

The Linux Security Journey

Version 6.0
October-2025

By Dr. Shlomi Boutnaru



Created using [Craiyon AI Image Generator](#)

Introduction.....	6
UID (User Identifier).....	7
RUID (Real User ID).....	8
EUID (Effective User ID).....	9
SUID (Saved User ID).....	10
GID (Group Identifier).....	11
EGID (Effective Group ID).....	12
RGID (Real Group ID).....	13
SGID (Saved Group ID).....	14
File Permissions.....	15
umask (Set File Mode Creation Mask).....	16
File Permissions are Not Cumulative.....	17
Sticky Bit.....	18
SUID Bit (Set User ID).....	19
SGID Bit (Set Group ID).....	20
ASLR (Address Space Layout Randomization).....	21
ASLR in Statically Linked ELF's.....	22
KASLR (Kernel Address Space Layout Randomization).....	23
Non-Executable Memory (NX Bit Support).....	24
SAS (Secure Attention Sequence).....	25
CryptoAPI (Cryptography Application Programming Interface).....	26
Cryptoloop.....	27
dm-crypt (Device Mapper Crypto Target).....	28
LUKS (Linux Unified Key Setup).....	29
Secure Computing Mode (seccomp).....	30
Linux Security Modules (LSM).....	31
LoadPin.....	32
SafeSetID.....	33
Yama.....	34
Keyrings.....	35
AppArmor (Application Armor).....	36
Primary Groups.....	37
Secondary Group.....	38
ACL (Access Control Lists).....	39
PAM (Pluggable Authentication Module).....	40
Capabilities.....	41
chroot (Change Root Directory).....	43
NetFilter.....	44
Chains (iptables).....	45
Table Types (iptables).....	46
nftables.....	47

Secure Execution Mode.....	48
dmesg_restrict.....	49
TCP Wrappers.....	50
TCP SYN Cookie Protection.....	51
UFW (Uncomplicated Firewall).....	52
Firewalld (Firewall Daemon).....	53
su (Substitute User).....	54
OpenSSH.....	55
Disable Kernel Modules.....	56
Kernel Module Signing.....	57
Disable Kexec (Disable Kernel Execution).....	58
USB Guard (Universal Serial Bus Guard).....	59
USB Auth (Universal Serial Bus Authorization).....	60
sudo (SuperUser Do).....	61
Authentication Logs.....	62
Reduced Access to Syscalls.....	63
PolKit (Authorization Manager).....	64
KSPP (Kernel Self Protection Project).....	65
Debug WX.....	66
Grsecurity.....	67
Kernel Modules Strict RWX (Read-Write-Execute) Protection.....	68
Kernel Strict RWX (Read-Write-Execute) Protection.....	69
Hardened User Copy.....	70
perf_event_paranoid.....	71
IPv4 RPF (Reverse Path Filtering).....	72
Reset Attack Mitigation.....	73
Guard Pages.....	74
Use Virtually Mapped Stack with Guard Pages.....	75
LKRG (Linux Kernel Runtime Guard).....	76
LVBS (Linux Virtualization Based Security).....	77
Blacklisting Kernel Modules.....	78
SecurityFS (Security File System).....	79
IMA (Integrity Measurement Architecture).....	80

Introduction

When starting to learn OS security I believe that there is a need for understanding multiple technologies and concepts. Because of that I have decided to write a series of short writeups aimed at providing the security vocabulary.

Overall, I wanted to create something that will improve the overall knowledge of Linux different security mechanisms included with Linux in writeups that can be read in 1-3 mins. I hope you are going to enjoy the ride.

Lastly, you can follow me on twitter - @boutnaru (<https://twitter.com/boutnaru>). Also, you can read my other writeups on medium - <https://medium.com/@boutnaru>. Lastly, You can find my free eBooks at <https://TheLearningJourneyEbooks.com>.

Lets GO!!!!!!

UID (User Identifier)

UID stands for “User Identifier”, it is a number associated with each Linux user in order to represent it in the Linux kernel. We can think about it like SID¹ in Windows which uniquely identifies a security principal. We can see the UID for a specific user defined in “/etc/passwd”.

In most Linux distributions UIDs 1-500 are reserved for system users, while in distributions like Ubuntu/Fedora the UID for a new user starts from 1000 - by default, the UID of root is 0². Moreover, we can also change the UID of a user with the “usermod”³ command in the following manner: “sudo usermod -u [NEW_UID] [USER_NAME]” - as shown in the screenshot below. Also, all processes executed by the user will get its UID - more on that in a future writeup.

