **on** or **off**.

sandbox/rename

type	boolean
default	on
query	yes

Turn Rename sandboxing **on** or **off**.

sandbox/symlink

type	boolean
default	on
query	yes

Turn Symlink sandboxing **on** or **off**.

sandbox/truncate

type	boolean
default	on
query	yes

Turn Truncate sandboxing **on** or **off**.

sandbox/chdir

type	boolean
default	off
query	yes
static	yes

Turn Chdir sandboxing **on** or **off**.

For performance reasons, this sandboxing is off by default and setting it on only works at startup. If not given at startup, Syd will just allow *chdir*(2) and *fchdir*(2) system calls at seccomp-bpf level. Turning this sandboxing off during runtime is still possible, in this case the respective system calls handlers will skip the access checks.

sandbox/readdir

type	boolean
default	on
query	yes

Turn Readdir sandboxing **on** or **off**.

sandbox/mkdir

type	boolean
default	on
query	yes

Turn Mkdir sandboxing **on** or **off**.

sandbox/rmdir

type	boolean
default	on
query	yes

Turn Rmdir sandboxing **on** or **off**.

sandbox/chown

type	boolean
default	on
query	yes

Turn Chown sandboxing **on** or **off**.

sandbox/chgrp

type	boolean
default	on
query	yes

Turn Chgrp sandboxing **on** or **off**.

sandbox/chmod

type	boolean
default	on
query	yes

Turn Chmod sandboxing **on** or **off**.

sandbox/chattr

type	boolean
default	on
query	yes

Turn Chattr sandboxing **on** or **off**.

sandbox/chroot

type	boolean
default	on
query	yes

Turn Chroot sandboxing **on** or **off**.

sandbox/utime

type	boolean
default	on
query	yes

Turn Utime sandboxing **on** or **off**.

sandbox/mkdev

type	boolean
default	on
query	yes

Turn Mkdev sandboxing **on** or **off**.

sandbox/mkfifo

type	boolean
default	on
query	yes

Turn Mkfifo sandboxing **on** or **off**.

sandbox/mktemp

type	boolean
default	on
query	yes

Turn Mktemp sandboxing **on** or **off**.

sandbox/net

type	boolean
default	on
query	yes

Turn Network sandboxing **on** or **off**.

sandbox/lock

type	boolean
query	yes
static	yes

Turn Landlock sandboxing **on** or **off**.

sandbox/force

type	boolean
query	yes

Turn Force sandboxing **on** or **off**.

sandbox/tpe

type	boolean
default	on
query	yes

Turn Trusted Path Execution (TPE) sandboxing **on** or **off**.

sandbox/crypt

type	boolean
default	off
query	yes

Turn Crypt sandboxing **on** or **off**.

To set this option **on**, a key must have already been specified with **crypt/key** or *syd*(1) will exit with the *errno*(3) **ENOKEY**.

Note, setting this sandboxing type to **on** implies **trace/allow_safe_kcapi:true** to allow cryptographic operations using the Kernel Cryptography API (KCAPI).

Note, setting this sandboxing type to **on** implies **trace/exit_wait_all:true** so as not to leave any ongoing encryption processes behind on sandbox process exit.

sandbox/proxy

type	boolean
default	off
query	yes
static	yes
oci	no

Turn Proxy sandboxing **on** or **off**.

Defaults to proxying through TOR. See the options "proxy/addr", "proxy/port", "proxy/ext/host", and "proxy/ext/port" to configure a different proxy.

Implies **unshare/net:true**.

Requires *syd-tor*(1) helper utility to be under PATH. *syd-tor*(1) is executed once at startup, it runs as a single process and this process runs at most as long as the owner Syd process. See the *syd-tor*(1) manual page for more information.

sandbox/pty

type	boolean
default	on
query	yes
static	yes
oci	no

Turn PTY sandboxing **on** or **off**.

Requires *syd-pty*(1) helper utility to be under PATH. *syd-pty*(1) is executed once at startup, it runs as a single process and this process runs at most as long as the owner Syd process. See the *syd-pty*(1) manual page for more information.

Note, this option has no effect unless both standard input and standard output are attached to a TTY at startup.

sandbox/mem

type	boolean
default	off
query	yes
static	yes

Turn Memory sandboxing **on** or **off**.

For performance reasons, this only works at startup. If not given at startup, Syd will just allow *brk(2)*, *mmap(2)*, *mmap2(2)*, and *mremap(2)* system calls at seccomp-bpf level. Turning this sandboxing off during runtime is still possible, in this case the respective system calls handlers will skip the access checks.

sandbox/pid

type	boolean
default	off
query	yes

Turn PID sandboxing **on** or **off**.

default/walk

type	string
default	deny

Specify the default action for Walk sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/stat

type	string
default	deny

Specify the default action for Stat sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/read

type	string
default	deny

Specify the default action for Read sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/write

type	string
default	deny

Specify the default action for Write sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/exec

type	string
default	deny

Specify the default action for Exec sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/ioctl

type	string
default	deny

Specify the default action for Ioctl sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/create

type	string
default	deny

Specify the default action for Create sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/delete

type	string
default	deny

Specify the default action for Delete sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "stop", "abort", "kill", "panic", or "exit", where the default is "deny".

default/rename

type	string
default	deny

Specify the default action for Rename sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "stop", "abort", "kill", "panic", or "exit", where the default is "deny".

default/symlink

type	string
default	deny

Specify the default action for Symlink sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "stop", "abort", "kill", "panic", or "exit", where the default is "deny".

default/truncate

type	string
default	deny

Specify the default action for Truncate sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/chdir

type	string
default	deny

Specify the default action for Chdir sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/readdir

type	string
default	deny

Specify the default action for Readdir sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/mkdir

type	string
default	deny

Specify the default action for Mkdir sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/rmdir

type	string
default	deny

Specify the default action for Rmdir sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/chown

type	string
default	deny

Specify the default action for Chown sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/chgrp

type	string
default	deny

Specify the default action for Chgrp sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/chmod

type	string
default	deny

Specify the default action for Chmod sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/chattr

type	string
default	deny

Specify the default action for Chattr sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/chroot

type	string
default	deny

Specify the default action for Chattr sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/utime

type	string
default	deny

Specify the default action for Utime sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/mkdev

type	string
default	deny

Specify the default action for Mkdev sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/mkfifo

type	string
default	deny

Specify the default action for Mkfifo sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/mktemp

type	string
default	deny

Specify the default action for Mktemp sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/net

type	string
default	deny

Specify the default action for Network sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/block

type	string
default	deny

Specify the action for IP blocklist violations.

The value must be exactly one of "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/force

type	string
default	deny

For force sandboxing, define the default action to take when the path of a binary is not in the Integrity Force map.

The value must be either one of "warn", "filter", "deny", "panic", "stop", "abort", "kill", "exit", where the default is "deny".

default/segvguard

type	string
default	deny

Specify the action for SegvGuard access violations.

The value must be exactly one of "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/tpe

type	string
default	deny

Specify the action for TPE sandboxing access violations.

The value must be exactly one of "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/mem

type	string
default	deny

Specify the action for Memory sandboxing access violations.

The value must be exactly one of "allow", "warn", "filter", "deny", "panic", "stop", "abort", "kill", or "exit", where the default is "deny".

default/pid

type	string
default	kill

Specify the action for PID sandboxing access violations.

The value must be either one of "warn", "filter", "stop", "abort", "kill", "exit", where the default is "kill".

default/lock

type	string
default	kill
static	yes

Specify the compatibility level for Lock sandboxing.

The value must be either one of **kill**, **deny**, **warn**. **kill** stands for the Landlock compatibility level **hard-requirement**, whereas **deny** stands for **soft-requirement** and **warn** stands for **best-effort**.

As of version 3.35.0, the default level has been promoted from **warn** to **kill** to adhere to the principle of secure defaults. Again, as of this version **ENOENT**, i.e. **No such file or directory** errors are fatal unless compatibility level is set to **best-effort** at startup using **default/lock:warn**.

For more information on Landlock compatibility levels, see: <https://landlock.io/rust-landlock/landlock/trait.Compatible.html>

unshare/mount

type	boolean
query	yes
static	yes
oci	no

Create Mount namespace on startup, implies **unshare/pid:true**.

unshare/uts

type	boolean
query	yes
static	yes
oci	no

Create UTS namespace on startup.

unshare/ipc

type	boolean
query	yes
static	yes
oci	no

Create IPC namespace on startup.

unshare/user

type	boolean
query	yes
static	yes
oci	no

Create User namespace on startup.

unshare/pid

type	boolean
query	yes
static	yes
oci	no

Create Pid namespace on startup, implies **unshare/mount:true**.

Syd mounts private *procfs*(5) in this mode.

As of version 3.37.2 *procfs*(5) is mounted with **hidepid=4** option which is Linux>=5.8.

As of version 3.39.0 *procfs*(5) is mounted with **subset=pid** option which is Linux>=5.8, unless **trace/allow_unsafe_proc_files** is set at startup.

unshare/net

type	boolean
query	yes
static	yes
oci	no

Create Net namespace on startup.

unshare/cgroup

type	boolean
query	yes
static	yes
oci	no

Create CGroup namespace on startup.

unshare/time

type	boolean
query	yes
static	yes
oci	no

Create Time namespace on startup. Syd resets the boot-time clock such that *uptime*(1) will report container uptime rather than host uptime. Use *time* command to override default and set alternative time.

root

type	string
static	yes
oci	no

Change the root mount to the given new root directory at startup using *pivot_root(2)*. Destination path arguments of *bind* commands are interpreted relative to this directory. The directories *\$root/dev*, and *\$root/proc* must exist to mount private filesystems. In addition, target paths of the *bind* commands must also be manually created by the user.

This option does nothing without *unshare/mount:1*.

As of version 3.23.14, symbolic links are not followed in any part of the root directory and path traversal using *..* is not permitted. In addition, root directory must be an absolute path, relative paths are not permitted.

As of version 3.35.0, the special keyword *tmpfs*, or shortly *tmp* or just *t*, is supported to make Syd mount a private *tmpfs(5)* filesystem as the root directory. In this mode, Syd is going to attempt to create target paths inside the private temporary filesystem. Similarly, as of version 3.45.0, the special keyword *ramfs*, or shortly *ram* or just *r*, is supported to make Syd mount a private *ramfs(5)* filesystem as the root directory. *ramfs(5)* is limited compared to *tmpfs(5)* and should only be preferred when the host Linux kernel isn't configured with the **CONFIG_TMPFS** option. The private root directory is mounted with the options *nodev*, *noexec*, *nosuid*, *nosymfollow*, *noatime*, and *mode=700*.

As of version 3.35.2, the special keywords *none* and *off* may be used to unset a previously set *root* directory.

root/map

type	boolean
static	yes
oci	no

Map current user to root in the sandbox on startup.

This option does nothing without *"unshare/user:1"*.

root/fake

type	boolean
static	yes

In **fakeroot** mode, the system will return a user/group id of **0**, mimicking the **root** user. This allows users to execute commands with apparent root privileges, without actual superuser rights. It's useful for tasks like package building where root-like environment is needed, but not actual root permissions.

time

type	i64
static	yes
oci	no

Set clock boottime and monotonic offset (seconds) in Time Namespace. To set boottime and monotonic offsets separately, use the options **time/boot**, and **time/mono**. This option is a shorthand to set both at the same time to the same offset. Use the keywords **none** or **off** to unset a previously set offset.

time/boot

type	i64
static	yes
oci	no

Set clock boottime offset (seconds) in Time Namespace. Use the keywords **none** or **off** to unset a previously set offset.

time/mono

type	i64
static	yes
oci	no

Set clock monotonic offset (seconds) in Time Namespace. Use the keywords **none** or **off** to unset a previously set offset.

uts/host

type	string
default	localhost

Set UTS host name in the sandbox. Name is limited to 64 characters. Name may be empty. Name may not have nul bytes. Default is **localhost**.

Useful when combined with **unshare/uts:true**. As of version 3.40.0, the value of this option is returned at *uname(2)* boundary in **nodename** field of the **utsname** structure regardless of the **unshare/uts** option.

uts/domain

type	string
default	(none)

Set NIS/YP domain name in the sandbox. Name is limited to 64 characters. Name may be empty. Name may not have nul bytes. Default is **(none)**.

Useful when combined with **unshare/uts:true**. As of version 3.40.0, the value of this option is returned at *uname(2)* boundary in **domainname** field of the **utsname** structure regardless of the **unshare/uts** option.

uts/version

type	string
default	<random>

Set version level of the operating system as returned in **version** field of the **utsname** structure at *uname(2)* boundary. Name is limited to 64 characters. Name may be empty. Name may not have nul bytes. Default is determined randomly at startup.

ioctl/allow

type	integer or string
default	[...]

Add to or remove a request from the *ioctl(2)* request allowlist. Accepts an unsigned 64-bit integer as argument. Prefix with **0x** for hexadecimal and **0o** for octal input. Use **ioctl/allow+<request>** to add to, and **ioctl/allow-<request>** to remove from the allowlist. As of version 3.38.0, *ioctl(2)* requests may also be specified by case-insensitive name and multiple requests may be added or removed by separating them as a comma-delimited list. Specifying *ioctl(2)* requests by name is strongly recommended because request numbers may vary by architecture which is handled transparently when the request is specified as a name. As of version 3.38.6, the *ioctl(2)* name may be prepended with an optional exclamation mark, i.e. **!**, to denote Syd should not return EINVAL ("Invalid argument") *errno(3)* in case the name is not defined for any of the current supported architectures. This allows for writing rules generic across multiple incompatible architectures.

By default the list contains the *ioctl(2)* requests FIOCLEX, FIONCLEX, FIONBIO, FIONREAD, FIOASYNC, FIOQSIZE, FIFREEZE, FITHAW, FS_IOC_FIEMAP, FIGETBSZ, FICLONE, FICLONERANGE, FIDEDUPERANGE, FS_IOC_GETFSUUID, FS_IOC_GETFSSYSFSPATH, and RNDGETENTCNT.

For rules added at startup deny rules have precedence over allow rules because the denylist is checked at kernel-space, whereas the allowlist is checked at user-space. For rules added after startup, the last matching rule wins.

ioctl/deny

type	integer or string
default	[...]
static	add is dynamic, remove is partly static

Add to or remove a request from the *ioctl(2)* request denylist. Accepts an unsigned 64-bit integer as argument. Prefix with **0x** for hexadecimal and **0o** for octal input. Use **ioctl/deny+<request>** to add to, and **ioctl/deny-<request>** to remove from the allowlist. As of version 3.38.0, *ioctl(2)* requests may also be specified by case-insensitive name and multiple requests may be added or removed by separating them as a comma-delimited list. Specifying *ioctl(2)* requests by name is strongly recommended because request numbers may vary by architecture which is handled transparently when the request is specified as a name. As of version 3.38.6, the *ioctl(2)* name may be prepended with an optional exclamation mark, i.e. **!**, to denote Syd should not return EINVAL ("Invalid argument") *errno(3)* in case the name is not defined for any of the current supported architectures. This allows for writing rules generic across multiple incompatible architectures.

By default the list of denylisted *ioctl(2)* requests are FIBMAP, FS_IOC_FSGETXATTR, FS_IOC_FSSETXATTR, FS_IOC_SETFLAGS, KDSETKEYCODE, KDSIGACCEPT, RNDADDDTOENTCNT, RNDGETPOOL, RNDADDENTROPY, RNDZAPENTCNT, RNDCLEARPOOL, SECCOMP_IOCTL_NOTIF_RECV, SECCOMP_IOCTL_NOTIF_ID_VALID, SECCOMP_IOCTL_NOTIF_ADDFD, SECCOMP_IOCTL_NOTIF_SET_FLAGS, TIOCCONS, TIOCLINUX, TIOCSETD, and TIOCSTI.

For security reasons, the *ioctl(2)* denylist is applied at the parent *seccomp-bpf* filter at startup. This means the *Syd* process is included in this restriction as well. This also means, removing elements from this list after startup has no effect. However, if *Ioctl* sandboxing was enabled at startup, adding new elements to the *ioctl(2)* denylist will further restrict the *ioctl(2)* request space.

For rules added at startup, deny rules have precedence over allow rules because the denylist is checked at kernel-space, whereas the allowlist is checked at user-space. For rules added after startup, the last matching rule wins.

Further reading about denylisted *ioctl(2)* requests:

- <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-1523>
- <https://a13xp0p0v.github.io/2017/03/24/CVE-2017-2636.html>
- <http://phrack.org/issues/52/6.html#article>
- <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=83efeeeb3d04b22aaed1df99bc70a48fe9>
- <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=8d1b43f6a6df7bcea20982ad376a000d9>
- <https://seclists.org/oss-sec/2024/q1/13>
- <https://seclists.org/oss-sec/2024/q1/14>
- <https://forums.grsecurity.net/viewtopic.php?f=7&t=2522>
- <http://lkml.indiana.edu/hypermail/linux/kernel/9907.0/0132.html>
- <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-11/msg07723.html>

mem/max

type	positive integer (u64)
default	0

This setting specifies the limit on per-process memory usage. Setting this value to **0** disables testing for this type of memory usage. Note, the value is parsed using the **parse-size** crate. Refer to their documentation for information on formatting. Setting an non-zero value with this option implies *sandbox/mem:on*.

mem/vm_max

type	positive integer (u64)
default	0

This setting specifies the limit on per-process virtual memory usage. Setting this value to **0** disables testing for this type of memory usage. Note, the value is parsed using the **parse-size** crate. Refer to their documentation for information on formatting. Setting an non-zero value with this option implies *sandbox/mem:on*.

pid/max

type	positive integer (u64)
default	0

This setting specifies the limit on the number of running tasks for pid sandboxing. Setting this value to 0 is functionally equivalent to setting *sandbox/pid* to *off*. Setting a non-zero value with this option implies *sandbox/pid:on*.

As of version 3.40.0, when *unshare/pid:true* is set, PID sandboxing counts and enforces the limit in the current PID namespace; on Linux 6.14 and newer, to account for the kernel's 300 reserved PIDs the namespaced `kernel.pid_max` is set to **max(pid/max, 301)** (or 512 on s390x), while on older kernels *kernel.pid_max sysctl(8)* is left unchanged.

bind

type	string-vec
static	yes
oci	no

This command causes Syd to bind mount a directory on startup. The format is **source-dir:target-dir:mount-options,...** where *the source and target* directories may be equal. Mount options are a comma-separated list of a combination of the following options:

- **ro** to mount the filesystem read-only.
- **nodev** to not interpret character or block special devices on the filesystem.
- **noexec** to not permit direct execution of any binaries on the mounted filesystem.
- **nosuid** to not honour set-user-ID and set-group-ID bits or file capabilities when executing programs from this filesystem. In addition, SELinux domain transitions require permission **nosuid_transition**, which in turn needs also policy capability **nnp_nosuid_transition**.
- **nosymfollow** to not follow symbolic links when resolving paths. Symbolic links can still be created, and *readlink(1)*, *readlink(2)*, *realpath(1)*, and *realpath(3)* all still work properly.
- **noatime** to not update inode access times on this filesystem (e.g. for faster access on the news spool to speed up news servers). This works for all inode types (directories too), so it implies **nodiratime**.
- **nodiratime** to not update directory inode access times on this filesystem. (This option is implied when **noatime** is set.)
- **relatime** to update inode access times relative to modify or change time.

Mount options may be omitted. If the source directory is not an absolute path, it is interpreted as the filesystem type rather than the source directory. This may be used to mount special filesystems such as *cgroupfs*, *overlayfs* or *tmpfs(5)* into the mount namespace. In this case, any mount options supported by this filesystem type may be submitted in options argument not just the ones listed above. You may find some examples below:

- `bind+/:ro`
- `bind+tmpfs:/tmp:noexec,size=16M`
- `bind+cgroup2:/sys/fs/cgroup:nodev,noexec,nosuid`
- `bind+overlay:/tmp/target:lowerdir=/tmp/lower,upperdir=/tmp/upper,workdir=/tmp/work,nosuid`
- `bind+devpts:/dev/pts:newinstance,ptmxmode=0600,mode=600,nosuid,noexec`
- `bind+ramfs:/tmp:nodev,noexec,nosuid`

- `bind+sysfs:/sys:nodev,noexec,nosuid`
- `bind+mqueue:/dev/mqueue:nodev,noexec,nosuid`

This option does nothing without **unshare/mount:true**.

This command may be used to create immutable containers. For example, the command **bind+/:ro** is functionally equivalent to **deny/write+/***** except the restriction happens at kernel VFS layer rather than at user level using *seccomp(2)* notify. Alternatively this can also be achieved at the kernel level using *landlock(7)*.

As of version 3.23.14, symbolic links are not followed in any part of the source or target directory paths and path traversal using `..` is not permitted. In addition, target directory must be an absolute path, relative paths are not permitted.

As of version 3.23.14, mounting the special *proc(5)* filesystem under a custom path is not permitted. Syd handles this mount itself specially after all bind mounts are processed.

crypt

type **string-vec**

Specifies a list of *glob(3p)* patterns to encrypt for **Crypt sandboxing**.

crypt/key

type **i32**
static **yes**

Specify *keyrings(7)* IDs of the 256-bit AES-CTR encryption key and HMAC-SHA256 authentication key for **Crypt sandboxing**. The ID must be a 32-bit integer. To set encryption and authentication keys separately, use the options **crypt/key/enc**, and **crypt/key/mac**. This option is a shorthand to set both at the same time to the same key serial ID.

Setting an encryption key with this option implies **sandbox/crypt:on**.

Session keyring must be attached to the user keyring or this will fail at startup with the EKEYREVOKED ("Key has been revoked") *errno(3)*. Use the *syd-key(1)* utility to safely generate a key and save to *keyrings(7)* interface.

crypt/key/enc

type **i32**
static **yes**

Specify *keyrings(7)* ID of the 256-bit AES-CTR encryption key for **Crypt sandboxing**. The ID must be a 32-bit integer.

Setting an encryption key with this option implies **sandbox/crypt:on**.

Session keyring must be attached to the user keyring or this will fail at startup with the EKEYREVOKED ("Key has been revoked") *errno(3)*. Use the *syd-key(1)* utility to safely generate a key and save to *keyrings(7)* interface.

crypt/key/mac

type	i32
static	yes

Specify *keyrings(7)* ID of the 256-bit HMAC-SHA256 authentication key for **Crypt sandboxing**. The ID must be a 32-bit integer.

Setting an encryption key with this option implies **sandbox/crypt:on**.

Session keyring must be attached to the user keyring or this will fail at startup with the EKEYREVOKED ("Key has been revoked") *errno(3)*. Use the *syd-key(1)* utility to safely generate a key and save to *keyrings(7)* interface.

crypt/tmp

type	string
default	mem
static	yes

Specify temporary backing directory for transparent file decryption. The argument must be an absolute path or the special value **mem**. The user must ensure this directory is secure as decrypted contents will be written to temporary files under this directory. Specify the special value **mem** to use anonymous files which live in RAM with a volatile backing storage created with *memfd_create(2)*. This is the default. The user is encouraged to specify this option for efficient handling of large files for **Crypt sandboxing**.

Setting this option implies **sandbox/crypt:on**.

force

type	string-vec
------	-------------------

Add or remove an integrity force rule for Force Sandboxing. The format is *force+/*path*:*hashhex*:*action** for addition and *force-/*path** for removal. Use *force^* to clear the Integrity Force map. Available actions are "warn", "filter", "deny", "panic", "stop", "abort", "kill" and "exit" where the default is "deny". *hashhex* is either a 8-character CRC32 checksum, 16-character CRC64 checksum, 32-character MD5 checksum, a 40-character SHA1 checksum, a 64-character SHA3-256 checksum, a 96-character SHA3-384 checksum or a 128-character SHA3-512 checksum.

