Yes, even **you** can harden your systemd services!

SoS Stage H @ 39C3

whoami/whoomee (she/they)

39C3

security theater?

- maybe. please don't leave your SSH server wide open and blame systemd afterwards.
- hardening involves complicated interactions between settings.
  - danger of leaving a service in a state that allows easy undoing of hardening!

defense in depth

- processes should be equipped with the least privileges necessary to do their job
- it should be easy to determine the resources a service has access to

the perfect citizen

- reads files from the usual, well defined paths
- limits write access to the appropriate directories (`/var/lib`, `/var/cache`) (cf. plugin/auto-update systems)
- does not execute executables from random places
- perfect world: we statically know the bounding set of paths (for binaries, libraries, etc.) required by the service - possible on NixOS and similar systems

common system service patterns

service started as root because... (*examples*)

- it needs to bind to a privileged port
- it needs to access secrets owned by the root user

but as a result:

- it *can* access secrets owned by root
- it *can* wipe your hard drive
- it *can* take over other privileged ports
- it *can* replace your sudo with tetris
- fs access sprinkled everywhere...

big picture: the one and only manpage

- which manpage contains the documentation for `Foo=`?

```
ask systemd.directives(7)!
```

the manual is your friend!

Almost every option mentioned here is documented in `systemd.exec(5)`

- what does the world look like for `myapp.service`?

`systemd-analyze unit-shell` (only on systemd v258+)

- shucks, which settings have I configured for `myapp.service`?

```
systemctl cat myapp.service
```

everything is a file, so we need files!

- fs access is usually one of the areas to be tinkered with
- different intensities of isolation, let's not go totally wild (yet)

tools in the toolbox: host fs access

`ReadWritePaths=` and `ReadOnlyPaths=`

`InaccessiblePaths=` and `TemporaryFileSystem=`

example:

```
ReadOnlyPaths=/
ReadWritePaths=/var/lib/myapp
InaccessiblePaths=/home/foo
TemporaryFileSystem=/home/bar
```

(but better ways exist!)

restricting executable paths

`ExecPaths=` and `NoExecPaths=` allow restricting executable loading

example:

```
NoExecPaths=/
ExecPaths=/usr/bin
```

fun with bind mounts and tmpfs

Who says the service-visible fs hierarchy needs to look anything like the host fs?

- hides host fs contents using tmpfs mounts: `TemporaryFileSystem=`
- build arbitrary filesystem structure: `BindPaths=` and `BindReadOnlyPaths=`
  - allows injecting files exactly where a service expects them!
  - useful because $people like to hardcode paths...

starting with an empty slate

- `ProtectSystem=strict`: everything* is mounted read-only!
  - except `/dev`, `/proc` and `/sys`
- `ProtectHome=tmpfs`: hide `/home` and `/root`
- these are almost always viable

/var is for persistent data…

- `StateDirectory=` under `/var/lib`
- `CacheDirectory=` under `/var/cache`
- `LogsDirectory=` under `/var/log`

in practice: use `%S/$STATE_DIRECTORY` and friends

```
StateDirectory=myapp
ExecStart=/.../myapp --data-dir %S/myapp
```

additional directories

- `RuntimeDirectory=` under `/run/` (defaults: *automatically* set up/removed)
- `ConfigurationDirectory=` under `/etc`
  - can be iffy: configuration editable via web UI etc.
  - NixOS *alternative*: bind mount config directly `BindReadOnlyPaths=${config}:/etc/myapp.conf`

even more directories...

- `PrivateTmp=`: private per-process `/tmp`
- `PrivateDevices=`
- `ProtectProc=noaccess/invisible` and `ProcSubset=pid`

system call filtering

- `SystemCallArchitectures=native`
- `SystemCallFilter=@system-service`
- `SystemCallErrorNumber=EPERM`

the boring ones

just set these to `True` because seriously, wtf?