Lastly, it is important to understand that there are three different types of UID: EUID (Effective UID), RUID (Real UID) and SUID (Saved UID).

```
root@localhost:/tmp/troller# id troller
uid=1000(troller) gid=1000(troller) groups=1000(troller)
root@localhost:/tmp/troller# usermod -u 1337 troller
root@localhost:/tmp/troller# id troller
uid=1337(troller) gid=1000(troller) groups=1000(troller)
```

¹ <https://medium.com/@boutnaru/windows-security-sid-security-identifier-d5a27567d4e5>

² <https://linuxhandbook.com/uid-linux/>

³ <https://linux.die.net/man/8/usermod>

RUID (Real User ID)

RUID stands for “Real UserID”, which is the user who initiated a specific operation⁴. Thus, we can say that it is basically the UID⁵ of the user that started the specific task/process⁶. Overall, RUID is the “uid” field of the Linux’s “struct cred” data structure⁷. This information is also included as part of the “Auxiliary Vector”⁸ in the “AT_UID” entry⁹.

Moreover, there could be specific syscalls that use only the real uid/group id (and not the effective uid - which I am going to detail about in a future writeup), one example of that is access¹⁰.

Lastly, another example of usage is by the “passwd” command line utility¹¹. When executing it gets the permissions of the root user. However, due to the fact the “real uid” is not changed by using a “suid bit”¹² we can’t change the password of a user which is not us (unless we are the “root” user) - as shown in the screenshot below.

```
[test@localhost ~]$ id  
uid=1000(test) gid=1000(test) groups=1000(test)  
[test@localhost ~]$ passwd test2  
passwd: You may not view or modify password information for test2.  
[test@localhost ~]$ passwd test  
Changing password for test.  
Current password: _
```

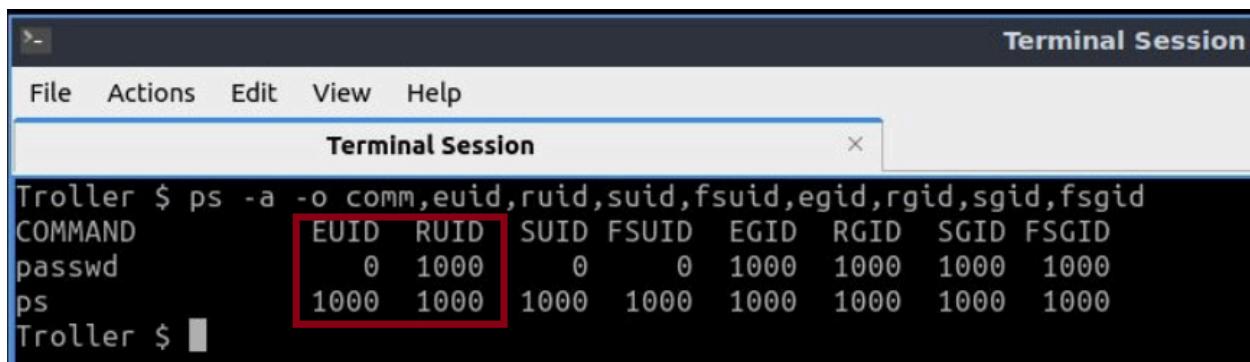
⁴ <https://linuxhint.com/difference-between-real-effective-user-id-in-linux-os/>
⁵ <https://medium.com/@boutnaru/the-linux-security-journey-uid-user-identifier-2f11bcf90ee8>
⁶ <https://www.geeksforgeeks.org/real-effective-and-saved-userid-in-linux/>
⁷ <https://elixir.bootlin.com/linux/v6.5.8/source/include/linux/cred.h#L119>
⁸ <https://medium.com/@boutnaru/linux-the-auxiliary-vector-auxv-cba527871b50>
⁹ <https://elixir.bootlin.com/linux/v6.5.8/source/include/uapi/linux/auxvec.h#L20>
¹⁰ <https://elixir.bootlin.com/linux/v6.5.8/source/fs/open.c#L368>
¹¹ <https://man7.org/linux/man-pages/man1/passwd.1.html>
¹² <https://medium.com/@boutnaru/linux-security-suid-bit-d4f553e7d99e>

EUID (Effective User ID)

EUID (Effective User ID) is what is mostly used for determining the permissions of a certain task (process/thread) in a Linux system¹³. By default, the EUID is equal to the value of RUID, there are also use cases in which those two are different¹⁴.