- *syd-sha(1)* is a helper tool to calculate checksums of files.
- *syd-path(1)* is a helper tool to write integrity force rules for binaries under PATH.

proxy/addr

type	IP address
default	127.0.0.1
static	yes
oci	no

Set internal address for Proxy sandboxing. This must be an IPv4 or an IPv6 address. Defaults to 127.0.0.1.

proxy/port

type	integer
default	9050
static	yes
oci	no

Set internal port for Proxy sandboxing. Defaults to 9050.

proxy/ext/host

type	Hostname or IP
default	127.0.0.1
static	yes
oci	no

Set external address for Proxy sandboxing. This must either be an IPv4 address or an IPv6 address or a hostname. If the argument does not parse as an IP address, Syd resolves the name using the system DNS resolver and selects a response IP randomly.

Defaults to "127.0.0.1", which may be overridden with the environment variable **SYD_PROXY_HOST** at startup.

proxy/ext/port

type	integer
default	9050
static	yes
oci	no

Set external port for Proxy sandboxing.

Defaults to 9050, which may be overridden with the environment variable **SYD_PROXY_PORT** at startup.

proxy/ext/unix

type	string
static	yes
oci	no

Set external UNIX domain socket for Proxy sandboxing.

The argument may also be set using the environment variable **SYD_PROXY_UNIX** at startup.

This option has precedence over the option "proxy/ext/host", ie. when both are given Syd will connect to the UNIX domain socket.

segvguard/expiry

type	integer (u64)
default	120

Specify SegvGuard expiry timeout in seconds. Set to 0 to disable SegvGuard.

segvguard/suspension

type	integer (u64)
default	600

Specify SegvGuard suspension timeout in seconds.

segvguard/maxcrashes

type	integer (u8)
default	5

Specify SegvGuard max crashes.

tpe/gid

type	integer (gid_t)
------	------------------------

Specify untrusted GID for Trusted Path Execution (TPE). By default, TPE is applied to users of all groups including root and this setting can be used to limit it to a certain group. To unset a previously set GID and return to the default state set "none" as the value.

tpe/negate

type	boolean
------	----------------

Negate GID logic for Trusted Path Execution (TPE). This turns "tpe/gid" from untrusted into trusted such that users belonging to this group will be exempt from TPE.

tpe/root_owned

type	boolean
------	----------------

Ensure file and parent directory are root-owned for Trusted Path Execution (TPE).

Note, this option will misbehave with "unshare/user:1" if the real root user is not mapped inside the container.

tpe/user_owned

type **boolean**

Ensure file and parent directory are user-owned or root-owned for Trusted Path Execution (TPE).

Note, this option may misbehave with "unshare/user:1" if the real root user is not mapped inside the container.

tpe/root_mount

type **boolean**

Ensure file and parent directory are on root filesystem for Trusted Path Execution (TPE).

This option may be used to pin all executions to a single safe mountpoint.

allow/walk

type **string-vec**

Specifies a list of *glob(3p)* patterns to allow for **Walk sandboxing**.

allow/stat

type **string-vec**

Specifies a list of *glob(3p)* patterns to allow for **Stat sandboxing**.

allow/read

type **string-vec**

Specifies a list of *glob(3p)* patterns to allow for **Read sandboxing**.

allow/write

type **string-vec**

Specifies a list of *glob(3p)* patterns to allow for **Write sandboxing**.

allow/exec

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to allow for **Exec sandboxing**.

allow/iocli

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to allow for **Iocli sandboxing**.

allow/create

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to allow for **Create sandboxing**.

allow/delete

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to allow for **Delete sandboxing**.

allow/rename

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to allow for **Rename sandboxing**.

allow/symlink

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to allow for **Symlink sandboxing**.

allow/truncate

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to allow for **Truncate sandboxing**.

allow/chdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to allow for **Chdir sandboxing**.

allow/readdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to allow for **Readdir sandboxing**.

allow/mkdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to allow for **Mkdir sandboxing**.

allow/rmdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to allow for **Rmdir sandboxing**.

allow/chown

type **string-vec**

Specifies a list of *glob*(3p) patterns to allow for **Chown sandboxing**.

allow/chgrp

type **string-vec**

Specifies a list of *glob*(3p) patterns to allow for **Chgrp sandboxing**.

allow/chmod

type **string-vec**

Specifies a list of *glob*(3p) patterns to allow for **Chmod sandboxing**.

allow/chattr

type	string-vec
------	------------

Specifies a list of *glob(3p)* patterns to allow for **Chattr sandboxing**.

allow/chroot

type	string-vec
------	------------

Specifies a list of *glob(3p)* patterns to allow for **Chroot sandboxing**.

allow/utime

type	string-vec
------	------------

Specifies a list of *glob(3p)* patterns to allow for **Utime sandboxing**.

allow/mkdev

type	string-vec
------	------------

Specifies a list of *glob(3p)* patterns to allow for **Mkdev sandboxing**.

allow/mkfifo

type	string-vec
------	------------

Specifies a list of *glob(3p)* patterns to allow for **Mkfifo sandboxing**.

allow/mktemp

type	string-vec
------	------------

Specifies a list of *glob(3p)* patterns to allow for **Mktemp sandboxing**.

allow/net/bind

type	string-vec
------	------------

Specifies a list of network address patterns to allow for **Bind network sandboxing**.

allow/net/accept

type	string-vec
------	-------------------

Specifies a list of network address patterns to allow for **Accept network sandboxing**.

allow/net/connect

type	string-vec
------	-------------------

Specifies a list of network address patterns to allow for **Connect network sandboxing**.

allow/net/sendfd

type	string-vec
------	-------------------

Specifies a list of network address patterns to allow for **SendFd network sandboxing**.

allow/net/link

type	string-vec
static	yes

Specifies a list of netlink families to allow for **Link network sandboxing**.

Accepts a comma-delimited list of the following items: **route, usersock, firewall, sock_diag, nflog, xfrm, selinux, iscsi, audit, fib_lookup, connector, netfilter, ip6_fw, dnrtmsg, kobject_uevent, generic, scsitransport, ecryptfs, rdma, crypto, and smc**. Use **all** to specify all families.

allow/lock/read

type	string-set
static	yes
default	<i>("/dev/null", "/proc")</i>

Specifies a set of beneath paths to grant file read access for **Lock** sandboxing. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_READ_FILE** and only applies to the content of the directory not the directory itself. As of version 3.21.0, this set includes the paths **"/dev/null"** and **"/proc"** by default as Syd is included in the Landlock sandbox and Syd requires read access to these paths to function correctly. As of version 3.46.0, path must not contain magic symbolic links or parent (**".."**) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (**".."**) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/write

type	string-set
static	yes
default	(<code>"/dev/null"</code>)

Specifies a set of beneath paths to grant file write access for **Lock** sandboxing. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_WRITE_FILE** and only applies to the content of the directory not the directory itself. As of version 3.21.0, this set includes the path `"/dev/null"` by default as Syd is included in the Landlock sandbox and Syd requires write access to this file to function correctly. As of version 3.46.0, path must not contain magic symbolic links or parent (`".."`) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (`".."`) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/exec

type	string-set
static	yes

Specifies a set of beneath paths to grant file execute access for **Lock** sandboxing. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_EXECUTE** and only applies to the content of the directory not the directory itself. As of version 3.46.0, path must not contain magic symbolic links or parent (`".."`) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (`".."`) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/ioctl

type	string-set
static	yes

Specifies a set of beneath paths to grant *ioctl*(2) access for **Lock** sandboxing. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_IOCTL_DEV** and only applies to the content of the directory not the directory itself. Landlock *ioctl*(2) support requires ABI 5 or later. Fifth Landlock ABI was introduced with Linux 6.10. On older kernels, this command is a no-op and is not going to confine *ioctl*(2) operations. As of version 3.46.0, path must not contain magic symbolic links or parent (`".."`) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (`".."`) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/create

type	string-set
static	yes

Specifies a set of beneath paths to grant file creation, rename and link access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_MAKE_REG** and only applies to the content of the directory not the directory itself. As of version 3.46.0, path must not contain magic symbolic links or parent (`".."`) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (`".."`) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/delete

type	string-set
static	yes

Specifies a set of beneath paths to grant file unlink, rename and link access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_REMOVE_FILE** and only applies to the content of the directory not the directory itself. As of version 3.46.0, path must not contain magic symbolic links or parent ("..") components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent ("..") components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/rename

type	string-set
static	yes

Specifies a set of beneath paths to grant access to link or rename a file from or to a different directory (i.e. reparent a file hierarchy) for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_REFER** and only applies to the content of the directory not the directory itself. Landlock rename support requires ABI 2 or later. Second Landlock ABI was introduced with Linux 5.19. On older kernels, this type of access is always denied with Landlock. As of version 3.46.0, path must not contain magic symbolic links or parent ("..") components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent ("..") components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/symlink

type	string-set
static	yes

Specifies a set of beneath paths to grant symbolic link creation, rename and link access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_MAKE_SYM** and only applies to the content of the directory not the directory itself. As of version 3.46.0, path must not contain magic symbolic links or parent ("..") components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent ("..") components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/truncate

type	string-set
static	yes
default	("dev/null")

Specifies a set of beneath paths to grant file truncation access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_TRUNCATE** and only applies to the content of the directory not the directory itself. Landlock file truncation support requires ABI 3 or later. Third Landlock ABI was introduced with Linux 6.2. On older kernels, this command is a no-op and is not going to confine file truncation operations. As

of version 3.21.0, this set includes the path `"/dev/null"` by default as Syd is included in the Landlock sandbox and Syd requires truncation access to this file to function correctly. As of version 3.46.0, path must not contain magic symbolic links or parent (`".."`) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (`".."`) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/readdir

type	string-set
static	yes
default	<code>("/proc")</code>

Specifies a set of beneath paths to grant directory list access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_READ_DIR** and applies to the directory and the directories beneath it. As of version 3.21.0, this set includes the directory `"/proc"` by default as Syd is included in the Landlock sandbox and Syd requires readdir access to this directory to function correctly. As of version 3.46.0, path must not contain magic symbolic links or parent (`".."`) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (`".."`) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/mkdir

type	string-set
static	yes

Specifies a set of beneath paths to grant directory creation and rename access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_MAKE_DIR** and only applies to the content of the directory not the directory itself. As of version 3.46.0, path must not contain magic symbolic links or parent (`".."`) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (`".."`) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/rmdir

type	string-set
static	yes

Specifies a set of beneath paths to grant directory deletion and rename access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_REMOVE_DIR** and only applies to the content of the directory not the directory itself. As of version 3.46.0, path must not contain magic symbolic links or parent (`".."`) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (`".."`) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/mkdev

type	string-set
static	yes

Specifies a set of beneath paths to grant block device creation access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_MAKE_BLOCK** and only applies to the content of the directory not the directory itself. As of version 3.46.0, path must not contain magic symbolic links or parent (“..”) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (“..”) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/mkcdev

type	string-set
static	yes

Specifies a set of beneath paths to grant character device creation access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_MAKE_CHAR** and only applies to the content of the directory not the directory itself. As of version 3.46.0, path must not contain magic symbolic links or parent (“..”) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (“..”) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed. Noop without **sandbox/lock:on**.

allow/lock/mkfifo

type	string-set
static	yes

Specifies a set of beneath paths to grant named pipe (FIFO) creation access for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_FS_MAKE_FIFO** and only applies to the content of the directory not the directory itself. As of version 3.46.0, path must not contain magic symbolic links or parent (“..”) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (“..”) components in them. Path may be relative in which case it is resolved relative to the directory where Syd was executed.

allow/lock/bind

type	(u16-set, string-set)
static	yes

Specifies a list of allowed *bind(2)* ports and UNIX domain socket paths for **Lock sandboxing**. This category corresponds to the Landlock access rights **LANDLOCK_ACCESS_NET_BIND_TCP** and **LANDLOCK_ACCESS_FS_MAKE_SOCKET** and only applies to the content of the directory not the directory itself. Argument is either a single port or a closed range in format **port1-port2**, or an absolute UNIX domain socket path. Landlock network support requires ABI 4 or later. Fourth Landlock ABI was introduced with Linux 6.7. On older kernels, this command is a no-op when specified with port arguments and does not do any network confinement. As of version 3.46.0, path must not contain magic symbolic links or parent (“..”) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (“..”) components in them. Noop without **sandbox/lock:on**.

allow/lock/connect

type	u16-set
static	yes

Specifies a list of allowed *connect(2)* ports for **Lock sandboxing**. This category corresponds to the Landlock access right **LANDLOCK_ACCESS_NET_BIND_CONNECT**. Argument is either a single port or a closed range in format **port1-port2**. Landlock network support requires ABI 4 or later. Fourth Landlock ABI was introduced with Linux 6.7. On older kernels, this command is a no-op and does not do any network confinement. Noop without **sandbox/lock:on**.

warn/walk

type	string-vec
------	-------------------

Specifies a list of *glob(3p)* patterns to warn for **Walk sandboxing**.

warn/stat

type	string-vec
------	-------------------

Specifies a list of *glob(3p)* patterns to warn for **Stat sandboxing**.

warn/read

type	string-vec
------	-------------------

Specifies a list of *glob(3p)* patterns to warn for **Read sandboxing**.

warn/write

type	string-vec
------	-------------------

Specifies a list of *glob(3p)* patterns to warn for **Write sandboxing**.

warn/exec

type	string-vec
------	-------------------

Specifies a list of *glob(3p)* patterns to warn for **Exec sandboxing**.

warn/ioctl

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Ioctl sandboxing**.

warn/create

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Create sandboxing**.

warn/delete

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Delete sandboxing**.

warn/rename

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Rename sandboxing**.

warn/symlink

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Symlink sandboxing**.

warn/truncate

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Truncate sandboxing**.

warn/chdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Chdir sandboxing**.

warn/readdir

type **string-vec**

Specifies a list of *glob(3p)* patterns to warn for **Readdir sandboxing**.

warn/mkdir

type **string-vec**

Specifies a list of *glob(3p)* patterns to warn for **Mkdir sandboxing**.

warn/rmdir

type **string-vec**

Specifies a list of *glob(3p)* patterns to warn for **Rmdir sandboxing**.

warn/chown

type **string-vec**

Specifies a list of *glob(3p)* patterns to warn for **Chown sandboxing**.

warn/chgrp

type **string-vec**

Specifies a list of *glob(3p)* patterns to warn for **Chgrp sandboxing**.

warn/chmod

type **string-vec**

Specifies a list of *glob(3p)* patterns to warn for **Chmod sandboxing**.

warn/chattr

type **string-vec**

Specifies a list of *glob(3p)* patterns to warn for **Chattr sandboxing**.

warn/chroot

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Chroot sandboxing**.

warn/utime

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Utime sandboxing**.

warn/mkdev

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Mkdev sandboxing**.

warn/mkfifo

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Mkfifo sandboxing**.

warn/mktemp

type **string-vec**

Specifies a list of *glob*(3p) patterns to warn for **Mktemp sandboxing**.

warn/net/bind

type **string-vec**

Specifies a list of network address patterns to warn for **Bind network sandboxing**.

warn/net/accept

type **string-vec**

Specifies a list of network address patterns to warn for **Accept network sandboxing**.

warn/net/connect

type **string-vec**

Specifies a list of network address patterns to warn for **Connect network sandboxing**.

warn/net/sendfd

type **string-vec**

Specifies a list of network address patterns to warn for **SendFd network sandboxing**.

deny/walk

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Walk sandboxing**.

deny/stat

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Stat sandboxing**.

deny/read

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Read sandboxing**.

deny/write

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Write sandboxing**.

deny/exec

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Exec sandboxing**.

deny/ioctl

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Ioctl sandboxing**.

deny/create

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Create sandboxing**.

deny/delete

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Delete sandboxing**.

deny/rename

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Rename sandboxing**.

deny/symlink

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Symlink sandboxing**.

deny/truncate

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Truncate sandboxing**.

deny/chdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Chdir sandboxing**.

deny/readdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Readdir sandboxing**.

deny/mkdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Mkdir sandboxing**.

deny/rmdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Rmdir sandboxing**.

deny/chown

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Chown sandboxing**.

deny/chgrp

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Chgrp sandboxing**.

deny/chmod

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Chmod sandboxing**.

deny/chattr

type **string-vec**

Specifies a list of *glob*(3p) patterns to deny for **Chattr sandboxing**.

deny/chroot

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Chroot sandboxing**.

deny/utime

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Utime sandboxing**.

deny/mkdev

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Mkdev sandboxing**.

deny/mkfifo

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Mkfifo sandboxing**.

deny/mktemp

type **string-vec**

Specifies a list of *glob(3p)* patterns to deny for **Mktemp sandboxing**.

deny/net/bind

type **string-vec**

Specifies a list of network address patterns to deny for **Bind network sandboxing**.

deny/net/accept

type **string-vec**

Specifies a list of network address patterns to deny for **Accept network sandboxing**.

deny/net/connect

type **string-vec**

Specifies a list of network address patterns to deny for **Connect network sandboxing**.

deny/net/sendfd

type **string-vec**

Specifies a list of network address patterns to deny for **SendFd network sandboxing**.

panic/walk

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Walk sandboxing**.

panic/stat

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Stat sandboxing**.

panic/read

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Read sandboxing**.

panic/write

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Write sandboxing**.

panic/exec

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Exec sandboxing**.

panic/iocctl

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Iocctl sandboxing**.

panic/create

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Create sandboxing**.

panic/delete

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Delete sandboxing**.

panic/rename

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Rename sandboxing**.

panic/symlink

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Symlink sandboxing**.

panic/truncate

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Truncate sandboxing**.

panic/chdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Chdir sandboxing**.

panic/readdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Readdir sandboxing**.

panic/mkdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Mkdir sandboxing**.

panic/rmdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Rmdir sandboxing**.

panic/chown

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Chown sandboxing**.

panic/chgrp

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Chgrp sandboxing**.

panic/chmod

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Chmod sandboxing**.

panic/chattr

type **string-vec**

Specifies a list of *glob*(3p) patterns to panic for **Chattr sandboxing**.

panic/chroot

type **string-vec**

Specifies a list of *glob(3p)* patterns to panic for **Chroot sandboxing**.

panic/utime

type **string-vec**

Specifies a list of *glob(3p)* patterns to panic for **Utime sandboxing**.

panic/mkdev

type **string-vec**

Specifies a list of *glob(3p)* patterns to panic for **Mkdev sandboxing**.

panic/mkfifo

type **string-vec**

Specifies a list of *glob(3p)* patterns to panic for **Mkfifo sandboxing**.

panic/mktemp

type **string-vec**

Specifies a list of *glob(3p)* patterns to panic for **Mktemp sandboxing**.

panic/net/bind

type **string-vec**

Specifies a list of network address patterns to panic for **Bind network sandboxing**.

panic/net/accept

type **string-vec**

Specifies a list of network address patterns to panic for **Accept network sandboxing**.

panic/net/connect

type **string-vec**

Specifies a list of network address patterns to panic for **Connect network sandboxing**.

panic/net/sendfd

type **string-vec**

Specifies a list of network address patterns to panic for **SendFd network sandboxing**.

stop/walk

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Walk sandboxing**.

stop/stat

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Stat sandboxing**.

stop/read

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Read sandboxing**.

stop/write

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Write sandboxing**.

stop/exec

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Exec sandboxing**.

stop/iocctl

type **string-vec**

Specifies a list of *glob*(3p) patterns to stop for **Iocctl sandboxing**.

stop/create

type **string-vec**

Specifies a list of *glob*(3p) patterns to stop for **Create sandboxing**.

stop/delete

type **string-vec**

Specifies a list of *glob*(3p) patterns to stop for **Delete sandboxing**.

stop/rename

type **string-vec**

Specifies a list of *glob*(3p) patterns to stop for **Rename sandboxing**.

stop/symlink

type **string-vec**

Specifies a list of *glob*(3p) patterns to stop for **Symlink sandboxing**.

stop/truncate

type **string-vec**

Specifies a list of *glob*(3p) patterns to stop for **Truncate sandboxing**.

stop/chdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to stop for **Chdir sandboxing**.

stop/readdir

type	string-vec
------	------------

Specifies a list of *glob*(3p) patterns to stop for **Readdir sandboxing**.

stop/mkdir

type	string-vec
------	------------

Specifies a list of *glob*(3p) patterns to stop for **Mkdir sandboxing**.

stop/rmdir

type	string-vec
------	------------

Specifies a list of *glob*(3p) patterns to stop for **Rmdir sandboxing**.

stop/chown

type	string-vec
------	------------

Specifies a list of *glob*(3p) patterns to stop for **Chown sandboxing**.

stop/chgrp

type	string-vec
------	------------

Specifies a list of *glob*(3p) patterns to stop for **Chgrp sandboxing**.

stop/chmod

type	string-vec
------	------------

Specifies a list of *glob*(3p) patterns to stop for **Chmod sandboxing**.

stop/chattr

type	string-vec
------	------------

Specifies a list of *glob*(3p) patterns to stop for **Chattr sandboxing**.

stop/chroot

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Chroot sandboxing**.

stop/utime

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Utime sandboxing**.

stop/mkdev

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Mkdev sandboxing**.

stop/mkfifo

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Mkfifo sandboxing**.

stop/mktemp

type **string-vec**

Specifies a list of *glob(3p)* patterns to stop for **Mktemp sandboxing**.

stop/net/bind

type **string-vec**

Specifies a list of network address patterns to stop for **Bind network sandboxing**.

stop/net/accept

type **string-vec**

Specifies a list of network address patterns to stop for **Accept network sandboxing**.

stop/net/connect

type **string-vec**

Specifies a list of network address patterns to stop for **Connect network sandboxing**.

stop/net/sendfd

type **string-vec**

Specifies a list of network address patterns to stop for **SendFd network sandboxing**.

abort/walk

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Walk sandboxing**.

abort/stat

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Stat sandboxing**.

abort/read

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Read sandboxing**.

abort/write

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Write sandboxing**.

abort/exec

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Exec sandboxing**.

abort/iocctl

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Iocctl sandboxing**.

abort/create

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Create sandboxing**.

abort/delete

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Delete sandboxing**.

abort/rename

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Rename sandboxing**.

abort/symlink

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Symlink sandboxing**.

abort/truncate

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Truncate sandboxing**.

abort/chdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to abort for **Chdir sandboxing**.

abort/readdir

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Readdir sandboxing**.

abort/mkdir

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Mkdir sandboxing**.

abort/rmdir

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Rmdir sandboxing**.

abort/chown

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Chown sandboxing**.

abort/chgrp

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Chgrp sandboxing**.

abort/chmod

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Chmod sandboxing**.

abort/chattr

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Chattr sandboxing**.

abort/chroot

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Chroot sandboxing**.

abort/utime

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Utime sandboxing**.

abort/mkdev

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Mkdev sandboxing**.

abort/mkfifo

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Mkfifo sandboxing**.

abort/mktemp

type **string-vec**

Specifies a list of *glob(3p)* patterns to abort for **Mktemp sandboxing**.

abort/net/bind

type **string-vec**

Specifies a list of network address patterns to abort for **Bind network sandboxing**.

abort/net/accept

type **string-vec**

Specifies a list of network address patterns to abort for **Accept network sandboxing**.

abort/net/connect

type **string-vec**

Specifies a list of network address patterns to abort for **Connect network sandboxing**.

abort/net/sendfd

type **string-vec**

Specifies a list of network address patterns to abort for **SendFd network sandboxing**.

kill/walk

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Walk sandboxing**.

kill/stat

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Stat sandboxing**.

