

Linux security and system hardening

Jeffrey Bencieux - <https://www.bencieux.fr>

September 29, 2026

Outline

Introduction	Partitioning
objectives	Disk encryption
general principles	Authentication
Secure bootloader chain	Pluggable Authentication
Kernel parameters	Modules (PAM)
Kernel configuration	SSH
Kernel command line	Kerberos realm
Sysctl interface	Autorisation
Linux kernel security mechanisms	file permissions
Modules	Capabilities
Linux Security Modules (LSM)	autorisation models
Memory protections	Services
eBPF	Tracability and detection
Cryptography API	logs
Network protections	audit subsystem and auditd
Process isolation	security software
Filesystems and partitioning	MISC
	User protections
	Conclusion

Outline

Introduction

- objectives

- general principles

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

Outline

Introduction

- objectives

- general principles

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

Objectives

- know and understand the main security mechanisms of a GNU/Linux operating system
- harden GNU/Linux operating systems
- secure by practice

Outline

Introduction

- objectives

- general principles

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

The attack surface of a software environment is the sum of the different points (for "attack vectors") where an unauthorized user (the "attacker") can try to enter data to, extract data from, or control a device or critical software in an environment. - Wikipedia

Attack surface reduction

- reduce the number of entry points
 - eg: disable useless services on a server

Attack surface reduction

- reduce the number of entry points
 - eg: disable useless services on a server
- improve security of existing entry points
 - eg: implement physical access control

Know your attack surface:

- how can a building being entered?
- what ports are open on a machine? Why?
- what feature of a kernel are enabled? Why?

Least privilege

Use the least needed privilege to do a given task.

Least privilege examples

- execute a systemd service as a service user, not as root

Least privilege examples

- execute a systemd service as a service user, not as root
- connect to your VPS server via SSH as a low-privileged user, then sudo with only authorized commands

Least privilege examples

- execute a systemd service as a service user, not as root
- connect to your VPS server via SSH as a low-privileged user, then sudo with only authorized commands
- use CAP_NET_RAW for tcpdump instead of making it setUID root

Defense in depth

Slow down attackers by thinking and implementing several successive defenses.

Defense in depth



Defense in depth examples

- physical control access with biometry factor

Defense in depth examples

- physical control access with biometry factor
- on the wire: 802.1x → IPSec → mTLS → data

Defense in depth examples

- physical control access with biometry factor
- on the wire: 802.1x → IPSec → mTLS → data
- Linux process: running in a container / on a dedicated VM /
on a dedicated hypervisor / on a dedicated machine

Defense in depth

- make attackers' life hard
- the myth of "safe spaces"
- know your assets, adapt your defenses

Several categories or range of attackers exists. Their level depends mostly on:

- money
- motivation

- state-sponsored (politics, economic interests)

Attackers models and threat analysis

- state-sponsored (politics, economic interests)
- criminals (mostly money)

Attackers models and threat analysis

- state-sponsored (politics, economic interests)
- criminals (mostly money)
- activists (ideology)

Attackers models and threat analysis

- state-sponsored (politics, economic interests)
- criminals (mostly money)
- activists (ideology)
- insiders (money or ideology)

Attackers models and threat analysis

- state-sponsored (politics, economic interests)
- criminals (mostly money)
- activists (ideology)
- insiders (money or ideology)
- hackers (fun)

Attackers models and threat analysis

- state-sponsored (politics, economic interests)
- criminals (mostly money)
- activists (ideology)
- insiders (money or ideology)
- hackers (fun)

Your defenses should always be adequate to whom you want to protect your assets again.

Before implementing security defenses, ask yourself:

- what should be protected?

Before implementing security defenses, ask yourself:

- what should be protected?
- against whom?

Before implementing security defenses, ask yourself:

- what should be protected?
- against whom?
- how much money and resources will be needed?

What are the attacker models for:

- your personal VPS
- a network share for an association
- a private company email server
- a governmental web server
- a missile-launching command server

What are the attacker models for:

- your personal VPS
- a network share for an association
- a private company email server
- a governmental web server
- a missile-launching command server

There is no rule of thumb to conceive "good models".

Risks should be identified and either:

- mitigated with an adequate security barrier
- accepted as residual by adequate authorities

Outline

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

An attacker has access to a physical device:

- what can he do?
- how can the boot chain be trusted?

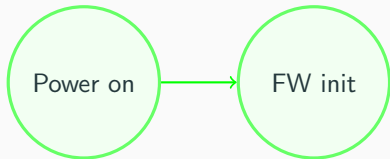
An attacker with physical access can:

- plug and unplug physical parts of the device
- change software parts such as the bootloader or the kernel image in `/boot`

Boot chain

1. Power on
2. Firmware init (boot ROM)
3. UEFI boot manager
4. Bootloader (GRUB, rEFInd...)
5. Kernel
6. Userland

Boot chain



Firmware initialization

- AKA ROM code
- dedicated Read-Only Memory
- contains the first software to execute when hardware is powered on

Boot chain



UEFI boot manager

- look for entries in NVRAM
- start EFI stubs located in ESP
- if NVRAM entry is a disk, look for path `\EFI\BOOT\BOOTx64.EFI`

EFI System Partition (ESP)

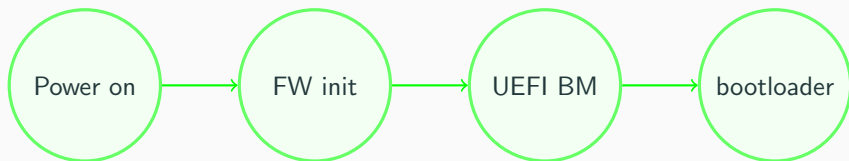
- FAT32 filesystem, mandatory
- Stores OS bootloaders or boot managers as `.efi` executable files
- EFI files are Portable Executable (PE)

EFI System Partition (ESP)

List EFI stubs

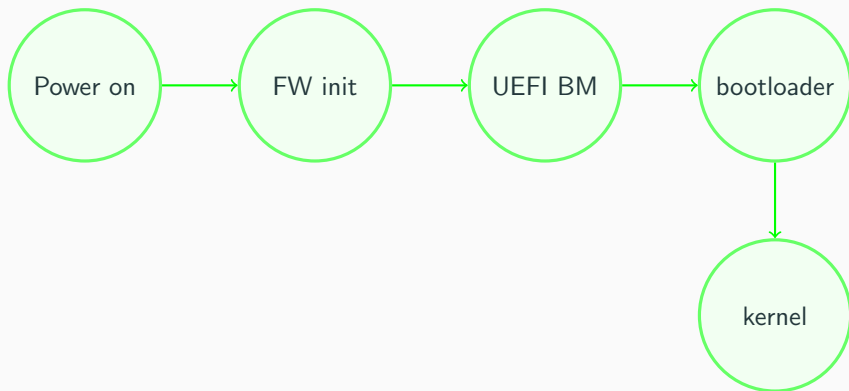
```
$ mount | grep /efi
/dev/nvme0n1p1 on /efi type vfat (rw,relatime,...)
$ tree /efi
/efi
├── EFI
│   ├── Boot
│   │   └── bootx64.efi
│   └── Microsoft
│       └── Boot
│           ...
│           ├── bootmgfw.efi
│           └── bootmgr.efi
│               ...
│               ├── refind
│               │   ├── drivers_x64
│               │   │   └── ext4_x64.efi
│               │   └── refind_x64.efi
│               ...
```

Boot chain

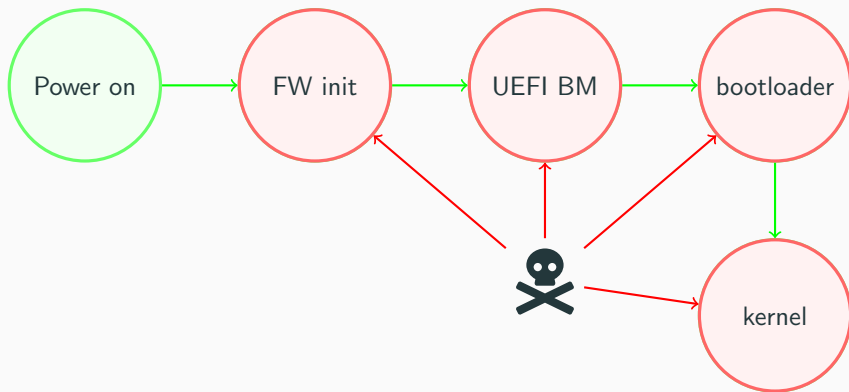


- bootloaders are EFI stubs
- usual suspects: GRUB, rEFInd
- in charge of loading kernel and initramfs from `/boot`
- has entries for multiple kernels or specific kernel command lines

Boot chain



Boot chain



How can the boot chain not be tampered with?

How can the boot chain not be tampered with?

- Cryptographic signatures

How can the boot chain not be tampered with?

- Cryptographic signatures
- UEFI Secure Boot

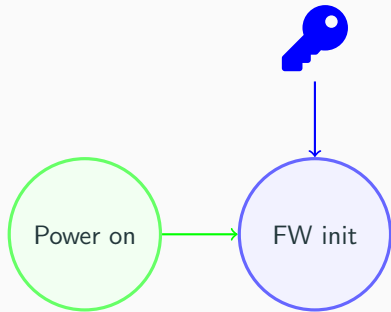
How can the boot chain not be tampered with?

- Cryptographic signatures
- UEFI Secure Boot
- Microsoft Certification Authority embedded in most hardware

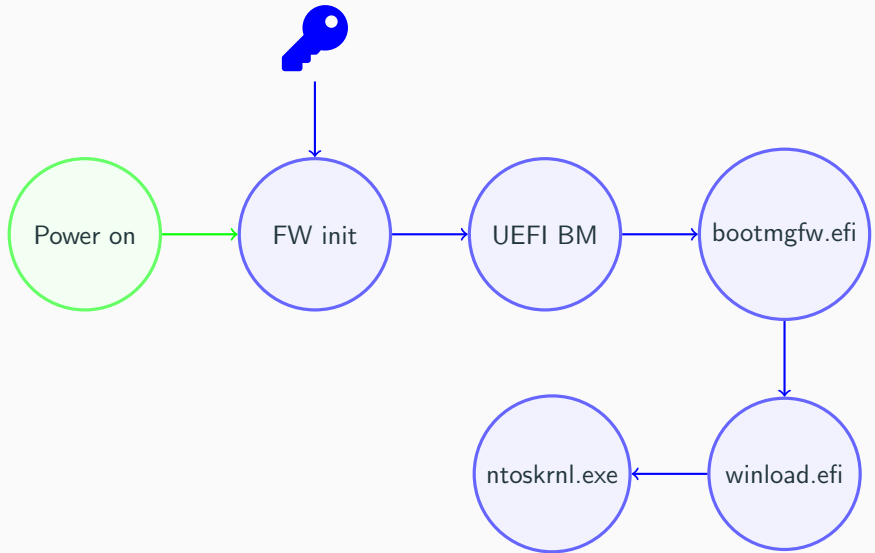
How can the boot chain not be tampered with?

- Cryptographic signatures
- UEFI Secure Boot
- Microsoft Certification Authority embedded in most hardware
- Every component verify the signature of the next one

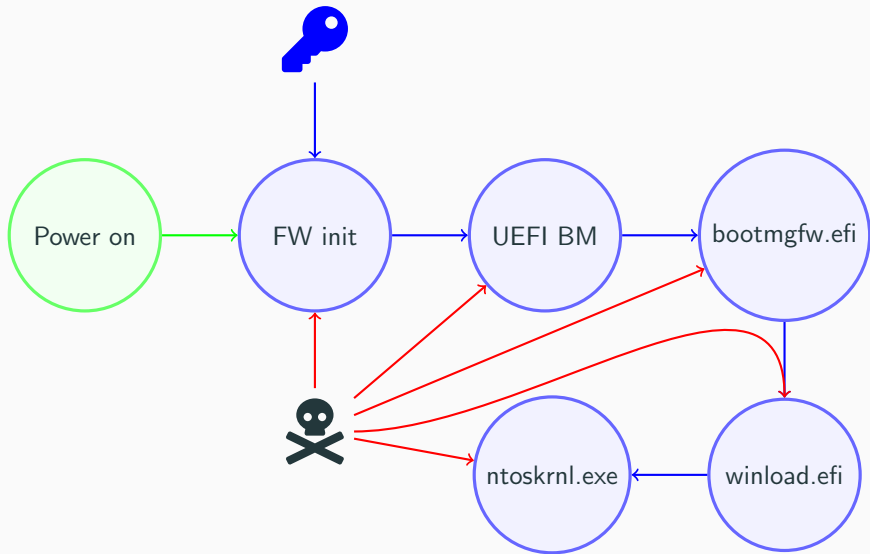
Microsoft Secure Boot chain



Microsoft Secure Boot chain



Microsoft Secure Boot chain



And Linux?

How to integrate Linux?

And Linux?

How to integrate Linux?

- add your own keys, then sign EFI stubs
 - not feasible by default

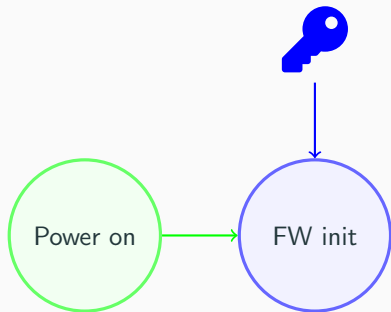
How to integrate Linux?

- add your own keys, then sign EFI stubs
 - not feasible by default
- SHIM: EFI stub signed by Microsoft keys

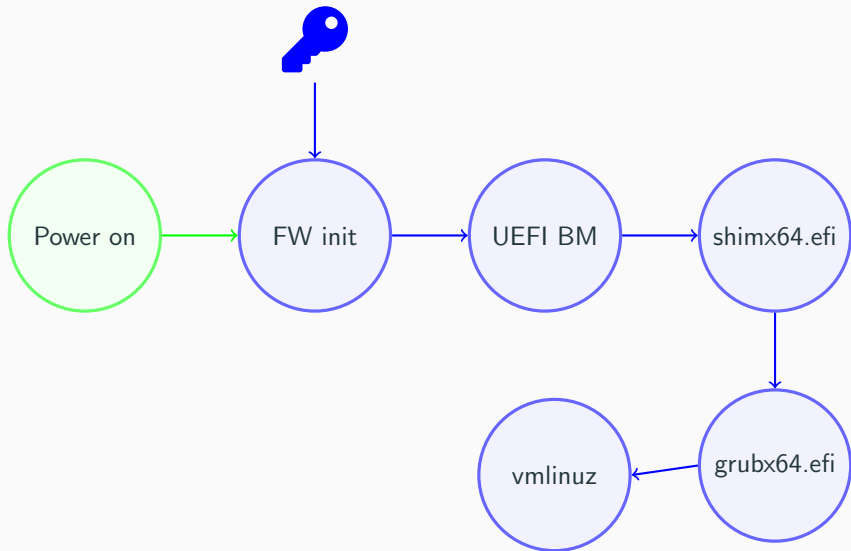
How to integrate Linux?

