SECCOMP your PostgreSQL

Joe Conway
joe@crunchydata.com
mail@joeconway.com
@josepheconway

Crunchy Data
2020-0202
Holistic Security

- Allow authorized access to your data
- Prevent unauthorized access
- Defense in Depth - many layers
  - Hardened Shell - perimeter security
  - Crunchy Core - in database security
  - Confinement - reduce attack surface ← This talk...
  - Instrumented - monitoring and alerting
Agenda

- SECCOMP background
- systemd SECCOMP support
- pgseccomp
**SECCOMP**

- SECCOMP ("SECure COMPuting with filters")
- Linux kernel syscall filtering mechanism
- Allows reduction of the kernel attack surface
- Prevents (or at least audit logs) normally unused syscalls

"A large number of system calls are exposed to every userland process with many of them going unused for the entire lifetime of the process. As system calls change and mature, bugs are found and eradicated. A certain subset of userland applications benefit by having a reduced set of available system calls. The resulting set reduces the total kernel surface exposed to the application. System call filtering is meant for use with those applications."

https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt
SECCOMP Strict Mode

- Original version of seccomp merged in 2005 into Linux 2.6.12
  - Process makes one-way transition into secure state
  - Enabled via the /proc, and later prctl syscall
  - Allowed syscalls: exit(), sigreturn(), read() and write() to already-open file descriptors
  - Otherwise terminate the process with SIGKILL
SECCOMP Filter Mode

- Second version added to Linux 3.5 in 2012
  - Also called seccomp-bpf
  - Uses BPF (but not eBPF)
  - Can restrict arbitrary list of specific syscalls
  - Process must have the CAP_SYS_ADMIN or must have no_new_privs bit set
  - Actions (simplified), in decreasing order of precedence:
    - SCMP_ACT_KILL - terminate as though killed by uncatchable SIGSYS
    - SCMP_ACT_ERRNO - return specified error number to caller
    - SCMP_ACT_LOG (since Linux Kernel 4.14) - allow, but log, syscall
    - SCMP_ACT_ALLOW - allow syscall

- libseccomp simplifies use of seccomp-bpf
Checking SECCOMP Support

```
grep SECCOMP /boot/config-$(uname -r)
CONFIG_SECCOMP=y
CONFIG_HAVE_ARCH_SECCOMP_FILTER=y
CONFIG_SECCOMP_FILTER=y
```
No New Privs

- Set `no_new_privs`: `prctl(PR_SET_NO_NEW_PRIVS, 1)`
  - Once set, inherited across `fork`, `clone`, and `execve`
  - Cannot be unset
  - In particular, seccomp filters persist
  - Unprivileged users only allowed seccomp filters if `no_new_privs` is set
libseccomp Basic Use

- Init seccomp with default action
- Add specific rules for syscalls of interest
- Load the filter
Behavior

- Once filter is loaded, cannot relax restrictions
- Process may load multiple filters
- All child processes inherit active filters
- Child processes can load own filters
- Highest precedence action taken per syscall
Example

```c
#include <seccomp.h>
#include <stdio.h>

int main(int argc, char *argv[]){
    int rc, i;
    scmp_filter_ctx ctx;
    for (i = 0; i < 3; i++){
        uint32_t write_action = i % 2 ? SCMP_ACT_LOG : SCMP_ACT_ALLOW;
        ctx = seccomp_init(SCMP_ACT_LOG);
        rc = seccomp_rule_add(ctx, write_action, SCMP_SYS(write), 0);
        rc = seccomp_load(ctx);
        printf("filter \%d loaded\n", i + 1);
        seccomp_release(ctx);
    }
    return 0;
}
```
Example

gcc -ggdb3 -O0 example.c -lseccomp
echo -n > /var/log/audit/audit.log && ./a.out

grep SECCOMP /var/log/audit/audit.log|grep -o -P 'sig=.syscall=\d{1,3}'
sig=0 arch=c000003e syscall=5
sig=0 arch=c000003e syscall=157
sig=0 arch=c000003e syscall=317
sig=0 arch=c000003e syscall=1
sig=0 arch=c000003e syscall=157
sig=0 arch=c000003e syscall=317
sig=0 arch=c000003e syscall=1
sig=0 arch=c000003e syscall=231