kill/read

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Read sandboxing**.

kill/write

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Write sandboxing**.

kill/exec

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Exec sandboxing**.

kill/ioc tl

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Ioc tl sandboxing**.

kill/create

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Create sandboxing**.

kill/delete

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Delete sandboxing**.

kill/rename

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Rename sandboxing**.

kill/symlink

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Symlink sandboxing**.

kill/truncate

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Truncate sandboxing**.

kill/chdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Chdir sandboxing**.

kill/readdir

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Readdir sandboxing**.

kill/mkdir

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Mkdir sandboxing**.

kill/rmdir

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Rmdir sandboxing**.

kill/chown

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Chown sandboxing**.

kill/chgrp

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Chgrp sandboxing**.

kill/chmod

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Chmod sandboxing**.

kill/chattr

type **string-vec**

Specifies a list of *glob(3p)* patterns to kill for **Chattr sandboxing**.

kill/chroot

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Chroot sandboxing**.

kill/utime

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Utime sandboxing**.

kill/mkdev

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Mkdev sandboxing**.

kill/mkfifo

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Mkfifo sandboxing**.

kill/mktemp

type **string-vec**

Specifies a list of *glob*(3p) patterns to kill for **Mktemp sandboxing**.

kill/net/bind

type **string-vec**

Specifies a list of network address patterns to kill for **Bind network sandboxing**.

kill/net/accept

type **string-vec**

Specifies a list of network address patterns to kill for **Accept network sandboxing**.

kill/net/connect

type **string-vec**

Specifies a list of network address patterns to kill for **Connect network sandboxing**.

kill/net/sendfd

type **string-vec**

Specifies a list of network address patterns to kill for **SendFd network sandboxing**.

exit/walk

type **string-vec**

Specifies a list of *glob(3p)* patterns to exit for **Walk sandboxing**.

exit/stat

type **string-vec**

Specifies a list of *glob(3p)* patterns to exit for **Stat sandboxing**.

exit/read

type **string-vec**

Specifies a list of *glob(3p)* patterns to exit for **Read sandboxing**.

exit/write

type **string-vec**

Specifies a list of *glob(3p)* patterns to exit for **Write sandboxing**.

exit/exec

type **string-vec**

Specifies a list of *glob(3p)* patterns to exit for **Exec sandboxing**.

exit/ioctl

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Ioctl sandboxing**.

exit/create

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Create sandboxing**.

exit/delete

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Delete sandboxing**.

exit/rename

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Rename sandboxing**.

exit/symlink

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Symlink sandboxing**.

exit/truncate

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Truncate sandboxing**.

exit/chdir

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Chdir sandboxing**.

exit/readdir

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to exit for **Readdir sandboxing**.

exit/mkdir

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to exit for **Mkdir sandboxing**.

exit/rmdir

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to exit for **Rmdir sandboxing**.

exit/chown

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to exit for **Chown sandboxing**.

exit/chgrp

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to exit for **Chgrp sandboxing**.

exit/chmod

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to exit for **Chmod sandboxing**.

exit/chattr

type	string-vec
------	-------------------

Specifies a list of *glob*(3p) patterns to exit for **Chattr sandboxing**.

exit/chroot

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Chroot sandboxing**.

exit/utime

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Utime sandboxing**.

exit/mkdev

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Mkdev sandboxing**.

exit/mkfifo

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Mkfifo sandboxing**.

exit/mktemp

type **string-vec**

Specifies a list of *glob*(3p) patterns to exit for **Mktemp sandboxing**.

exit/net/bind

type **string-vec**

Specifies a list of network address patterns to exit for **Bind network sandboxing**.

exit/net/accept

type **string-vec**

Specifies a list of network address patterns to exit for **Accept network sandboxing**.

exit/net/connect

type **string-vec**

Specifies a list of network address patterns to exit for **Connect network sandboxing**.

exit/net/sendfd

type **string-vec**

Specifies a list of network address patterns to exit for **SendFd network sandboxing**.

append

type **string-vec**

Specifies a list of *glob(3p)* patterns to files that should be made append-only for **Write sandboxing**.

If a path is append-only, Syd adds **O_APPEND** and removes **O_TRUNC** from flags on any sandbox granted attempt to *open(2)* this path. Unsetting the **O_APPEND** flag using *fcntl(2)* **F_SETFL** command is prevented. Similarly, any attempt to *rename(2)*, *truncate(2)* and *unlink(2)* the file is prevented. This is typically useful for history and log files.

mask

type **string-map**

Specifies a list of *glob(3p)* patterns to mask for **Read & Write sandboxing**.

If a path is masked, Syd returns a file descriptor to **/dev/null** on any sandbox granted attempt to *open(2)* this path. Masking can effectively be used to hide the contents of a file in a more relaxed and compatible way than denying read/write access to it. *stat(2)* calls on a masked file returns the original file metadata and a masked file may be executed. After a successful mask operation, the mask path is *not* checked for sandbox access.

As of version 3.35.1, the default mask path **/dev/null** may be changed by specifying a colon-separated extra path to the mask-add command, e.g. **mask+/**dev**/[**fn**]null:**dev**/zero** when both of the paths **/dev/full** and **/dev/null** will be masked with the path **/dev/zero**. The mask path must be a fully canonicalized path without symbolic links.

As of version 3.36.0, the default mask path may be overridden for directories by specifying an additional colon-separated extra path to the mask-add command, e.g. **mask+/**proc**/**acpi**/***:**dev**/null:**var**/empty** when the path **/proc/acpi/wakeup** which is a regular file will return **/dev/null** at *open(2)* boundary but the directory **/proc/acpi** and any subdirectory within will return **/var/empty** at *open(2)* boundary. The mask path must be a fully canonicalized path without symbolic links.

This feature provides a non-privileged alternative to the *bind* command because it does not require the creation of a mount namespace. Moreover, *mask* commands may be specified dynamically after startup using the *syd(2)* API allowing for fine-tuned and/or incremental confinement.

block

type **ip-range**

Specifies a range of IP networks to be blocked when specified as the target address of **connect** group system calls which are *connect(2)*, *sendto(2)*, *sendmsg(2)*, *sendmmsg(2)* and when received as the source address in return from *accept(2)* and *accept4(2)* system calls for IPv4 and IPv6 family sockets. Use **block+<net>** and **block-<net>** to add and remove ip networks from the range. Alternatively the range can also be populated by including **ipset** and **netset** files from within Syd configuration. Use **block^** to clear the list and **block!** to simplify the ip range by aggregating networks together. **block!** is useful to call after importing big IP blocklists, it helps reduce memory consumption and improve matching performance. Below is a configuration snippet that imports Feodo and DShield blocklists:

```
# Enable IP blocklists
# Source: https://github.com/firehol/blocklist-ipsets.git
include /usr/src/blocklist-ipsets/feodo.ipset
include /usr/src/blocklist-ipsets/feodo_badips.ipset
include /usr/src/blocklist-ipsets/dshield.netset
include /usr/src/blocklist-ipsets/dshield_1d.netset
include /usr/src/blocklist-ipsets/dshield_30d.netset
include /usr/src/blocklist-ipsets/dshield_7d.netset
include /usr/src/blocklist-ipsets/dshield_top_1000.ipset
block!
```

cmd/exec

type **command**

Makes Syd execute an external command without sandboxing. The process is executed in a new process group with its standard input attached to **/dev/null**. Standard output and standard error file descriptors are inherited. Syd also ensures no non-standard file descriptors leak into the new process utilizing the *close_range(2)* system call. Current working directory is changed to the root directory, i.e. */*. The *umask(2)* is set to 077. The program name and arguments must be separated with the **US** (unit separator, hex: 0x1f, octal: 037) character. To ease usage, the *syd-exec(1)* helper utility is provided to construct a sandbox command of this type:

```
; syd -puser -mlock:exec -- sh -c 'test -c $(syd-exec echo hello world)'
hello world
;
```

load

type **integer (fd) or string (profile-name)**

Read configuration from the given file descriptor, the file must be open for reading. Syd uses *pidfd_getfd(2)* to acquire the file descriptor and reads sandbox configuration from it. This command is useful to load a set of sandbox commands into Syd in a single step and is typically used with **reset**, e.g:

```
int fd = open("/tmp", O_RDWR | O_TMPFILE | O_CLOEXEC, 0);
if (fd == -1) errx(1, "Failed to open temporary file");

const char *syd = "sandbox:stat/\n\nallow/stat+/**\ndenied/stat+/\n\nlock:on\n";
errx(write(fd, syd, strlen(syd)) == -1, "Failed to write config");
errx(lseek(fd, 0, SEEK_SET) == -1, "Failed to seek in file");

char load[64];
sprintf(load, "/dev/syd/load/%d", fd);
errx(stat("/dev/syd/reset", NULL) == -1, "Failed to reset syd");
errx(stat(load, NULL) == -1, "Failed to load syd profile");

errx(execvp("/bin/sh", (char *[]){"/bin/sh", "-l", NULL}) == -1, "execvp failed");
```

Due to security reasons, this command is only available via the virtual *stat(2)* call, it may not be used with the **-m** command line switch or in a configuration file.

As of version 3.30.0, this command may be used to load builtin profiles, when Syd falls back to parsing the "load" argument as a profile name if parsing the argument as a file descriptor fails.

trace/allow_safe_setuid

type	boolean
static	yes

Enable **SafeSetID** and retain the Linux capability **CAP_SETUID**. This option is implied at startup if any UID transits were defined with the **setuid** command. This feature allows Syd to change UID simultaneously with the sandbox process. Because NPTL uses reserved signals to ensure all threads share the same UID/GID, setting this option disables the SROP mitigator. See the **Enhanced Execution Control (EEC)** section of the *syd(7)* manual page for more information.

trace/allow_safe_setgid

type	boolean
static	yes

Enable **SafeSetID** and retain the Linux capability **CAP_SETGID**. This option is implied at startup if any GID transits were defined with the **setuid** command. This feature allows Syd to change GID simultaneously with the sandbox process. Because NPTL uses reserved signals to ensure all threads share the same UID/GID, setting this option disables the SROP mitigator. See the **Enhanced Execution Control (EEC)** section of the *syd(7)* manual page for more information.

setuid

type	[(uid, uid)]
static	yes

Add, remove a UID transition or reset UID transitions. Only a single transition from a source UID can be defined. Target UID can not be lower than the build default **11**, which is typically the **operator** user. Defining a UID transit with this option implies **trace/allow_safe_setuid:true**.

Usage:

```
setuid+0:65534      # Define a UID transition from root to nobody.
setuid+root:nobody  # Same as above but using user names.
setuid-0:65534      # Remove a previously defined UID transition.
setuid~0            # Remove all UID transitions matching source UID.
setuid~             # Remove all UID transitions.
```

setgid

type	[(gid, gid)]
static	yes

Add, remove a GID transition or reset GID transitions. Only a single transition from a source GID can be defined. Target GID can not be lower than the build default **14**, which is typically the **uucp** user. Defining a GID transit with this option implies **trace/allow_safe_setgid:true**.

Usage:

```
setgid+0:65534      # Define a GID transition from root to nogroup.
setgid+root:nogroup # Same as above but using group names.
setgid-0:65534      # Remove a previously defined GID transition.
setgid~0            # Remove all GID transitions matching source GID.
setgid~             # Remove all GID transitions.
```

trace/allow_unsafe_cbpf

type	boolean
static	yes

A boolean specifying whether Syd should allow additional *seccomp(2)* cbpf filters to be installed by sandbox processes. By default, this is denied to mitigate confused deputy problems and *errno(3)* is set to **EINVAL**, i.e. **Invalid argument**, for compatibility reasons. On the one hand, stacked *seccomp(2)* cbpf filters allow for incremental confinement and therefore added hardening, on the other hand they may be abused to install system call filters with more precedent actions than user-notify thereby bypassing Syd's own *seccomp(2)* cbpf filters. To quote the *seccomp_unotify(2)*: "... a user-space notifier can be bypassed if the existing filters allow the use of *seccomp(2)* or *prctl(2)* to install a filter that returns an action value with a higher precedence than **SECCOMP_RET_USER_NOTIF** (see *seccomp(2)*)." Setting the option **trace/allow_unsafe_prctl:true** overrides this option and allows the **PR_SET_SECCOMP** *prctl(2)* operation inside the sandbox. This may be changed in the future for clearer separation of mitigations.

trace/allow_unsafe_ebpf

type	boolean
static	yes

Allows direct eBPF use inside the Syd sandbox using the *bpf(2)* system call, whose unprivileged use is permitted since Linux-4.4. On the one hand, eBPF programs can be used for additional hardening, on the other hand eBPF is a frequent source of vulnerabilities due to churn, complexity, improper validation and complexity of validation. eBPF may also be abused to implement efficient and portable rootkits.

Note, as of version 3.37.0, Syd drops the capability **CAP_BPF** and denies the privileged *bpf(2)* commands **BPF_MAP_CREATE** and **BPF_PROG_LOAD** with the *errno(3)* **EPERM**, i.e. **Operation not permitted**, regardless of the value of this option. This is in consistence with the Linux kernel checks for the **kernel.unprivileged_bpf_disabled** *sysctl(8)*. Consult the *bpf(2)* and *capabilities(7)* manual pages for more information about the **CAP_BPF** Linux capability which is implemented in Linux-5.8 or newer.

trace/allow_unsafe_dumpable

type	boolean
static	yes

A boolean specifying whether Syd should skip from setting its process dumpable attribute to false. This allows core dumps for the Syd process, and allows debugging/profiling/tracing the Syd process. You should not set this option unless you're developing Syd.

trace/allow_unsafe_exec_ldso

type	boolean
------	----------------

A boolean specifying whether *ld.so(8)* exec indirection should be allowed. This is not allowed by default to harden noexec boundaries.

trace/allow_unsafe_exec_libc

type	boolean
static	yes

A boolean specifying whether turning on secure-execution mode for libc should be skipped. Refer to the **Enforcing AT_SECURE and UID/GID Verification** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_exec_memory

type	boolean
static	yes

Specify whether the Memory-Deny-Write-Execute (MDWE) protections should be bypassed. See **Memory-Deny-Write-Execute Protections** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_exec_nopie

type	boolean
------	----------------

A boolean specifying whether execution of non-PIE binaries should be allowed. This is generally not recommended but may be necessary on some systems. Refer to the **Enforcing Position-Independent Executables (PIE)** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_exec_null

type	boolean
static	yes

A boolean specifying whether exec calls with NULL argument and environment pointers should be allowed. Refer to the **Enhanced execve and execveat Syscall Validation** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_exec_stack

type	boolean
------	----------------

A boolean specifying whether execution of binaries with executable stack should be allowed. This is generally not recommended but may be necessary on some systems. Refer to the **Enforcing Non-Executable Stack** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_exec_script

type	boolean
static	yes

Opt out of file vetting for interpreted exec. When off (default) on Linux 6.14 and newer, Syd sets `SECBIT_EXEC_RESTRICT` and `SECBIT_EXEC_RESTRICT_FILE_LOCKED` at startup so interpreters/dynamic linkers must only execute a file if *execveat(2)* with `AT_EXECVE_CHECK` flag on its file descriptor would succeed (FD-based check avoids TOCTOU). When on, Syd does not set these bits (legacy behavior). No-op on kernels < 6.14. Bits are unprivileged-settable; locks make the policy sticky across exec. Refer to the **Securebits and Kernel-Assisted Executability** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_exec_interactive

type	boolean
static	yes

Opt out of interactive snippet denial. When off (default) on Linux 6.14 and newer, Syd sets `SECBIT_EXEC_DENY_INTERACTIVE` and its lock so interpreters refuse interactive code (-e, -c, REPL, etc.) unless content arrives via an FD and passes `AT_EXECVE_CHECK` of *execveat(2)*. When on, Syd does not set these bits (legacy behavior). No-op on kernels < 6.14. Bits are unprivileged-settable; locks persist the policy across exec. Refer to the **Securebits and Kernel-Assisted Executability** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_exec_speculative

type	boolean
static	yes

A boolean specifying whether speculation controls should not be set to enable Speculative Execution mitigations using the *prctl(2)* interface at startup. When this option is enabled, the *prctl(2)* operations `PR_GET_SPECULATION_CTRL`, and `PR_SET_SPECULATION_CTRL` are allowed within the sandbox. Refer to the **Speculative Execution Mitigation** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_ptrace

type	boolean
static	yes

A boolean specifying whether *ptrace*(2) should be used to secure the exec handler. Setting this option to true effectively removes the *ptrace*(2) dependency from the sandbox. This is necessary to trace *syd* together with its children, e.g. with **strace -f**. **Warning**, this option makes *syd*(1) keep the **CAP_SYS_PTRACE** capability and disables Force Sandboxing, SegvGuard and the exec-TOCTOU mitigator. It allows the sandbox process to trivially break out of the sandbox by e.g. attaching to the *syd*(1) main thread with *ptrace*(2) and getting a handle to the *seccomp*(2) notify file descriptor. Therefore, **this option should** only be used in trusted environments.

trace/allow_unsafe_perf

type	boolean
static	yes

A boolean specifying whether perf calls should be allowed within the sandbox.

As of version 3.40.0, the *prctl*(2) operations **PR_TASK_PERF_EVENTS_ENABLE**, and **PR_TASK_PERF_EVENTS_DISABLE** are also allowed if this option is set at startup.

trace/allow_unsafe_create

type	boolean
------	----------------

A boolean specifying whether to allow unsafe file creation. Refer to the **Trusted File Creation** section of the *syd*(7) manual page for more information.

trace/allow_unsafe_filename

type	boolean
------	----------------

A boolean specifying whether the restrictions on file names should be lifted. By default, file names with control characters, forbidden characters or invalid UTF-8 are denied with **EINVAL** as necessary. Read **Enhanced Path Integrity Measures** of the *syd*(7) manual page for more information.

trace/allow_unsafe_hardlinks

type	boolean
------	----------------

A boolean specifying whether to allow unsafe hardlink targets. Refer to the **Trusted Hardlinks** section of the *syd*(7) manual page for more information.

trace/allow_unsafe_machine_id

type	boolean
------	----------------

Specify whether the sandbox substitutes *machine-id(5)* with a synthetic, per-Syd random identifier or exposes the host value. The substitution is done at *open(2)* boundary after the access checks grants access to this file. The files **/etc/hostid** and **/var/adm/hostid** which are part of the *gethostid(3)* interface of POSIX.1-2008 are also substituted as part of this mitigation. When **false** (default), at startup Syd computes a SHA3-512 digest using AT_RANDOM bytes and formats the result as a 128-character lowercase hexadecimal sandbox ID; the first 32 characters of this string (**which** must not be all zeroes) are presented in place of *machine-id(5)*, */etc/hostid*, and */var/adm/hostid* to limit information leakage. Users may override the sandbox ID by setting **SYD_ID** environment variable to a 128-character lowercase hexadecimal string that satisfies the same non-all-zero 32-character prefix constraint. When **true**, no substitution is performed and the real system *machine-id(5)*, */etc/hostid*, and */var/adm/hostid* files are made visible to the sandbox process (i.e., the mitigation is disabled). Refer to the following links for more information:

- <https://man7.org/linux/man-pages/man5/machine-id.5.html>
- <https://pubs.opengroup.org/onlinepubs/9699919799/functions/gethostid.html>

trace/allow_unsafe_proc_files

type	boolean
static	yes

Specifies whether internal *procfs(5)* should NOT be mounted with the option **subset=pid**. This option is a no-op unless **unshare/pid:true** is also set.

trace/allow_unsafe_proc_pid_status

type	boolean
------	----------------

A boolean specifying whether masking security-sensitive fields in *proc_pid_status(5)* files should be disabled. Refer to the **Hardening** *proc_pid_status(5)* section of the *syd(7)* manual page for more information.

trace/allow_unsafe_magiclinks

type	boolean
------	----------------

A boolean specifying whether */proc* magic links should be followed even when per-process directory id differs from the caller process id. Magic links are symbolic link-like objects that are most notably found in *proc(5)*; examples include **/proc/pid/exe** and **/proc/pid/fd/***. See *symlink(7)* for more details. Unknowingly opening magic links can be risky for some applications. Examples of such risks include the following:

- If the process opening a pathname is a controlling process that currently has no controlling terminal (see *credentials(7)*), then opening a magic link inside **/proc/pid/fd** that happens to refer to a terminal would cause the process to acquire a controlling terminal.
- In a containerized environment, a magic link inside **/proc** may refer to an object outside the container, and thus may provide a means to escape from the container.

Because of such risks, Syd denies access to magic links which do not belong to the current process by default.

As of version 3.36.0, *ioctl(2)* requests to magic links are denied unless this option is set.

trace/allow_unsafe_symlinks

type	boolean
------	----------------

A boolean specifying whether to allow following symlinks in *untrusted* directories. *Untrusted* directories are either group-writable, world-writable, or have the sticky-bit set. Refer to the **Trusted** Symbolic Links section of the *syd(7)* manual page for more information.

