Linux system hardening thanks to systemd

Timothée Ravier

French Network and Information Security Agency (ANSSI)

RMLL 2017

Goal of this talk

Goal of this talk

- Increase the security of standard Linux distributions
- Use security features made available to userspace by the Linux kernel
- Take advantage of their integration into systemd
- Simplify deployments and help system maintenance

systemd “how-to” in three slides

systemd?

- Integrated in most Linux distributions as a replacement for SysVinit
- Handle system boot up and manage system services
- Responsible for environment setup for system daemons
- Init scripts are replaced by declarative configuration files: **units**

Unit?

To display the current configuration of a service:

```
# systemctl cat php-fpm.service

# /usr/lib/systemd/system/php-fpm.service

[Unit]
Description=The PHP FastCGI Process Manager
After=network.target

[Service]
Type=notify
PIDFile=/run/php-fpm/php-fpm.pid
ExecStart=/usr/bin/php-fpm --nodaemonize
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Command

Unit?

To display the current configuration of a service:

```
# systemctl cat php-fpm.service

# /usr/lib/systemd/system/php-fpm.service

[Unit]
Description=The PHP FastCGI Process Manager
After=network.target

[Service]
Type=notify
PIDFile=/run/php-fpm/php-fpm.pid
ExecStart=/usr/bin/php-fpm --nodaemonize
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Corresponding file

Unit?

To display the current configuration of a service:

```
# systemctl cat php-fpm.service

# /usr/lib/systemd/system/php-fpm.service

[Unit]
Description=The PHP FastCGI Process Manager
After=network.target

[Service]
Type=notify
PIDFile=/run/php-fpm/php-fpm.pid
ExecStart=/usr/bin/php-fpm --nodaemonize
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Who?
When?

Unit?

To display the current configuration of a service:

```
# systemctl cat php-fpm.service

# /usr/lib/systemd/system/php-fpm.service

[Unit]
Description=The PHP FastCGI Process Manager
After=network.target

[Service]
Type=notify
PIDFile=/run/php-fpm/php-fpm.pid
ExecStart=/usr/bin/php-fpm --nodaemonize
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

What?
How?

Unit?

To display the current configuration of a service:

```
# systemctl cat php-fpm.service

# /usr/lib/systemd/system/php-fpm.service

[Unit]
Description=The PHP FastCGI Process Manager
After=network.target

[Service]
Type=notify
PIDFile=/run/php-fpm/php-fpm.pid
ExecStart=/usr/bin/php-fpm --nodaemonize
PrivateTmp=true

[Install]
WantedBy=multi-user.target
```

Why?

Example: switching to an unprivileged user and group

Edit the service configuration:

```
# systemctl edit php-fpm.service
```

Example: switching to an unprivileged user and group

Edit the service configuration:

```
# systemctl edit php-fpm.service
```

add the following content:

```
[Service]
User=http
Group=www
```

Example: switching to an unprivileged user and group

Edit the service configuration:

```
# systemctl edit php-fpm.service
```

add the following content:

```
[Service]
User=http
Group=www
```

and make those changes effective:

```
# systemctl daemon-reload
# systemctl restart php-fpm.service
```

Taking advantage of security features from the Linux kernel

Filtering access to system calls using *seccomp-bpf*

Concept

- Restrict which system calls are available to a process
- Also applies to child processes

Filtering access to system calls using *seccomp-bpf*

Concept

- Restrict which system calls are available to a process
- Also applies to child processes

Example

```
[Service]
SystemCallFilter=~chroot
SystemCallFilter=~@obsolete
```

Filtering access to system calls using *seccomp-bpf*

Concept

- Restrict which system calls are available to a process
- Also applies to child processes

Example

```
[Service]
SystemCallFilter=~chroot
SystemCallFilter=~@obsolete
```

Beware

- Can be bypassed with ptrace on kernels < 4.8
- Solution: add a filter for the ptrace system call:

```
[Service]
SystemCallFilter=~ptrace
```

Linux capabilities

Concept

- Restrict privileges granted to a process (potentially running as root)
- Grant a subset of root privileges to an unprivileged process

Linux capabilities

Concept

- Restrict privileges granted to a process (potentially running as root)
- Grant a subset of root privileges to an unprivileged process

Example

```
[Service]
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
AmbientCapabilities=CAP_NET_BIND_SERVICE
```

Linux capabilities

Concept

- Restrict privileges granted to a process (potentially running as root)
- Grant a subset of root privileges to an unprivileged process