./get_syscalls.sh  #helper script from pgseccomp
exit_group,fstat,prctl,seccomp,write
Analysis

- **First loop**
  - Nothing is logged before the first filter is loaded
  - Filter 1 allows `write` and defaults other syscalls to log
  - `fstat (#5) called first time printf is called`
  - `printf` requires `write (#1)`, not logged due to allow rule

- **Second loop**
  - Filter 2 adds a log rule for `write`
  - `seccomp filter load requires prctl (#157) and seccomp (#317)`
  - `fstat` no longer called by `printf`

- **Third loop**
  - `seccomp filter load requires prctl (#157) and seccomp (#317)`
  - Filter 3 adds a allow rule for `write`
  - But rule is ineffective and the `write` call is still logged

- Process exit requires `exit_group (#231)`
Example

```c
#include <seccomp.h>
#include <stdio.h>
int main(int argc, char *argv[]){
    int rc, i;
    scmp_filter_ctx ctx;
    for (i = 0; i < 3; i++){
        uint32_t write_action = i % 2 ? SCMP_ACT_LOG : SCMP_ACT_ALLOW;
        ctx = seccomp_init(SCMP_ACT_LOG);
        rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(seccomp), 0);
        rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(prctl), 0);
        rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(exit_group), 0);
        rc = seccomp_rule_add(ctx, SCMP_ACT_ALLOW, SCMP_SYS(fstat), 0);
        rc = seccomp_rule_add(ctx, write_action, SCMP_SYS(write), 0);
        rc = seccomp_load(ctx);
        printf("filter %d loaded\n", i + 1);
        seccomp_release(ctx);
    }
}
```
Example

gcc -ggdb3 -O0 example.c -lseccomp
echo -n > /var/log/audit/audit.log && ./a.out

grep SECCOMP /var/log/audit/audit.log|grep -o -P 'sig=.*syscall=\d{1,3}'
sig=0 arch=c000003e syscall=1
sig=0 arch=c000003e syscall=1

./get_syscalls.sh
write
Analysis

- `printf requires write (#1)`
- Once restricted (filter 2 `write_action = SCMP_ACT_LOG`) cannot relax (filter 3 `write_action = SCMP_ACT_ALLOW`)
- Consequently `write` continues to be logged
Overview

- systemd supports SECCOMP filtering via config options
- Advantages:
  - Sysadmin control over production services
  - Possibly more difficult to subvert than service configured option
- Disadvantages:
  - Requires coordination/cooperation between service admin and sysadmin
  - Requires extra allowed syscalls
  - Limited SECCOMP configuration flexibility
  - Lack of SCMP_ACT_LOG option
SystemCallFilter - Whitelist

- Space delimited list of allowed syscalls (*SCMP_ACT_ALLOW* whitelist)
- Default action *SCMP_ACT_KILL*
- Immediate process termination with the SIGSYS signal
- Can override with *SystemCallErrorNumber*: specifies action *SCMP_ACT_ERRNO*
- May be specified more than once: filters merged
SystemCallFilter - Blacklist

- If first character of the list is ~ (tilde) effect is inverted (blacklist)
- Blacklist elements may have suffix :<errno>
- <errno> is 0 - 4095 or errno name, e.g. EACCES
- Causes SCMP_ACT_ERRNO action rather than SCMP_ACT_KILL per element
- Whitelisting is recommended
SystemCallFilter - Predefined Sets

- Predefined sets of syscalls are provided
- Starts with "@" character followed by name
- E.g.: @system-service, @file-system, ...
- Set contents may change between systemd versions
- List of syscalls depends on kernel version and arch for which systemd was compiled
- New kernel syscalls might be added to these sets
- Use `systemd-analyze syscall-filter` to enumerate filter sets
SystemCallErrorNumber