As of version 3.42.0, sending symlink file descriptors with *sendmsg(2)*, and *sendmmsg(2)* system calls using SCM_RIGHTS control messages is not permitted unless this option is set.

trace/allow_unsafe_namespace

type	string-vec
static	yes

A list of namespaces to allow creation under the sandbox. Must be a comma-separated list of **mount**, **uts**, **ipc**, **user**, **pid**, **net**, **cgroup** and **time**. The special value **all** is supported as a placeholder to specify all namespaces. An invocation of this command overrides all previous invocations, ie only the list of subnamespaces in the last invocation of this command will be allowed. By default, subnamespace creation is not allowed. As of version 3.35.2, the system calls *sethostname(2)* and *setdomainname(2)* are only allowed in the sandbox if **uts** subnamespace is allowed. This is similar to the mount family system calls which are only allowed if **mount** subnamespace is allowed.

trace/allow_unsafe_nice

type	boolean
static	yes

A boolean specifying whether process and I/O priority changes are allowed for the sandbox. See the **Process Priority and Resource** Management section of the *syd(7)* manual page for more information.

trace/allow_unsafe_nocookie

type	boolean
static	yes

A boolean specifying whether enforcement of syscall argument cookies should be disabled. See the **Syscall Argument Cookies** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_nomseal

type	boolean
static	yes

A boolean specifying whether read-only sealing critical regions of the Syd sandbox policy using *mseal(2)* when sandbox is locked should be disabled. See the **Memory Sealing of Sandbox Policy Regions on Lock** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_sigreturn

type	boolean
static	yes

A boolean specifying whether signal counting to mitigate Sigreturn Oriented Programming, aka SROP, should be disabled. Refer to the **Mitigation against Sigreturn Oriented Programming (SROP)** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_chown

type	boolean
static	yes

Makes Syd keep the capability **CAP_CHOWN** and sandbox process will inherit the capability from Syd.

trace/allow_unsafe_chroot

type	boolean
static	yes

Disable Chroot sandboxing and turn *chroot(2)* system call into a no-op. Refer to the explanation of **chroot** sandbox category in the SANDBOXING section of the *syd(7)* manual page for more information.

trace/allow_unsafe_pivot_root

type	boolean
static	yes

Turn *pivot_root(2)* system call into a no-op rather than unconditionally denying it with the *errno(3)* **EPERM**. Refer to the explanation of **chroot** sandbox category in the SANDBOXING section of the *syd(7)* manual page for more information.

trace/allow_unsafe_oob

type	boolean
static	yes

Allow the **MSG_OOB** flag for *send(2)*, *sendto(2)*, *sendmsg(2)*, and *sendmmsg(2)* system calls to send out-of-band data. Refer to the **Denying MSG_OOB Flag in send System Calls** subsection of the *syd(7)* manual page for more information.

trace/allow_unsafe_open_kfd

type	boolean
------	---------

A boolean specifying whether *open(2)* calls to AMD KFD character devices should be continued in the sandbox process rather than opening them in the Syd emulator thread and sending the file descriptor. The **/dev/kfd** character device requires per-application access to the GPU device, therefore opening the device in the Syd emulator thread and then continuing the subsequent *ioctl(2)* system calls in the sandbox process is going to return **EBADF**, i.e. **Bad file number**. Until Syd has a way to fully emulate the *ioctl(2)* request space and is able to call the *ioctl(2)* system call directly from Syd emulator threads, this option may be used to access such character devices. **Setting this option opens** a TOCTOU attack vector, whereby the sandbox process can open an arbitrary file instead of the character device in question! Syd applies the following mitigations to limit the scope of the attack vector:

- Syd **continues** the system call if and only if **O_RDWR** is set in the flags argument.
- Syd does not **continue** the system call if at least one of the flags **O_CREAT**, **O_TRUNC** or **O_TMPFILE** is set in the flags argument.
- Syd returns **ENOSYS**, i.e. **Function not implemented**, for the *openat2(2)* system call rather than **continuing** it in the sandbox process to prevent the **struct open_how** pointer indirection to bypass the restrictions applied to the flags argument. Refer to the *openat2(2)* manual page for more information.
- This option may be changed at runtime, and it is highly recommended to unset this option using the *syd(2)* virtual system call API right after the character device is opened.

trace/allow_unsafe_open_path

type	boolean
------	---------

A boolean specifying whether the mitigation to turn **O_PATH** file descriptors into **O_RDONLY** file descriptors for safe emulation should be disabled. With this option, syd continues the *open(2)* system calls with the **O_PATH** in the sandbox process which opens a TOCTOU vector.

trace/allow_unsafe_mkbdev

type	boolean
static	yes

Specify whether unsafe block device access should be allowed. When set, Syd does not drop the capability **CAP_MKNOD** on startup for itself, but it is still dropped for the sandbox process. This allows:

- block device creation with *mknod(2)*.
- *ioctl(2)* calls on block devices.
- open block devices with *open(2)*.
- list block devices with *getdents64(2)*.

trace/allow_unsafe_mkcdev

type	boolean
static	yes

Specify whether unsafe character device creation should be allowed. When set, Syd does not drop the capability **CAP_MKNOD** on startup for itself, but it is still dropped for the sandbox process. This allows creation of character devices with *mknod(2)*.

trace/allow_unsafe_stat_bdev

type	boolean
------	----------------

Specify whether *stat(2)* family calls on block devices should return last access and modification times as-is. Refer to the **Device Sidechannel Mitigations** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_stat_cdev

type	boolean
------	----------------

Specify whether *stat(2)* family calls on character devices should return last access and modification times as-is. Refer to the **Device Sidechannel Mitigations** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_notify_bdev

type	boolean
------	----------------

Specify whether unsafe event generation for *fanotify_mark(2)* and *inotify_add_watch(2)* system calls should be allowed for block devices. Refer to the **Device Sidechannel Mitigations** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_notify_cdev

type	boolean
------	----------------

Specify whether unsafe event generation for *fanotify_mark(2)* and *inotify_add_watch(2)* system calls should be allowed for character devices. Refer to the **Device Sidechannel Mitigations** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_cpu

type	boolean
static	yes

Specify whether CPU emulation system calls should be allowed. By default, as of version 3.22.1, Syd denies the *modify_ldt(2)*, *subpage_prot(2)*, *switch_endian(2)*, *vm86(2)*, and *vm86old(2)* system calls, which are associated with CPU emulation functionalities. Enabling this option (*trace/allow_unsafe_cpu:1*) permits these calls, thus relaxing the restriction. This option should be used with caution, as allowing these system calls can introduce potential vulnerabilities by enabling processes to modify CPU state or memory protections. Use this setting only in trusted environments where the execution of these system calls is necessary.

trace/allow_unsafe_deprecated

type	boolean
static	yes

Specify whether deprecated system calls such as *remap_file_pages(2)*, *stime(2)*, and *uselib(2)* should be allowed. Refer to the output of the command **syd-ls deprecated** for the full list of deprecated system calls for your installation.

trace/allow_unsafe_keyring

type	boolean
static	yes

Specify whether the *add_key(2)*, *keyctl(2)*, and *request_key(2)* system calls should be allowed. Enabling this setting permits key management within the sandbox, which can introduce security risks by allowing keyring manipulations. Use only in trusted environments.

trace/allow_unsafe_pipe

type	boolean
static	yes

Allow creating notification pipes using the "O_NOTIFICATION_PIPE" flag to the *pipe2(2)* system call. See the "Denying O_NOTIFICATION_PIPE Flag in pipe2" subsection of the *syd(7)* manual page for more information.

trace/allow_unsafe_pkey

type **boolean**
static **yes**

Specifies whether the *pkey_alloc(2)*, *pkey_free(2)*, and *pkey_mprotect(2)* system calls should be allowed. By default, these calls are denied to enhance security. Setting this option to true enables these system calls, allowing the use of memory protection keys. This option should be used with caution and only in trusted environments where the use of these system calls is necessary.

trace/allow_unsafe_madvise

type **boolean**
static **yes**

Specifies whether *madvise(2)* system call should NOT be hardened. By default, only a subset of advice are permitted. see **syd-ls madvise**. Refer to the **madvise(2) Hardening** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_mbind

type **boolean**
static **yes**

Specifies whether the *mbind(2)* system call should be allowed. By default, this call is denied to enhance security, as changing NUMA memory policy and triggering page migration over large address ranges can be abused to create prolonged kernel work and resource pressure, which can serve as a denial-of-service vector. It may also make memory placement more predictable, weakening certain mitigation techniques. Enable only if required for compatibility with applications that need explicit NUMA policy control.

trace/allow_unsafe_msgsnd

type **boolean**
static **yes**

Specifies whether the *msgsnd(2)* system call should be allowed. By default, this call is denied to enhance security as the ability of this system call to allocate large, contiguous blocks of memory in the kernel heap is often used to orchestrate kernel heap spraying attacks. See the "Mitigation Against Heap Spraying" section of the *syd(7)* manual page for more information.

trace/allow_unsafe_page_cache

type **boolean**
static **yes**

Specifies whether the system calls *cachestat(2)* and *mincore(2)* should be allowed. By default, these calls are denied to enhance security as it has been documented that they can be misused to perform page-cache attacks. See the "Mitigation against Page Cache Attacks" section of the *syd(7)* manual page for more information.

trace/allow_unsafe_time

type	boolean
static	yes

A boolean specifying whether system calls which adjust the system time are allowed. Note, this also causes Syd to keep the **CAP_SYS_TIME** capability. Use **syd-ls time** to see the list of system calls allowed by this setting.

trace/allow_unsafe_uring

type	boolean
static	yes

A boolean specifying whether system calls of the *io_uring*(7) interface are allowed. Normally, these are denied because they may be used to bypass path sandboxing. Use **syd-ls uring** to see the list of system calls allowed by this setting.

trace/allow_unsafe_xattr

type	boolean
------	----------------

A boolean specifying whether the extended attributes restrictions on *user.syd*, *security*, and *trusted* namespaces should be lifted. If this option is not set only sandbox processes with access to the sandbox lock can view or change these extended attribute namespaces.

trace/allow_unsafe_caps

type	boolean
static	yes

A boolean specifying whether Syd should skip dropping Linux capabilities at startup. This setting can be used to construct *privileged* containers and should be used with extreme care.

trace/allow_unsafe_cap_fixup

type	boolean
static	yes

Opt out of hardened UID/capability transitions. When off (default), Syd clears **SECBIT_KEEP_CAPS**, sets **SECBIT_NO_SETUID_FIXUP**, and applies their lock bits at startup so capabilities are dropped when all UIDs become nonzero and are not implicitly gained or adjusted by later setuid-style UID changes; capability sets then only change via explicit *capset*(2) and *prctl*(2) calls. When on, Syd leaves **SECBIT_KEEP_CAPS** and **SECBIT_NO_SETUID_FIXUP** (and their locks) as inherited from the parent, preserving the kernel's traditional "setuid fixup" behavior and any **PR_SET_KEEPCAPS** use by the application (legacy behavior). No-op on kernels that do not support securebits. Refer to the **Securebits and Kernel-Assisted Executability** section of the *syd*(7) manual page for more information.

trace/allow_unsafe_env

type	boolean
static	yes

Specify whether unsafe environment variables should be allowed into the environment of the sandbox process. See **syd-ls env** for the list of unsafe environment variables.

trace/allow_safe_kcapi

type	boolean
------	----------------

Specify whether access to the Linux kernel cryptography API (aka: "KCAPI") should be allowed when network sandboxing is on. This option has no effect when network sandboxing is off.

As most things in life, cryptography has good and evil uses: KCAPI is convenient as it may be used to implement cryptography without depending on user-space libraries such as OpenSSL but it may also enable malicious code to efficiently turn itself into ransomware. Adhering to the goal to be secure by default Syd disallows this access by default.

Note, Syd does not hook into *setsockopt(2)* and the "ALG_SET_KEY" operation to set the encryption key is directly handled by the host kernel therefore the encryption key is not copied into Syd's address space.

Note again, Syd hooks into *bind(2)*, *sendto(2)*, *sendmsg(2)*, and *sendmmsg(2)* but **not** *read(2)*, *write(2)*, *recv(2)*, or *splice(2)*. To reduce syscall overhead, user is recommended to use the unhooked system calls when they can to interact with KCAPI.

trace/allow_safe_syslog

type	boolean
static	yes

Specify whether unprivileged sandbox processes can access Syd's *syslog(2)* emulation using *dmesg(8)*. Unprivileged processes include the set of *all* sandbox processes with the sandbox lock "off", and *all* but the initial sandbox process with the sandbox lock set to "exec". Note, this option has nothing to do with access to the host syslog which is never allowed.

trace/allow_safe_bind

type	boolean
static	yes

Specify whether the socket address arguments of successful *bind(2)* calls should be allowed for *connect(2)*, *sendto(2)*, *sendmsg(2)*, and *sendmmsg(2)* system calls.

Note, these addresses are allowed globally and not per-process for usability reasons. Thus, for example, a process which forks to call *bind(2)* will have its address allowed for their parent as well.

trace/allow_unsafe_bind

type	boolean
static	yes

Specify whether the Linux capability **CAP_NET_BIND_SERVICE**, which allows a process to *bind(2)* to ports lower than 1024, should be retained. When this option is set, Syd keeps the capability on startup for itself, but it is still dropped for the sandbox process.

trace/allow_unsafe_socket

type	boolean
static	yes

Specify whether unsafe socket families should be allowed. When set, Syd does not drop the capability **CAP_NET_RAW** on startup for itself, but it is still dropped for the sandbox process. This allows:

- use of RAW and PACKET sockets.
- bind to any address for transparent proxying.
- make use of the *ping(1)* command.

trace/allow_unsupp_socket

type	boolean
------	----------------

Specify whether unsupported socket families such as netlink sockets should be allowed access when network sandboxing is on. By default Syd allows sandboxed access to unix, ipv4 and ipv6 sockets. This option has no effect when network sandboxing is off.

As of version 3.16.6 Syd allows access to algorithm sockets with the **trace/allow_safe_kcapi** option rather than with this option. Algorithm sockets are used to interact with the Linux kernel cryptography API.

As of version 3.42.0, Transparent Inter-Process Communication (AF_TIPC) sockets at *socketpair(2)* boundary are only permitted if this option is set to true.

trace/allow_unsafe_personality

type	boolean
static	yes

Specify whether *personality(2)* restrictions should be lifted. See **syd-ls** personality for the list of allowlisted *personality(2)* personas. See the **Personality Syscall Restrictions** of the *syd(7)* manual page for more information.

trace/allow_unsafe_prctl

type	boolean
static	yes

Specify whether *prctl*(2) restrictions should be lifted. See **syd-ls prctl** for the list of allowed prctl requests.

trace/allow_unsafe_prlimit

type	boolean
static	yes

Specify whether *prlimit*(2) restrictions should be lifted.

trace/allow_unsafe_mqueue

type	boolean
static	yes

Specify whether unsafe permissions in mode argument of *mq_open*(2) system call should be permitted. See the **Shared Memory Permissions Hardening** section of the *syd*(7) manual page for more information.

trace/allow_unsafe_rseq

type	boolean
static	yes

Specify whether unsafe Restartable Sequences with the *rseq*(2) system call should be permitted. See the **Denying Restartable Sequences** section of the *syd*(7) manual page for more information.

trace/allow_unsafe_shm

type	boolean
static	yes

Specify whether unsafe permissions in mode arguments of *shmget*(2), *msgget*(2), and *semget*(2) system calls and the **IPC_SET** operation of *shmctl*(2), *msgctl*(2), and *semctl*(2) system calls should be permitted. See the **Shared Memory Permissions Hardening** section of the *syd*(7) manual page for more information.

trace/allow_unsafe_sysinfo

type	boolean
static	yes

Specify whether the *sysinfo(2)* randomizer should be disabled at startup. If this option is set at startup the *sysinfo(2)* system call becomes allowed and provides identical info to the files **/proc/loadavg** and **/proc/meminfo** which are disabled by default by common profiles such as the **linux** and **user** profiles. Notably this mitigation is unset for the **paludis** profile because leaking this side-channel is irrelevant for package builds.

trace/allow_unsafe_syslog

type	boolean
static	yes

Specify whether the Linux capability **CAP_SYSLOG** should be retained. This allows the process to perform privileged *syslog(2)* operations. This is useful when sandboxing a service such as syslogd.

trace/allow_unsafe_sync

type	boolean
static	yes

Specify whether the *sync(2)* and *syncfs(2)* system calls should be allowed inside the sandbox. By default these system calls are turned into no-ops to prevent potential local DoS, however it may be useful to disable this restriction in scenarios where sync is actually expected to work such as when sandboxing databases.

trace/allow_unsafe_memfd

type	boolean
------	----------------

A boolean specifying whether secret memory file descriptors and executable memory file descriptors should be enabled. By default Syd strips the **MFD_EXEC** and adds the **MFD_NOEXEC_SEAL** flag to *memfd_create(2)* flags argument. This ensures the memory file descriptor can never be made executable. The **MFD_NOEXEC_SEAL** flag requires Linux-6.3 or newer therefore on older kernels this option must be enabled to make memory file descriptors work. However, the user should be aware that allowing encrypted memory file descriptors does allow an attacker to bypass Exec, Force and TPE sandboxing and execute denylisted code.

trace/allow_unsafe_uname

type	boolean
static	yes

A boolean specifying whether *uname(2)* hardening should be disabled. Refer to the **Hardened uname(2)** section of the *syd(7)* manual page for more information.

trace/allow_unsafe_vmsplice

type	boolean
static	yes

Specify whether the *vmsplice(2)* system call should be allowed inside the sandbox. By default this system call is not permitted. Refer to the **Restricting vmsplice System Call** section of the *syd(7)* manual page for more information.

trace/deny_dotdot

type	boolean
------	---------

Specify whether `..` components should be denied during path resolution for *chdir(2)* and *open(2)* family system calls. This is useful in mitigating path traversal attacks. See **Path Resolution Restriction For Chdir and Open Calls** of the *syd(7)* manual page for more information.

trace/deny_exec_elf32

type	boolean
------	---------

Deny the execution of 32-bit ELF binaries.

trace/deny_exec_elf_dynamic

type	boolean
------	---------

Deny the execution of dynamically linked ELF binaries.

trace/deny_exec_elf_static

type	boolean
------	---------

Deny the execution of statically linked ELF binaries.

trace/deny_exec_script

type	boolean
------	----------------

Deny the execution of scripts (files with `#!<interpreter>` on first line).

The `execve(2)` TOCTOU mitigations do not cover this option which means **the functionality is vulnerable to TOCTOU**. This allows an attacker to execute a script whose path is denylisted. This TOCTOU is limited to scripts and requires the interpreter binary to be allowlisted for exec. Hence this vulnerability does not allow an attacker to execute denylisted binaries. This is why the user is recommended to deny the respective interpreter binaries for execution instead for a safe and secure approach.

On Linux-6.14 and newer, kernel-assisted executability provides a safe way to deny execution of scripts in cooperation with enlightened interpreters. Refer to the **Securebits and Kernel-Assisted** Executability section of the `syd(7)` manual page for more information.

trace/deny_tsc

type	boolean
static	yes

Specify whether reading the timestamp counter should be denied. Without an accurate timer, many timing attacks are going to be harder to perform.

- This works on **x86 only**.
- This breaks time related calls in the **vDSO**, which can be trivially worked around by writing a **LD_PRELOAD** library to call the respective system calls directly. See **libsydtime**, <https://lib.rs/libsydtime>, for a reference implementation.
- This has a negative performance impact on programs that rely on `gettimeofday(2)` being a **vDSO** call.

trace/exit_wait_all

type	boolean
static	yes

Specify whether Syd should wait for all processes to exit before exiting. By default, Syd exits with the eldest process and any leftover processes in the background are automatically killed.

trace/force_cloexec

type	boolean
------	----------------

Specify whether the `"O_CLOEXEC"` flag should be enforced for all `creat(2)`, `open(2)`, `openat(2)`, `openat2(2)`, `memfd_create(2)`, `socket(2)`, `accept(2)`, and `accept4(2)` system calls made by the sandbox process. When this feature is enabled, Syd ensures that every file descriptor opened by the sandbox process is automatically set with the `"O_CLOEXEC"` flag, which prevents these file descriptors from being inherited by newly executed programs. This measure enhances security by closing file descriptors during `exec(3)` calls, thereby mitigating the risk of file descriptor leakage which could lead to unauthorized access to sensitive files or resources. The feature can be toggled at runtime using Syd's virtual `stat(2)` API, providing flexible control over the confinement level of sandboxed processes.

trace/force_rand_fd

type **boolean**

Specify whether file descriptors returned by all *creat(2)*, *open(2)*, *openat(2)*, *openat2(2)*, *memfd_create(2)*, *socket(2)*, *accept(2)*, and *accept4(2)* system calls made by the sandbox process should be randomized. When this feature is enabled, Syd specifies a random available slot (rather than the lowest-numbered one) to the **SECCOMP_IOCTL_NOTIF_ADDFD** operation which is used to install a file descriptor to the sandbox process. Randomizing file descriptor numbers makes it significantly harder for an attacker to predict or deliberately reuse critical descriptors, thereby raising the bar against file-descriptor reuse and collision attacks. Note that enabling this may break programs which rely on the POSIX guarantee that *open(2)* returns the lowest available descriptor. This behavior can be toggled at runtime via Syd's virtual *stat(2)* API, allowing operators to enable or disable descriptor randomization without restarting or recompiling the sandboxed process. We're also cooperating with the HardenedBSD project to implement a similar feature in the BSD kernel. Refer to the following link for more information: <https://git.hardenedsbsd.org/hardenedsbsd/HardenedBSD/-/issues/117>

This feature uses the *kcmp(2)* system call and requires a Linux kernel configured with the **CONFIG_KCMP** option. On a kernel without this option, all system calls that are part of this feature will return **ENOSYS (Function not implemented)**.

As of version 3.38.0, this option is enabled for the **user** profile.

trace/force_ro_open

type **boolean**

Specify whether creating and writing *open(2)* family system calls should be denied regardless of the path argument. This option is restricted to *creat(2)*, *open(2)*, *openat(2)*, and *openat2(2)* system calls and provided for convenience. To stop all write-like access completely, including e.g. *mkdir(2)*, *truncate(2)* etc., use the **readonly** profile instead which uses the rule "deny/wrset/**" to prevent all write-like access. See "PROFILES" section of the *syd(5)* manual page for more information.

trace/force_no_symlinks

type **boolean**

Specify whether path resolution for the *open(2)* family is forced to use the **RESOLVE_NO_SYMLINKS** resolve flag. This flag is forced during path canonicalization, therefore this mitigation applies to all hooked path system calls, not just the *open(2)* family. When enabled, traversal of symbolic links is disallowed during lookup; all pathname components must be non-symlink entries. This affects only the pathname resolution step and does not modify other flags or access checks.

trace/force_no_magiclinks

type **boolean**

Specify whether path resolution for the *open(2)* family is forced to use the **RESOLVE_NO_MAGICLINKS** resolve flag. This flag is forced during path canonicalization, therefore this mitigation applies to all hooked path system calls,

not just the *open(2)* family. When enabled, traversal of magic links (such as special *proc(5)* links that do not behave like regular symbolic links) is disallowed during lookup; all pathname components must be non-magiclink entries. This affects only the pathname resolution step and does not modify other flags or access checks.

trace/force_no_xdev

type	boolean
------	----------------

Specify whether path resolution for the *open(2)* family is forced to use the **RESOLVE_NO_XDEV** resolve flag. This flag is forced during path canonicalization, therefore this mitigation applies to all hooked path system calls, not just the *open(2)* family. When enabled, traversal of mount points, including bind mounts, is disallowed during lookup; the path must reside on the same mount as the directory referenced by *dirfd* (or the current working directory when *dirfd* == *AT_FDCWD*). This affects only the pathname resolution step and does not modify other flags or access checks.

trace/force_umask

type	octal
static	yes

Specify an umask mode to force. To unset a previously configured force umask use -1 as the value. As of version 3.15.6, *chmod(2)* family system calls also honour force umask for added hardening. As of version 3.22.1, this setting does not apply to directory creation for *mkdir(2)* and *mkdirat(2)* system calls. As of version 3.26.2, this setting does not apply to UNIX domain socket creation for *bind(2)* system calls, and non-regular file creation for *mknod(2)* and *mknodat(2)* system calls.

trace/memory_access

type	integer
default	2
static	yes

Set mode on cross memory attach and *proc_pid_mem(5)* usage. Cross memory attach is done using the system calls *process_vm_readv(2)* and *process_vm_writev(2)* which requires a Linux kernel configured with the **CONFIG_CROSS_MEMORY_ATTACH** option enabled. Supported modes are:

- **0**: Use cross memory attach if available, use *proc_pid_mem(5)* otherwise.
- **1**: Use */proc/pid/mem(5)* unconditionally.
- **2**: Use cross memory attach unconditionally.