Example

```
[Service]
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
AmbientCapabilities=CAP_NET_BIND_SERVICE
```

Beware

- Some capabilities are equivalent to full root privileges
- Avoid blacklists. Whitelist only the capabilities effectively used

For more details, see: https://forums.grsecurity.net/viewtopic.php?f=7&t=2522

Mount namespaces

Concept

- Each service can get its own filesystem hierarchy
- Hide arbitrary paths or turn them read-only

Mount namespaces

Concept

- Each service can get its own filesystem hierarchy
- Hide arbitrary paths or turn them read-only

Example

```
[Service]
InaccessiblePaths=/etc/secrets
ProtectSystem=full
```

Mount namespaces

Concept

- Each service can get its own filesystem hierarchy
- Hide arbitrary paths or turn them read-only

Example

```
[Service]
InaccessiblePaths=/etc/secrets
ProtectSystem=full
```

Beware

- Reversible if CAP_SYS_ADMIN or mount system call is available:

```
[Service]
CapabilityBoundingSet=~CAP_SYS_ADMIN
SystemCallFilter=~@mount
```

Getting your hands dirty (cow?)

Practical example: sandboxing the Dirty CoW

- Vulnerability CVE-2016-5195
- *Local root* made public in October 2016
- Impacted every kernel from the version 2.6.22, released in 2007
- Race condition in the memory management code handling Copy-on-Write

Practical example: sandboxing the Dirty CoW

Exploit vector

- Race condition triggered by the `madvise` system call

Options to mitigate the impact

- Block the `madvise` system call

Configuration

```
[Service]
SystemCallFilter=~madvise
```

Practical example: sandboxing the Dirty CoW

Exploit vector

- Indirect access to memory using the `ptrace` system call and `/proc/self/mem`

Options to mitigate the impact

- Block the `ptrace` system call
- Remove access to the `proc` virtual filesystem

Configuration

```
[Service]
SystemCallFilter=~ptrace
InaccessiblePaths=/proc
```

See https://lists.freedesktop.org/archives/systemd-devel/2017-April/038634.html and https://github.com/systemd/systemd/pull/5985 for more details.

Practical example: sandboxing the Dirty CoW

Exploit vector

- Vulnerable code may be reachable from drivers exposed in /dev

Options to mitigate the impact

- Remove access to most hardware drivers available from /dev

Configuration

```
[Service]
PrivateDevices=yes
```

Practical example: The Good, the Bad and the socket

- Vulnerability CVE-2016-8655
- *Local root*
- Race condition in `AF_PACKET` type sockets leading to Use-After-Free in kernel context
- Creating `AF_PACKET` sockets requires `CAP_NET_RAW`
- May be obtained via unprivileged user namespace (Linux $\geqslant$ 3.8)

Practical example: The Good, the Bad and the socket

Exploit vector

- AF_PACKET sockets

Options to mitigate the impact

- Restrict socket type availability

Configuration

Minimal version with a blacklist:

```
[Service]
RestrictAddressFamilies=~AF_PACKET
```

Better option using a whitelist:

```
[Service]
RestrictAddressFamilies=AF_INET AF_INET6 AF_UNIX
```

Practical example: The Good, the Bad and the socket

Exploit vector

- CAP_NET_RAW capability

Options to mitigate the impact

- Block acquisition of the CAP_NET_RAW capability

Configuration

```
[Service]
CapabilityBoundingSet=~CAP_NET_RAW
```

Practical example: The Good, the Bad and the socket

Exploit vector

- Unrestricted availability of unprivileged user namespace

Options to mitigate the impact

- Restrict access to user namespaces

Configuration

```
[Service]
RestrictNamespaces=~user
```

Notice

- Requires systemd $\geqslant$ 233

Practical example: systemd versus the crashing tweet

- Vulnerability CVE-2016-7795
- Denial of Service targeting systemd
- Raise an assertion in the daemon running as PID 1
- Pause process execution thus reducing functionality available on the system

Practical example: systemd versus the crashing tweet

Exploit vector

- Incorrect handling of empty notification events sent through `/run/systemd/notify`

Options to mitigate the impact

- Restrict access to the `/run/systemd/notify` socket

Configuration

```
[Service]
InaccessiblePaths=/run/systemd
```

Conclusion

Conclusion

- Simplified interface to help setup kernel security features
- Easy to setup and maintain
- Does not replace applying updates
- Hardening features applied only to system services

Thank you

Contact:

- ✉ timothee.ravier@ssi.gouv.fr
- travier@mastodon.etalab.gouv.fr
- @siosm