- add your own keys, then sign EFI stubs
 - not feasible by default
- SHIM: EFI stub signed by Microsoft keys
- use Linux distributions' own cryptographic keys to sign the kernel

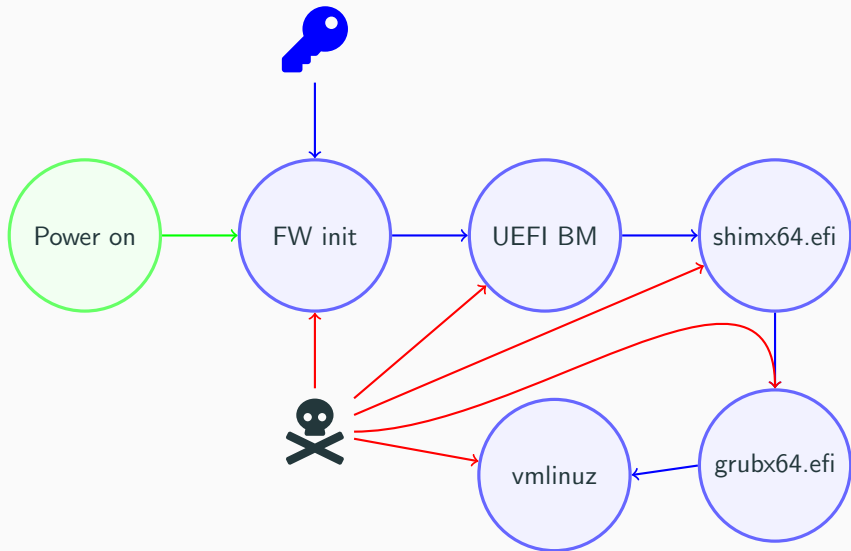
Linux secure boot chain



Linux secure boot chain



Linux secure boot chain



Secure Boot signatures

```
$ bootctl
  System:
    Firmware: n/a (n/a)
  Firmware Arch: x64
    Secure Boot: disabled
  TPM2 Support: yes
  Measured UKI: no
  Measured OS: no
  Boot into FW: supported
  Platform Lang: en_US.UTF-8
...
Available Boot Loaders on ESP:
  ESP: /efi (/dev/disk/by-partuuid/1b6c07e0-50e6-4
      bdd-82c0-c7d5bd305408)
  File: └─ /efi//EFI/BOOT/bootx64.efi
```

Secure Boot signatures

...

Boot Loaders Listed in EFI Variables:

Title: rEFInd Boot Manager

ID: 0x0000

Status: active , boot-order

Partition: /dev/disk/by-partuuid/1b6c07e0-50e6-4bdd-82c0-c7d5bd305408

File:  /efi//EFI/REFIND/REFIND_X64.EFI

Title: Windows Boot Manager

ID: 0x0003

Status: active , boot-order

Partition: /dev/disk/by-partuuid/1b6c07e0-50e6-4bdd-82c0-c7d5bd305408

File:  /efi//EFI/MICROSOFT/BOOT/BOOTMGFW.EFI

Checking signatures

```
$ sbverify /efi/EFI/Microsoft/Boot/bootmgfw.efi  
Signature verification OK  
$ sbverify /efi/EFI/efind/efind_x64.efi  
Signature verification failed
```

Secure Boot signatures

Self-signing

```
$ openssl req -x509 -sha256 -newkey rsa:4096 -noenc -keyout  
mykey.key -out mycert.crt
```

```
$ sbsign --key mykey.key --cert mycert.crt --output refind_x64_signed.efi  
        refind_x64.efi  
Signing Unsigned original image
```

```
$ sbverify --cert mycert.crt refind_x64_signed.efi  
Signature verification OK
```

Trusted Platform Module (TPM)

- How can cryptographic keys be stored securely?

Trusted Platform Module (TPM)

- How can cryptographic keys be stored securely?
- Trusted Platform Module (TPM): dedicated hardware chip

Trusted Platform Module (TPM)

- How can cryptographic keys be stored securely?
- Trusted Platform Module (TPM): dedicated hardware chip
- stores secrets for secure boot integrity checks

Trusted Platform Module (TPM)

- How can cryptographic keys be stored securely?
- Trusted Platform Module (TPM): dedicated hardware chip
- stores secrets for secure boot integrity checks
- can also be used for Full Disk Encryption (FDE) secrets' storage

Trusted Platform Module (TPM)

- How can cryptographic keys be stored securely?
- Trusted Platform Module (TPM): dedicated hardware chip
- stores secrets for secure boot integrity checks
- can also be used for Full Disk Encryption (FDE) secrets' storage
- TPM 2.0 necessary for Windows 11, no requirements on Linux distributions

Trusted Platform Module (TPM)

- contains "entries" called Platform Configuration Registries (PCR)
- each PCR has a specific role¹ for system integrity

¹https://wiki.archlinux.org/title/Trusted_Platform_Module

Trusted Platform Module (TPM)

Number	Use
PCR0	Core System Firmware executable code (aka Firmware)
PCR1	Core system firmware data/host platform configuration; typically contains serial and model numbers
PCR2	Extended or pluggable executable code; includes option ROMs on pluggable hardware
PCR3	Extended or pluggable firmware data; includes information about pluggable hardware
PCR4	Boot Manager Code and Boot Attempts. Measures the boot manager and the devices that the firmware tried to boot from.
PCR5	oot Manager Configuration and Data. Can measure configuration of boot loaders; includes the GPT Partition Table.
PCR6	Resume from S4 and S5 Power State Events
PCR7	SecureBoot state
PCR8	Hash of the kernel command line
PCR9	Hash of the initramfs and EFI Load Options Linux (measures the initramfs and EFI Load Options, essentially the kernel cmdline options)

- virtual TPM for virtualized systems

Check for TPM presence

```
$ ls -l /dev/tpm*
```

```
$ ls -ld /sys/kernel/security/tpm*
```

- BlackLotus (2023): UEFI bootkit² for Windows using CVE-2022-21894 to bypass Secure Boot

²<https://www.welivesecurity.com/2023/03/01/blacklotus-uefi-bootkit-myth-confirmed/>

- BlackLotus (2023): UEFI bootkit² for Windows using CVE-2022-21894 to bypass Secure Boot
- brings a legitimate but vulnerable `bootmgfw.efi`

²<https://www.welivesecurity.com/2023/03/01/blacklotus-uefi-bootkit-myth-confirmed/>

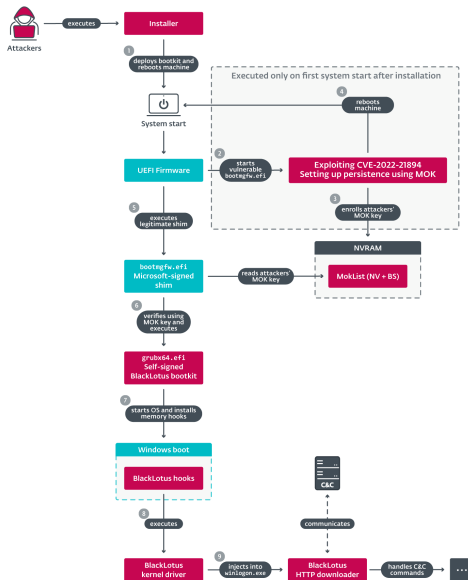
- BlackLotus (2023): UEFI bootkit² for Windows using CVE-2022-21894 to bypass Secure Boot
- brings a legitimate but vulnerable `bootmgfw.efi`
- adds an attacker key to authorized keys and self-signed EFI stubs: chain of trust is maintained

²<https://www.welivesecurity.com/2023/03/01/blacklotus-uefi-bootkit-myth-confirmed/>

- BlackLotus (2023): UEFI bootkit² for Windows using CVE-2022-21894 to bypass Secure Boot
- brings a legitimate but vulnerable `bootmgfw.efi`
- adds an attacker key to authorized keys and self-signed EFI stubs: chain of trust is maintained
- persistent across reboots and kernel reinstalls

²<https://www.welivesecurity.com/2023/03/01/blacklotus-uefi-bootkit-myth-confirmed/>

BlackLotus flow



- LoJax, a firmware malware³

³<https://www.welivesecurity.com/2018/09/27/lojax-first-uefi-rootkit-found-wild-courtesy-sednit-group/>

Questions?

Introduction

Secure bootloader chain

Kernel parameters

- Kernel configuration

- Kernel command line

- Sysctl interface

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

Introduction

Secure bootloader chain

Kernel parameters

- Kernel configuration

- Kernel command line

- Sysctl interface

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

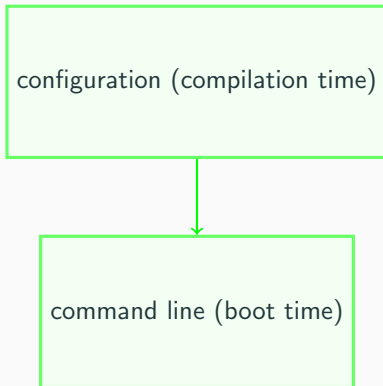
Tracability and detection

MISC

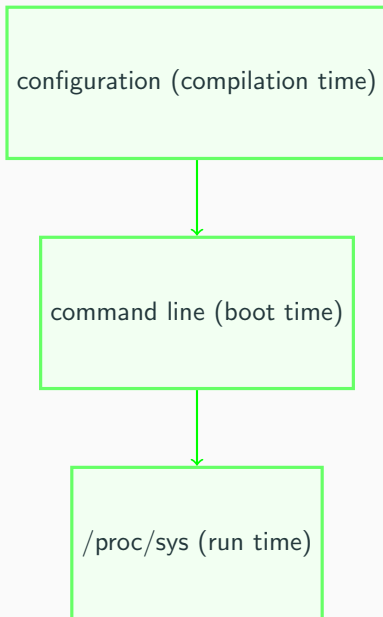
Conclusion

configuration (compilation time)

Kernel options order



Kernel options order



Reducing the kernel attack surface

- Why compiling its own kernel?

Reducing the kernel attack surface

- Why compiling its own kernel?
- disabling unused features
 - hence reducing attack surface

Reducing the kernel attack surface

- Why compiling its own kernel?
- disabling unused features
 - hence reducing attack surface
- enabling optional security features
 - making attackers' life harder

Maintaining a custom kernel has significant maintenance costs:

- monitoring kernel changes
- CI/CD for kernel image
- deployment of custom kernels
- debug

References

- the kernel source code
- LWN.net
- Linux Kernel Self-Protection Project

Kernel configuration file

- .config file with CONFIG_* directives

Kernel configuration file

- .config file with CONFIG_* directives
- many security-related directives, focusing here on ANSSI recommendations

Compile a kernel

```
$ make defconfig
# for Qemu
$ make kvm_guest.config
# customizing configuration
$ make menuconfig

$ make -j<nb_processors>
$ make -j<nb_processors> modules
```

Introduction

Secure bootloader chain

Kernel parameters

- Kernel configuration

- Kernel command line

- Sysctl interface

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

The kernel command line

- parameters given to the kernel at boot time

The kernel command line

- parameters given to the kernel at boot time
- protect your boot manager to avoid tampering of the command line
 - eg: password in GRUB

The kernel command line

- parameters given to the kernel at boot time
- protect your boot manager to avoid tampering of the command line
 - eg: password in GRUB
- once booted, located in `/proc/cmdline`

kernel command line example

Showing kernel command line

```
$ cat /proc/cmdline  
root=PARTUUID=80e54d76-7244-3347-b610-d2ba0cb124ab rw  
add_efi_memmap archisobasedir=arch archisolabel=  
ARCH_202107 initrd=boot\amd-ucode.img initrd=boot\  
initramfs-linux.im
```


Introduction

Secure bootloader chain

Kernel parameters

- Kernel configuration

- Kernel command line

- Sysctl interface

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

- parameters given to the kernel at run time

- parameters given to the kernel at run time
- pseudo file system exposed at `/proc/sys`, writable only by privileged users

Sections of interest for us:

- `/proc/sys/debug`: debugging features
- `/proc/sys/net`: network-related parameters
- `/proc/sys/kernel`: general kernel stuff, LSM and security

Querying /proc/sys

Query /proc/sys

```
$ cat /proc/sys/net/ipv4/ip_forward  
0
```

```
$ sysctl net.ipv4.ip_forward  
net.ipv4.ip_forward = 0
```

```
$ sysctl -a  
abi.vsyscall32 = 1  
debug.exception-trace = 1  
debug.kprobes-optimization = 1  
[...]
```

- `https://docs.kernel.org/admin-guide/sysctl/index.html`
- `https://wiki.archlinux.org/title/Kernel_parameters`

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

- Modules

- Linux Security Modules (LSM)

- Memory protections

- eBPF

- Cryptography API

- Network protections

- Process isolation

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

Several mechanisms came out throughout the years:

- module signature
- Linux Security modules (LSM)
- eBPF filters
- memory protections: DEP, ASLR...
- cryptography APIs
- ...

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

- Modules

- Linux Security Modules (LSM)

- Memory protections

- eBPF

- Cryptography API

- Network protections

- Process isolation

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

Kernel modules reminder

- piece of code that can be loaded into the kernel on demand

Kernel modules reminder

- piece of code that can be loaded into the kernel on demand
- ELF binary with `.ko` extension

Kernel modules reminder

- piece of code that can be loaded into the kernel on demand
- ELF binary with `.ko` extension
- located in `/lib/modules/<kernel_version>`

- attacker is root, thus can insert kernel modules

Threat model

- attacker is root, thus can insert kernel modules
- attacker can write new kernel modules somehow

- attacker is root, thus can insert kernel modules
- attacker can write new kernel modules somehow
- how to ensure modules' integrity?

Threat model

- attacker is root, thus can insert kernel modules
- attacker can write new kernel modules somehow
- how to ensure modules' integrity?
 - cryptographic signatures

Kernel module signature

- needs a specific kernel configuration

Configuring signature

```
# Enable module signature support
CONFIG_MODULE_SIG=y
# Require signatures (refuse unsigned)
CONFIG_MODULE_SIG_FORCE=y
# Automatically sign all modules at build
CONFIG_MODULE_SIG_ALL=y
# Use SHA-512 for signing hash
CONFIG_MODULE_SIG_SHA512=y
# Signing key path
CONFIG_MODULE_SIG_KEY="certs/signing_key.pem"
```

Kernel module signature

Checking configuration

```
$ zgrep CONFIG_MODULE_SIG /proc/config.gz
# or grep CONFIG_MODULE_SIG /boot/config-<kernel-version>
CONFIG_MODULE_SIG_FORMAT=y
CONFIG_MODULE_SIG=y
# CONFIG_MODULE_SIG_FORCE is not set
CONFIG_MODULE_SIG_ALL=y
CONFIG_MODULE_SIG_SHA512=y
CONFIG_MODULE_SIG_HASH="sha512"
CONFIG_MODULE_SIG_KEY="certs/signing_key.pem"
CONFIG_MODULE_SIG_KEY_TYPE_ECDSA=y
```

Kernel module signature

kernel has a public keyring in `/proc/keys`

Listing public keys

```
# cat /proc/keys
00604794 |—Q—      2 perm 3f030000      0      0 keyring      _ses: 1
06d62632 |—Q—      1 perm 0b0b0000      0      0 user          invocation_id: 16
084173d5 |—Q—     14 perm 3f030000      0      0 keyring      _ses: 1
084c5e05 |———      1 perm 1f010000      0      0 asymmetri Canonical Ltd. Master
      Certificate Authority: [...]: X509.rsa 8e345a63 []
...
```

Kernel module signature

- useless if signature required is not enforced

Kernel module signature

- useless if signature required is not enforced
- how to deal with third-party drivers?