From a security point of view, these two modes of access have an important distinction where cross memory attach honours page protections of the target process, however using */proc/pid/mem(5)* does not. This makes direct *proc_pid_mem(5)* access dangerous in that a Syd deputy process may be confused into corrupting or even controlling memory regions the sandbox process otherwise does not have direct access to. This is the main reason why mode **2** has been added as of version 3.32.6 as a secure default alternative to the previous default mode **0** whose fallback behaviour can be unpredictable and is against the idea of secure defaults. Therefore as of version 3.32.6, the user is asked to change the memory access mode explicitly if their Linux kernel is not configured with the **CONFIG_CROSS_MEMORY_ATTACH** option. You may also use the environment variables **SYD_NO_CROSS_MEMORY_ATTACH** and **SYD_PROC_PID_MEM_FALLBACK**, see the "ENVIRONMENT" section of the *syd(1)* manual page for more information. For further information about the security impact of *proc_pid_mem(5)* writes refer to the following links:

- <https://lore.kernel.org/lkml/202403011451.C236A38@keescook/T/>
- <https://lwn.net/Articles/476947/>
- <https://issues.chromium.org/issues/40089045>

```
; strace -q -eprocess_vm_readv -fc -- syd -poff -pD -mtrace/memory_access:0 true
% time      seconds  usecs/call      calls      errors syscall
-----
100.00      0.000031      10          3          process_vm_readv
-----
100.00      0.000031      10          3          total
; strace -q -eprocess_vm_readv -fc -- syd -poff -pD -mtrace/memory_access:1 true
; strace -q -eprocess_vm_readv -fc -- syd -poff -pD -mtrace/memory_access:2 true
% time      seconds  usecs/call      calls      errors syscall
-----
100.00      0.000008         2          3          process_vm_readv
-----
100.00      0.000008         2          3          total
```

trace/sync_seccomp

type	boolean
default	true
static	yes

Use synchronous mode for seccomp-notify so each Syd syscall handler thread wakes up on the same CPU as the respective sandbox thread that executed the system call. This option makes no functional difference and typically helps with performance. Use *perf(1)* to benchmark seccomp synchronous mode on your system:

```
; perf bench sched seccomp-notify
# Running 'sched/seccomp-notify' benchmark:
# Executed 1000000 system calls
Total time: 6.736 [sec]
6.736395 usecs/op
148447 ops/sec
; perf bench sched seccomp-notify --sync-mode
# Running 'sched/seccomp-notify' benchmark:
# Executed 1000000 system calls
Total time: 4.188 [sec]
4.188846 usecs/op
238729 ops/sec
```

PATTERN MATCHING

Syd uses shell-style pattern matching for allowlists and filters. The matching code is based on *rsync(1)*. See the **PATTERN MATCHING RULES** section of the *rsync(1)* manual for more information. Notably, Syd applies the **triple star** extension to patterns, i.e. **/dev/***** matches both **/dev** and any file recursively under **/dev**. Note also, Syd gets patterns from multiple sources: a configuration file, a profile, the **-m** command line switch, or a *stat(1)* call with **/dev/syd** prefix. There is no precedence between different sources. All patterns in a list are compiled together in an array and pattern matching during access control happens in a single step where **the last** matching pattern decides the outcome.

ADDRESS MATCHING

Syd has a simple address scheme to match network addresses. The addresses can either be a **glob** pattern to match **UNIX** and **abstract UNIX** socket addresses, or **IP CIDR** followed by a port range to match **IPv4** and **IPv6** addresses. Port range can either be a single port or a closed range in format **port1-port2**. The address and the port range must be split by the character **!**. The precedence logic is same as **Pattern Matching** where **the last matching pattern decides the outcome**.

In addition there are some **aliases**, you may use instead of specifying an address:

- **any**: Expanded to **any4** + **any6**.
- **any4**: Expanded to **0.0.0.0/0** which matches the whole Ipv4 address space.
- **any6**: Expanded to **::/0** which matches the whole Ipv6 address space.
- **loopback**: Expanded to **loopback4** + **loopback6**.
- **loopback4**: Expanded to **127.0.0.0/8**
- **loopback6**: Expanded to **::1/128**
- **linklocal**: Expanded to **linklocal4** + **linklocal6**.
- **linklocal4**: Expanded to **169.254.0.0/16**
- **linklocal6**: Expanded to **fe80::/10**
- **local**: Expanded to **local4** + **local6**.
- **local4**: Expanded to four addresses as defined in **RFC1918**: - **127.0.0.0/8** - **10.0.0.0/8** - **172.16.0.0/12** - **192.168.0.0/16**
- **local6**: Expanded to four addresses: - **::1/128** - **fe80::/7** - **fc00::/7** - **fec0::/7**

SECURITY

The interface is only available if the sandbox lock is not set for the calling process. Similarly, command-line option parsing and configuration file parsing stops once a **lock:on** clause is executed.

RETURN VALUE

For *stat(2)* calls, on success, zero is returned. On error, -1 is returned, and *errno* is set to indicate the error.

For *open(2)* calls, on success the new file descriptor (a nonnegative integer) is returned. The file descriptor is randomized. On error, -1 is returned and *errno* is set to indicate the error.

On a successful call the *stat(2)* buffer has the following fields masked, other fields are equivalent to the character device */dev/null*:

- Inode is derived from the first 16 hex characters of **SYD_ID**, converted to u64 using native endianness; returns 0 if invalid or not set. Refer to the ENVIRONMENT section of the *syd(1)* manual page for more information on how **SYD_ID** is generated.
- Mode field represents the file type (character device) and permissions, with special bits set as follows: sticky bit for *unshare/mount:1*, SUID bit for *unshare/user:1*, SGID bit for *unshare/net:1*, user read/write/exec bits for Read, Write and Exec sandboxing, group read bit for Stat sandboxing, group write bit for Proxy sandboxing, group exec bit for TPE sandboxing, world read bit for Lock sandboxing, world write bit for Crypt sandboxing, and world exec bit for Force sandboxing.

- Nlink field represents the lower 32 bits of sandboxing capabilities, encoded as: *lock, walk, stat, read, write, exec, ioctl, create, delete, rename, symlink, truncate, chdir, readdir, mkdir, rmdir, chown, chgrp, chmod, chattr, chroot, utime, mkbdev, mkcdev, mkfifo, mktemp, net/bind, net/connect, net/sendfd, force, proxy, and pty.*
- Device type represents *syd(2)* API version (major, minor)
- Access, creation, and modification times are non-zero constants.

ERRORS

EBUSY Attempted to edit a setting at runtime that must be configured at startup

EEXIST Attempted to add an UID/GID transition for SafeSetID but a transition with the same source UID/GID exists.

EINVAL Sandbox command is syntactically incorrect.

ENOENT Result of the given sandbox query is false (e.g. **test -c /dev/syd/sandbox/stat?**).

ENOENT Sandbox lock is on, no commands are allowed.

ENOKEY Crypt sandboxing is on but no encryption key was supplied.

EKEYREVOKED Session keyring is not linked to the user keyring for Crypt sandboxing.

ENODATA SafeSetID is on but no UID/GID transits were defined for the current user/group.

EOPNOTSUPP Sandbox command is not supported.

SEE ALSO

syd(1), syd(5), syd(7), syd-ls(1) open(2), stat(2), perf(1), pledge(2), ptrace(2), seccomp(2), strace(1), glob(3p), io_uring(7), gdb(1), valgrind(1), wordexp(3)

- **syd** homepage: <https://sydbox.exherbo.org>
- **libsyd** homepage: <https://libsyd.exherbo.org>
- **gosyd** homepage: <https://gosyd.exherbo.org>
- **plsyd** homepage: <https://plsyd.exherbo.org>
- **pysyd** homepage: <https://pysyd.exherbo.org>
- **rbsyd** homepage: <https://rbsyd.exherbo.org>
- **syd.el** homepage: <https://sydel.exherbo.org>

- **libsydttime** homepage: <https://lib.rs/libsydttime>
- **LandLock** homepage: <https://landlock.io/>
- **vDSO** wiki: <https://en.wikipedia.org/wiki/VDSO>
- **parse-size** documentation: https://docs.rs/parse-size/1.0.0/parse_size/
- **ipnetwork** documentation: <https://docs.rs/ipnetwork>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd(1)

NAME

syd - Rock solid application kernel

SYNOPSIS

syd [-acefhlpqxE_{VP}] [--] {command [arg...]}

syd --api

syd --check

syd --el

syd --sh

DESCRIPTION

Syd is a utility leveraging the *seccomp*(2) system call for sandboxing processes on Linux systems version 5.19 or later. It enables fine-grained control over a process's filesystem and network access *without requiring root privileges*. Syd is designed for ease of use across a wide array of architectures, including **x86**, **x86_64**, **x32**, **armv7**, **aarch64**, **loongarch64**, **mips**, **mips64**, **mips64el**, **ppc**, **ppc64**, **ppc64le**, **riscv64**, and **s390x** embodying the principle of providing simple, flexible, and robust access control to Linux users.

The core functionality of Syd revolves around restricting a process's resource access through several mechanisms:

- **Bind Mounts**: Utilized within a mount namespace to enforce restrictions at the **Virtual File System (VFS)** level, such as **read-only**, **nodev**, **noexec**, **nosuid**, and **nosymfollow**.
- **Landlock**: Employs read-only and read-write path restrictions at the kernel level.
- **seccomp-bpf**: Applies Secure Computing user filters for kernel-space sandboxing.
- **seccomp-notify**: Enables sandboxing in kernel space with user space fallback for dereferencing pointer arguments in system calls, including pathnames and network addresses. Access checks utilize UNIX shell-style patterns and CIDR notation, defaulting to denying system calls with **EACCES** while attempting to emulate successful calls to mitigate **Time-of-Check to Time-of-Use (TOCTOU)** attack vectors.

Prerequisites for Syd include a Linux kernel supporting *pidfd_getfd*(2) and *pidfd_send_signal*(2) system calls, **SECCOMP_USER_NOTIF_FLAG_CONTINUE** operation in the Secure Computing facility, and preferably the **CONFIG_CROSS_MEMORY_ATTACH** kernel option. For syscall emulation, Syd uses the seccomp operation **SECCOMP_IOCTL_NOTIF_ADDFD**. Moreover Syd sets the **SECCOMP_FILTER_FLAG_WAIT_KILLABLE_RECV** flag to correctly handle interrupts during tracing. While *Linux version 5.19 or later is required*, for Landlock support Syd requires a kernel configured with the option **CONFIG_LSM_LANDLOCK** supporting *Landlock ABI version 3*, with *syd-lock*(1) available as a helper program to verify kernel support. Linux kernel options **CONFIG_KCMP** and **CONFIG_UNIX_DIAG** are recommended.

Syd is committed to maintaining rigorous security standards by strictly delimiting the resource space accessible to sandboxed processes. In the **SECURITY** section of the *syd(7)* manual page, a detailed enumeration of the security hardening measures implemented by Syd is provided, along with optional configurations to relax certain restrictions. This flexibility allows for the accommodation of a diverse range of processes within the sandbox environment.

The approach to security within Syd is methodically designed to balance robust protection with operational flexibility, ensuring that users have the ability to fine-tune the sandboxing mechanisms to meet specific requirements. By offering insights into the hardening techniques and customization options, Syd empowers users to navigate the trade-offs between security and functionality effectively.

OPTIONS

The following options are understood:

-h, --help	Show usage and exit.
-V, --version	Show version and exit.
-C, --check	Print sandboxing support information about the current system and exit.
-v, --verbose	Increase verbosity, equivalent to incrementing log/verbose by one.
-c	Login shell compatibility Causes command to be executed under a shell with the user profile. The shell to execute is <i>/bin/sh</i> by default. Use the environment variable SYD_SHELL to override.
-f	Login shell compatibility. Causes Syd to parse the user profile on startup.
-l, --login	Login shell compatibility Causes Syd to parse the user profile on startup.
-q	Enable quick boot mode for faster startup times. This must be passed as the first option or it will be ignored. See the explanation of the environment variable SYD_QUICK_BOOT for the safety of this option.
-x	Enable trace aka "dry run" mode. In this mode Syd will allow system calls even if they raise access violations. This mode with extended logging can be used to build sandboxing profiles in an automated way. See <i>pandora(1)</i> which is a tool that uses Syd's trace mode to automatically generate sandbox profiles.
-m config	Configure sandbox during init, may be repeated.
-p name	Use a sandbox profile during init, may be repeated.
-P path	Run a configuration file during init, may be repeated.
-a alias	Set alias of the command. Passed as argv[0] to the program.
-e	Use -e var=val to put var=val in the environment for command, may be repeated. Use -e var to remove var from the environment for command, may be repeated. Use -e var= to pass-through an unsafe environment variable, may be repeated.
-E mode	Export secure computing rules with the given format to standard output and exit. Mode must be one of bpf or pfc : bpf , aka Berkeley Packet Filter is a binary, machine readable format, whereas pfc , aka Pseudo Filter Code is a textual, human readable format.
--api	Output <i>syd(2)</i> API specification in JSON format. This specification is intended to ease generation of language bindings. This specification is also available via the magic path /dev/syd .
--el	Output syd.el which is the Emacs Lisp implementation of Syd <i>stat(2)</i> interface. This file is also available via the magic path /dev/syd.el .
--sh	Output a shell script which defines the esyd helper function. This file is also available via the magic path /dev/syd.sh . Works with POSIX sh, bash and zsh. You may use eval "\$(syd --sh)" in your shell init file.

INVOCATION

Syd executes a command with the specified arguments under a sandbox and exits with the same status. The sandbox may be constructed by command-line arguments and configuration files. *syd(2)* API is available for dynamic configuration if the sandbox lock allows it. An IPC socket may be configured with the *ipc* command to configure Syd through a UNIX socket. Refer to the *syd(2)* manual page for more information.

ENVIRONMENT

SYD_ID	Specify sandbox id as 128 lowercase hexadecimal characters. The first 32 characters may not be all zeroes. If this variable is not set by the user at startup, Syd generates it by hashing AT_RANDOM bytes with SHA3-512. Syd panics if user passes the value in incorrect format.
SYD_IPC	Specify UNIX socket address for runtime configuration. Equivalent to the ipc command, see the <i>syd(2)</i> manual page. ipc command has precedence over this environment variable.
SYD_LOG	Set log level to emerg , alert , crit , error , warn , notice , info or debug .
SYD_LOG_BUF_LEN	Set <i>syslog(2)</i> ring buffer capacity. By default, the ring buffer is allocated on the stack with an architecture-dependent size. Setting this variable makes Syd allocate the ring buffer on the heap with the user-specified size. Note, the value is parsed using the parse-size crate. Refer to their documentation for information on formatting.
SYD_LOG_FD	Set log file descriptor, defaults to <i>stderr(3)</i> . Negative values are permitted as a shorthand to disable logging. Positive values must be valid FDs or Syd will exit with EBADF .
SYD_PDS	Set parent-death signal using signal name or number.
SYD_PID_FN	Set pid filename, makes Syd write its process ID to this file at startup. The file must not exist and is going to be created with user-only read permissions.
SYD_NPROC	Set the number of core syscall handler threads, defaults to the number of CPUs. The number must be at least 1.
SYD_NPROC_MAX	Set the number of maximum syscall handler threads, defaults to <code>usize::MAX</code> . The number must be greater than SYD_NPROC .
SYD_SHELL	Pick the shell to spawn when invoked as a login shell, defaults to /bin/sh .
SYD_DUMP_SCOMP	Export secure computing rules with the given format, equivalent to the -E option.
SYD_SKIP_SCOMP	Skip <i>seccomp(2)</i> confinement of per-Syd threads. This is unsafe and should only used for profiling. Syd honours RUST_BACKTRACE environment variable when this is set.
SYD_FORCE_NO_SYMLINKS	Force RESOLVE_NO_SYMLINKS resolve flag at <i>open(2)</i> boundary. Equivalent to trace/force_no_symlinks:1 .
SYD_FORCE_NO_MAGICLINKS	Force RESOLVE_NO_MAGICLINKS resolve flag at <i>open(2)</i> boundary. Equivalent to trace/force_no_magiclinks:1 .
SYD_FORCE_NO_XDEV	Force RESOLVE_NO_XDEV resolve flag at <i>open(2)</i> boundary. Equivalent to trace/force_no_xdev:1 .
SYD_FORCE_CLOEXEC	Force close-on-exec for file descriptors.

SYD_FORCE_RAND_FD	Equivalent to trace/force_cloexec:1 . Use randomized file descriptors to harden against fd reuse.
SYD_FORCE_RO_OPEN	Equivalent to trace/force_rand_fd:1 . Reject creating and writing <i>open(2)</i> calls.
SYD_FORCE_TTY	Equivalent to trace/force_ro_open:1 . Force TTY output which is pretty-printed JSON.
SYD_QUIET_TTY	Force quiet TTY output which is line-oriented JSON.
SYD_PROXY_HOST	Override the default value of proxy/ext/host , If the value is a hostname and not an IP address, Syd resolves this hostname at startup and selects a response IP randomly.
SYD_PROXY_PORT	Override the default value of proxy/ext/port .
SYD_PROXY_UNIX	Set the default value for proxy/ext/unix which overrides proxy/ext/host .
SYD_QUICK_BOOT	Enable quick boot mode, this makes Syd startup noticeably faster: However, quick boot removes a layer of defense against some container breaks! Use this if you frequently re-execute <i>syd(1)</i> or <i>syd-oci(1)</i> , as Exherbo Linux does during <i>cave-generate-metadata(1)</i> .
SYD_NO_CROSS_MEMORY_ATTACH	Disable cross memory attach and use <i>proc_pid_mem(5)</i> unconditionally.
SYD_PROC_PID_MEM_FALLBACK	By default, Syd uses cross memory attach unconditionally. Setting this variable causes Syd to fall back to <i>proc_pid_mem(5)</i> automatically, if <i>process_vm_readv(2)</i> or <i>process_vm_writev(2)</i> fails with ENOSYS , which indicates kernel support is missing for these system calls. The variable SYD_NO_CROSS_MEMORY_ATTACH has precedence over this variable.
SYD_ASSUME_KERNEL	Override <i>uname(2)</i> to get host Linux kernel version used for feature detection. Syd reports the major and minor kernel version at <i>uname(2)</i> boundary within the sandbox. The micro version is randomized per-Syd run to prevent information leaks.
SYD_PALUDIS_LPATH	Override <i>sandbox/lpath</i> option for the <i>paludis</i> profile, defaults to off.
SYD_PALUDIS_IOCTL	Override <i>sandbox/ioctl</i> option for the <i>paludis</i> profile, defaults to off.
SYD_USER_LPATH	Override <i>sandbox/lpath</i> option for the <i>user</i> profile, defaults to on.
CARGO_BIN_EXE_syd-pty	Path to the <i>syd-pty(1)</i> utility. Default is to search PATH .
CARGO_BIN_EXE_syd-tor	Path to the <i>syd-tor(1)</i> utility. Default is to search PATH .

LOGGING

There're eight log levels: emerg, alert, crit, error, warn, notice, info, and debug. Log level may be set with the **SYD_LOG** environment variable. Logs go to standard error unless a file descriptor is specified with the environment variable **SYD_LOG_FD**. The messages of severity warn and above are also sent to *syslog(3)* unless the environment variable **SYD_NO_SYSLOG** is set.

Syd logs in JSON lines. Below is a list of some of the commonly used keys and their meanings:

KEY	DESCRIPTION
ctx	Context of the log entry, e.g. access , safesetid , segvguard etc.
cap	Sandbox capability
act	Sandbox action: Allow , Warn , Deny , Panic , Stop , Abort , Kill or Exit
pid	Process ID
path	Path argument of the syscall
addr	Network address argument of the syscall, e.g. 127.0.0.1:22

unix	UNIX socket address argument of the syscall
ipv	IP version of the network address in the addr field (4 or 6)
abs	True if the socket address in the unix field is an abstract UNIX socket
sys	Name of the syscall
arch	Architecture of the syscall
args	Arguments of the syscall
src	Origin of the syscall in format path+offset Use, e.g. objdump -D path grep offset to display the syscall instruction
cmd	Process name, or command line if log output is a TTY or log feature is enabled
cwd	Current working directory of the process
uid	User ID
time	Timestamp in ISO8601-compatible format, currently YYYYMMDDThhmmssZ Time format may change but it will always remain ISO8601-compatible. Formatting errors fallback to printing the timestamp as an integer.
err	Error information
msg	Miscellaneous informational messages, mostly used with the info log level
tip	Informational messages on how to configure the sandbox

EXIT CODES

Syd exits with the same exit code as the sandbox process itself. If the sandbox process exits with a signal, Syd exits with 128 plus the value of the signal. In case there was an error in spawning or waiting for the sandbox process, Syd exits with **errno** indicating the error condition. E.g. **syd true** returns **0**, **syd false** returns **1**, and **syd -- syd true** returns **16** which stands for **EBUSY** which means **Device or resource busy** indicating there is already a secure computing filter loaded.

BENCHMARKS

The table below lists the benchmark runs we ran for Syd:

1: compile kernel	sydbox-{ 1,3 }	https://gitlab.exherbo.org/-/snippets/253
2: compile kernel	sydbox-{ 1,3 }	https://gitlab.exherbo.org/-/snippets/253
3: unpack compressed tarball	sydbox-{ 1,3 }, Gentoo sandbox	https://gitlab.exherbo.org/-/snippets/253
4: compile kernel	sydbox-{ 1,3 }, Gentoo sandbox	https://gitlab.exherbo.org/-/snippets/259
5: compile kernel in a Podman container	syd-oci, crun, runc, youki, gvisor	https://gitlab.exherbo.org/-/snippets/261
6: compile kernel in a Podman container	syd-oci, crun, runc, youki, gvisor	https://gitlab.exherbo.org/-/snippets/262
7: run sqlite-bench	no-syd, syd, syd+crypt	https://gitlab.exherbo.org/-/snippets/275

SEE ALSO

syd(2), *syd(5)*, *syd(7)*, *syd-lock(1)*, *syd-ls(1)*

- **syd** homepage: <https://sydbox.exherbo.org/>
- **libsyd** homepage: <https://libsyd.exherbo.org/>
- **pandora** homepage: https://lib.rs/pandora_box
- **paludis** homepage: <http://paludis.exherbo.org/>
- **Landlock** homepage: <https://landlock.io>

- **Path** wiki: [https://en.wikipedia.org/wiki/Path_\(computing\)](https://en.wikipedia.org/wiki/Path_(computing))
- **Unix domain socket** wiki: https://en.wikipedia.org/wiki/Unix_domain_socket
- **IPv4** wiki: <https://en.wikipedia.org/wiki/IPv4>
- **IPv6** wiki: <https://en.wikipedia.org/wiki/IPv6>
- **TOCTOU** wiki: https://en.wikipedia.org/wiki/Time-of-check_to_time-of-use
- **VFS** wiki: https://en.wikipedia.org/wiki/Virtual_file_system
- **ipnetwork** documentation: <https://docs.rs/ipnetwork>
- **Enabling Logging**: https://docs.rs/env_logger/latest/env_logger/#enabling-logging

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-aes(1)

NAME

syd-aes - AES-CTR encryption and decryption utility

SYNOPSIS

syd-aes [-hv] -e|-d -k <key-serial> -i <iv-hex>

DESCRIPTION

The **syd-aes** utility uses the Linux Kernel Cryptography API to encrypt and decrypt data using AES-CTR mode using *keyrings(7)* to select the encryption/decryption key to use without copying key material into userspace. It supports both encryption and decryption operations, with the key serial provided as a 32-bit ID and IV provided as a hexadecimal string. Given data from standard input, **syd-aes** performs the specified operation and outputs the result to standard output. **syd-aes** uses pipes and *splice(2)* to transfer data using zero-copy, and therefore able to encrypt/decrypt files of arbitrary size.

OPTIONS

-h	Display help.
-v	Enable verbose mode. If standard error is a terminal, print progress updates periodically, similar to dd(1) .
-e	Encrypt the input data.
-d	Decrypt the input data.
-k <key-serial>	Key serial ID (32-bit integer). The kernel key that syd-aes (via <code>ALG_SET_KEY_BY_KEY_SERIAL</code>) will read must grant the caller search permission -- i.e. have the <code>KEY_(POS USR GRP OTH)_SEARCH</code> permission bit(s) set so the kernel can locate and copy the key data into the crypto API; otherwise the operation will be denied (EPERM: "Operation not permitted").
-i <iv>	Hex-encoded IV (128 bits).

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd-key(1)*, *splice(2)*, *keyrings(7)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-asm(1)

NAME

syd-asm - Disassemble raw CPU instructions from standard input

SYNOPSIS

syd-asm [-h] [-a *arch*]

syd-asm [-h] -a *list*

DESCRIPTION

syd-asm reads CPU instructions as raw bytes or hexadecimal encoded from standard input and disassembles them. The disassembled instructions are printed in JSON format as one instruction per-line. The disassembly is done natively for architectures **x86**, **x86_64**, **x32**, **arm**, **aarch64**, and **riscv64** and falls back to GNU *objdump*(1) for other architectures. There's no support for LLVM *objdump*(1) yet.