- Overrides default action
- Causes SCMP_ACT_ERRNO action rather than SCMP_ACT_KILL as default
- Disadvantage: SCMP_ACT_ERRNO actions are not audit logged by default
SystemCallArchitectures

- List of architecture identifiers to include in the system call filter
- Recommended to limit permitted syscall arch, so secondary ABIs may not circumvent restrictions
- Probably want native
NoNewPrivileges

- Ensures process and children can never gain new privileges through `execve()`
- systemd documentation claims overridden by `SystemCallFilter`
- Empirical evidence suggests otherwise (addressed later)
PostgreSQL Implementation

- Whitelist derivation painful
  - No audit log entries for SCMP_ACT_ERRNO actions
  - Cannot use SCMP_ACT_LOG default action
- Therefore started with pgseccomp list (derivation covered later)
- Added additional syscalls via trial and error
- Determined NoNewPrivileges needed to be set
Before

LOI="(Seccomp|NoNewPrivs|Speculation_Store_Bypass)"
for pid in $(ps -fu postgres|tail -n+2|tr -s " ":|cut -d " " -f2)
do
echo "${pid} - $(cat /proc/${pid}/status |grep -E ${LOI})"
done;
17811 - NoNewPrivs: 0
Seccomp: 0
Speculation_Store_Bypass: thread vulnerable

[...]

17850 - NoNewPrivs: 0
Seccomp: 0
Speculation_Store_Bypass: thread vulnerable
Edit postgresql.service

# 94 total syscalls
SystemCallFilter=accept access arch_prctl ...
<br>...<long list>...
<br>... wait4 write

# Uncomment to make error return the default instead of SIGSYS
#SystemCallErrorNumber=EACCES

SystemCallArchitectures=native

# Note commented out for now
#NoNewPrivileges=yes
Restart PostgreSQL

# after editing service file, reload daemon
systemctl daemon-reload

# not for production use - clear out audit log
echo -n > /var/log/audit/audit.log

# start/restart postgresql service
systemctl restart postgresql@12-main.service
LOI="(Seccomp|NoNewPrivs|Speculation_Store_Bypass)"
for pid in $(ps -fu postgres|tail -n+2|tr -s " "|cut -d " -f2)
do
    echo "${pid} - $(cat /proc/${pid}/status |grep -E ${LOI})"
done;
18355 - NoNewPrivs: 0
Seccomp: 2
Speculation_Store_Bypass: thread force mitigated

[...]

18362 - NoNewPrivs: 0
Seccomp: 2
Speculation_Store_Bypass: thread force mitigated
Edit postgresql.service

# 94 total syscalls
SystemCallFilter=accept access arch_prctl ...<long list>... wait4 write

# Uncomment to make error return the default instead of SIGSYS
#SystemCallErrorNumber=EACCES

SystemCallArchitectures=native

# Uncomment this time...
NoNewPrivileges=yes
Restart PostgreSQL

# after editing service file, reload daemon
systemctl daemon-reload

# not for production use - clear out audit log
echo -n > /var/log/audit/audit.log

# start/restart postgresql service
systemctl restart postgresql@12-main.service
LOI="(Seccomp|NoNewPrivs|Speculation_Store_Bypass)"
for pid in $(ps -fu postgres|tail -n+2|tr -s " "|cut -d " " -f2)
do
echo "${pid} - $(cat /proc/${pid}/status |grep -E ${LOI})"
done;
18593 - NoNewPrivs: 1
Seccomp: 2
Speculation_Store_Bypass: thread force mitigated

[...]