Kernel module signature

- useless if signature required is not enforced
- how to deal with third-party drivers?
 - manually sign drivers or add vendor key to kernel keyring

Kernel module signature

- useless if signature required is not enforced
- how to deal with third-party drivers?
 - manually sign drivers or add vendor key to kernel keyring
- how to deal with Dynamic Kernel Module Support (DKMS)?

Kernel module signature

- useless if signature required is not enforced
- how to deal with third-party drivers?
 - manually sign drivers or add vendor key to kernel keyring
- how to deal with Dynamic Kernel Module Support (DKMS)?
 - automate the signing process in post-install scripts

Kernel module signature

Manual signature

```
$ openssl req -new -noenc -utf8 -sha256 -days 36500 -batch -x509  
-outform PEM -out kernel_key.pem -keyout kernel_key.pem
```

```
$ hexdump -C efivarfs.ko
```

```
...  
0000ceb0  00 00 00 00 00 00 00 00 08 c3 00 00 00 00 00 00 | .....  
0000cec0  93 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....  
0000ced0  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....  
0000cee0
```

```
$ /usr/src/linux-headers-$(uname -r)/scripts/sign-file sha512 kernel_key.pem  
kernel_cert.pem efivarfs.ko
```

```
$ hexdump -C efivarfs.ko | tail
```

```
0000d090  b2 e0 66 81 c6 e4 32 6e 8a fd de 96 e7 86 ac fd |..f...2n.....|  
0000d0a0  00 00 02 00 00 00 00 00 00 00 01 c0 7e 4d 6f 64 |.....~Mod|  
0000d0b0  75 6c 65 20 73 69 67 6e 61 74 75 72 65 20 61 70 |ule signature ap|  
0000d0c0  70 65 6e 64 65 64 7e 0a |pended~.|  
0000d0c8
```

Takeaway

- efficient but hard to implement in production

Takeaway

- efficient but hard to implement in production
- still vulnerable to existing modules flows

Takeaway

- efficient but hard to implement in production
- still vulnerable to existing modules flows
- Bring Your Own Vulnerable Driver (BYOVD) attacks

Takeaway

- efficient but hard to implement in production
- still vulnerable to existing modules flows
- Bring Your Own Vulnerable Driver (BYOVD) attacks
- always check what modules are installed on a system

- <https://www.kernel.org/doc/html/v4.10/admin-guide/module-signing.html>
- <https://www.cyberask.co.uk/posts/hardening-linux-kernel-modules-via-module-signing-enfo.html>

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

- Modules

- Linux Security Modules (LSM)

- Memory protections

- eBPF

- Cryptography API

- Network protections

- Process isolation

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

- what is a Linux Security Module (LSM)?

Linux Security Modules (LSM)

- what is a Linux Security Module (LSM)?
- API hooking several tactical locations in the kernel

Linux Security Modules (LSM)

- what is a Linux Security Module (LSM)?
- API hooking several tactical locations in the kernel
- mostly used for Mandatory Access Control (MAC)

Linux Security Modules (LSM)

- what is a Linux Security Module (LSM)?
- API hooking several tactical locations in the kernel
- mostly used for Mandatory Access Control (MAC)
- LSM are called one after another but are not all compatible

Linux Security Modules hook implementation

hooks are implemented in `security/security.c`:

```
1882 /**
1883  * security_inode_getattr() — Check if getting file attributes is allowed
1884  * @path: file
1885  *
1886  * Check permission before obtaining file attributes.
1887  *
1888  * Return: Returns 0 if permission is granted.
1889  */
1890 int security_inode_getattr(const struct path *path)
1891 {
1892     if (unlikely(IS_PRIVATE(d_backing_inode(path->dentry))))
1893         return 0;
1894     return call_int_hook(inode_getattr, path);
1895 }
```

Linux Security Modules hook implementation

and later called in relevant parts of the kernel, here `fs/stat.c`:

```
254 int vfs_getattr(const struct path *path, struct kstat *stat,
255                 u32 request_mask, unsigned int query_flags)
256 {
257     int retval;
258
259     retval = security_inode_getattr(path);
260     if (unlikely(retval))
261         return retval;
262     return vfs_getattr_nosec(path, stat, request_mask, query_flags);
263 }
```

Linux Security Modules hook implementation

LSMs can then "hook" it with specific code, eg

security/apparmor/lsm.c:

```
1664 static struct security_hook_list apparmor_hooks[] __ro_after_init = {  
[...]  
1685     LSM_HOOK_INIT( inode_getattr , apparmor_inode_getattr ),
```

Mainline LSM

- AppArmor
- BPF
- Capability
- Integrity Policy Enforcement (IPE)
- Landlock
- LoadPin
- Lockdown
- SELinux
- SafeSetID
- Smack
- Tomoyo
- YAMA

Linux Security modules (LSM)

Checking configured LSMs

```
$ zgrep -E CONFIG.LSM /proc/config.gz
CONFIG_LSM="landlock,lockdown,yama,integrity,bpf" capability,landlock,lockdown,
yama,bpf

$ zgrep -E CONFIG_SECURITY_(SELINUX|APPARMOR|TOMOYO) /proc/config.gz
CONFIG_SECURITY_SELINUX=y
CONFIG_SECURITY_SELINUX_BOOTPARAM=y
CONFIG_SECURITY_SELINUX_DEVELOP=y
CONFIG_SECURITY_SELINUX_AVC_STATS=y
CONFIG_SECURITY_SELINUX_SIDTAB_HASH_BITS=9
[...]
# CONFIG_SECURITY_SELINUX_DEBUG is not set
CONFIG_SECURITY_TOMOYO=y
CONFIG_SECURITY_TOMOYO_MAX_ACCEPT_ENTRY=2048
CONFIG_SECURITY_TOMOYO_MAX_AUDIT_LOG=1024
# CONFIG_SECURITY_TOMOYO_OMIT_USERSPACE_LOADER is not set
CONFIG_SECURITY_TOMOYO_POLICY_LOADER="/usr/bin/tomoyo-init"
CONFIG_SECURITY_TOMOYO_ACTIVATION_TRIGGER="/usr/lib/systemd/systemd"
# CONFIG_SECURITY_TOMOYO_INSECURE_BUILTIN_SETTING is not set
CONFIG_SECURITY_APPARMOR=y
# CONFIG_SECURITY_APPARMOR_DEBUG is not set
CONFIG_SECURITY_APPARMOR_INTROSPECT_POLICY=y
CONFIG_SECURITY_APPARMOR_HASH=y
```

Checking loaded LSMs

```
$ cat /sys/kernel/security/lsm  
capability , landlock , lockdown , yama , bpf
```

- MAC security extension for the kernel

- MAC security extension for the kernel
- "profiles" containing allowed operations for applications

- MAC security extension for the kernel
- "profiles" containing allowed operations for applications
- filesystem access, Linux capabilities, network sockets, DBus messaging, inter-process communication (IPC)...

- security policy is defined next to software, by application maintainers/developpers

- security policy is defined next to software, by application maintainers/developpers
- vulnerabilities in software cannot grant arbitrary rights on arbitrary files anymore

- security policy is defined next to software, by application maintainers/developpers
- vulnerabilities in software cannot grant arbitrary rights on arbitrary files anymore
- it can still grant rights on some files

AppArmor example

Profile /etc/apparmor.d/sbin.syslogd:

```
profile syslogd /{usr/,}{bin,sbin}/syslogd flags=(complain) {  
[...]  
capability sys_tty_config ,  
capability dac_override ,  
capability dac_read_search ,  
capability setuid ,  
capability setgid ,  
capability syslog ,  
  
unix (receive) type=dgram ,  
unix (receive) type=stream ,  
  
/dev/log                               wl ,  
/var/lib/*/dev/log                     wl ,  
/dev/kmsg                              r ,  
/proc/kmsg                             r ,  
  
/dev/tty*                             w ,  
/dev/xconsole                          rw ,  
/etc/syslog.conf                       r ,  
/etc/syslog.d/                         r ,  
/etc/syslog.d/*                        r ,  
/{usr/,}{bin,sbin}/syslogd            rmix ,  
/var/log/**                            rw ,  
@{run}/syslog.pid                     krwl ,  
@{run}/syslogd.pid                    krwl ,  
@{run}/utmp                            rw ,
```

Check AppArmor status

```
# aa-status
apparmor module is loaded.
136 profiles are loaded.
15 profiles are in enforce mode.
   /usr/bin/man
   /usr/bin/pidgin
[...]

# journalctl -x -k --grep=apparmor
kernel: Command line: root=/dev/sda1 lsm=apparmor
kernel: Kernel command line: root=/dev/sda1 lsm=apparmor
kernel: AppArmor: AppArmor initialized
kernel: AppArmor: AppArmor Filesystem Enabled
kernel: AppArmor: AppArmor sha256 policy hashing enabled
systemd[1]: systemd 259-1 running in system mode (+PAM +AUDIT +SELINUX +APPARMOR
      +IMA +IPE +SMACK +SECCOMP +GCRYPT -GNUTLS +OPENSSL +ACL +BLKID +CURL +
      ELFUTILS +FIDO2 >
kernel: audit: type=1400 audit(1785852748.153:2): apparmor="STATUS" operation="
      profile_load" profile="unconfined" name="lpassword" pid=188 comm="
      apparmor_parser"
kernel: audit: type=1400 audit(1785852748.155:3): apparmor="STATUS" operation="
      profile_load" profile="unconfined" name="Discord" pid=190 comm="
      apparmor_parser"
```

AppArmor takeaway

- application-specific

AppArmor takeaway

- application-specific
- no more arbitrary read/writes for attackers on application with profiles

AppArmor takeaway

- application-specific
- no more arbitrary read/writes for attackers on application with profiles
- easier to deploy than other MAC-based LSMs

- Security Enhanced Linux (SELinux), also a MAC implementation (by the NSA)

- Security Enhanced Linux (SELinux), also a MAC implementation (by the NSA)
- defines "labels" corresponding to security contexts on system resources (file, process, port...) and policies

- Security Enhanced Linux (SELinux), also a MAC implementation (by the NSA)
- defines "labels" corresponding to security contexts on system resources (file, process, port...) and policies
 - labels have four elements: user, role, type and category
 - policies defines labels for domain types and transitions

- Security Enhanced Linux (SELinux), also a MAC implementation (by the NSA)
- defines "labels" corresponding to security contexts on system resources (file, process, port...) and policies
 - labels have four elements: user, role, type and category
 - policies defines labels for domain types and transitions
- deny by default policy

- Security Enhanced Linux (SELinux), also a MAC implementation (by the NSA)
- defines "labels" corresponding to security contexts on system resources (file, process, port...) and policies
 - labels have four elements: user, role, type and category
 - policies defines labels for domain types and transitions
- deny by default policy
- system-wide

Check SELinux status

```
# sestatus
SELinux status:                enabled
SELinuxfs mount:              /sys/fs/selinux
SELinux root directory:      /etc/selinux
Loaded policy name:           default
Current mode:                 permissive
Mode from config file:       permissive
Policy MLS status:           enabled
Policy deny_unknown status:   allowed
Memory protection checking:   actual (secure)
Max kernel policy version:    35
```

Check SELinux security contexts

```
# ps auxZ | grep bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 root 350 0.0 0.5 8492
    5288 pts/0 Ss 10:09 0:00 -bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 root 377 0.0 0.2 6736
    2432 pts/0 S+ 10:09 0:00 grep bash

/tmp# ls -lZ test.txt
-rw-r--r--. 1 root root unconfined_u:object_r:user_tmp_t:s0 6 Aug  5 10:10 test.
txt

# semanage port -l | grep 443
http_port_t            tcp            80, 443, 488, 8008, 8009, 8443, 8448
```

Label on files are stored as an extended attribute:

SELinux extended attributes

```
# getfattr -n security.selinux test.txt  
# file: test.txt  
security.selinux="unconfined_u:object_r:user_tmp_t:s0"
```

Check SELinux security contexts

```
# seinfo -x --type=user_tmp_t /etc/selinux/default/policy/policy.35
Types: 1
    type user_tmp_t, file_type, non_auth_file_type, non_security_file_type,
        polymember, tmpfile, ubac_constrained_type, user_home_content_type;

# what files are categorized with a label
# semanage fcontext -l | grep user_tmp_t
/tmp/gconfd-[^/]+                                directory
    unconfined_u:object_r:user_tmp_t:s0
/tmp/gconfd-root                                directory
    unconfined_u:object_r:user_tmp_t:s0

# what domain types can do what to a given label
# sesearch -A -t user_tmp_t | head
allow NetworkManager_t file_type:filesystem getattr;
allow NetworkManager_t user_tmp_t:sock_file { append getattr open write };
allow accountsd_t user_tmp_t:dir { getattr ioctl lock open read search };
allow accountsd_t user_tmp_t:file { getattr ioctl lock open read };
allow admin_mail_t user_tmp_t:dir { getattr ioctl lock open read search };
allow admin_mail_t user_tmp_t:file { append getattr ioctl lock open read write
};
allow aide_t file_type:dir { getattr ioctl lock open read search };
allow aide_t file_type:file { getattr ioctl lock open read };
allow aide_t file_type:lnk_file { getattr read };
```

Check SELinux policy violations

```
# ausearch -m AVC | grep 'denied'
type=AVC msg=audit(1785917367.378:31): avc: denied { read } for pid=252 comm=
="auditctl" name="audit" dev="sda1" ino=34158 scontext=system_u:system_r:
auditctl_t:s0 tcontext=system_u:object_r:auditd_log_t:s0 tclass=dir
permissive=1
type=AVC msg=audit(1785917367.378:32): avc: denied { open } for pid=252 comm=
="auditctl" path="/var/log/audit/audit.log" dev="sda1" ino=17048 scontext=
system_u:system_r:auditctl_t:s0 tcontext=system_u:object_r:auditd_log_t:s0
tclass=file permissive=1
type=AVC msg=audit(1785917367.378:32): avc: denied { read } for pid=252 comm=
="auditctl" name="audit.log" dev="sda1" ino=17048 scontext=system_u:
system_r:auditctl_t:s0 tcontext=system_u:object_r:auditd_log_t:s0 tclass=
file permissive=1
type=AVC msg=audit(1785917367.490:48): avc: denied { getattr } for pid=238
comm="dbus-daemon" name="/" dev="pidfs" ino=1 scontext=system_u:system_r:
system_dbusd_t:s0 tcontext=system_u:object_r:unlabeled_t:s0 tclass=
filesystem permissive=1
type=AVC msg=audit(1785917367.530:55): avc: denied { getattr } for pid=259
comm="systemd-logind" name="/" dev="pidfs" ino=1 scontext=system_u:system_r:
systemd_logind_t:s0 tcontext=system_u:object_r:unlabeled_t:s0 tclass=
filesystem permissive=1
```

- system-wide, deny by default policy

- system-wide, deny by default policy
- very efficient, no more arbitrary read/writes for attackers

- system-wide, deny by default policy
- very efficient, no more arbitrary read/writes for attackers
- hard to deploy and maintain

- `https://docs.kernel.org/admin-guide/LSM/index.html`
- `https://landlock.io`
- `https://wiki.apparmor.net/about/apparmor_vs_selinux`

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

- Modules

- Linux Security Modules (LSM)

- Memory protections

- eBPF

- Cryptography API

- Network protections

- Process isolation

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

Direct Execution Prevention (DEP)

- attackers exploiting memory bugs need to write to pages and execute these

Direct Execution Prevention (DEP)