OPTIONS

-
- h** Display help.
 - a** Specify alternative architecture, such as **x86**, **x86_64** and **aarch64**.
Use **list** to print the list of libseccomp supported architectures.
-

SEE ALSO

syd(1), *syd*(2), *syd*(5), *syd*(7), *objdump*(1)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-aux(1)

NAME

syd-aux - Print auxiliary vector information

SYNOPSIS

syd-aux [*-hrs*]

DESCRIPTION

Print auxiliary vector information.

If **-r** is given print hexadecimal-encoded AT_RANDOM cookie.

If **-s** is given exit with success if AT_SECURE is set.

OPTIONS

-h Display help and exit.

-r Print hexadecimal-encoded AT_RANDOM cookie.

-s Exit with success if AT_SECURE is set.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd-elf(1)*, *syd-ldd(1)*, *getauxval(3)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-bit(1)

NAME

syd-bit - Utility to flip bits in files

SYNOPSIS

syd-bit [-h] -i <idx> <file>

syd-bit [-h] -r <file>

DESCRIPTION

The **syd-bit** utility flips the given bit or a random bit in the specified file. It provides a simple way to simulate bit-flip attacks.

OPTIONS

-h	Display help.
-i <idx>	Flip the bit at index <idx> in the file
-r	Flip a random bit in the file

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd-aes(1)*, *syd-key(1)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-cap(1)

NAME

syd-cap - Print information on Linux capabilities

SYNOPSIS

syd-cap [*-h*]

DESCRIPTION

Print information on Linux capabilities.

OPTIONS

-h Display help and exit.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd-aux(1)*, *syd-elf(1)*, *syd-ldd(1)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-cat(1)

NAME

syd-cat - Tool to parse, validate and display *syd(5)* configuration

SYNOPSIS

syd-cat [-hjJmM] [-p *name*] <*path*>...

DESCRIPTION

Given a list of paths, parses and validates *syd(5)* configuration.

Prints configuration to standard output on success.

Supported configuration file extensions are **.ipset**, **.netset**, and **.syd-3**.

OPTIONS

-h	Display help.
-j	Display Syd configuration as JSON.
-J	Display Syd configuration as compact JSON.
-m magic	Run a magic command at init, may be repeated.
-M magic	Run a magic command at exit, may be repeated.
-p name	Display rules of the profile with the given name. Use list as name to display the list of profiles.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd(7)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-cpu(1)

NAME

syd-cpu - Print the number of CPUs

SYNOPSIS

syd-cpu [*-hlp*]

DESCRIPTION

Print the number of CPUs.

OPTIONS

-
- h** Display help.
 - l** Print the number of logical CPUs (default).
 - p** Print the number of physical CPUs.
-

SEE ALSO

syd(1), *syd(2)*, *syd(5)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-dns(1)

NAME

syd-dns - Resolve hostname into IPs using system DNS resolver

SYNOPSIS

syd-dns [-hr46] hostname

syd-dns [-R] IPv4/6 address

DESCRIPTION

Resolve hostname into IPs using system DNS resolver.

Given -R, perform a reverse-DNS lookup for the given IPv4/6 address using the system DNS resolver.

OPTIONS

-h Display help and exit.

-4 Print only IPv4 addresses

-6 Print only IPv6 addresses

-r Print a random IP picked using *getrandom*(2)

-R Perform a reverse DNS lookup using *getnameinfo*(3)

SEE ALSO

syd(1), *syd*(2), *syd*(5), *syd-net*(1), *getrandom*(2), *getnameinfo*(3)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbbox/sydbbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbbox/-/issues>. Discuss in #sydbbox on Libera Chat or in #sydbbox:mailstation.de on Matrix.

syd-elf(1)

NAME

syd-elf - Print executable file information

SYNOPSIS

syd-elf [-36dhpstxX] *binary|script*

DESCRIPTION

Given a binary, print file name and ELF information.

Given a script, print file name and "SCRIPT".

The information line is a list of fields delimited by colons.

OPTIONS

-h Display help and exit.

-3 Exit with success if the given binary is 32-bit.

-6 Exit with success if the given binary is 64-bit.

-d Exit with success if the given binary is dynamically linked.

-s Exit with success if the given binary is statically linked.

-p Exit with success if the given binary is a Position Independent Executable (PIE).

-t Print the type of the file as an abbreviation.

-x Exit with success if the given executable is a script.

-X Exit with success if the given binary has executable stack.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd-ldd(1)*, *ldd_(1)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-emacs(1)

NAME

syd-emacs - Convenience wrapper to run Emacs under Syd

SYNOPSIS

syd-emacs [*command-line switches*] [*files...*]

DESCRIPTION

syd-emacs is a convenience wrapper to run Emacs under Syd. All command-line arguments are passed directly to *emacs(1)*.

FILES

<code>/dev/syd.el</code>	Emacs Lisp <i>syd(2)</i> API library. Access assumes sandbox lock is accessible, ie "lock:off" or "lock:exec".
<code>~/.emacs.d/init.syd-3</code>	Emacs Syd profile, if this file does not exist, the "lib" profile is used instead. Note, the "lib" profile turns all sandboxing off and sets "lock:exec" to allow access to the virtual file "/dev/syd.el". If you do not want to turn all sandboxing off, you're encouraged to configure a profile with this file. Ensure to add "lock:exec" in the end so that the initial <i>emacs(1)</i> process can access the sandbox to load "/dev/syd.el". Ensure to call "(syd-lock :lock-on)" from within <i>emacs(1)</i> when you're done configuring <i>syd(1)</i> using the <i>syd(2)</i> API. If you do not want to allow access to the sandbox lock, you're encouraged to call <i>syd(1)</i> manually with <i>emacs(1)</i> .
<code>~/.emacs.d/syd.log</code>	This is just a convenience wrapper. Emacs Syd log file, access violations are logged to this file. <i>syd-emacs(1)</i> opens this file, and sets SYD_LOG_FD environment variable to the value of the file descriptor. The file is opened for create+append-only.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *emacs(1)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbbox/sydbbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbbox/-/issues>. Discuss in #sydbbox on Libera Chat or in #sydbbox:mailstation.de on Matrix.

syd-env(1)

NAME

syd-env - Run a command with the environment of the process with the given PID

SYNOPSIS

syd-env pid [-i] [name=value]... {command [arg...]}

syd-env -e <eval-str>

DESCRIPTION

syd-env utility runs a command with the environment of the process with the given PID. It is similar to the *env(1)* utility except it allows picking the environment of an arbitrary process.

Given "-e" with a string argument, **syd-env** performs environment expansion and command substitution using *wordexp(3)*.

SECURITY

wordexp(3) child process is executed in a confined environment with a timeout of 3 seconds. Confinement is done using Landlock, namespaces and seccomp.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *env(1)*, *wordexp(3)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-exec(1)

NAME

syd-exec - Construct a sandbox command to execute a process outside syd

SYNOPSIS

syd-exec {command [arg...]}

DESCRIPTION

The **syd-exec** utility may be used to construct a sandbox command to execute a process outside syd. See the documentation of the **cmd/exec** sandbox command in *syd(2)* manual page for more information on its usage.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *exec(3)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-fd(1)

NAME

syd-fd - Interact with remote file descriptors

SYNOPSIS

syd-fd [-h] [-p *pid*] [-f *remote_fd[:local_fd]*]... {*command* [*args...*]}

DESCRIPTION

The **syd-fd** utility can be used to interact with remote file descriptors on Linux systems. Given only a PID argument with **-p**, it lists the open files of the process with the given PID in line-oriented compact JSON format. Given no PID argument it lists the open files of the current process. The **-f** argument can be used to transfer remote file descriptors using *pidfd_getfd(2)* which requires Linux \geq 5.6. Optionally a colon-delimited local file descriptor may be specified as target fd. Specify **rand** as target to duplicate the file descriptor to a random available file descriptor slot. If a command is given it is executed and the file descriptors are transferred to the process. If no command is given `"/bin/sh"` is executed.

OPTIONS

-h	Display help.
-p <i>pid</i>	Specify process ID. If not given, list fds of current process.
-f <i>remote_fd[:local_fd]</i>	Specify remote fd to transfer. Optionally specify colon-separated local fd as target, or rand for random target.

EXIT CODES

syd-fd exits with the same exit code as the command.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd-lock(1)*, *syd-pds(1)*, *pidfd_getfd(1)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbbox/sydbbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbbox/-/issues>. Discuss in #sydbbox on Libera Chat or in #sydbbox:mailstation.de on Matrix.

syd-fork(1)

NAME

syd-fork - Fork fast in an infinite loop.

SYNOPSIS

syd-fork [-h]

DESCRIPTION

If no arguments are provided, the program will quickly create an infinite number of child processes through forking. This is implemented using inline assembly on x86, x86_64, arm, and aarch64 architectures, making it significantly faster and more efficient than the bash fork bomb.

When the **-h** flag is passed, a warning message will be displayed advising that this program is intended for stress-testing the pid limiter and should not be used for any other purpose. It is not intended to be used as a joke and should be used with caution. Use of the program is at the user's own risk. To stress-test the pid limiter, run the program with no arguments. The program will quickly create an infinite number of child processes through forking and it will quickly reach the maximum number of processes that the system can handle.

OPTIONS

-h Display help.

EXAMPLES

To run syd-fork on April 1st at 8:00 AM UTC, you can use the *at(1)* command. This is useful for scheduling the program to execute at a specific time for testing or demonstration purposes. Ensure that the *at(1)* daemon is running on your system and that you have permission to schedule jobs with *at(1)*.

```
$ echo "syd-fork" | at 08:00 April 1
```

NOTES

Distribution maintainers are recommended to *ln(1)* "syd-fork" to "syd-fuck" under an NSFW option to help treat anger issues.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *at(1)*, *fork(2)*, *ln(1)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-hex(1)

NAME

syd-hex - Hex-encode/decode the given file or standard input

SYNOPSIS

syd-hex [*-hdeflsC*] <file|->

DESCRIPTION

Given a file, hex-encode the file and print.

Given no positional arguments or "-" as argument, hex-encode standard input and print.

Use **-d** to hex-decode rather than encode.

Use **-s** with **-d** to hex-decode data with newline-delimited chunks.

Use **-C** with **-s** to warn and continue in case of read or encoding errors.

Use **-f** to force print decoded hex to TTY (insecure). By default, decoded hex output is *not* written to standard output if it's attached to a TTY.

Use **-l** <human-size> to exit after size bytes are read.

Use **-l** <line-count> with **-s** to exit after count lines are read.

OPTIONS

-h	Display help.
-d	Hex-decode.
-e	Hex-encode (default).
-f, --force-tty	Force output to TTY (insecure).
-l <human-size>	Exit after size bytes are read.
-l <line-count>	Exit after count lines are read in stream mode.
-s, --stream	Enable stream mode with -d when data is hex-decoded with newline-delimited chunks.
-C, --continue-on-failure	Continue in case of read or encoding errors in stream mode.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *tty(1)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbbox/sydbbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbbox/-/issues>. Discuss in #sydbbox on Libera Chat or in #sydbbox:mailstation.de on Matrix.

syd-info(1)

NAME

syd-info - Print system information

SYNOPSIS

syd-info [-h]

DESCRIPTION

Print system information.

System information is acquired using the *sysinfo(2)* system call.

OPTIONS

-h Display help and exit.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *sysinfo(2)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-key(1)

NAME

syd-key - Utility to generate encryption keys and save to *keyrings(7)*

SYNOPSIS

syd-key [-hpP] [-d keydesc] [-t keytype] [-k keyring]

DESCRIPTION

The **syd-key** utility generates random 256-bit encryption keys using Linux Kernel's random number generator. The key is saved to Linux *keyrings(7)* and key serial ID is printed as a decimal 32-bit integer to standard output.

OPTIONS

-h	Display help.
-p	Read passphrase from the controlling TTY (NOT <i>stdin(3)</i>) and derive key material. syd-key will refuse to read the passphrase from <i>stdin(3)</i> for safety, unless -P is explicitly supplied. The passphrase is hashed using SHA3-256; the resulting digest is stored with <i>keyrings(7)</i> interface. The passphrase buffer is zeroized after use. Key serial ID is printed on <i>stdout(3)</i> . If none of -pP is supplied, a random key is generated using <i>getrandom(2)</i> with GRND_RANDOM flag instead.
-P	Read passphrase from <i>stdin(3)</i> and derive key material. This option is intended for non-interactive use (for example, piping a passphrase from a password manager). Use with caution: reading a passphrase from <i>stdin(3)</i> can be less secure than reading from the controlling TTY because it may be observable by other processes, recorded in shell constructs, or otherwise leaked by the environment. The passphrase is hashed using SHA3-256; the resulting digest is stored with <i>keyrings(7)</i> interface. The passphrase buffer is zeroized after use. Key serial ID is printed on <i>stdout(3)</i> . If none of -pP is supplied, a random key is generated using <i>getrandom(2)</i> with GRND_RANDOM flag instead.
-d kdesc	Specify alternative key description. Default is SYD-3-CRYPT .
-t ktype	Specify alternative key type. Default is user .
-k kring	Specify alternative key ring ID. Default is KEY_SPEC_USER_KEYRING . May be exactly one of thread , process , session , user or user-session . May also be a 32-bit decimal number specifying a keyring ID.

CAVEATS

Keys in *keyrings(7)* are identified by their (type, description) pair. When **syd-key** invokes *add_key(2)* with a type/description that already exists in the target keyring, the kernel will update the existing key's payload instead of creating a new key. To force creation of a new key, use a unique description (for example by appending a UUID or timestamp).

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd-aes(1)*, *getrandom(2)*, *add_key(2)*, *keyrings(7)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-ldd(1)

NAME

syd-ldd - Print shared object dependencies in a secure way

SYNOPSIS

syd-ldd [*option*]... *file*...

DESCRIPTION

The **syd-ldd** utility is meant to be used as a secure alternative to *ldd*(1). It creates a syd sandbox and runs *ldd*(1) under it with restricted privileges.

INVOCATION

syd-ldd utility is equivalent to invoking the following command:

```
syd
-pimutable
-msandbox/read:on
-msandbox/stat:off
-msandbox/exec:on
-msandbox/write:on
-msandbox/net:on
-msandbox/lock:on
-mallow/read+/etc/ld-*.path
-mallow/read+/etc/locale.alias
-mallow/read+/usr/share/locale*/**/*.mo
-mallow/read+/usr/share/locale*/locale.alias
-mallow/read+/usr/lib*/locale*/locale-archive
-mallow/read+/usr/lib*/**/gconv-modules*
-mallow/read+/usr/**/LC_{ALL,COLLATE,CTYPE,IDENTIFICATION,MESSAGES}
-mallow/read+/**/*.so.[0-9]*
-mallow/exec+/lib**/ld-linux*.so.[0-9]
-mallow/exec+/usr/lib*/**/ld-linux*.so.[0-9]
-mallow/write+/dev/null
-mallow/lock/read+/
-mallow/lock/write+/dev/null
-mallow/read,write+/dev/tty
-mallow/read,exec+/path/to/ldd
/path/to/ldd -- args...
```

SEE ALSO

syd(1), *syd*(2), *syd*(5), *ldd*(1)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-lock(1)

NAME

syd-lock - Run a program under *landlock(7)*

SYNOPSIS

```
syd-lock [-bchrvwASUV] [-C level] [-E errata] [-F flag]... [-l category[,category...][+|-]path[port[-port]]... {command  
[args...]}
```

DESCRIPTION

syd-lock utility runs a program under *landlock(7)*. The program is confined by the given *landlock(7)* categories. Supported categories are *read*, *write*, *exec*, *ioctl*, *create*, *delete*, *rename*, *symlink*, *truncate*, *readdir*, *mkdir*, *rmdir*, *mkbdev*, *mkcdev*, *mkfifo*, *bind*, and *connect*. Categories other than *bind* and *connect* must specify paths to be confined. Both absolute and relative paths are permitted. Path must not contain magic symbolic links or parent (“..”) components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent (“..”) components in them. Categories *bind* and *connect* must specify a network port or closed port range separated by dash. Zero is a valid port number to confine binds and connects to ephemeral ports. *bind* category also supports absolute UNIX domain socket paths to confine their creation via *mknod(2)*. For full details and specific behavior of each *landlock(7)* category, refer to the **Sandboxing** and **Lock Sandboxing** sections of the *syd(7)* manual page.

OPTIONS

-h	Display help.
-v	Be verbose. Print <i>landlock(7)</i> status to <i>stderr(3)</i> before running the program.
-V	Print <i>landlock(7)</i> ABI version on <i>stdout(3)</i> .
-A	Print <i>landlock(7)</i> ABI version on <i>stdout(3)</i> and exit with it as exit code. Use for scripting.
-l cat[,cat...][+ -]path[port[-port]]	<p>Add or remove a <i>landlock(7)</i> rule with categories and an associated resource (path or port), may be repeated.</p> <p>Join categories and resource by either a “+” (plus) for add or a “-” (minus) for remove.</p> <p>Resource must be a path for all categories except <i>bind</i> and <i>connect</i>.</p> <p>Resource must be a port or a dash-delimited closed port range for <i>bind</i> and <i>connect</i>.</p> <p>Resource may also be a UNIX domain socket path for <i>bind</i> to confine <i>mknod(2)</i> with S_IFSOCK.</p> <p>Both absolute and relative paths are permitted for all categories except <i>bind</i> which requires an absolute path.</p> <p>Paths are stored as hash sets and ports as fixed bit sets to make stacking options simple and predictable.</p>

-C level	Set <i>landlock(7)</i> compatibility level. Must be one of <i>hard-requirement</i> , <i>soft-requirement</i> , and <i>best-effort</i> . Default is <i>hard-requirement</i> to adhere to the principle of secure defaults. Level can be given shortly as <i>hard</i> (or <i>h</i>), <i>soft</i> (or <i>s</i>) and <i>best</i> (or <i>b</i>).
-E errata	Query supported <i>landlock(7)</i> errata fixes. Use <i>-E list</i> to print list of known erratas. The argument may be a name or number. Use a number to query undefined erratas. Multiple erratas may be specified split by commas.
-F flags	Set <i>landlock_restrict_self(2)</i> flags. Use <i>-F list</i> to print a list of flags. See the FLAGS section for information on flags and their functionality.
-S	Enable scoped signals introduced with <i>landlock(7)</i> ABI 6.
-U	Enable scoped UNIX abstract sockets introduced with <i>landlock(7)</i> ABI 6.
-r path	Specify a read-only path, may be repeated. Equivalent to <i>-l read,readdir,exec,iocctl+path</i> .
-w path	Specify a read-write path, may be repeated. Equivalent to <i>-l all+path</i> .
-b port[-port]	Specify a port for <i>bind(2)</i> , may be repeated. Equivalent to <i>-l bind+port</i> .
-c port[-port]	Specify a port for <i>connect(2)</i> , may be repeated. Equivalent to <i>-l connect+port</i> .

CONFIGURATION

landlock(7) categories and their associated resources (paths or ports) are given with the *-l* option. This option accepts a comma separated list of categories, followed by either a "+" (plus) or a "-" (minus) symbol indicating to add or remove the given rule. Rulesets store paths as hash sets and ports as fixed bit sets to allow for simple and predictable stacking of multiple *-l* options. Use *-V* option to check for *landlock(7)* support in the Linux kernel. The specific support level may be determined by the exit code. Use *-A* option to check for *landlock(7)* ABI version.

ABI

landlock(7) ABI versioning makes it possible to adjust the security policy according to kernel capabilities. **syd-lock** has support for *landlock(7)* ABI 7 which is new in Linux-6.15. See the **HISTORY** section for information on when each *landlock(7)* ABI was introduced to the Linux kernel.

SETS

As of version 3.38.0, multiple categories may be specified split by commas and the following sets are defined to streamline sandbox profile composition. Names are intentionally chosen to be consistent with OpenBSD's *pledge(2)*:

all	All filesystem access rights
rpath	read, readdir
wpath	write, truncate
cpath	create, delete, rename
dpath	mkbdev, mkcdev
spath	mkfifo, symlink
tpath	mkdir, rmdir
inet	bind, connect

COMPATIBILITY LEVELS

As of version 3.35.0, *landlock(7)* compatibility level may be set using the *-C* option to one of the following levels: *hard-requirement*, or just *hard* or *h*, *soft-requirement*, or just *short* or *s*, and *best-effort*, or just *best* or *b*. Default is *hard-requirement* to adhere to the principle of secure defaults. In this level the requested *landlock(7)* restrictions are taken into account only if they are supported by the running system; if any requested feature is not supported, the operation returns a compatibility error and the sandbox is not entered. File *open(2)* errors during sandbox setup, including the *ENOENT* ("No such file or directory") *errno(3)*, return a fatal error in this level. In *soft-requirement* level the requested restrictions are taken into account if they are supported by the running system, or the entire sandboxing request is silently ignored otherwise; no compatibility error is returned. In *best-effort* level the requested restrictions are taken into account if they are supported by the running system, and any unsupported restrictions are silently ignored; no compatibility error is returned. In *soft-requirement* and *best-effort* levels file *open(2)* errors with the *ENOENT* ("No such file or directory") *errno(3)* are silently ignored. Other file *open(2)* errors are fatal.

FLAGS

As of version 3.38.0, *landlock(7)* flags may be set using the *-F* option. Flags may be specified using their names or numerical values. Multiple flags may be set at once by specifying them as a comma-separated list. Flags are supported beginning with *landlock(7)* ABI 7 which is new in Linux-6.15. List of supported flags are given below. Setting a flag on an unsupported ABI is a NO-OP unless otherwise noted.

log_same_exec_off	1: Disables logging of denied accesses originating from the thread creating the <i>landlock(7)</i> domain, as well as its children, as long as they continue running the same executable code (i.e., without an intervening <i>execve(2)</i> call). This is intended for programs that execute unknown code without invoking <i>execve(2)</i> , such as script interpreters. Programs that only sandbox themselves should not set this flag, so users can be notified of unauthorized access attempts via system logs. This flag requires <i>landlock(7)</i> ABI 7 support which is new in Linux-6.15.
log_new_exec_on	2: Enables logging of denied accesses after an <i>execve(2)</i> call, providing visibility into unauthorized access attempts by newly executed programs within the created <i>landlock(7)</i> domain. This flag is recommended only when all potential executables in the domain are expected to comply with the access restrictions, as excessive audit log entries could make it more difficult to identify critical events. This flag requires <i>landlock(7)</i> ABI 7 support which is new in Linux-6.15.
log_subdomains_off	4: Disables logging of denied accesses originating from nested <i>landlock(7)</i> domains created by the caller or its descendants. This flag should be set according to runtime configuration, not hardcoded, to avoid suppressing important security events. It is useful for container runtimes or sandboxing tools that may launch programs which themselves create <i>landlock(7)</i> domains and could otherwise generate excessive logs. Unlike log_same_exec_off , this flag only affects future nested domains, not the one being created. This flag requires <i>landlock(7)</i> ABI 7 support which is new in Linux-6.15.

SECURITY

As of version 3.35.0, the default *landlock(7)* compatibility level has been changed from *best-effort* to *hard-requirement*, and *ENOENT* (No such file or directory) errors are made fatal unless level is set to *best-effort*. This adheres to the principle of secure defaults and above all avoids the silent and dangerous trap where a non-existing file or directory which had been denied access (and skipped) at startup is created after and *landlock(7)* ends up allowing access to the newly created file or directory. For more information, see: <https://landlock.io/rust-landlock/landlock/trait.Compatible.html>

As of version 3.46.0, path must not contain magic symbolic links or parent ("..") components. Path is permitted to contain regular symbolic links. These regular symbolic symlinks are permitted to resolve to targets with parent ("..") components in them. *bind* category requires absolute UNIX socket paths. Path may be relative for other categories in which case it is resolved relative to the directory where *syd-lock(1)* was executed.