18600 - NoNewPrivs: 1
Seccomp: 2
Speculation_Store_Bypass: thread force mitigated
pgseccomp Overview

- SECCOMP filtering via PostgreSQL config options

  **Advantages:**
  - PostgreSQL admin control
  - More flexibility
  - Supports `SCMP_ACT_LOG` option
  - Ensures `SCMP_ACT_ERRNO` actions logged
  - Possible to lock down session more tightly than postmaster
  - Possible to lock down some roles more tightly than others

  **Disadvantages:**
  - Possibly less resilient to subversion than systemd method
    → but risk mitigated/eliminated through good practices
Implementation

- PostgreSQL extension
- Loaded via `shared_preload_libraries`
- Postmaster global filter loaded on service start via `_PG_init()`
- Global filter config settings `PGC_POSTMASTER`: change requires **restart**
- Client backend session filter loaded on session start via `ClientAuthentication_hook`
- Session filter config settings `PGC_SIGHUP`: change requires **reload**
- Provides `seccomp_filter()` table function: describes installed merged filter
Enabling Configuration

- Requires `shared_preload libraries = 'pgseccomp'
- `pgseccomp.enabled`: Overall on/off switch
Global Filter Configuration

- `pgseccomp.global_syscall_allow/log/error/kill`: Lists of syscalls per action at Postmaster level
- `pgseccomp.global_syscall_default`: Default postmaster action allow/log/error/kill
Session Filter Configuration

- `pgseccomp.session_syscall_allow/log/error/kill`: Lists of syscalls per action at session level
- `pgseccomp.session_syscall_default`: Default session action allow/log/error/kill
- `pgseccomp.session_roles`: List of roles with customized syscall lists
  - `session_syscall_allow/log/error/kill/default.<role>`: Per role customized syscall lists
Client Filter Configuration

SELECT set_client_filter(default_action, allow_list, log_list, error_list, kill_list);
Step 1 - postgresql.conf

```plaintext
pgseccomp.enabled = on

pgseccomp.global_syscall_default = 'allow'
pgseccomp.global_syscall_allow = ''
pgseccomp.global_syscall_log = ''
pgseccomp.global_syscall_error = ''
pgseccomp.global_syscall_kill = ''

pgseccomp.session_syscall_default = 'log'
# Note '*' means use global list
pgseccomp.session_syscall_allow = '*'
pgseccomp.session_syscall_log = '*'
pgseccomp.session_syscall_error = '*'
pgseccomp.session_syscall_kill = '*'
```
Step 2 - auditd.conf

Modify `/etc/audit/auditd.conf`

- Derivation of syscall list floods log
- Ensure log lines not lost
- Do not rotate log

```
disp_qos = 'lossless'
change max_log_file_action = 'ignore'
```
Step 3 and 4 - Prepare for Testing

# clear and restart auditd
systemctl stop auditd.service # if running
echo -n "" > /var/log/audit/audit.log
systemctl start auditd.service

# restart postgresql
systemctl restart postgresql@12-main.service
Step 5 - Exercise PostgreSQL

- Exercise postgres as much as possible; for example:
  - Your application regression tests
  - Other random testing of relevant postgres features
  - PostgreSQL regression tests
    ```
    make installcheck-world
    make check world EXTRA_REGRESS_OPTS=--temp-config=<dir>/tmp.conf
    ```
- Note: at this point audit.log will start growing very quickly
Step 6 - Process Results

- a) Stop auditd service
- b) Run the provided `get_syscalls.sh` script
- c) Cut and paste the result as the value of `pgseccomp.session_syscall_allow`
Step 7 - Derive Global Filter Lists

- Set `pgseccomp.global_syscall_default = 'log'`
- Repeat steps 3-5
- Repeat step 6a and 6b
- Cut and paste the result as the value of `pgseccomp.global_syscall_allow`
Step 8 - Iterate

- Iterate
- Add new syscalls found to global and session allow lists
- Stop when `get_syscalls.sh` output is empty
Step 9 - Adjust

- Change global and session defaults to `error` or `kill`
- Reduce allow lists if desired
Step 10 - Ongoing

- Monitor postgres and audit.log
- Adjust or react as required
Example #1 - Block readlink Syscall

CREATE TABLESPACE regress_tblspace LOCATION '/tmp/testtblspc';

SELECT * FROM seccomp_filter()
WHERE syscall = 'readlink' AND context = 'session';

<table>
<thead>
<tr>
<th>syscall</th>
<th>syscallnum</th>
<th>filter_action</th>
<th>context</th>
</tr>
</thead>
<tbody>
<tr>
<td>readlink</td>
<td>89</td>
<td>session-&gt;allow</td>
<td>session</td>
</tr>
</tbody>
</table>