- attackers exploiting memory bugs need to write to pages and execute these
- with DEP, pages of memory cannot be marked Writable **and** Executable (W+X)

Direct Execution Prevention (DEP)

- attackers exploiting memory bugs need to write to pages and execute these
- with DEP, pages of memory cannot be marked Writable **and** Executable (W+X)
- `.rodata` section is marked RO and `.text` as R+X

Direct Execution Prevention (DEP)

- attackers exploiting memory bugs need to write to pages and execute these
- with DEP, pages of memory cannot be marked Writable **and** Executable ($W+X$)
- `.rodata` section is marked RO and `.text` as $R+X$
- exploits such as buffer overflows can no longer write to $W\oplus X$ pages

Direct Execution Prevention (DEP)

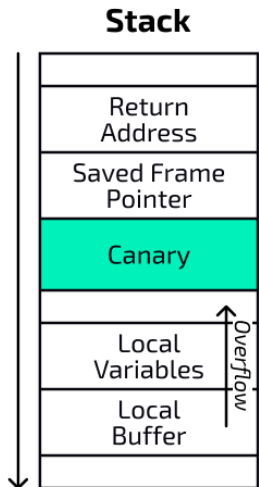
Enable DEP

```
# kernel text and rodata memory will be made read-only, and  
  non-text memory will be made non-executable  
CONFIG_STRICT_KERNEL_RWX=y  
CONFIG_ARCH_OPTIONAL_KERNEL_RWX=y  
CONFIG_ARCH_HAS_STRICT_KERNEL_RWX=y  
CONFIG_DEBUG_WX=y
```

- kernel stack is a target for attackers

- kernel stack is a target for attackers
- stack corruption prevention: canaries

Stack canaries



Enable stack protections

```
# adds -fstack-protector-strong to CC flags
CONFIG_STACKPROTECTOR=y
CONFIG_STACKPROTECTOR_STRONG=y
# adds buffer pages between kernel stacks
CONFIG_VMAP_STACK=y
# detects stack corruption when calling scheduler
CONFIG_SCHED_STACK_END_CHECK=y
```

Direct Memory Access (DMA)

- conceptual bypass of memory checks

Direct Memory Access (DMA)

- conceptual bypass of memory checks
- physical direct memory access by devices such as NICs or graphic cards

Direct Memory Access (DMA)

- conceptual bypass of memory checks
- physical direct memory access by devices such as NICs or graphic cards
- kernel direct memory access by userland programs

Direct Memory Access (DMA)

- conceptual bypass of memory checks
- physical direct memory access by devices such as NICs or graphic cards
- kernel direct memory access by userland programs
- unless absolutely needed, should be disabled

Direct Memory Access (DMA)

Disable Direct Memory Accesses

```
# Disable direct access to physical memory from kernel
#CONFIG.DEVMEM is not set
# Disable direct access to kernel memory from userland
#CONFIG.DEVKMEM is not set
# Disable ACPI dangerous usage that can lead to direct
  physical
  memory write
# CONFIG_ACPI_CUSTOM_METHOD is not set
```

Old memory-related features

Disable old features

```
# Disable VDSO backward compatibility , needed to enable ASLR
# CONFIG_COMPAT_VDSO is not set
# Disable vsyscall table , not needed by libc anymore and can
  serve as
# ROP gadget source
CONFIG_LEGACY_VSYSCALL_NONE =y
CONFIG_LEGACY_VSYSCALL_EMULATE =n
CONFIG_LEGACY_VSYSCALL_XONLY =n
CONFIG_X86_VSYSCALL_EMULATION =n
```

Leaking memory information

- attackers needs to know at what addresses are located kernel structures and code

Leaking memory information

- attackers needs to know at what addresses are located kernel structures and code
- kernel memory adresses should only be available to privileged users

Disable old features

```
# dmesg only readable by privileged users  
CONFIG_SECURITY_DMESG_RESTRICT=y
```

```
# Forbids supplying a virtual ELF core file of the live  
kernel
```

```
#CONFIG_PROC_KCORE is not set
```

Kernel Address Space Layout Randomization (KASLR)

- ASLR randomizes the position of memory addresses for key data areas of userland processes

Kernel Address Space Layout Randomization (KASLR)

- ASLR randomizes the position of memory addresses for key data areas of userland processes
- and for the kernel? KASLR randomizes the address at which the kernel is decompressed in memory

Enable KASLR

```
# Randomize the address of the kernel image
CONFIG_RANDOMIZE_BASE=y
# Randomize the kernel memory sections
CONFIG_RANDOMIZE_MEMORY=y
# Support for randomizing kernel stack offset on syscall
  entry
CONFIG_RANDOMIZE_KSTACK_OFFSET=y
```

Enable KASLR in /proc/sys

```
kernel.randomize_va_space=2
```

Check if KASLR is enabled

```
$ head -n 3 /proc/kallsyms
ffffffff85000000 T srs0_alias_untrain_ret
ffffffff85000000 T _stext
ffffffff85000000 T _text
```

```
# reboot
```

```
$ head -n 3 /proc/kallsyms
ffffffffffb6c00000 T srs0_alias_untrain_ret
ffffffffffb6c00000 T _stext
ffffffffffb6c00000 T _text
```

Control Flow Integrity (CFI)

- with DEP, $W\oplus X$ pages, exploits are forced to use existing executable code (eg: ROP gadgets)
 - control flow hijacking

Control Flow Integrity (CFI)

- with DEP, W $\oplus$ X pages, exploits are forced to use existing executable code (eg: ROP gadgets)
 - control flow hijacking
- avoid redirection of execution by ensuring return addresses are the expected ones

Control Flow Integrity (CFI)

- with DEP, W $\oplus$ X pages, exploits are forced to use existing executable code (eg: ROP gadgets)
 - control flow hijacking
- avoid redirection of execution by ensuring return addresses are the expected ones
- control flow paths defined at compile time

Control Flow Integrity (CFI) methods

- Software: Clang kernel CFI: – `fsanitize=kcfi`
 - hash of the function pointer's signature type stored before function implementation and checked when making indirect calls
 - hopefully merged in GCC 17 too

Control Flow Integrity (CFI) methods

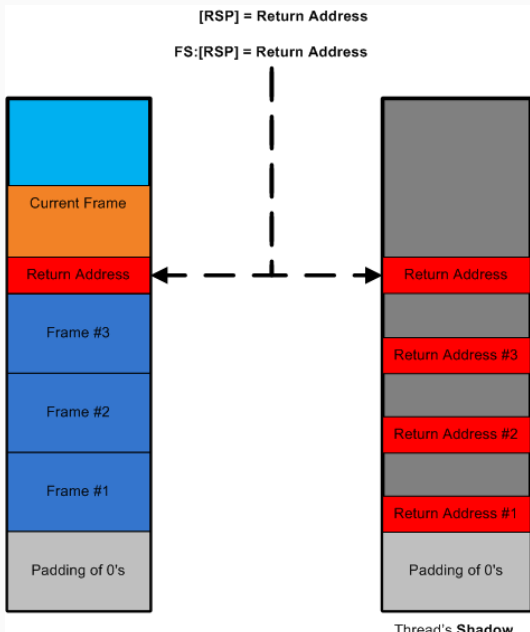
- Software: Clang kernel CFI: `-fsanitize=kcfi`
 - hash of the function pointer's signature type stored before function implementation and checked when making indirect calls
 - hopefully merged in GCC 17 too
- Hardware shadow stacks: Intel Control-Flow Enforcement Technology (CET)

Control Flow Integrity (CFI)

Enable shadow stacks

```
# only working if hardware supports it  
CONFIG_X86_USER_SHADOW_STACK=y
```

Control Flow Integrity (CFI) methods



- <https://lwn.net/Articles/422487>
- <https://messervices.cyber.gouv.fr/guides/recommandations-de-securite-relatives-un-systeme-gnuli>
- <https://lwn.net/Articles/569635/>
- <https://lwn.net/Articles/810077/>
- <https://lwn.net/Articles/1056601/>

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

- Modules

- Linux Security Modules (LSM)

- Memory protections

- eBPF

- Cryptography API

- Network protections

- Process isolation

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

- extended Berkeley Packet Filters (eBPF)

- extended Berkeley Packet Filters (eBPF)
- originally: filters for network packets within the kernel
 - name not relevant anymore

- extended Berkeley Packet Filters (eBPF)
- originally: filters for network packets within the kernel
 - name not relevant anymore
- sort of virtual machine with specific instructions in the kernel

- extended Berkeley Packet Filters (eBPF)
- originally: filters for network packets within the kernel
 - name not relevant anymore
- sort of virtual machine with specific instructions in the kernel
- restricted environment, kernel verifier, result stored in userspace-available maps

- extended Berkeley Packet Filters (eBPF)
- originally: filters for network packets within the kernel
 - name not relevant anymore
- sort of virtual machine with specific instructions in the kernel
- restricted environment, kernel verifier, result stored in userspace-available maps
- used as an alternative to develop kernel code or modules

- Cilium
- Falco
- Tetragon
- ...

1. coder codes an eBPF program

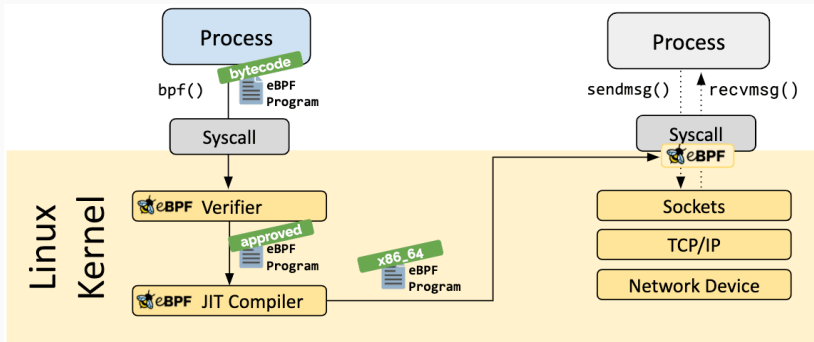
1. coder codes an eBPF program
2. compiler produces an eBPF bytecode

1. coder codes an eBPF program
2. compiler produces an eBPF bytecode
3. coder codes and compiles a userland program to load eBPF object file

1. coder codes an eBPF program
2. compiler produces an eBPF bytecode
3. coder codes and compiles a userland program to load eBPF object file
4. kernel verify and attach eBPF program to either: kprobe, tracepoint, LSM hook...

1. coder codes an eBPF program
2. compiler produces an eBPF bytecode
3. coder codes and compiles a userland program to load eBPF object file
4. kernel verify and attach eBPF program to either: kprobe, tracepoint, LSM hook...
5. when attach point is called, eBPF program is called too

1. coder codes an eBPF program
2. compiler produces an eBPF bytecode
3. coder codes and compiles a userland program to load eBPF object file
4. kernel verify and attach eBPF program to either: kprobe, tracepoint, LSM hook...
5. when attach point is called, eBPF program is called too
6. both eBPF program and userland one write to maps



4

⁴<https://ebpf.io/fr-fr/what-is-ebpf/>

Does eBPF helps to secure Linux?

Does eBPF helps to secure Linux?

- yes: security checks close to kernel code, fast development and performance of security tools

Does eBPF helps to secure Linux?

- yes: security checks close to kernel code, fast development and performance of security tools
- and no: broaden kernel attack surface, new ingress vector

/proc/sys recommendations

```
# kernel BPF restricted to privileged users
kernel.unprivileged_bpf_disabled =1
# Enables hardening for the BPF JIT compiler for all users
net.core.bpf_jit_harden=2
```

- <https://lwn.net/Articles/740157/>
- <https://ebpf.io/fr-fr/what-is-ebpf/>

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

- Modules

- Linux Security Modules (LSM)

- Memory protections

- eBPF

- Cryptography API

- Network protections

- Process isolation

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

- used for cryptographic operations from both kernel and userland programs: hash, cipher, RNG...

⁵<https://docs.kernel.org/crypto/userspace-if.html>

- used for cryptographic operations from both kernel and userland programs: hash, cipher, RNG...
- lots of algorithms implemented and linked directly in the kernel

⁵<https://docs.kernel.org/crypto/userspace-if.html>

- used for cryptographic operations from both kernel and userland programs: hash, cipher, RNG...
- lots of algorithms implemented and linked directly in the kernel
- userland access is now deprecated⁵

⁵<https://docs.kernel.org/crypto/userspace-if.html>

Check kernel crypto API

```
$ cat /proc/crypto
name          : ccm(aes)
driver        : ccm_base(ctr-aes-aesni-avx,cbcmac-aes-lib)
module        : ccm
priority      : 400
refcnt        : 3
selftest      : passed
internal      : no
type          : aead
async         : no
blocksize     : 1
ivsize        : 16
maxauthsize   : 16
geniv         : <none>
[...]
```

- kernelland users: dm-crypt, IPSec, 802.11...
- userland users: depends

- kernelland users: dm-crypt, IPSec, 802.11...
- userland users: depends
- userland API controlled by CONFIG_CRYPT_USER_API_*

- kernelland users: dm-crypt, IPSec, 802.11...
- userland users: depends
- userland API controlled by CONFIG_CRYPT_USER_API_*
- large attack surface in the kernel
- many vulnerabilities found throughout the years

CopyFail CVE-2026-31431

- flaw in `algif_aead` crypto API module, used by IPSec, allow to write past a buffer size boundary by 4 bytes
- performance optimization (in-place copy) allowing source and destination buffers for AEAD operations to be shared

- flaw in `algif_aead` crypto API module, used by IPSec, allow to write past a buffer size boundary by 4 bytes
- performance optimization (in-place copy) allowing source and destination buffers for AEAD operations to be shared
- mixing a 4-bytes writing primitive in the page cache with the `splice ()` system call gives an arbitrary file write

- flaw in `algif_aead` crypto API module, used by IPSec, allow to write past a buffer size boundary by 4 bytes
- performance optimization (in-place copy) allowing source and destination buffers for AEAD operations to be shared
- mixing a 4-bytes writing primitive in the page cache with the `splice ()` system call gives an arbitrary file write
- which gives a LPE

- flaw in `algif_aead` crypto API module, used by IPSec, allow to write past a buffer size boundary by 4 bytes
- performance optimization (in-place copy) allowing source and destination buffers for AEAD operations to be shared
- mixing a 4-bytes writing primitive in the page cache with the `splice ()` system call gives an arbitrary file write
- which gives a LPE
- bug was sitting there since 2017

- how to mitigate CopyFail?

- how to mitigate CopyFail?
- disable IPSec? Bug still there

- how to mitigate CopyFail?
- disable IPSec? Bug still there
- disable crypto API in the kernel, yes but potential breaks

- if it is deprecated what should we do?