Consider combining *syd-lock*(1) use with *syd-mdwe*(1) to get W^X memory protections. See *syd-mdwe*(1) manual page for more information.

HISTORY

- 1st *landlock*(7) ABI was introduced with Linux-5.13.
- 2nd *landlock*(7) ABI was introduced with Linux-5.19.
- 3rd *landlock*(7) ABI was introduced with Linux 6.2.
- 4th *landlock*(7) ABI was introduced with Linux 6.7.
- 5th *landlock*(7) ABI was introduced with Linux 6.10.
- 6th *landlock*(7) ABI was introduced with Linux 6.12.
- 7th *landlock*(7) ABI was introduced with Linux 6.15.

Refer to the following links for more information:

- <https://git.kernel.org/stable/c/17ae69aba89dbfa2139b7f8024b757ab3cc42f59>
- <https://git.kernel.org/stable/c/cb44e4f061e16be65b8a16505e121490c66d30d0>
- <https://git.kernel.org/stable/c/299e2b1967578b1442128ba8b3e86ed3427d3651>
- <https://git.kernel.org/stable/c/136cc1e1f5be75f57f1e0404b94ee1c8792cb07d>
- <https://git.kernel.org/stable/c/2fc0e7892c10734c1b7c613ef04836d57d4676d5>
- <https://git.kernel.org/stable/c/e1b061b444fb01c237838f0d8238653afe6a8094>
- <https://git.kernel.org/stable/c/72885116069abdd05c245707c3989fc605632970>

EXIT STATUS

syd-lock exits with the same code as the child process on clean exit. On unclean termination, exit code is set to 128 plus signal number. In case executing the child process fails *syd-lock*(1) exits with the *errno*(3) number. *syd-lock -A* exits with the *landlock*(7) ABI version as exit code.

syd-lock -E exits with one of the following exit codes:

-
- | | |
|----------|---------------------------------|
| 0 | All erratas are available. |
| 1 | Some erratas are not available. |
| 2 | No erratas are available. |
-

syd-lock -V exits with one of the following exit codes based on support for the latest *landlock*(7) ABI:

-
- | | |
|------------|--------------------|
| 0 | Fully enforced |
| 1 | Partially enforced |
| 2 | Not enforced |
| 127 | Not supported |
-

syd-lock exits with **22** (EINVAL) for invalid CLI arguments.

EXAMPLES

```
$ syd-lock wget -O/dev/null chesswob.org
$ syd-lock -l read,exec+/ wget -O/dev/null chesswob.org
/dev/null: Permission denied
$ syd-lock -l read,exec+/ -l write+/dev/null wget -O/dev/null chesswob.org
Prepended http:// to 'chesswob.org'
--2025-04-30 16:24:35-- http://chesswob.org/
Resolving chesswob.org (chesswob.org)... 95.216.39.164, fe80::468a:5bff:fe88:2141
Connecting to chesswob.org (chesswob.org)|95.216.39.164|:80... failed: Permission denied.
Connecting to chesswob.org (chesswob.org)|fe80::468a:5bff:fe88:2141|:80... failed: Permission denied.
Retrying.

^C
$ syd-lock -l read,exec+/ -l write+/dev/null -l connect+80 -l connect+443 wget -O/dev/null chesswob.org
Prepended http:// to 'chesswob.org'
--2025-04-30 16:25:59-- http://chesswob.org/
Resolving chesswob.org (chesswob.org)... 95.216.39.164, fe80::468a:5bff:fe88:2141
Connecting to chesswob.org (chesswob.org)|95.216.39.164|:80... connected.
HTTP request sent, awaiting response... 302 Moved Temporarily
Location: https://www.chesswob.org/ [following]
--2025-04-30 16:25:59-- https://www.chesswob.org/
Loaded CA certificate '/etc/ssl/certs/ca-certificates.crt'
Resolving www.chesswob.org (www.chesswob.org)... 95.216.39.164, fe80::468a:5bff:fe88:2141
Connecting to www.chesswob.org (www.chesswob.org)|95.216.39.164|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 148827 (145K) [text/html]
Saving to: '/dev/null'

/dev/null                100%[=====>] 145.34K  --.-KB/s   in 0.01s

2025-04-30 16:25:59 (11.9 MB/s) - '/dev/null' saved [148827/148827]
$
```

SEE ALSO

landlock(7), *syd(1)*, *syd(2)*, *syd(5)*, *syd(7)*, *syd-mdwe(1)*, *syd-ofd(1)*, *syd-pds(1)*, *syd-sec(1)*

syd homepage: <https://sydbox.exherbo.org/>

Landlock homepage: <https://landlock.io/>

Landlock documentation: <https://docs.kernel.org/userspace-api/landlock.html>

Landlock admin guide: <https://docs.kernel.org/admin-guide/LSM/landlock.html>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-ls(1)

NAME

syd-ls - List unsafe directories, capabilities, system calls, environment variables, ioctl requests, personalities, and prctl options

SYNOPSIS

syd-ls [*set*]

DESCRIPTION

The **syd-ls** utility prints the names of the system calls which belong to the given set. Available sets are **cpu**, **dead**, **deny**, **ebpf**, **futex**, **hook**, **nice**, **noop**, **pkey**, **ptrace**, **safe**, **setid**, **time**, and **uring**.

If set is **drop**, **syd-ls** prints the list of Linux *capabilities*(7) that are dropped at startup.

If set is **env**, **syd-ls** prints the list of unsafe environment variables.

If set is **ioctl**, **syd-ls** prints the list of allowed ioctl requests.

If set is **prctl**, **syd-ls** prints the list of allowed prctl options.

If set is **personality**, **syd-ls** prints the list of allowed personalities.

Given no set, **syd-ls** lists all files in the current working directory. In this mode, *getdents64*(2) is used directly. Use to list files in untrusted directories with huge number of files. File names are printed hex-encoded, delimited by newline, use *syd-hex*(1) to decode. See **EXAMPLES** section for more information.

EXAMPLES

```
[alip@caissa tmp]$ mkdir test; cd test
[alip@caissa test]$ for i in {1..10000000}; do :>$i; done
[alip@caissa test]$ df -i .
Filesystem      Inodes    IUsed IFree IUse% Mounted on
tmpfs           10048576 10000228 48348   100% /tmp
[alip@caissa test]$ syd-ls | head -n5 | syd-hex -dfs
.
..
10000000
99999999
99999998
[alip@caissa test]$ for ls in syd-ls gls 9ls 'busybox ls'; do
> sync
> sudo sh -c 'echo 3 > /proc/sys/vm/drop_caches'
> echo "[*] $ls"
> time $ls >/dev/null
> done
[*] syd-ls
syd-ls: Listed 10000002 files in 2.882764582 seconds.

real    0m2.889s
user    0m0.883s
```

```
sys      0m1.995s
[*] gls

real     0m7.548s
user     0m5.724s
sys      0m1.803s
[*] 9ls

real     0m15.306s
user     0m2.523s
sys      0m12.743s
[*] busybox ls

real     0m18.011s
user     0m11.178s
sys      0m6.786s
[alip@caissa test]$ gls --version | head -n1
ls (GNU coreutils) 9.6
[alip@caissa test]$ pacman -Ss 9base | head -n1
extra/9base 6-9 [installed]
[alip@caissa test]$ busybox | head -n1
BusyBox v1.36.1 () multi-call binary.
```

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd-hex(1)*, *getdents(2)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-mdwe(1)

NAME

syd-mdwe - Run a program under Memory-Deny-Write-Execute protections

SYNOPSIS

syd-mdwe [-hms] {command [args...]}

DESCRIPTION

syd-mdwe utility runs a program under Memory-Deny-Write-Execute (MDWE) protections. The protections can be applied using *prctl(2)* and *seccomp(2)*, These protections are identical to what Syd applies by default.

OPTIONS

-
- h** Display help.
 - m** Enable MDWE protections using *prctl(2)* PR_SET_MDWE (default: both).
 - s** Enable MDWE protections using *seccomp(2)* (default: both).
-

EXAMPLES

Running pax-test once standalone and once under *syd-mdwe(1)* on a 6.8 kernel we get the following differences:

PaX Testcase	standalone	mdwe
Executable anonymous mapping	Killed	Killed
Executable bss	Killed	Killed
Executable data	Killed	Killed
Executable heap	Killed	Killed
Executable stack	Killed	Killed
Executable shared library bss	Killed	Killed
Executable shared library data	Killed	Killed
Executable anonymous mapping (mprotect)	Vulnerable	Killed
Executable bss (mprotect)	Vulnerable	Killed
Executable data (mprotect)	Vulnerable	Killed
Executable heap (mprotect)	Vulnerable	Killed
Executable stack (mprotect)	Vulnerable	Killed
Executable shared library bss (mprotect)	Vulnerable	Killed
Executable shared library data (mprotect):	Vulnerable	Killed
Writable text segments	Vulnerable	Killed

The test was performed with paxtest-0.9.15:

```
PaXtest - Copyright(c) 2003-2016 by Peter Busser <peter@adamantix.org> and Brad Spengler <spender@grsecurity.net>
Released under the GNU Public Licence version 2 or later
```

```
Mode: 1
Blackhat
Kernel:
Linux syd 6.8.0-syd-13213-g70293240c5ce #9 SMP PREEMPT_DYNAMIC Mon Mar 25 04:40:47 CET 2024 x86_64 GNU/Linux
```

EXIT STATUS

On clean exit, **syd-mdwe** exits with the same code as the child process. On unclean termination, exit code is set to 128 plus signal number. In case executing the child process fails **syd-mdwe** exits with the *errno*(3) number.

CAVEATS

By default, *prctl*(2) error setting *PR_SET_MDWE*(2const) is not fatal. Use the **-m** option to make this error fatal. This utility does not work on MIPS architectures where Linux requires executable stack.

SEE ALSO

syd(1), *syd*(2), *syd*(5), *syd-lock*(1), *syd-ofd*(1), *syd-pds*(1), *syd-sec*(1), *seccomp*(2), *prctl*(2), *PR_SET_MDWE*(2const)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-net(1)

NAME

syd-net - Tool to aggregate IP networks

SYNOPSIS

syd-net [-h] <path>...

DESCRIPTION

Aggregates and outputs a list of IP networks compiled from the given list of files or standard input.

OPTIONS

-h Display help.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd(7)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-mem(1)

NAME

syd-mem - Calculate the memory usage of the given process or the parent process

SYNOPSIS

syd-mem [-sHV] [*pid*]

DESCRIPTION

The **syd-mem** utility calculates the memory usage of the given process or the parent process. Source of information is the per-process file *proc_pid_smmaps_rollup(5)* or *proc_pid_smmaps(5)* when *-s* option is given.

OPTIONS

-
- H** Print human-formatted size
 - V** Print virtual memory size
 - s** Use *proc_pid_smmaps(5)* rather than the rollup file.
This method is inefficient and is provided for benchmarking.
-

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *proc(5)*, *proc_pid_smmaps(5)*, *proc_pid_smmaps_rollup(5)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-oci(1)

NAME

syd-oci - OCI container runtime

SYNOPSIS

syd-oci [*OPTIONS*] [*COMMAND*]

DESCRIPTION

syd-oci is an OCI container runtime implementation for *syd*(1).

All common subcommands are supported: create, start, state, kill, delete, pause, resume, exec, run, list, ps, spec, events, features, update. Checkpoint/restore support is planned in the near future. See the respective youki issue, <https://github.com/youki-dev/youki/issues/142>, for more information.

INTEGRATION

syd-oci is a thin wrapper around *youki*(1) that integrates the *syd*(1) sandbox into containers. It is compatible with *docker*(1) and *podman*(1). To get syd-oci, you should build *syd*(1) with the **oci** feature. To use syd-oci with *docker*(1) you have two options: Either start *dockerd*(8) manually with the option **--add-runtime=syd-oci=/bin/syd-oci**, and do for example **docker run -it --runtime=syd-oci alpine** when starting containers, or add the following snippet to your */etc/docker/daemon.json* file:

```
{
  "runtimes": { "syd-oci": { "path": "/bin/syd-oci" } },
  "default-runtime": "syd-oci"
}
```

You may need to adapt the path to syd-oci depending on your installation. To use with *podman*(1) is similar, just pass **--runtime=/bin/syd-oci** as an option to **podman run**.

CONFIGURATION

The configuration directory of syd-oci is one of the following:

- For system-wide containers: **/etc/syd/oci**
- For rootless containers, one of the following: - **\${XDG_CONFIG_HOME}/syd/oci** where XDG_CONFIG_HOME is usually **~/.config**. - **\${HOME}/.syd/oci** if XDG_CONFIG_HOME is not set.

`syd-oci` attempts to configure the `syd(1)` sandbox in the following order, and parses the first file or profile it locates and stops processing, the environment variable **SYD_OCI_NO_CONFIG** may be set to skip to the final step:

- If `hostname` and `domainname` is defined for the container, try to load `${SYD_CONFIG_DIR}/${hostname}.${domainname}.syd-3`.
- If `domainname` is defined for the container, try to load `${SYD_CONFIG_DIR}/${domainname}.syd-3`.
- If `hostname` is defined for the container, try to load `${SYD_CONFIG_DIR}/${hostname}.syd-3`.
- Try to load `${SYD_CONFIG_DIR}/default.syd-3`
- Load the builtin **oci** profile. This profile is designed to be combined with *pandora(1)* and learning mode. See **`syd-cat -p oci`** for the list of rules.

`SYD_CONFIG_DIR` in the items above refer to the configuration directory. Refer to *syd(5)* for the syntax of *syd(1)* configuration files and *syd(2)* for a list of configuration items *syd(1)* understands. A *vim(1)* syntax highlighting file is also provided to easily edit *syd(1)* configuration files. Use **`syd-cat file.syd-3`** to check a *syd(1)* configuration file for syntax errors.

Finally, note that the **include** directives in the configuration files are searched within the container image. This allows you to provide additional image-based sandbox configuration. One possible use could be to store cryptographic checksums of all executables and their dependent dynamic libraries in an include file in the image and then use this with Force Sandboxing for binary verification, see *syd(7)* for more information on Force Sandboxing.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd(7)*, *pandora(1)*, *docker(1)*, *dockerd(8)*, *podman(1)*, *youki(1)*

- **syd** homepage: <https://sydbox.exherbo.org/>
- **youki** homepage: <https://containers.github.io/youki/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in `#sydbox` on Libera Chat or in `#sydbox:mailstation.de` on Matrix.

syd-ofd(1)

NAME

syd-ofd - Take a lock on a file, then execute into another program

SYNOPSIS

syd-ofd [-n | -N] [-t timeout] [-d fd] [-s=-r | -x=-w] file {command [arg...]}

DESCRIPTION

syd-ofd takes a lock on a file, then executes into another program. It is functionally identical to the *s6-setlock(1)* utility except it uses OFD locks which are new in POSIX 2024. This lock type is also known as "file-private locks" and is open file description-based rather than process based like the old-style POSIX locks that *s6-setlock(1)* uses. This allows *syd(1)* to pass the file descriptor to the sandbox process and close its own copy while the sandbox process still holding the lock which is not possible with old style locks and *s6-setlock(1)*.

OPTIONS

-h	Display help.
-n	Nonblocking lock. If syd-ofd cannot acquire the lock, it will exit 11 ("EAGAIN": Try again) immediately.
-N	Blocking lock. syd-ofd will wait until it can acquire the lock. This is the default.
-t timeout	Timed lock. If syd-ofd cannot acquire the lock after timeout milliseconds, it will exit 4 ("EINTR": Interrupted system call).
-s=-r	Shared lock. Other shared locks on the same file will not prevent the lock from being acquired (but an exclusive lock will). The -r option is retained for compatibility with the <i>s6-setlock(1)</i> utility.
-x=-w	Exclusive lock. This is the default. The -w option is retained for compatibility with the <i>s6-setlock(1)</i> utility.
-d fd	Make the lock visible in <i>program</i> on file descriptor <i>fd</i> .

EXIT STATUS

On clean exit, **syd-ofd** exits with the same code as the child process. On unclean termination, exit code is set to 128 plus signal number. In case executing the child process fails **syd-ofd** exits with the *errno(3)* number.

SECURITY

syd-ofd uses *openat*(2) with the resolve flags `RESOLVE_NO_MAGICLINKS` and `RESOLVE_NO_SYMLINKS` when opening or creating the lock file. Consequently, if any component of the specified path is a symbolic link, the call fails and returns error code 40 (ELOOP: "Too many symbolic links encountered"). In the same manner, the presence of any `..` (dot-dot) component in the lock-file path causes the call to fail and return error code 13 (EACCES: "Permission denied"). This restriction is intended to mitigate confused-deputy vulnerabilities during lock-file creation and related file operations by preventing an intermediary or less-privileged actor from redirecting the operation via symbolic links or parent-directory (`..`) components to an unintended filesystem location.

SEE ALSO

syd(1), *syd*(2), *syd*(5), *syd-lock*(1), *syd-mdwe*(1), *syd-pds*(1), *syd-sec*(1), *s6-setlock*(1), *fcntl*(2), *F_OFD_SETLKW*(2*const*), *openat2*(2)

- **syd** homepage: <https://sydbox.exherbo.org/>
- **s6-setlock** manpage: <https://skarnet.org/software/s6/s6-setlock.html>
- **POSIX 2024** *fcntl*(2) manpage: <https://pubs.opengroup.org/onlinepubs/9799919799/functions/fcntl.html>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-path(1)

NAME

syd-path - Write Force sandboxing rules for binaries and list executables under PATH

SYNOPSIS

syd-path [-h1235cCeklmpsw]

DESCRIPTION

Write Force sandboxing rules for binaries under PATH.

If at least one of the various **-e** options is specified, list executables with specified information under PATH.

OPTIONS

-h	Display help.
-c	Calculate CRC64 checksum (insecure).
-C	Calculate CRC32 checksum (insecure).
-m	Calculate MD5 checksum (insecure, portage/paludis vdb compat).
-1	Calculate SHA1 checksum (insecure).
-2	Calculate SHA3-256 checksum.
-3	Calculate SHA3-384 checksum.
-5	Calculate SHA3-512 checksum (default).
-k	Use action kill (default).
-w	Use action warn.
-l num	Limit by number of entries.
-p path	Specify alternative PATH.
-s	Prefix rules with <code>"/dev/syd/"</code> .
-e32	List 32-bit ELF executables under PATH (conflicts with -e64).
-e64	List 64-bit ELF executables under PATH (conflicts with -e32).
-ed	List dynamically linked ELF executables under PATH (conflicts with -es).
-es	List statically linked ELF executables under PATH (conflicts with -ed).
-ep	List PIE executables under PATH (conflicts with -eP).
-eP	List non-PIE executables under PATH (conflicts with -ep).
-ex	List scripts under PATH.
-eX	List binaries with executable stack under PATH.

BUGS

This tool will skip any failure silently.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-pause(1)

NAME

syd-pause - Block forever (until signaled), optionally ignoring selected signals

SYNOPSIS

syd-pause [*-t*] [*-h*] [*-a*] [*-q*] [*-b*] [*-i*] [*-p signals*]

DESCRIPTION

syd-pause is a tiny, long-lived process that simply waits until it is terminated by a signal. By default, it honors all standard termination signals; options allow you to ignore specific signals so the process continues running when they are delivered. This is useful as a minimal placeholder, supervisor target, or synchronization sentinel in service pipelines and sandboxes.

OPTIONS

--help	Display help.
-t	Ignore SIGTERM.
-h	Ignore SIGHUP.
-a	Ignore SIGALRM.
-q	Ignore SIGQUIT.
-b	Ignore SIGABRT.
-i	Ignore SIGINT.
-p signals	Ignore the comma-separated list of signal numbers given in <i>signals</i> (e.g. -p 1,2,3,15). Numbers must be valid per <i>signal(7)</i> . This flag can be combined with the short flags above.

EXIT STATUS

On normal termination by a signal, **syd-pause** exits 0. On errors **syd-pause** exits with the corresponding *errno(3)* value.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *syd-ofd(1)*, *s6-pause(1)*, *pause(2)*

- **syd** homepage: <https://sydbox.exherbo.org/>
- **s6-pause** manpage: <https://skarnet.org/software/s6/s6-pause.html>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-pds(1)

NAME

syd-pds - Run a command with parent death signal set

SYNOPSIS

syd-pds [-h] [-s *signal*] {*command* [*args...*]}

DESCRIPTION

The *syd-pds*(1) utility runs a command with the parent death signal set. When the parent process dies, the specified signal will be delivered to the command. The signal defaults to SIGKILL.

OPTIONS

-h	Display help.
-s <i>signal</i>	Set parent death signal to the specified signal. Defaults to SIGKILL.

EXIT STATUS

syd-pds exits with the same code as the child process.

SEE ALSO

syd(1), *syd*(2), *syd*(5), *syd-lock*(1), *syd-mdwe*(1), *syd-ofd*(1), *syd-sec*(1), *PR_SET_PDEATHSIG*(2)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-poc(1)

NAME

syd-poc - POC||GTFO! Demonstrate various sandbox break vectors.

SYNOPSIS

syd-poc [-h] [command] [args...]

DESCRIPTION

syd-poc is a simple utility to demonstrate proof of concepts for various sandbox break vectors. Use this tool to break the chains of your imagination and find new, novel ways to break out of sandboxen and keep us posted to spread the fun!

OPTIONS

-h Display help.

BUGS

User must ensure the benign path is at least as long as the target path or there's a risk out-of-bounds write typically followed by a segmentation fault.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-pty(1)

NAME

syd-pty - PTY to STDIO bidirectional forwarder

SYNOPSIS

syd-pty [-dh] [-x <x-size>] [-y <y-size>] -p <pid-fd> -i <pty-fd>

DESCRIPTION

Forwards data between the given *pty*(7) main file descriptor, and *stdio*(3).

PID file descriptor is used to track the exit of Syd process.

OPTIONS

-h	Display help.
-d	Run in debug mode without confinement.
-p pid-fd	PID file descriptor of Syd process.
-i pty-fd	PTY main file descriptor.
-x x-size	Specify window row size (default: inherit).
-y y-size	Specify window column size (default: inherit).

USAGE

syd-pty(1) is not meant to be used as a standalone tool. Syd invokes *syd-pty*(1) at startup when PTY sandboxing is set to on with **sandbox/pty:on**. See the PTY SANDBOXING section in *syd*(7) manual page for more information. If you want to use *syd-pty*(1) in your own project, what you need to do first is to pass it a *non-blocking* PID fd of your own process with the **-p <pid-fd>** argument so *syd-pty*(1) can simultaneously exit with it. Care should be given at this stage as PID file descriptors are **O_CLOEXEC** by default. Next create a new pseudoterminal with *posix_openpt*(3) or *openpty*(3) and pass the main end of the file descriptor pair to *syd-pty*(1) with the **-i <pty-fd>** argument.

IMPLEMENTATION

syd-pty(1) is designed with performance, security, and privacy in mind, utilizing advanced techniques such as edge-triggered *epoll(7)* for efficient event notification and full asynchronous operations to handle pseudoterminal activities without blocking. It employs zero-copy data transfer using the *splice(2)* system call to move data directly between file descriptors within the kernel, ensuring high performance and data privacy. To facilitate bidirectional communication, *syd-pty(1)* uses two pairs of pipes, allowing seamless data flow between *pty(7)* and *stdio(3)*. Additionally, *syd-pty(1)* confines its execution environment using seccomp and Landlock, restricting system calls and file accesses to minimize the attack surface.