(1 row)

-- Use readlink syscall; should work
WITH a(tsid) AS
(SELECT oid FROM pg_tablespace WHERE spcname = 'regress_tblspace')
SELECT pg_tablespace_location(tsid) FROM a;

<table>
<thead>
<tr>
<th>pg_tablespace_location</th>
</tr>
</thead>
<tbody>
<tr>
<td>/tmp/testtblspc</td>
</tr>
</tbody>
</table>

(1 row)
Example #1 - Block readlink Syscall

-- move 'readlink' from session allow to session error in postgresql.conf
-- requires reload and new session
SELECT * FROM seccomp_filter()
WHERE filter_action LIKE '%error' AND context = 'session';
    syscall | syscallnum | filter_action | context
--------+------------+----------------+---------
  readlink |       89 | session->error | session
(1 row)

-- Use readlink syscall; should fail with permission denied error
WITH a(tsid) AS
(SELECT oid FROM pg_tablespace WHERE spcname = 'regress_tblspace')
SELECT pg_tablespace_location(tsid) FROM a;
ERROR: could not read symbolic link "pg_tblspc/363574": Permission denied
Example #2 - Cannot Reduce Restriction on nanosleep

-- add 'nanosleep' to session allow in postgresql.conf
-- requires reload and new session
-- note that it is not listed in the global allow config

-- 'nanosleep' syscall is limited by postmaster global default
SELECT * FROM seccomp_filter()
WHERE filter_action LIKE 'global%'
AND context = 'session';
  syscall  | syscallnum | filter_action | context
-----------+------------+---------------+---------
  nanosleep |    35       | global->log   | session
(1 row)
Example #3 - Block clone Selectively

CREATE LANGUAGE plperlu;
CREATE OR REPLACE FUNCTION cat(location text) RETURNS text AS $$
    use IPC::Run3; my @cmd = ("cat", $_[0]);
    run3 \@cmd, undef, \my $cout , \my $cerr;
    if ($?!=0){return($cerr)} else {return($cout)};
$$ LANGUAGE plperlu;

-- With settings gathered per the above, and
-- pgseccomp.session_syscall_default = 'log', this throws the following
-- syscalls into audit.log: arch_prctl,clone,dup,dup2,execve,fadvise64,
-- getgroups,pipe,set_robust_list,wait4
SELECT cat('pg_hba.conf');

# PostgreSQL Client Authentication Configuration File
# ===================================================

[...]
Example #3 - Block clone Selectively

-- now add 'clone' to session error in postgresql.conf
-- requires reload and new session
----------------------------------------
-- pgseccomp.session_syscall_error = 'clone'

```
SELECT cat('pg_hba.conf');
ERROR: Permission denied at line 4.
CONTEXT: PL/Perl function "cat"
```
Example #3 - Block clone Selectively

-- add following in postgresql.conf
-- requires reload and new session
----------------------------------------
-- pgseccomp.session_roles = 'joe'
-- session_syscall_error.joe = ''

\c - joe
SELECT cat('pg_hba.conf');

# PostgreSQL Client Authentication Configuration File
# ==============================================================
# # PostgreSQl Client Authentication Configuration File
# # ==============================================================

[...]
Example #4 - Block clone Client-side

GRANT EXECUTE ON FUNCTION
  set_client_filter(text,text,text,text,text) TO joe;
\c - joe
SELECT cat('pg_hba.conf');

-------------------------------------------------------------------------
# PostgreSQL Client Authentication Configuration File
# ==============================================================
[...]
SELECT set_client_filter('error',null,null,'clone',null);
  set_client_filter
-------------------
  OK
(1 row)

SELECT cat('pg_hba.conf');
ERROR: run3(): Permission denied saving STDOUT at line 4.
CONTEXT: PL/Perl function "cat"
Questions?

Thank You!
mail@joeconway.com
joe@crunchydata.com
@josepheconway