⁶https://www.schneier.com/essays/archives/1999/03/cryptography_the_imp.html

- if it is deprecated what should we do?
 - disable userland access in kernel configuration
 - use well-known userland implementation of crypto algorithms such as OpenSSL
 - do not implement your own crypto⁶

⁶https://www.schneier.com/essays/archives/1999/03/cryptography_the_imp.html

Hardening crypto API access

```
$ grep CONFIG_CRYPT_USER_API_ .config
# CONFIG_CRYPT_USER_API_HASH is not set
# CONFIG_CRYPT_USER_API_SKCIPHER is not set
# CONFIG_CRYPT_USER_API_RNG is not set
# CONFIG_CRYPT_USER_API_AEAD is not set
```

- `https://docs.kernel.org/crypto/userspace-if.html`
- `https://xint.io/blog/copy-fail-linux-distributions`

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

- Modules

- Linux Security Modules (LSM)

- Memory protections

- eBPF

- Cryptography API

- Network protections

- Process isolation

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

- network is among the first things attackers checks

- network is among the first things attackers checks
- expose only necessary ports

- network is among the first things attackers checks
- expose only necessary ports
- kernel needs packet filtering software

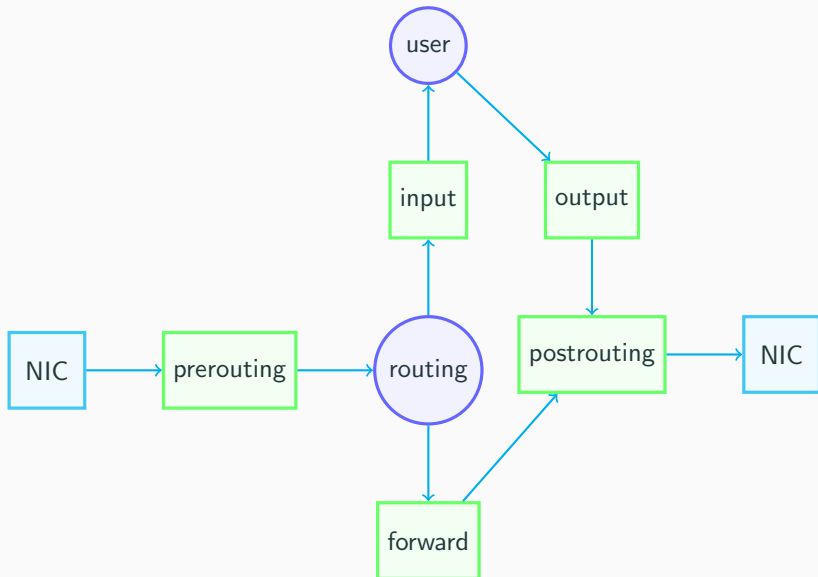
- kernel implementation: netfilter

- kernel implementation: netfilter
- userland tools: iptables and nftables
 - main difference is history and code duplication, but both rely on netfilter
 - we will focus on nftables

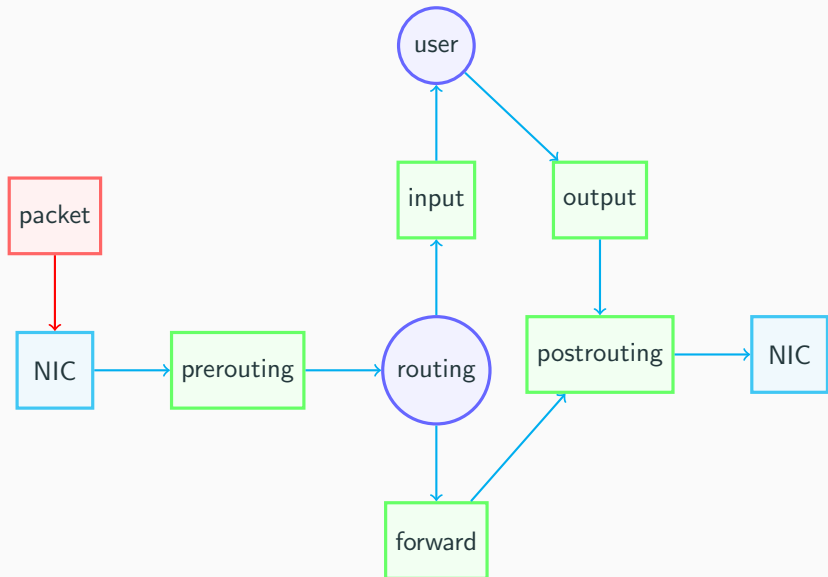
- netfilter "hooks" network functions in kernel code along packet's path

- netfilter "hooks" network functions in kernel code along packet's path
- userland programs such as nftables can add rules that will be proceeded by these hooks

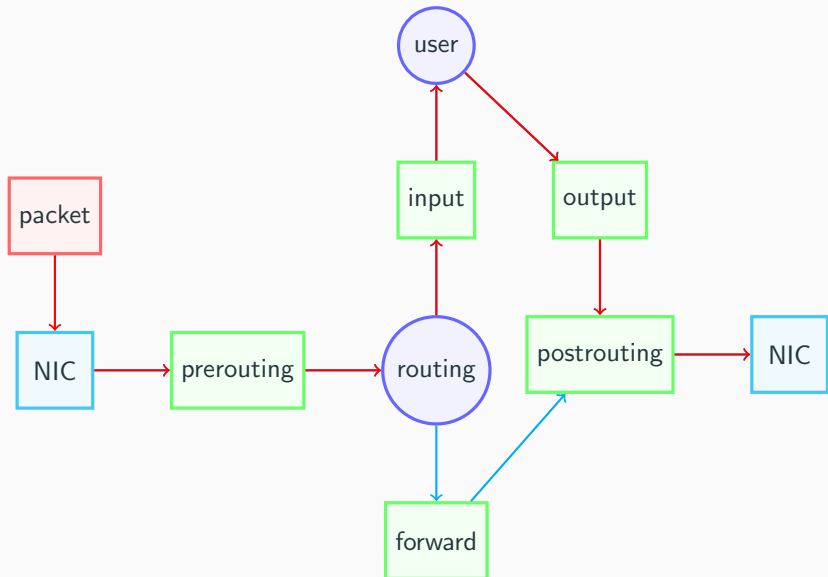
Netfilter flow



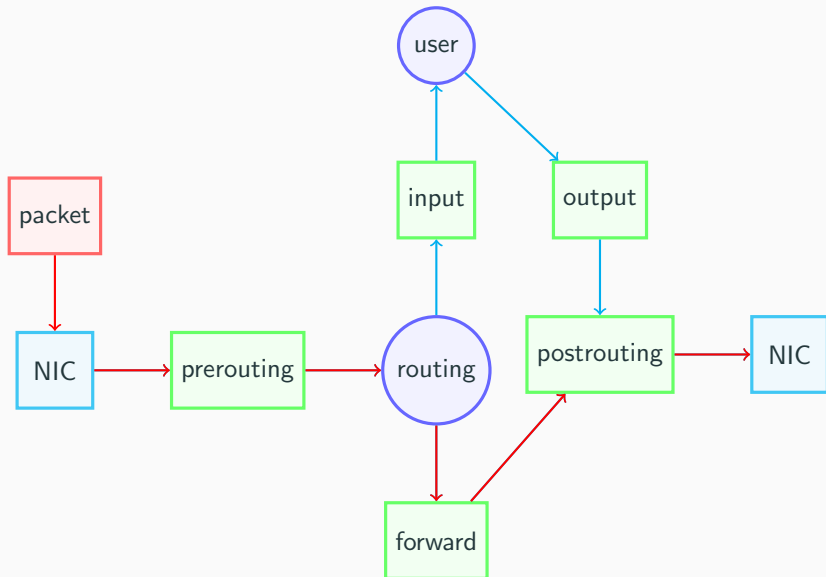
Netfilter flow



Netfilter flow



Netfilter flow



nftables example

nftables.conf workstation

```
#!/sbin/nft -f
flush ruleset

table ip filter {
    chain input {
        type filter hook input priority 0; policy drop;

        ct state {established , related} accept

        iifname "lo" accept
    }
    chain output {
        type filter hook output priority 100; policy drop;

        ct state invalid drop comment
        ct state {established , related} accept

        oifname "eth0" tcp dport { 22, 25, 80, 443, 465, 587 } ct state new accept
        oifname "eth0" udp dport 53 ct state new accept
        oifname "lo" accept
    }
    chain forward {
        type filter hook forward priority 0; policy drop;
    }
}
```

check nftables

```
# Check nftables service is up, running and enabled
# systemctl status nftables.service
* nftables.service — nftables
    Loaded: loaded (/usr/lib/systemd/system/nftables.service; enabled; preset:
           enabled)

# Check there are NFT rules
# nft list ruleset
table inet filter {
    chain input {
        type filter hook input priority filter; policy drop
    [...]

# Check that policies are set to "drop"
```

- <https://www.netfilter.org/>
- https://docs.kernel.org/networking/nf_flowtable.html
- <https://www.sobyte.net/post/2022-04/understanding-netfilter-and-iptables/>

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

- Modules

- Linux Security Modules (LSM)

- Memory protections

- eBPF

- Cryptography API

- Network protections

- Process isolation

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

- how to isolate processes from each other on Linux?

- how to isolate processes from each other on Linux?
- control groups (cgroups)
- namespaces

- how to isolate processes from each other on Linux?
- control groups (cgroups)
- namespaces
- main features behind container software such as LXC and Docker

- isolate resource usage of processes
 - memory, CPU, disk I/O, network...

- isolate resource usage of processes
 - memory, CPU, disk I/O, network...
- also used for resource monitoring

- isolate resource usage of processes
 - memory, CPU, disk I/O, network...
- also used for resource monitoring
- possible to expand/restrict resources on the fly

- a process can only be part of one cgroup

- a process can only be part of one cgroup
- cgroups are processes trees attached to *controllers*

- a process can only be part of one cgroup
- cgroups are processes trees attached to *controllers*
- existing controllers
 - cpu
 - io
 - memory
 - ...

Check available cgroups controllers

```
$ cat /sys/fs/cgroup/cgroup.controllers  
cpuset cpu io memory hugetlb pids rdma misc dmem
```

Check a process control groups membership

```
# current shell
$ cat /proc/$$/cgroup
0::/user.slice/user-1000.slice/session-1.scope
```


Check cgroups process tree

```
$ find /sys/fs/cgroup/ -type d
/sys/fs/cgroup/
/sys/fs/cgroup/sys-fs-fuse-connections.mount
/sys/fs/cgroup/sys-kernel-config.mount
/sys/fs/cgroup/sys-kernel-debug.mount
/sys/fs/cgroup/dev-mqueue.mount
/sys/fs/cgroup/user.slice
/sys/fs/cgroup/user.slice/user-1000.slice
/sys/fs/cgroup/user.slice/user-1000.slice/user@1000.service
/sys/fs/cgroup/user.slice/user-1000.slice/user@1000.service/
    session.slice
...
/sys/fs/cgroup/user.slice/user-1000.slice/user@1000.service/
    app.slice
...
/sys/fs/cgroup/user.slice/user-1000.slice/user@1000.service/
    init.scope
/sys/fs/cgroup/user.slice/user-1000.slice/session-1.scope
/sys/fs/cgroup/sys-kernel-tracing.mount
/sys/fs/cgroup/init.scope
```

Check cgroups consumption

```
$ systemd-cgtop
```

CGroup	Tasks	%CPU	Memory	Input/s	Output/s
/	1046	18.9	3.1G	0B	175.1K
user.slice	824	10.3	4.6G	—	—
user.slice/user-1000.slice	824	10.3	4.6G	—	—
user.slice/user-1000.slice/session-1.scope	755	9.8	4.5G	—	—
...					

Also used by:

- systemd
- runc
- docker
- LXC
- YARN

- isolate set of processes to see and interact with only a given subset of resources

Eight namespaces as of today:

- mount: isolate FS
- IPC: isolate inter-process communications such as message queues
- UTS: isolate hostname
- PID: isolate process identifiers
- network: isolate network stack
- user: isolate user identifiers
- time: isolate system time (kernel clocks)
- cgroup: isolate cgroup hierarchy

Check namespaces of a process

```
$ ls -l /proc/self/ns/  
total 0  
lrwxrwxrwx 1 jeff jeff 0 cgroup -> 'cgroup:[4026531835]'  
lrwxrwxrwx 1 jeff jeff 0 ipc -> 'ipc:[4026531839]'  
lrwxrwxrwx 1 jeff jeff 0 mnt -> 'mnt:[4026531832]'  
lrwxrwxrwx 1 jeff jeff 0 net -> 'net:[4026531833]'  
lrwxrwxrwx 1 jeff jeff 0 pid -> 'pid:[4026531836]'  
lrwxrwxrwx 1 jeff jeff 0 pid_for_children -> 'pid:[4026531836]'  
lrwxrwxrwx 1 jeff jeff 0 time -> 'time:[4026531834]'  
lrwxrwxrwx 1 jeff jeff 0 time_for_children -> 'time:[4026531834]'  
lrwxrwxrwx 1 jeff jeff 0 user -> 'user:[4026531837]'  
lrwxrwxrwx 1 jeff jeff 0 uts -> 'uts:[4026531838]'
```

List processes in the same namespace

```
$ lsns -t net -T
```

NS	TYPE	NPROCS	PID	USER	NETNSID	NSFS	COMMAND
4026531833	net	37	594	jeff	unassigned	—zsh	
4026532721	net	1	21789	jeff	unassigned	/usr/lib/firefox/firefox	[...]
4026532763	net	1	1236	jeff	unassigned	/usr/lib/firefox/firefox	[...]

```
$ sudo lsns -t net -T
```

NS	TYPE	NPROCS	PID	USER	NETNSID	NSFS	COMMAND
4026531833	net	251	1	root	unassigned	/usr/lib/systemd/systemd	[...]
4026532583	net	1	1145	rtkit	unassigned	/usr/lib/rtkit-daemon	
4026532646	net	1	1186	polkitd	unassigned	/usr/lib/polkit-1/polkitd	[...]
4026532721	net	1	21789	jeff	unassigned	/usr/lib/firefox/firefox	[...]
4026532763	net	1	1236	jeff	unassigned	/usr/lib/firefox/firefox	[...]

Namespaces use from userland is dictated by three system calls:

- `clone()`: when creating a new process, move it to a new namespace using `CLONE_NEW*` flags

Namespaces use from userland is dictated by three system calls:

- `clone()`: when creating a new process, move it to a new namespace using `CLONE_NEW*` flags
- `unshare()`: move calling process to a new namespace

Namespaces use from userland is dictated by three system calls:

- `clone()`: when creating a new process, move it to a new namespace using `CLONE_NEW*` flags
- `unshare()`: move calling process to a new namespace
- `setns()`: move calling process to a given existing namespace

- isolate mount tree and thus filesystem hierarchy view

- isolate mount tree and thus filesystem hierarchy view
- much more secure than `chroot()`

- isolate mount tree and thus filesystem hierarchy view
- much more secure than `chroot()`
- eg: processes in child namespace cannot see mountpoints belonging to parent

- isolate mount tree and thus filesystem hierarchy view
- much more secure than `chroot()`
- eg: processes in child namespace cannot see mountpoints belonging to parent
- still need to `pivot_root` inside a dedicated FS

Let's imagine you are in a container:

- What if you share the mount namespace with the host?

Let's imagine you are in a container:

- What if you share the mount namespace with the host?
 - escalate privileges with write access to host files (eg: write a new key to `/root/.ssh/authorized_keys` and SSH to the host)

Let's imagine you are in a container:

- What if you share the mount namespace with the host?
 - escalate privileges with write access to host files (eg: write a new key to `/root/.ssh/authorized_keys` and SSH to the host)
- What if you have a different mount namespace than the host?
 - if it exists, abuse mountpoints pointing to the host (eg: symbolic links in mountbinds)

- each net namespace has its own network stack, routing tables, netfilter rules, socket table, /proc/net directory...