SECURITY

syd-pty(1) implements comprehensive security measures to mitigate risks associated with running outside the Syd sandbox, thus preventing potential Meddler-in-the-Middle (MITM) attacks against containers. Seccomp filters are meticulously configured to allow only necessary syscalls. Moreover, executable memory is disallowed to prevent code injection attacks. Landlock and namespaces, if available, enforce further restrictions by disallowing all filesystem and network access, providing an additional layer of security. Additionally, the main PTY file descriptor is placed into exclusive mode via the **TIOCEXCL** *ioctl(2)* request. This prevents any further opens of the secondary PTY device (save for processes with the **CAP_SYS_ADMIN** capability), thereby reducing the attack surface for unauthorized eavesdropping or input injection at the device layer as part of a defense-in-depth strategy. These combined techniques ensure that even if *syd-pty(1)* is compromised, the scope of malicious actions is significantly limited, maintaining the integrity and security of the overall system.

ENVIRONMENT

SYD_PTY_DEBUG	Run in debug mode without confinement, equivalent to the -d option
SYD_PTY_RULES	Print seccomp rules in human-readable format to standard error at startup

BUGS

splice(2) support for ttys was removed in commit 36e2c7421f02a22f71c9283e55fdb672a9eb58e7 (merged for Linux 6.5) and later restored in commit 9bb48c82aced07698a2d08ee0f1475a6c4f6b266 (merged for Linux 6.6). When running under a problematic Linux kernel *syd-pty(1)* will exit with 22, aka **EINVAL** or **Invalid argument**. See the following links for more information:

- <https://git.kernel.org/linus/36e2c7421f02a22f71c9283e55fdb672a9eb58e7>
- <https://git.kernel.org/linus/9bb48c82aced07698a2d08ee0f1475a6c4f6b266>

SEE ALSO

syd(1), *syd(2)*, *syd(7)*, *stdio(3)*, *pty(7)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbbox/sydbbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbbox/-/issues>. Discuss in #sydbbox on Libera Chat or in #sydbbox:mailstation.de on Matrix.

syd-read(1)

NAME

syd-read - Print resolved symbolic links or canonical file names

SYNOPSIS

syd-read [-hmnzBDFMNPRUX] [-c n] [-d dir] [-p pid] path...

DESCRIPTION

The **syd-read** utility prints resolved symbolic links or canonical file names. By default last component may exist, other components must exist.

OPTIONS

-h	Display help.
-c n	Cycle through the path list n times, useful for benchmarking.
-d dir	Resolve relative to the given directory.
-p pid	Resolve from the perspective of the given process ID.
-m	All components of the paths must exist, conflicts with -M .
-M	Last component must not exist, other components must exist, conflicts with -m .
-B	Resolve beneath the given directory, useful with -d dir . Implies -P , conflicts with -R .
-R	Treat the given directory as root directory, useful with -d dir . Implies -P , conflicts with -B .
-D	Do not traverse through ”.” components.
-X	Do not traverse through mount points.
-F	Do not follow symbolic links for any of the path components.
-N	Do not follow symbolic links for the last path component.
-P	Do not resolve /proc magic symbolic links.
-U	Resolve unsafe /proc magic symbolic links.
-n	Do not output the trailing delimiter.
-z	End each output line with NUL not newline.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *readlink(1)*, *realpath(3)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbbox/sydbbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbbox/-/issues>. Discuss in #sydbbox on Libera Chat or in #sydbbox:mailstation.de on Matrix.

syd-rnd(1)

NAME

syd-rnd - Print AT_RANDOM bytes in various formats

SYNOPSIS

syd-rnd [*-hinr*]

DESCRIPTION

Given no arguments, print AT_RANDOM bytes in lower hexadecimal format.

Given **-r**, print raw bytes.

Given **-i**, print an unsigned 64-bit integer.

Given **-n**, print a human-readable name.

OPTIONS

-
- h** Display help.
 - r** Print raw bytes.
 - i** Print an unsigned 64-bit integer.
 - n** Print a human-readable name.
-

SEE ALSO

syd(1), *syd(2)*, *syd(5)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-run(1)

NAME

syd-run - Run a program inside a container with the given process ID

SYNOPSIS

syd-run [-hvacimnptuU] pid {command [arg...]}

DESCRIPTION

The *syd-run(2)* utility runs a program inside a *syd(1)* container with the given process ID. This requires the use of system calls *setns(2)*, and *pidfd_open(2)* which require Linux-5.8 or newer. Note, entering *pid_namespaces(7)* and *time_namespaces(7)* is a privileged operation, whereas entering *cgroup_namespaces(7)*, *ipc_namespaces(7)*, *mount_namespaces(7)*, *network_namespaces(7)*, and *uts_namespaces(7)* is unprivileged when combined with *user_namespaces(7)* provided that unprivileged *user_namespaces(7)* support is enabled in the Linux kernel.

OPTIONS

-
- h** Display help.
 - v** Be verbose. Print informational messages on standard error.
 - a** Auto-detect namespaces to enter. PID and Time namespaces are excluded. This is the default.
 - c** Enter into CGroup namespace.
 - i** Enter into IPC namespace.
 - m** Enter into mount namespace.
 - n** Enter into network namespace.
 - p** Enter into PID namespace.
 - t** Enter into time namespace.
 - u** Enter into UTS namespace.
 - U** Enter into user namespace.
-

EXIT STATUS

On clean exit, *syd-run(1)* exits with the same code as the child process. On unclean termination, exit code is set to 128 plus signal number. In case executing the child process fails *syd-run(1)* exits with the *errno(3)* number.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *setns(2)*, *pidfd_open(2)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-sec(1)

NAME

syd-sec - Print secure bits or run command with secure bits set

SYNOPSIS

syd-sec [-*ahikprsxAIKPRSX*] {*command* [*args...*]}

DESCRIPTION

Given no arguments, print information on process secure bits in compact JSON.

Given no command, one or more of the secure bit options [-*aikprsx*] may be given to test for secure bits. Use capital letter options, [-*AIKPRSX*], to test for locked versions of respective secure bits.

Given a command and arguments, with at least one of the secure bit options [-*aikprsxAIKRSX*] set the specified securebits, execute the command and exit with the same status.

OPTIONS

-h Display help and exit.

-p, -P Set/test process no_new_privs attribute.

-r, -R Set/test secure bit SECBIT_NOROOT.

-s, -S Set/test secure bit SECBIT_NO_SETUID_FIXUP.

-k, -K Set/test secure bit SECBIT_KEEP_CAPS.

-a, -A Set/test secure bit SECBIT_NO_CAP_AMBIENT_RAISE.

-x, -X Set/test secure bit SECBIT_EXEC_RESTRICT_FILE.

-i, -I Set/test secure bit `SECBIT_EXEC_DENY_INTERACTIVE`.

SECURE BITS

Securebit	Description	CAP_SETPCAP required?
<code>NO_NEW_PRIVS</code>	When set, <i>execve</i> (2) will not grant new privileges (e.g., set-user-ID/set-group-ID mode bits and file capabilities are ignored). Inherited across <i>fork</i> (2), <i>clone</i> (2), and <i>execve</i> (2); once set, cannot be unset.	No
<code>NOROOT</code>	Disable special handling of UID 0 for gaining capabilities on <i>exec</i> /setuid. <code>NOROOT_LOCKED</code> is lock for <code>NOROOT</code> (prevents further changes; irreversible).	Yes
<code>NO_SETUID_FIXUP</code>	Stop kernel adjustments to permitted/effective/ambient capability sets when effective/filesystem UIDs toggle between 0 and nonzero. <code>NO_SETUID_FIXUP_LOCKED</code> is lock for <code>NO_SETUID_FIXUP</code> (prevents further changes; irreversible).	Yes
<code>KEEP_CAPS</code>	Allow retaining permitted capabilities when switching all UIDs from 0 to nonzero; always cleared on <i>execve</i> (2). <code>KEEP_CAPS_LOCKED</code> is lock for <code>KEEP_CAPS</code> (prevents further changes; irreversible).	Yes
<code>NO_CAP_AMBIENT_RAISE</code>	Disallow raising ambient capabilities via <i>prctl</i> (<i>PR_CAP_AMBIENT_RAISE</i>). <code>NO_CAP_AMBIENT_RAISE_LOCKED</code> is lock for <code>NO_CAP_AMBIENT_RAISE</code> (prevents further changes; irreversible).	Yes
<code>EXEC_RESTRIC_FILE</code>	Interpreter/dynamic linker should execute a file only if <i>execveat</i> (2) with <code>AT_EXECVE_CHECK</code> on the related file descriptor succeeds. <code>EXEC_RESTRIC_FILE_LOCKED</code> is lock for <code>EXEC_RESTRIC_FILE</code> (prevents further changes; irreversible).	No
<code>EXEC_DENY_INTERACTIVE</code>	Interpreter should not accept interactive user commands; content via a file descriptor is allowed only if <i>execveat</i> (2) with <code>AT_EXECVE_CHECK</code> succeeds. <code>EXEC_DENY_INTERACTIVE_LOCKED</code> is lock for <code>EXEC_DENY_INTERACTIVE</code> (prevents further changes; irreversible).	No

EXIT STATUS

When querying secure bits, **syd-sec** exits with success if all the specified secure bits are set in process secure bits. When running a command, **syd-sec** exits with the same code as the child process. If *PR_SET_SECUREBITS*(2const) *prctl*(2) operation fails prior to command execution, **syd-sec** exits with *errno*(3).

SEE ALSO

syd(1), *syd*(2), *syd*(5), *syd-lock*(1), *syd-mdwe*(1), *syd-ofd*(1), *syd-pds*(1), *PR_GET_SECUREBITS*(2const), *PR_SET_SECUREBITS*(2const)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-sh(1)

NAME

syd-sh - Simple confined shell based on *wordexp*(3)

SYNOPSIS

syd-sh [-helsx] [--] [*command_file* [argument...]]

syd-sh [-helx] -c *command_string* [*command_name* [argument...]]

DESCRIPTION

syd-sh is a simple confined shell based on *wordexp*(3). Each command is executed in its own confined environment. Confinement is done using Landlock, namespaces and seccomp. Command timeout is 3 seconds.

OPTIONS

-
- h** Display help.
 - c** Read commands from the given *command_string* operand. No commands are read from standard input.
 - e** If not interactive, exit immediately if any untested command fails.
 - l** Ignored, login shell compatibility
 - s** Read commands from the standard input. If no operands and the **-c** is not specified, the **-s** option is assumed.
 - x** Write each command to standard error (preceded by a ”+ ”) before it is executed. Useful for debugging.
-

SEE ALSO

syd(1), *syd*(2), *syd*(5), *wordexp*(3)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-sha(1)

NAME

syd-sha - Calculate SHA3-512 checksum of the given file or standard input

SYNOPSIS

syd-sha [-bcChmx1235] <file|->

DESCRIPTION

Given a file, calculate the checksum of the file.

Given no positional arguments or "-" as argument, calculate checksum of standard input.

Use **-b** to print binary output rather than hex-encoded string.

OPTIONS

-
- h** Display help.
 - b** Binary output.
 - x** Hexadecimal output (default).
 - c** Calculate CRC64 checksum (insecure).
 - C** Calculate CRC32 checksum (insecure).
 - m** Calculate MD5 checksum (insecure, portage/paludis vdb compat).
 - 1** Calculate SHA1 checksum (insecure).
 - 2** Calculate SHA3-256 checksum.
 - 3** Calculate SHA3-384 checksum.
 - 5** Calculate SHA3-512 checksum (default).
-

SEE ALSO

syd(1), *syd(2)*, *syd(5)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-size(1)

NAME

syd-size - Print and parse human-formatted sizes

SYNOPSIS

syd-size *size|human-size*

DESCRIPTION

Given a number, **syd-size** prints human-formatted size.

Given a string, **syd-size** parses human-formatted size into bytes and prints it.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*

syd homepage: <https://sydbox.exherbo.org/>

parse-size documentation: https://docs.rs/parse-size/latest/parse_size/

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-stat(1)

NAME

syd-stat - Print detailed information about a process in JSON format

SYNOPSIS

syd-stat [*pid*]

DESCRIPTION

syd-stat utility prints detailed information about a process with the given process ID. The information is printed in line-oriented **JSON** format and may be further mangled with tools such as *jq*(1). Source of information are the per-process files *proc_pid_stat*(5) and */proc/pid/status*(5).

SEE ALSO

syd(1), *syd*(2), *syd*(5), *proc_pid_stat*(5), *proc_pid_status*(5), *jq*(1)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-sys(1)

NAME

syd-sys - Lookup syscalls, errno, ioctl, open flags, and signals by number or regular expression

SYNOPSIS

syd-sys [-heios] [-a arch] number|regex

syd-sys -a list

syd-sys [-uU]

DESCRIPTION

Given a number, **syd-sys** prints the matching syscall name.

Given a regex, **syd-sys** prints case-insensitively matching syscall names.

Use **-e** to query *errno*(3) numbers.

Use **-i** to query *ioctl*(2) requests.

Use **-o** to query *open*(2) flags.

Use **-s** to query *signal*(7) numbers.

Use **-u** to list UNIX domain socket inodes using *netlink*(7).

Use **-U** to list UNIX domain socket inodes using *proc_net*(5).

OPTIONS

-
- h** Display help.
 - a** Specify alternative architecture, such as **x86**, **x86_64** and **aarch64**.
Use **list** to print the list of libseccomp supported architectures.
 - e** Query *errno*(3) numbers
 - i** Query *ioctl*(2) requests
 - o** Query *open*(2) flags
 - s** Query *signal*(7) numbers
 - u** List UNIX domain socket inodes using *netlink*(7)
 - U** List UNIX domain socket inodes using *proc_net*(5)
-

SEE ALSO

syd(1), *syd(2)*, *syd(5)*, *errno(3)*, *ioctl(2)*, *open(2)*, *syscall(2)*, *signal(7)*, *netlink(7)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-test(1)

NAME

syd-test - Run syd integration tests

SYNOPSIS

syd-test [*<regex>*|*<number>*|*<number>*..*<number>*]..

DESCRIPTION

The **syd-test** utility may be used to run syd integration tests.

Requires **syd-test-do** utility to be in **PATH**.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-tck(1)

NAME

syd-tck - Measure runtime in Hardware Ticks

SYNOPSIS

syd-tck *{command [args...]}*

DESCRIPTION

syd-tck utility runs the given command with optional arguments and measures its runtime in hardware ticks, prints information about it and exits with the same exit code as the program or 128 plus the signal value if the command was terminated.

OUTPUT

```
true      code:0 total:0.00s td:285 freq:2494511485Hz prec:0.40ns pid:1 tc:322191649243995
```

Sample output looks like this on x86-64, below are explanations of each field:

1. The name of the command
2. Exit code
3. Total runtime in seconds
4. Total runtime in ticks (tick duration)
5. Frequency in Hertz
6. Precision in nanoseconds
7. Processor ID
8. Tick counter

PORTABILITY

syd-tck only works on architectures aarch64 and x86-64.

SEE ALSO

syd(1), *syd(2)*, *syd(5)*

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-tor(1)

NAME

syd-tor - SOCKS Proxy Forwarder

SYNOPSIS

syd-tor [-dh] -p <pid-fd> -i <socket-fd> [-o addr:port] [-u unix-sock]

DESCRIPTION

Receives listening socket from fd and forwards traffic to addr:port or UNIX socket.

External address must either be an IPv4, or an IPv6 address or path to a UNIX domain socket, defaults to **127.0.0.1:9050**.

PID file descriptor is used to track the exit of Syd process.

OPTIONS

-h	Display help.
-d	Run in debug mode without confinement.
-p pid-fd	PID file descriptor of Syd process.
-i socket-fd	Socket file descriptor to receive the listening socket from.
-o ext-addr	Specify external address to forward traffic to. Address may be an IPv4/IPv6 address in format "addr:port". Defaults to "127.0.0.1:9050".
-u unix-sock	Specify UNIX domain socket to forward traffic to. This option has precedence over -o .

USAGE

syd-tor(1) is not meant to be used as a standalone tool. Syd invokes *syd-tor*(1) at startup when Proxy sandboxing is set to on with "sandbox/proxy:on". See the PROXY SANDBOXING section in *syd*(7) manual page for more information. If you want to use *syd-tor*(1) in your own project, what you need to do first is to pass it a *non-blocking* PID fd of your own process with the "-p <pid-fd>" argument so *syd-tor*(1) can simultaneously exit with it. Care should be given at this stage as PID file descriptors are "O_CLOEXEC" by default. Next, create a UNIX socket-pair, enter a network namespace, bring up the loopback device, *bind*(2) a socket to a port on it and then send this socket file descriptor through the write end of the socket-pair with *sendmsg*(2) "SCM_RIGHTS" option. Finally pass the read end of the socket-pair to *syd-tor*(1) with the "-i <socket-fd>" argument.

IMPLEMENTATION

syd-tor(1) is designed with performance, security, and privacy in mind, utilizing advanced techniques such as edge-triggered *epoll*(7) for efficient event notification and full asynchronous operations to handle multiple connections without blocking. It employs zero-copy data transfer using the *splice*(2) system call to move data directly between file descriptors within the kernel, ensuring high performance and data privacy. To facilitate bidirectional communication, *syd-tor*(1) uses two pairs of pipes, allowing seamless data flow between the client and the external address. Additionally, *syd-tor*(1) confines its execution environment using *seccomp*(2) and *landlock*(7), restricting system calls and file accesses to minimize the attack surface. Compared to *socks*, which uses the Tokio runtime for asynchronous I/O, *syd-tor*(1) emphasizes kernel-level efficiency and security, making it a robust solution for SOCKS proxy forwarding.

SECURITY

syd-tor(1) implements comprehensive security measures to mitigate risks associated with running outside the Syd sandbox, thus preventing potential Meddler-in-the-Middle (MITM) attacks against containers. *seccomp*(2) filters are carefully configured to allow only the required syscalls. The *socket*(2) syscall is limited to a single domain, type, and protocol, while *connect*(2) is restricted to a single memory address, preventing unauthorized network connections. *sigaction*(2), and *rt_sigaction*(2) system calls are not permitted to install new signal handlers. *pipe*(2), *socket*(2), *connect*(2), *accept*(2), and *shutdown*(2) system calls are protected by syscall argument cookies determined randomly using *getrandom*(2) with GRND_RANDOM at startup. Refer to the **Syscall Argument Cookies** section of the *syd*(7) manual page for more information on argument cookies. These mitigations are most effective on 64-bit architectures, but on 32-bit systems the *socketcall*(2) interface may be exploited. Therefore, additional protective measures are implemented. On Linux 6.10 and later, the memory area containing the external network address is sealed using *mseal*(2) and surrounding memory is protected with guard pages to prevent overflow or adjacent corruption. Executable memory is also disallowed to prevent code injection attacks. If available, *landlock*(7) and *namespaces*(7) impose further restrictions by disallowing all filesystem access, thereby providing an extra layer of security. Together, these techniques ensure that even if *syd-tor*(1) is compromised, the scope of malicious actions is significantly limited, maintaining the overall integrity and security of the system.

ENVIRONMENT

SYD_TOR_DEBUG	Run in debug mode without confinement, equivalent to the "-d" option
SYD_TOR_RULES	Print seccomp rules in human-readable format to standard error at startup

CAVEATS

The *syd-tor*(1) process runs as a single process and can potentially hit file descriptor (FD) limits due to the number of FDs it opens per connection. Each client connection involves six FDs: one for the client socket, one for the external socket, and four for the pipes used for bidirectional data transfer (two pipes with an input and output FD each). To mitigate this, *syd-tor*(1) sets the file-max limit to the hard limit by overriding the soft limit at startup. However, in some cases, this may not be sufficient, and manual adjustment of FD limits may be necessary.

SEE ALSO

syd(1), *syd*(2), *syd*(7)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbbox/sydbbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbbox/-/issues>. Discuss in #sydbbox on Libera Chat or in #sydbbox:mailstation.de on Matrix.

syd-tty(1)

NAME

syd-tty - Print the controlling terminal of the given process

SYNOPSIS

syd-tty [*pid*]

DESCRIPTION

The **syd-tty** utility prints the controlling terminal of the given process. It is similar to the *tty*(1) utility except it allows printing the controlling terminal of an arbitrary process.

SEE ALSO

syd(1), *syd*(2), *syd*(5), *tty*(1), *ttyname*(3)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-utc(1)

NAME

syd-utc - Print UTC date and time in JSON format

SYNOPSIS

syd-utc

DESCRIPTION

syd-utc utility prints UTC date and time. The information is printed in line-oriented **JSON** format and may be further mangled with tools such as *jq*(1). Source of information is the Realtime Clock using *clock_gettime*(2).

SEE ALSO

syd(1), *syd*(2), *syd*(5), *clock_gettime*(2), *jq*(1)

syd homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-uts(1)

NAME

syd-uts - Print name and information about the current kernel in JSON format

SYNOPSIS

syd-uts [*-hdmnrsv*]

DESCRIPTION

syd-uts utility prints name and information about the current kernel. The information is printed in line-oriented **JSON** format and may be further mangled with tools such as *jq*(1). Source of information is the *uname*(2) system call. The options **-d**, **-m**, **-n**, **-r**, **-s**, and **-v** may be used to print individual items of the **utsname** structure verbatim. If many options are given at a time the items are printed as a dot-separated list on a single line.

OPTIONS

-
- | | |
|-----------|--|
| -h | Display help. |
| -s | Print name of the operating system implementation. |
| -n | Print network name of this machine. |
| -r | Print release level of the operating system. |
| -v | Print version level of the operating system. |
| -m | Print machine hardware platform. |
| -d | Print NIS or YP domain name of this machine. |
-

SEE ALSO

syd(1), *syd*(2), *syd*(5), *uname*(2), *jq*(1)

- **syd** homepage: <https://sydbox.exherbo.org/>

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbox/sydbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbox/-/issues>. Discuss in #sydbox on Libera Chat or in #sydbox:mailstation.de on Matrix.

syd-x(1)

NAME

syd-x - Check executability of files, list executables of processes

SYNOPSIS

syd-x [-hcv] [-l pid...] [files...]

DESCRIPTION

syd-x checks executability of the given files or lists executables of the given processes. The file executability check uses the *execveat*(2) system call with the flag "AT_EXECVE_CHECK" on Linux>=6.14 and falls back to the *faccessat*(2) system call with the flag "X_OK" on older Linux. If the *-l* option is specified, list mode is activated: all following arguments are interpreted as process IDs, not files. In list mode, executable file listing is done using the "PROCMAP_QUERY" *ioctl*(2) request on Linux>=6.11 and falls back to parsing the *proc_pid_maps*(5) file textually on older Linux.

OPTIONS

-h	Display help.
-c	Exit with success if the "AT_EXECVE_CHECK" flag is supported.
-l pid	Specify a process ID to list the executables of, may be repeated.
-v	Print file check status information on standard error.

EXIT STATUS

syd-x exits with 0 on success or with the errno number on failure.

SEE ALSO

syd(1), *syd*(2), *syd*(5)

- **syd** homepage: <https://sydbox.exherbo.org/>
- Executability check: https://docs.kernel.org/next/userspace-api/check_exec.html

AUTHORS

Maintained by Ali Polatel. Up-to-date sources can be found at <https://gitlab.exherbo.org/sydbbox/sydbbox.git> and bugs/patches can be submitted to <https://gitlab.exherbo.org/groups/sydbbox/-/issues>. Discuss in #sydbbox on Libera Chat or in #sydbbox:mailstation.de on Matrix.