- `ProtectHostname=`
- `ProtectClock=`
- `ProtectKernelTunables=`
- `ProtectKernelModules=`
- `ProtectKernelLogs=`
- `RestrictSUIDSGID=`
- `RestrictNamespaces=`
- `LockPersonality=`: who needs support for WYSE UNIX System V/386?

matter of interpretation

`MemoryDenyWriteExecute=`

(not compatible with dotnet *and* friends)

being useful to the outside world

`RestrictAddressFamilies=` is almost always viable:

`AF_UNIX`/`AF_INET`/`AF_INET6`

some assembly required:

- `IPAddressAllow=`, `IPAddressDeny=`

  services have their own minds on which address to bind…
- `PrivateNetwork=`

going further: confinement on NixOS

```
systemd.services.»service-name«.confinement.enable = true
```

```
RootDirectory=/run/confinement/%n
RuntimeDirectory=confinement/%n
BindReadOnlyPaths=/nix/store/...-glibc-2.40-66
BindReadOnlyPaths=/nix/store/...-gcc-14.3.0-libgcc
...
```

systemd credentials

use for:

- TLS certificates (perhaps shared by multiple services)
- all kinds of secrets

start with `LoadCredential=/path/on/host:/path/for/service`

- set up inside runtime directory
  e.g. `/run/credentials/myapp.service`
- credentials *are* not visible outside the service!

capabilities

root in kernel*: check euid == 0

capabilities

root in kernel*: check euid == 0

capabilities: fine-grained access

- `CAP_NET_BIND_SERVICE`: bind port < 1024
- `CAP_SYS_NICE`: change scheduler settings (nice/realtime priority)

capabilities

root in kernel*: check euid == 0

capabilities: fine-grained access

- `CAP_NET_BIND_SERVICE`: bind port < 1024
- `CAP_SYS_NICE`: change scheduler settings (nice/realtime priority)

use `CapabilityBoundingSet=` and `NoNewPrivileges=`
(and `AmbientCapabilities=` if required)

who needs users after all?

manually editing `/etc/passwd` and `/etc/group` is so 2010!

- danger of UID/GID reuse
- who owns that uid=941 zombie file lying around suspiciously?

dynamic users to the rescue

- `DynamicUser=true` allocates a uid/gid at runtime based on rules*
- designed for use with `ProtectSystem=strict` and `ProtectHome`
- doesn't allow any access outside `StateDirectory=`, `RuntimeDirectory=` etc.
- users and groups are synthesized via `nss-systemd(8)`

dynamic users to the rescue

- `DynamicUser=true` allocates a uid/gid at runtime based on rules*
- designed for use with `ProtectSystem=strict` and `ProtectHome`
- doesn't allow any access outside `StateDirectory=`, `RuntimeDirectory=` etc.
- users and groups are synthesized via `nss-systemd(8)`

N.B. this feature is often misunderstood and the most likely cause why your hardened service configuration doesn't work!

too much time? finding stuff to harden

```
sh-5.3$ sudo systemd-analyze security
UNIT                                 EXPOSURE PREDICATE HAPPY
dbus.service                              9.6 UNSAFE    😨
dovecot.service                           3.5 OK        🙂
emergency.service                         9.5 UNSAFE    😨
fail2ban.service                          5.8 MEDIUM    😐
getty@tty1.service                        9.6 UNSAFE    😨
knot.service                              1.8 OK        🙂
nginx.service                             1.6 OK        🙂
nix-daemon.service                        9.6 UNSAFE    😨
nscd.service                              8.2 EXPOSED   🙁
postfix.service                           6.1 MEDIUM    😐
postgresql.service                        1.3 OK        🙂
```

hardening in practice

```
systemd-analyze security myapp.service
```

typical fails

most over-hardening issues occur at startup!

but sometimes…

- service tries to read random path after a week of runtime
- service wants to do weird things with `/tmp`
  (try not to use `/tmp` to share files between processes)
- seldomly used feature/API endpoint requires specific fs access/syscall

tips

safe and useful:

- filesystem restrictions
- `LockPersonality=` and friends

use caution:

- network sandboxing

recap: the manual is your friend!

debugging hardening settings:

- show settings: `systemctl cat myapp.service`
- `systemd-analyze security` (beware of snakeoil!)

inspecting sandboxed services:

- `systemd-analyze unit-shell` (only on systemd v258+)
- `findmnt --task="$(systemctl show --value -p MainPID myapp.service)"`

takeaways

- use `systemd.directives(7)` to find systemd options
- read through `systemd.exec(5)`
- use `systemd-analyze security` for fun and profit
- allow specifying secrets via file paths
- writing a service: try to separate state, logs, cache, configuration

have fun collecting systemd-analyze points!

Get in touch 👋

✉ @violet_cookie_bytes@social.tchncs.de

DECT WHOOO / GSM NIXOS

@whoami@chaos.social

⇧my blog