- each net namespace has its own network stack, routing tables, netfilter rules, socket table, `/proc/net` directory...
- eg: processes in different net namespaces can thus bind to the 80 port

isolate process in network namespace

```
# Host namespace
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
    default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
[...]
```

```
2: wlp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    group default qlen 1000
    link/ether ca:fe:ca:fe:ca:fe brd ff:ff:ff:ff:ff:ff
    altname wlxac12032d691e
    inet 192.168.1.53/24 brd 192.168.1.255 scope global dynamic noprefixroute
        wlp1s0
[...]
```

```
$ ping -c 1 google.com
PING google.com (64.233.178.102) 56(84) bytes of data:
64 bytes from lcyulk-in-f102.1e100.net (64.233.178.102): icmp_seq=1 ttl=107 time
=95.1 ms
[...]
```

```
# unshare --net bash
$ ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

$ ping google.com
ping: google.com: Temporary failure in name resolution
```

Let's imagine you are in a container:

- What if you share the network namespace with the host?

Let's imagine you are in a container:

- What if you share the network namespace with the host?
 - network reconnaissance on other containers, VMs, physical hosts...
 - bypass firewall restrictions

Let's imagine you are in a container:

- What if you share the network namespace with the host?
 - network reconnaissance on other containers, VMs, physical hosts...
 - bypass firewall restrictions
- What if you have a different network namespace than the host?
 - check if you have `CAP_NET_ADMIN` capability

- each user namespace has its own user and group ID number space

- each user namespace has its own user and group ID number space
- UID and GID are different inside or outside of a given namespace

- each user namespace has its own user and group ID number space
- UID and GID are different inside or outside of a given namespace
- allow for processes to have an unprivileged user ID outside a namespace but UID 0 (root) inside it

Check current user namespace

```
# Host namespace
$ cat /proc/$$/uid_map
    0          0 429496729

# unshare --user --map-root-user bash
# cat /proc/$$/uid_map
    0          1000          1
```

Let's imagine you are in a container:

- What if you share the user namespace with the host?

Let's imagine you are in a container:

- What if you share the user namespace with the host?
 - if you can escalate privileges to UID 0, you are root of the host

Let's imagine you are in a container:

- What if you share the user namespace with the host?
 - if you can escalate privileges to UID 0, you are root of the host
- What if you have a different user namespace than the host?
 - if you escape, you are still an unprivileged user on the host

Let's imagine you are in a container:

- What if you share the user namespace with the host?
 - if you can escalate privileges to UID 0, you are root of the host
- What if you have a different user namespace than the host?
 - if you escape, you are still an unprivileged user on the host
 - if you are mapped to UID 0 in the container, you have access to more kernel features

- unprivileged users reaching kernel features reserved to privileged ones led to several vulnerabilities findings. Eg:
 - CVE-2022-0185: heap buffer overflow in FS context subsystem
 - CVE-2024-1086: use after free in netfilter
 - CVE-2026-63917: use after free in IPv6 virtual tunnel interface (VTI)

- unprivileged users reaching kernel features reserved to privileged ones led to several vulnerabilities findings. Eg:
 - CVE-2022-0185: heap buffer overflow in FS context subsystem
 - CVE-2024-1086: use after free in netfilter
 - CVE-2026-63917: use after free in IPv6 virtual tunnel interface (VTI)
- user namespace is a security barrier, but only compared to running privileged containers

Going further

- <https://docs.kernel.org/admin-guide/cgroup-v1/cgroups.html>
- <https://docs.kernel.org/admin-guide/cgroup-v2.html>
- <https://chrisdown.name/talks/cgroupv2/cgroupv2-fosdem.pdf>
- <https://kernel-internals.org/cgroups/namespaces/>
- https://lwn.net/Articles/531114/#series_index
- <https://edera.dev/stories/user-namespaces-are-not-a-security-boundary>

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Partitioning

Disk encryption

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Partitioning

Disk encryption

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

Why partitioning?

- separating usage on disk (eg: user data and system binaries)

Why partitioning?

- separating usage on disk (eg: user data and system binaries)
- avoiding denial of service of a full disk (eg: `/var/log` filling up `/`)

Why partitioning?

- separating usage on disk (eg: user data and system binaries)
- avoiding denial of service of a full disk (eg: `/var/log` filling up `/`)
- using custom file systems (ext4, XFS...)

Why partitioning?

- separating usage on disk (eg: user data and system binaries)
- avoiding denial of service of a full disk (eg: `/var/log` filling up `/`)
- using custom file systems (ext4, XFS...)
- applying security restrictions per partition

- when mounting a partition, security flags can be applied
- most useful ones are:
 - nodev: ignore device block files
 - noexec: ignore execution rights
 - nosuid: ignore SetUID and SetGID bits on files

Before installing a particular system, ask yourself:

- what should be partitioned separately and why?

Before installing a particular system, ask yourself:

- what should be partitioned separately and why?
- which mount flags should be applied when mounting these partitions?

partition schema

mountpoint	options	description
/	none	root
/boot	nosuid,nodev,noexec	boot partition
/opt	nosuid,nodev	additional SW
/tmp	nosuid,nodev,noexec	temporary files
/srv	nosuid,nodev	served file (web server etc.)
/home	nosuid,nodev,noexec	users HOME
/proc	hidepid=2	system and process info
/usr	nodev	system binaries
/var	nosuid,nodev,noexec	variables files
/var/log	nosuid,nodev,noexec	logs
/var/tmp	nosuid,nodev,noexec	temporary files written to disk

- works only if people respects the UNIX hierarchy⁷

⁷see man hier

- works only if people respects the UNIX hierarchy⁷
- usual errors:
 - /<folder> containing all custom software
 - /home containing executables

⁷see man hier

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Partitioning

Disk encryption

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

Why encrypting?

- protect against "cold" disk attacks

Why encrypting?

- protect against "cold" disk attacks
- protect against physical attackers stealing disks

Why encrypting?

- protect against "cold" disk attacks
- protect against physical attackers stealing disks
- also protect against unattended data leaks
 - recycling disks without wiping data

Why encrypting?

- protect against "cold" disk attacks
- protect against physical attackers stealing disks
- also protect against unattended data leaks
 - recycling disks without wiping data
- also protect against remote attacker with hypervisor rights

Different types of disk encryption:

- Full Disk Encryption (FDE)
- Encrypted container

Different types of disk encryption:

- Full Disk Encryption (FDE)
- Encrypted container

Disk encryption relies on symmetric cryptography

Full Disk Encryption (FDE)

- everything is encrypted on the disk but the MBR or UEFI start blocks

Full Disk Encryption (FDE)

- everything is encrypted on the disk but the MBR or UEFI start blocks
- if hardware-backed, the boot blocks can be encrypted too
 - also called Self Encrypting Drive (SED)

Full Disk Encryption (FDE)

- everything is encrypted on the disk but the MBR or UEFI start blocks
- if hardware-backed, the boot blocks can be encrypted too
 - also called Self Encrypting Drive (SED)
- key is stored in the disk itself, some vendors stores it in the TPM

Full Disk Encryption (FDE)

- everything is encrypted on the disk but the MBR or UEFI start blocks
- if hardware-backed, the boot blocks can be encrypted too
 - also called Self Encrypting Drive (SED)
- key is stored in the disk itself, some vendors stores it in the TPM
- usually AES 128 or 256 bits stored in the disk controller

Full Disk Encryption (FDE)

- everything is encrypted on the disk but the MBR or UEFI start blocks
- if hardware-backed, the boot blocks can be encrypted too
 - also called Self Encrypting Drive (SED)
- key is stored in the disk itself, some vendors stores it in the TPM
- usually AES 128 or 256 bits stored in the disk controller
- better performance, transparent for the disk user

HW-based Full Disk Encryption (FDE)

Drawbacks:

- in cloud environments, does not protect against storage administrators (they have the keys)

HW-based Full Disk Encryption (FDE)

Drawbacks:

- in cloud environments, does not protect against storage administrators (they have the keys)
- more difficult to implement RAID

SW-based Full Disk Encryption (FDE)

- less performance, more flexibility

SW-based Full Disk Encryption (FDE)

- less performance, more flexibility
- keys are (usually) stored on HW chips: TPM

SW-based Full Disk Encryption (FDE)

- less performance, more flexibility
- keys are (usually) stored on HW chips: TPM
- usual suspects:
 - Windows: bitlocker
 - Linux: LUKS (dm-crypt)
 - Mac OS: FileVault, iOS Data Protection

SW-based Full Disk Encryption (FDE)

Drawbacks:

- TPM-only encryption protects against specific scenarios
 - if an attacker steals the TPM, he can decrypt the disk
 - cloud administrator could just boot the machine, wait for decryption and dump memory/volumes

SW-based Full Disk Encryption (FDE)

Drawbacks:

- TPM-only encryption protects against specific scenarios
 - if an attacker steals the TPM, he can decrypt the disk
 - cloud administrator could just boot the machine, wait for decryption and dump memory/volumes
 - **always** have a PIN/passphrase when using TPM-backed encryption
 - also possible to use a physical token (smartcard, ubikey...)

- transparent block device encryption subsystem

- transparent block device encryption subsystem
- use routines from kernel crypto API (rings a bell?) to encrypt block devices

- transparent block device encryption subsystem
- use routines from kernel crypto API (rings a bell?) to encrypt block devices
- *The user can basically specify one of the symmetric ciphers, an encryption mode, a key (of any allowed size), an iv generation mode and then the user can create a new block device in /dev. Writes to this device will be encrypted and reads decrypted. You can mount your filesystem on it as usual or stack dm-crypt device with another device like RAID or LVM volume.*

Check for dm-crypt support

```
$ zgrep -E 'CONFIG_DM_CRYPT' /proc/config.gz  
CONFIG_DM_CRYPT=m
```

```
$ sudo dmsetup targets
```

```
9:57:16 (0)
```

striped	v1.7.0
linear	v1.5.0
error	v1.7.0

- interacting with dm-crypt can be done with low-level `dmsetup` tool

- interacting with dm-crypt can be done with low-level `dmsetup` tool
- prefer `cryptsetup`

- Linux Unified Key Setup

- Linux Unified Key Setup
- platform-independent standard on-disk format for use in various tools

- Linux Unified Key Setup
- platform-independent standard on-disk format for use in various tools
- adds a standardized header before encrypted data

- Linux Unified Key Setup
- platform-independent standard on-disk format for use in various tools
- adds a standardized header before encrypted data
- if the header is corrupted, the data cannot be decrypted

- Linux Unified Key Setup
- platform-independent standard on-disk format for use in various tools
- adds a standardized header before encrypted data
- if the header is corrupted, the data cannot be decrypted
- implemented in `cryptsetup`

- Linux Unified Key Setup
- platform-independent standard on-disk format for use in various tools
- adds a standardized header before encrypted data
- if the header is corrupted, the data cannot be decrypted
- implemented in `cryptsetup`
- allow for advanced features such as:
 - multiple keyslots

Create an encrypted volume

```
$ dd if=/dev/zero of=ext4_volume.img bs=1M count=128
$ sudo cryptsetup luksFormat ./ext4_volume.img
$ hexdump -C ./ext4_volume.img | head
00000000  4c 55 4b 53 ba be 00 02  00 00 00 00 00 00 40 00  |LUKS .....@.|
00000010  00 00 00 00 00 00 00 03  00 00 00 00 00 00 00 00  |.....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
*
00000040  00 00 00 00 00 00 00 00  73 68 61 32 35 36 00 00  |.....sha256..|
```

Create an encrypted volume

```
$ dd if=/dev/zero of=ext4_volume.img bs=1M count=128
$ sudo cryptsetup luksFormat ./ext4_volume.img
$ hexdump -C ./ext4_volume.img | head
00000000  4c 55 4b 53 ba be 00 02  00 00 00 00 00 00 40 00  |LUKS.....@.|
00000010  00 00 00 00 00 00 00 03  00 00 00 00 00 00 00 00  |.....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |.....|
*
00000040  00 00 00 00 00 00 00 00  73 68 61 32 35 36 00 00  |.....sha256..|
```

Open an encrypted volume

```
$ sudo cryptsetup open ./ext4_volume.img mydrive
Enter passphrase for ./ext4_volume.img:
$ ls -lh /dev/mapper/mydrive
lrwxrwxrwx 1 root root 7 Sep 12 13:41 /dev/mapper/mydrive -> ../dm-0
$ sudo mkfs.ext4 /dev/mapper/mydrive
mke2fs 1.47.4 (6-Mar-2025)
Creating filesystem with 28672 4k blocks and 28672 inodes
[...]
$ sudo mount /dev/mapper/mydrive /mnt/usb
$ ls /mnt/usb
lost+found
```


Add key to encrypted volume

```
$ sudo cryptsetup luksAddKey ./ext4_volume.img  
Enter any existing passphrase:  
Enter new passphrase for key slot:  
Verify passphrase:
```

Security with LUKS:

- carefully choose your passphrases

Security with LUKS:

- carefully choose your passphrases
- do not use shared passphrase across several devices
 - unless you need a breakglass passphrase

Security with LUKS:

- carefully choose your passphrases
- do not use shared passphrase across several devices
 - unless you need a breakglass passphrase
- use MFA with a USB stick, a smartcard, a yubikey...

- `https://gitlab.com/cryptsetup/cryptsetup/-/wikis/DMCCrypt`
- `https://man7.org/linux/man-pages/man8/cryptsetup.8.html`

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

 Pluggable Authentication Modules (PAM)

 SSH

 Kerberos realm

Autorisation

Tracability and detection

MISC

Conclusion

- verifying the user is who he claims to be (ie verifying identity)

- verifying the user is who he claims to be (ie verifying identity)
- the more MFA there is, the better

- verifying the user is who he claims to be (ie verifying identity)
- the more MFA there is, the better
- passwords are to be avoided when possible
 - prefer cryptographic keys / tokens / certificates

How can a user logs in a Linux system?

- Pluggable Authentication Modules (PAM)
- SSH key based authentication
- Delegated identity providers (LDAP/Kerberos)

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

 Pluggable Authentication Modules (PAM)

 SSH

 Kerberos realm

Autorisation

Tracability and detection

MISC

Conclusion

Pluggable Authentication Modules (PAM)

- collection of libraries to provide authentication API to userland programs

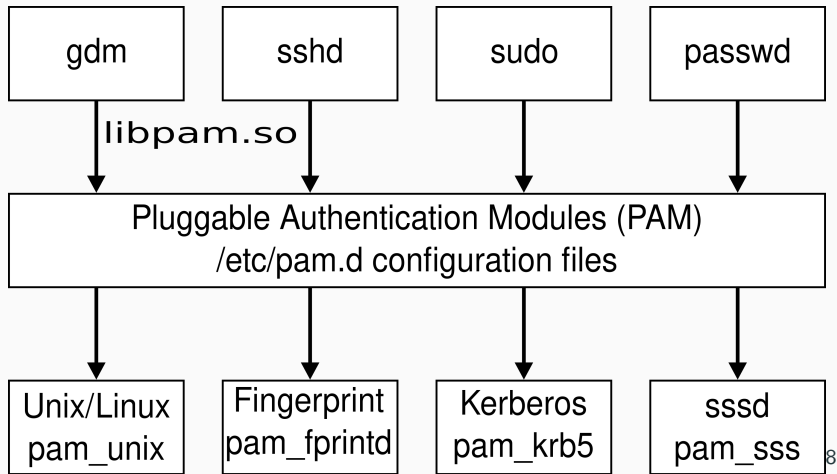
Pluggable Authentication Modules (PAM)

- collection of libraries to provide authentication API to userland programs
- binaries such as `login` or `su` use PAM

Pluggable Authentication Modules (PAM)

- collection of libraries to provide authentication API to userland programs
- binaries such as `login` or `su` use PAM
- system admins can choose how user can authenticate, using or not some modules

Pluggable Authentication Modules (PAM)



⁸<https://www.redhat.com/en/blog/pluggable-authentication-modules-pam>

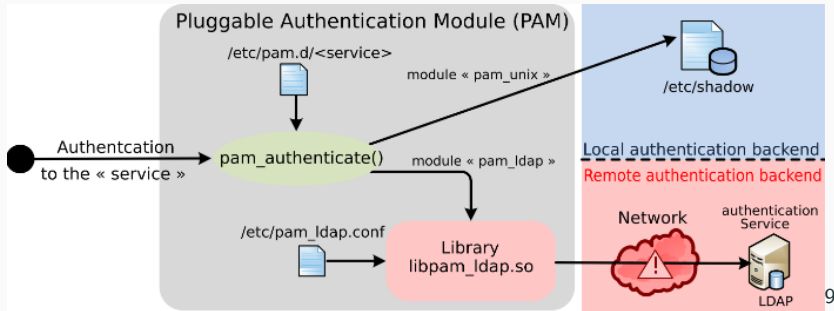
<https://www.redhat.com/en/blog/pluggable-authentication-modules-pam>

Pluggable Authentication Modules (PAM)

PAM has several "mechanisms":

- `auth`: handles the authentication of the requester and establishes the rights of the account
- `account`: checks that the requested account is available
- `session`: relates to session setup and termination
- `password`: used to modify the authentication token associated with an account (expiration or change)

Pluggable Authentication Modules (PAM)



⁹https://messervices.cyber.gouv.fr/documents-guides/linux_configuration-en-v2.pdf

hardening with PAM

Several modules can be used to harden Linux authentication:

- `pam_faillock`: temporarily block an account after too many unsuccessful login attempts
- `pam_time`: restrict logins to specific time slots
- `pam_passwdqc`: password policy enforcement
- `pam_wheel`: restrict root access to a given group
- `pam_mktemp`: enforce per-user `/tmp` private folders
- `pam_namespace`: enforce per-user namespace

hardening with PAM

Hardening with PAM examples

```
# Password complexity
$ cat /etc/pam.d/passwd
password required pam_pwquality.so minlen =12 minclass =3 \
    dcredit =0 ucredit =0 lcredit =0 \
    ocredit =0 maxrepeat =1

# Delay after too many unsuccessful login attempts
$ cat /etc/pam.d/login
$ cat /etc/pam.d/sshd
auth required pam_faillock.so deny=3 unlock_time =300
```

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

- Pluggable Authentication Modules (PAM)

- SSH

- Kerberos realm

Autorisation

Tracability and detection

MISC

Conclusion

- most used administration protocol on UNIX systems

- most used administration protocol on UNIX systems
- must be hardened accordingly

- authentication entirely handled by the `sshd` binary if `UsePam=no`

- authentication entirely handled by the `sshd` binary if `UsePam=no`
- authentication when using crypto keys does not go through PAM auth mechanism even if `UsePam=yes`
 - still goes through account, session though

- enforce protocol version 2
- do not log in as root
- do not use password authentication
- use secure key sizes and crypto algorithms
- disable unused features (TCP forwarding, X11 forwarding...)
- if possible, restrict who can connect via SSH with an allow-list
- disconnect idle connections

SSH server hardening

```
$ cat /etc/ssh/sshd_config
Protocol 2
AllowAgentForwarding no
AllowTcpForwarding no
GatewayPorts no
X11Forwarding no
PermitRootLogin no
AllowUsers myadmin
PasswordAuthentication no
PermitEmptyPasswords no
PubkeyAuthentication yes
MaxAuthTries 5
ClientAliveInterval 3600
ClientAliveCountMax 3
LogLevel VERBOSE
# Careful with algorithms, depends on sshd version
KexAlgorithms sntrup761x25519-sha512@openssh.com,curve25519-sha256,curve25519-sha256@libssh.org,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,umac-128-etm@openssh.com
HostKeyAlgorithms ssh-ed25519,ssh-ed25519-cert-v01@openssh.com,rsa-sha2-512,rsa-sha2-256
```

SSH client key generation

```
$ ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519
$ chmod 600 ~/.ssh/id_ed25519
$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa
$ chmod 600 ~/.ssh/id_rsa
```

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

- Pluggable Authentication Modules (PAM)

- SSH

- Kerberos realm

Autorisation

Tracability and detection

MISC

Conclusion

- when Linux systems are part of a Kerberos realm, authentication is delegated to KDC
 - eg: when they are part of an active directory domain

- when Linux systems are part of a Kerberos realm, authentication is delegated to KDC
 - eg: when they are part of an active directory domain
- configuration in `/etc/krb5.conf`

- when Linux systems are part of a Kerberos realm, authentication is delegated to KDC
 - eg: when they are part of an active directory domain
- configuration in `/etc/krb5.conf`
- users TGTs are stored as files, often in `/tmp/krb5cc_*`

- when Linux systems are part of a Kerberos realm, authentication is delegated to KDC
 - eg: when they are part of an active directory domain
- configuration in `/etc/krb5.conf`
- users TGTs are stored as files, often in `/tmp/krb5cc_*`
- service accounts secrets are stored in key tables, often in `/etc/krb5.keytab`

hardening kerberos client configuration

```
$ cat /etc/krb5.conf
[libdefaults]
    default_realm = EXAMPLE.COM
    # Restrict algorithms to AES
    permitted_enctypes = aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96
    default_tgs_enctypes = aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96
    default_tkt_enctypes = aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96
    # Lower expiration threshold for TGTs
    ticket_lifetime = 10h
    renew_lifetime = 7d
```

- these files must be protected, restrict permissions

- these files must be protected, restrict permissions
- domain-joined Linux hosts must not be a privilege escalation mean in the domain
 - if the hosts are virtualized, check who has hypervisor administration rights

- <https://man7.org/linux/man-pages/man8/pam.8.html>
- <https://docs.rockylinux.org/10/guides/security/pam/>
- https://archive.fosdem.org/2026/events/attachments/LG9LLV-linux-pam_demystified_and_beyond/slides/267560/fosdem_20_w6uru2i.pdf
- https://messervices.cyber.gouv.fr/documents-guides/NT_OpenSSH.pdf

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

- file permissions

- Capabilities

- autorisation models

- Services

Tracability and detection

MISC

Conclusion

- what can an authenticated user do on a system?

- what can an authenticated user do on a system?
- least privilege

How does Linux handles autorisation?

- file permissions

How does Linux handles autorisation?

- file permissions
- setUID, setGID, sticky bits

How does Linux handles autorisation?

- file permissions
- setUID, setGID, sticky bits
- sudo

How does Linux handles autorisation?

- file permissions
- setUID, setGID, sticky bits
- sudo
- capabilities

How does Linux handles autorisation?

- file permissions
- setUID, setGID, sticky bits
- sudo
- capabilities
- MAC: SELinux, AppArmor...

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

- file permissions

- Capabilities

- autorisation models

- Services

Tracability and detection

MISC

Conclusion

- also called Discretionary Access Control (DAC)

- also called Discretionary Access Control (DAC)
- three permissions: read, write and execute

- also called Discretionary Access Control (DAC)
- three permissions: read, write and execute
- three classes: owner, group and others

- also called Discretionary Access Control (DAC)
- three permissions: read, write and execute
- three classes: owner, group and others
- POSIX

- the user mask (umask) determines default permissions assigned on new files and directories

- the user mask (umask) determines default permissions assigned on new files and directories
- ANSSI recommends to set it to 0027
 - read and write for owner
 - read for group

- the user mask (umask) determines default permissions assigned on new files and directories
- ANSSI recommends to set it to 0027
 - read and write for owner
 - read for group
- often the cause for bad permissions with tools such as `ansible`

Changing default umask (per-user)

```
$ tail -n 1 ~/.zshrc
umask 0027
$ echo foo > /tmp/test
$ ls -l /tmp/test
-rw-r----- 1 jeff jeff 4 Sep 13 17:02 /tmp/test
```

- `setUID`, `setGID`: allow to execute a file under its owner identity

- setUID, setGID: allow to execute a file under its owner identity
- sticky bit: on a directory, means only the owner of a file inside it can move, rename or delete the file

- setUID, setGID: allow to execute a file under its owner identity
- sticky bit: on a directory, means only the owner of a file inside it can move, rename or delete the file
- POSIX

- set*ID root binaries have largely be exploited by attackers for years¹⁰

¹⁰<https://gtfobins.org/>

- set*ID root binaries have largely be exploited by attackers for years¹⁰
- main issue is that too many privileges are given to these binaries, how to reduce it?

¹⁰<https://gtfobins.org/>

- set*ID root binaries have largely be exploited by attackers for years¹⁰
- main issue is that too many privileges are given to these binaries, how to reduce it?
 - drop privileges, but that is only palliative
 - Linux capabilities

¹⁰<https://gtfobins.org/>

Find set*ID binaries

```
$ find / -type f -perm -4000 -ls 2>/dev/null
 2400 56 -rwsr-xr-x   1 root root    55688 Oct 10  2025 /usr/bin/umount
   355 20 -rwsr-xr-x   1 root root    18824 Oct 10  2025 /usr/bin/newgrp
25250 84 -rwsr-xr-x   1 root root    84360 Oct 10  2025 /usr/bin/su
  2894 88 -rwsr-xr-x   1 root root    88568 Sep 17  2025 /usr/bin/gpasswd
[...]
```

```
$ find / -type f -perm -2000 -ls 2>/dev/null
 8222 52 -rwxr-sr-x   1 root crontab 52008 Dec 15  2025 /usr/bin/crontab
21320 24 -rwxr-sr-x   1 root mail   23104 Dec 31  2024 /usr/bin/dotlockfile
  2793 32 -rwxr-sr-x   1 root shadow 31256 Sep 17  2025 /usr/bin/expiry
[...]
```

privilege escalation with set*ID file

```
$ find / -type f -perm -4000 -ls 2>/dev/null
105  228 -rwsr-xr-x 1 root  root  233040 Sep 13 10:14 /usr/bin/find

$ find . -exec /bin/bash -p \; -quit
bash-5.3#
```

- check set*id binaries on your systems and remove unused/dangerous ones
- use sticky bit on /tmp

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

- file permissions

- Capabilities

- autorisation models

- Services

Tracability and detection

MISC

Conclusion

- division of root privileges on Linux
- allow users to have more fine-grained capabilities on the system

- division of root privileges on Linux
- allow users to have more fine-grained capabilities on the system
- 41 capabilities as of today

- division of root privileges on Linux
- allow users to have more fine-grained capabilities on the system
- 41 capabilities as of today
- can be assigned to a file or to a process

- different "sets" of capabilities
 - inherited
 - permitted
 - effective
 - bounding
 - ambient
- sets are logically ANDed and ORed following an algorithm to compute effective capabilities when using `execve()`¹¹

¹¹<https://man7.org/linux/man-pages/man7/capabilities.7.html>

non-exhaustive list:

- CAP_SYS_ADMIN
- CAP_SETUID/CAP_SETGID
- CAP_SYS_PTRACE
- CAP_SYS_MODULE
- CAP_CHOWN
- CAP_FOWNER
- CAP_DAC_READ_SEARCH
- CAP_DAC_OVERRIDE
- CAP_SETFCAP

listing process capabilities

```
$ cp $(which zsh) .  
$ sudo setcap cap_net_raw=eip ./zsh  
$ ./zsh  
$ cat /proc/$$/status | grep Cap  
CapInh: 0000000800000000  
CapPrm: 0000000000002000  
CapEff: 0000000000002000  
CapBnd: 000001 ffffffff  
CapAmb: 0000000000000000  
$ capsh —decode=0000000000002000  
0x0000000000002000=cap_net_raw
```

capabilities privilege escalation

```
$ cp $(which python) .  
$ sudo setcap cap_setuid=eip ./python  
./python -c 'import os; os.setuid(0);os.system("/bin/bash")  
;'
```

[root@machine]#

dangerous capabilities

- capabilities are potent
- respect the least privilege principle

dangerous capabilities

- capabilities are potent
- respect the least privilege principle
 - do not give a process unneeded capabilities
 - same for containers

running docker container with least privilege

```
$ docker run --cap-drop=ALL --cap-add=NET_BIND_SERVICE -p  
80:80 nginx:alpine
```

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

- file permissions

- Capabilities

- autorisation models

- Services

Tracability and detection

MISC

Conclusion

- Discretionary Access Control (DAC)
 - each element handles its own permissions: UNIX files

- Discretionary Access Control (DAC)
 - each element handles its own permissions: UNIX files
- Mandatory Access Control (MAC)
 - central policies handles permissions for the entire system: SELinux

- Discretionary Access Control (DAC)
 - each element handles its own permissions: UNIX files
- Mandatory Access Control (MAC)
 - central policies handles permissions for the entire system: SELinux
- MAC is considered more secure

- Discretionary Access Control (DAC)
 - each element handles its own permissions: UNIX files
- Mandatory Access Control (MAC)
 - central policies handles permissions for the entire system: SELinux
- MAC is considered more secure
- but DAC and MAC are complementary on Linux

- Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC)

- Role-Based Access Control (RBAC)
- defining roles and assigning it to users

Role-Based Access Control (RBAC)

- Role-Based Access Control (RBAC)
- defining roles and assigning it to users
- preferred these days as less strict than MAC but stricter than DAC

Role-Based Access Control (RBAC)

- Role-Based Access Control (RBAC)
- defining roles and assigning it to users
- preferred these days as less strict than MAC but stricter than DAC
- eg: Kubernetes

- What model should we choose?

- What model should we choose?
- depends on the security need

- What model should we choose?
- depends on the security need
- but DAC + MAC is a good fit on most systems

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

- file permissions

- Capabilities

- autorisation models

- Services

Tracability and detection

MISC

Conclusion

- privileges given to services must be chosen carefully

- privileges given to services must be chosen carefully
- least privilege

- privileges given to services must be choosen carefully
- least privilege
- services running as root is prohibited

How to give a service privileges it needs?

- service user

How to give a service privileges it needs?

- service user
- capabilities on service binary/process

create a service user

```
# Creates a user with low UID, no login mean and no home
$ sudo useradd -M -r -s /usr/bin/nologin svc_myservice

# Check
$ sudo grep svc_myservice /etc/passwd /etc/shadow
/etc/passwd:svc_myservice:x:952:952::/home/svc_myservice:/usr/bin/nologin
/etc/shadow:svc_myservice:!:20715::::::
```

create a systemd service

```
[Unit]
Description=My Service
After=network.target

[Service]
Type=simple
User=svc.myservice
WorkingDirectory=/opt/myservice
ExecStart=/usr/bin/python3 -m http.server 80
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
AmbientCapabilities=CAP_NET_BIND_SERVICE
Restart=on-failure
RestartSec=5

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

start a systemd service

```
# start service
$ sudo systemctl start my

# check
sudo systemctl status my
my.service — My Service
   Loaded: loaded (/etc/systemd/system/my.service; disabled; preset: disabled)
   Active: active ( running ) since XXXX; 4s ago
 Invocation: 3510f8899b454f0ab29c7316c9615e62
    Main PID: 52044 (python3)
       Tasks: 1 (limit: 18402)
      Memory: 12.3M (peak: 12.4M)
         CPU: 67ms
    CGroup: /system.slice/my.service
           52044 /usr/bin/python3 -m http.server 80

$ sudo ss -lptn
State      Recv-Q      Send-Q      Local Address:Port      Peer Address:Port
Process
LISTEN      0          5           0.0.0.0:80              0.0.0.0:*    users:(((" python3",pid
=52044,fd=3))
```


- <https://man7.org/linux/man-pages/man7/capabilities.7.html>
- <https://www.redhat.com/en/blog/mastering-systemd>

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

- logs

- audit subsystem and auditd

- security software

MISC

Conclusion

Why tracing?

- compliance with laws or standards

Why tracing?

- compliance with laws or standards
- know what users are doing on your systems

Why tracing?

- compliance with laws or standards
- know what users are doing on your systems
- debug

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

- logs

- audit subsystem and auditd

- security software

MISC

Conclusion

- logs are vital for auditing and detecting

- logs are vital for auditing and detecting
- care should be taken on local retention duration

- logs are vital for auditing and detecting
- care should be taken on local retention duration
- always use a log collector on the network

Different logging systems:

- system logs (syslog)

Different logging systems:

- system logs (syslog)
- systemd-journald

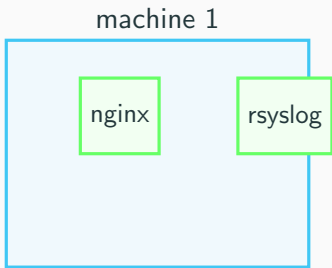
- standard for data logging

- standard for data logging
- client-server architecture

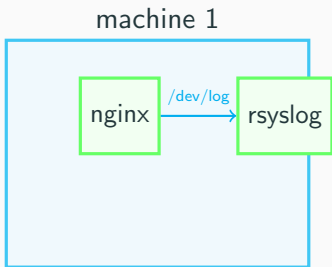
- standard for data logging
- client-server architecture
- a server handles logging for other software (clients) on the system: rsyslog, syslog-ng

- standard for data logging
- client-server architecture
- a server handles logging for other software (clients) on the system: rsyslog, syslog-ng
- clients can connect locally or remotely (syslog protocol)

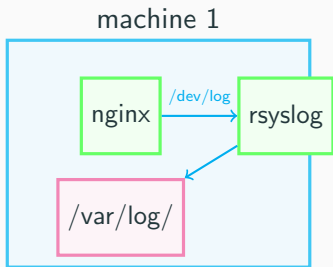
Syslog flow



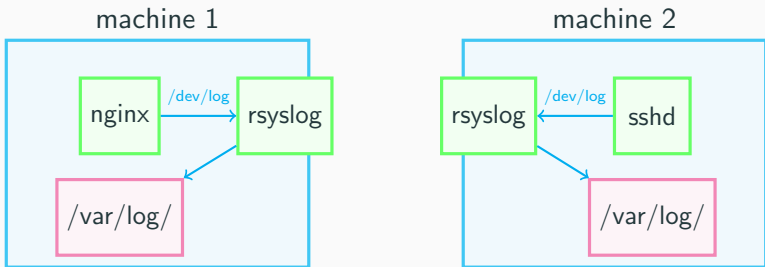
Syslog flow



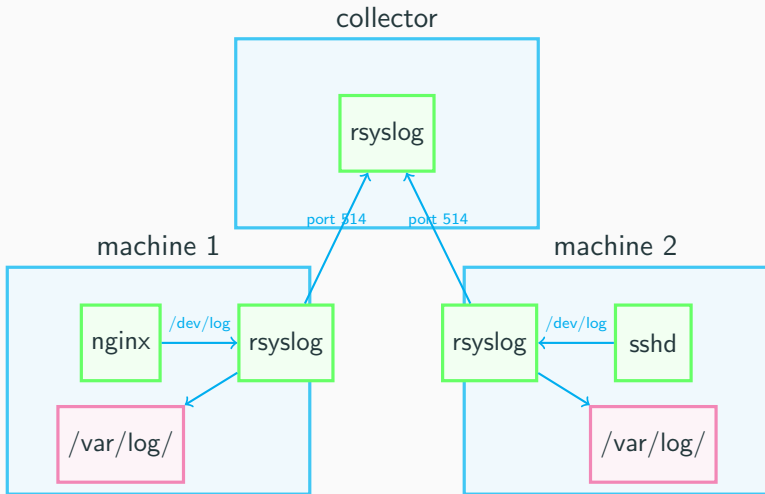
Syslog flow



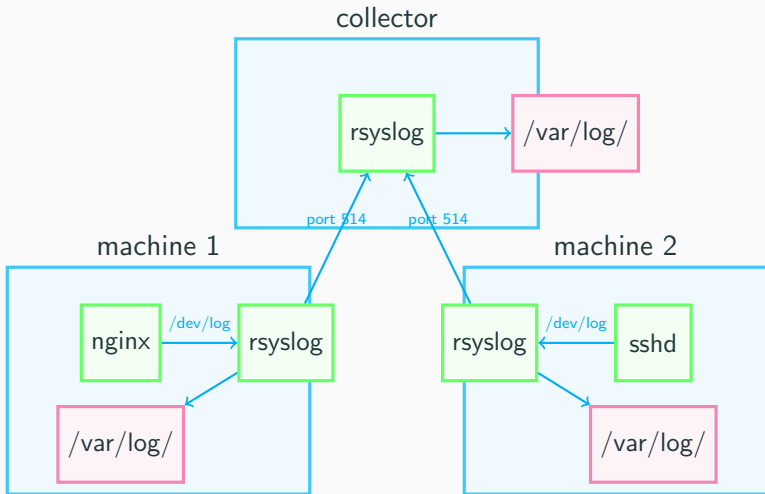
Syslog flow



Syslog flow



Syslog flow



hunting in system logs

```
# find SSH connections
sudo grep -i ssh /var/log/auth.log | tail -n 3
2026-09-19T13:49:07.636797+00:00 vps-XXXX sshd-session[948436]: Invalid user
    admin from 92.118.39.77 port 59736
2026-09-19T13:49:07.852211+00:00 vps-XXXX sshd-session[948436]: Connection
    closed by invalid user admin 92.118.39.77 port 59736 [preauth]
```

- most Linux distributions use it as of today

- most Linux distributions use it as of today
- collects logs directly from systemd services

- most Linux distributions use it as of today
- collects logs directly from systemd services
- logs are stored in `/var/log/journal/` in binary format and indexed

- most Linux distributions use it as of today
- collects logs directly from systemd services
- logs are stored in `/var/log/journal/` in binary format and indexed
- no more `grep`, use `journalctl` instead

list SSH service logs

```
# journalctl -u ssh --since "30 min ago"
Sep 19 15:32:35 debian sshd-session[321]: Accepted password for root from
10.0.2.2 port 56032 ssh2
Sep 19 15:32:35 debian sshd-session[321]: pam_unix(sshd:session): session opened
for user root(uid=0) by root(uid=0)
```

list SSH service logs

```
# journalctl -u ssh --since "30 min ago"
Sep 19 15:32:35 debian sshd-session[321]: Accepted password for root from
10.0.2.2 port 56032 ssh2
Sep 19 15:32:35 debian sshd-session[321]: pam_unix(sshd:session): session opened
for user root(uid=0) by root(uid=0)
```

Attentive readers would have notice the `pam_unix` above, meaning that:

```
$ grep UsePAM /etc/ssh/sshd_config
UsePAM yes
```

grep in journald logs

```
# journalctl —since "50 min ago" —grep="failed"
Sep 19 16:13:52 debian unix_chkpwd[795]: password check failed for user (root)
Sep 19 16:13:54 debian sshd-session[793]: Failed password for root from 10.0.2.2
      port 48134 ssh2
Sep 19 16:13:55 debian unix_chkpwd[796]: password check failed for user (root)
Sep 19 16:13:57 debian sshd-session[793]: Failed password for root from 10.0.2.2
      port 48134 ssh2
Sep 19 16:13:59 debian unix_chkpwd[797]: password check failed for user (root)
Sep 19 16:14:01 debian sshd-session[793]: Failed password for root from 10.0.2.2
      port 48134 ssh2
```

- because of history, some machines have both syslog daemon and journald running

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

- logs

- audit subsystem and auditd

- security software

MISC

Conclusion

- kernel subsystem in charge of auditing security events

- kernel subsystem in charge of auditing security events
- "hooks" located at strategic location in kernel code

- kernel subsystem in charge of auditing security events
- "hooks" located at strategic location in kernel code
- messages are sent to a specific audit log (`/var/log/audit/audit.log`)

- kernel subsystem in charge of auditing security events
- "hooks" located at strategic location in kernel code
- messages are sent to a specific audit log (`/var/log/audit/audit.log`)
- audit rules

Check if auditd is enabled

```
$ systemctl status auditd
auditd.service — Security Audit Logging Service
   Loaded: loaded (/usr/lib/systemd/system/auditd.service;
           disabled; preset: disabled)
   Active: inactive (dead)
     Docs: man:auditd(8)
           https://github.com/linux-audit/audit-
           documentation
$ sudo systemctl start auditd
$ sudo systemctl enable auditd
```

- rules are located in
 - `/etc/audit/audit.rules`
 - `/etc/audit/rules.d`
- two types of rules: watch or system call

auditd rules example

```
$ cat /etc/audit/rules.d/example.rules  
-w /etc/passwd -p r -k passwd_read  
-a always,exit -S openat,openat2 -F a0=1 -k syscall_open
```

auditd event

```
$ cat /etc/passwd
[...]
```

```
# ausearch -k passwd_read
```

```
time->Sat Sep 19 17:04:48 2026
```

```
type=PROCTITLE msg=audit(1789830288.973:184): proctitle=636174002
F6574632F706173737764
```

```
type=PATH msg=audit(1789830288.973:184): item=0 name="/etc/passwd" inode=32546 dev
=08:01 mode=0100644 ouid=0 ogid=0 rdev=00:00 obj=system_u:object_r:etc_t:s0
nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
```

```
type=CWD msg=audit(1789830288.973:184): cwd="/root"
```

```
type=SYSCALL msg=audit(1789830288.973:184): arch=c000003e syscall=257
```

```
success=yes exit=3 a0=ffffffffffffff9c a1=7ffcb1f98e0e a2=0 a3=0
```

```
items=1 ppid=354 pid=1003 audit=0 uid=0 gid=0 euid=0 suid=0 fsuid=0
```

```
egid=0 sgid=0 fsgid=0 tty=pts0 ses=1 comm="cat" exe="/usr/bin/cat" subj=
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key="passwd_read"
```


- very useful in incident response

- very useful in incident response
- if enabled and if rules are relevant

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

- logs

- audit subsystem and auditd

- security software

MISC

Conclusion

Dedicated software exists for the purposes of detection and/or prevention of malware:

Dedicated software exists for the purposes of detection and/or prevention of malware:

- antiviruses

Dedicated software exists for the purposes of detection and/or prevention of malware:

- antiviruses
- Endpoint Detection and Response (EDR)

Dedicated software exists for the purposes of detection and/or prevention of malware:

- antiviruses
- Endpoint Detection and Response (EDR)
- Intrusion Detection System (IDS)
 - Host-based (HIDS)
 - Network (NIDS)

Dedicated software exists for the purposes of detection and/or prevention of malware:

- antiviruses
- Endpoint Detection and Response (EDR)
- Intrusion Detection System (IDS)
 - Host-based (HIDS)
 - Network (NIDS)
- lots of editors pick more than one of the above in their products

(open-source) security software examples

- Wazuh: EDR/XDR
- OSSEC: HIDS
- OpenEDR: EDR

- these software should ensure innocuity

- these software should ensure innocuity
- the principle of least privileges also apply to it

- these software should ensure innocuity
- the principle of least privileges also apply to it
- because SW is tagged "security", does not mean we can blindly trust it

- these software should ensure inocuity
- the principle of least privileges also apply to it
- because SW is tagged "security", does not mean we can blindly trust it
- having detection, even with a low number of signatures, is always better than being blind

- <https://www.rfc-editor.org/info/rfc5424/>
<https://wiki.archlinux.org/title/Systemd/Journal>
<https://man7.org/linux/man-pages/man8/auditd.8.html>

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

 User protections

Conclusion

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

 User protections

Conclusion

- USB devices is still an infection vector widely used

- USB devices is still an infection vector widely used
- control what USB devices are connected to workstations/servers

- USB devices is still an infection vector widely used
- control what USB devices are connected to workstations/servers
- Linux: usbguard

- framework implementing allow and deny list for USB devices in the kernel

- framework implementing allow and deny list for USB devices in the kernel
- rules stored in
 - `/etc/usbguard/rules.conf`
 - `/etc/usbguard/rules.d/`

check if usbguard is active

```
# systemctl status usbguard
usbguard.service — USBGuard daemon
   Loaded: loaded (/usr/lib/systemd/system/usbguard.service; enabled; preset:
          enabled)
   Active: active (running) since Sat 2026-09-26 10:01:30 CEST; 1min 56s ago
 Invocation: 85cf141ecb544f569fe842f1cd48552f
    Docs: man:usbguard-daemon(8)
   Process: 678 ExecStart=/usr/sbin/usbguard-daemon -f -s -c /etc/usbguard/
          usbguard-daemon.conf (code=exited, status=0/SUCCESS)
[...]
```

list and allow devices

```
# usbguard list-devices
1: allow 1d6b:0002 Linux Foundation 2.0 root hub serial "0000:00:14.0" name "
    EHCI Host Controller" hash "h1V..." parent-hash "" via-port "usb1" with-
    interface 09:00:00
2: block 046d:c52b Logitech, Inc. Unifying Receiver serial "" name "USB Receiver
    " hash "a7b..." parent-hash "h1V..." via-port "1-1" with-interface {
    03:01:01 03:01:02 }
3: reject 0aa0:0001 Unknown Rogue Drive serial "12345" name "Flash Drive" hash "
    z9x..." parent-hash "h1V..." via-port "1-2" with-interface 08:06:50

# usbguard allow-device 3
```


- <https://usbguard.github.io/>

- <https://github.com/a13xp0p0v/kernel-hardening-checker>
- <https://github.com/jeffbencteux/sysctlchk>

Questions?

Introduction

Secure bootloader chain

Kernel parameters

Linux kernel security mechanisms

Filesystems and partitioning

Authentication

Autorisation

Tracability and detection

MISC

Conclusion

- a Linux system is as secure as the people administering it

- a Linux system is as secure as the people administering it
- respect principles of:
 - attack surface reduction
 - least privilege
 - defense in depth

- a Linux system is as secure as the people administering it
- respect principles of:
 - attack surface reduction
 - least privilege
 - defense in depth
- focus on understanding what is the security need and adapt your defenses to it

security professionals:

- CISO

security professionals:

- CISO
- Security Operations Center (SOC)

security professionals:

- CISO
- Security Operations Center (SOC)
- Computer Emergency Response Team (CERT)
 - aka Computer Security Incident Response Team (CSIRT)

security professionals:

- CISO
- Security Operations Center (SOC)
- Computer Emergency Response Team (CERT)
 - aka Computer Security Incident Response Team (CSIRT)
- auditors / pentesters

The End. Questions?