Moreover, there are use cases in which the EUID is different than the RUID for enabling a non-privileged user to access files which are accessed only by privileged users like root¹⁵. For example the case of the utility “passwd” that needs to alter “/etc/shadow” when the user changes a password - as shown in the screenshot below.

Lastly, the EUID is stored in the “euid” field of “struct cred”¹⁶ that is pointed from the PCB (Process Control Block)/TCB (Thread Control Block) data structure in Linux, which is “struct task_struct”¹⁷.



COMMAND	EUID	RUID	SUID	FSUID	EGID	RGID	SGID	FSGID
passwd	0	1000	0	0	1000	1000	1000	1000
ps	1000	1000	1000	1000	1000	1000	1000	1000

¹³ <https://linuxhandbook.com/uid-linux/>

¹⁴ <https://medium.com/@boutnaru/the-linux-security-journey-ruid-real-user-id-b23abcbea9c6>

¹⁵ <https://www.geeksforgeeks.org/real-effective-and-saved-userid-in-linux/>

¹⁶ <https://elixir.bootlin.com/linux/v6.8/source/include/linux/cred.h#L117>

¹⁷ <https://medium.com/@boutnaru/linux-kernel-task-struct-829f51d97275>

SUID (Saved User ID)

In this context SUID stands for “Saved User ID” (and it is different from SUID bit¹⁸). It is used when we have a task (process/thread) executing with high privilege (such as root, but not limited to that) which needs to do something in an unprivileged manner. Due to the fact, we want to work in a “least privilege” principle¹⁹, we need to use the high privileges only when it is a must.

Thus, we use the SUID in order to save the EUID²⁰ and then do the change which causes the task to execute as an unprivileged user. After finishing the operation/s the EUID is taken back from the SUID²¹.

Lastly, we can use the “setresuid” syscall for setting a different value between EUID and SUID²² - as shown in the screenshot below. We can see that we can set euid=0 if our uid=0 but we can’t do that if uid!=0.

```
#define _GNU_SOURCE
#include <unistd.h>
#include <stdio.h>

void main()
{
    printf("#####SUID exmple#####\n");
    uid_t euid=1000, ruid=1000, suid=0;
    int val=setresuid(ruid,euid,suid);
    printf("setresuid: ret value: %d\n*****\n",val);
    ruid=-1;euid=-1;suid=-1;
    val=getresuid(&ruid,&euid,&suid);
    printf("getresuid: retv value: %d\neuid=%d,ruid=%d,suid=%d\n*****\n",val,euid,ruid,suid);
    euid=0,ruid=0,suid=0;
    val=getresuid(&ruid,&euid,&suid);
    printf("setresuid: ret value: %d\n*****\n",val);
    euid=1000,ruid=1000;suid=1000;
    val=setresuid(ruid,euid,suid);
    printf("setresuid: ret value: %d\n*****\n",val);
    euid=0;ruid=0;suid=0;
    val=setresuid(ruid,euid,suid);
    printf("setresuid: ret value: %d\n*****\n",val);

}

Troller# ./code
#####SUID exmple#####
setresuid: ret value: 0
*****
getresuid: retv value: 0
euid=1000,ruid=1000,suid=0
*****
setresuid: ret value: 0
*****
setresuid: ret value: 0
*****
setresuid: ret value: -1      setresuid fails
*****
Troller# █
```

¹⁸ <https://medium.com/@boutnaru/linux-security-suid-bit-d4f553e7d99e>

¹⁹ <https://www.techtarget.com/searchsecurity/definition/principle-of-least-privilege-POLP>

²⁰ <https://medium.com/@boutnaru/the-linux-security-journey-euid-effective-user-id-65f351532b79>

²¹ <https://stackoverflow.com/questions/32455684/difference-between-real-user-id-effective-user-id-and-saved-user-id>

²² <https://man7.org/linux/man-pages/man2/setresuid.2.html>

GID (Group Identifier)

GID stands for “Group Identifier”, it is a number associated with each Linux group in order to represent it in the Linux kernel. We can think about it like SID²³ in Windows which uniquely identifies a security principal. We can see the GID for a specific group defined in “/etc/group”.

Moreover, by using groups we can manage which resources users of a group can access. We can get the GID of a group using the “getent” command line utility²⁴ - as shown in the screenshot below. For creating a new group we can use the “groupadd”²⁵ command line utility - as also shown in the screenshot below.

Also, there are Linux distributions that reserve the GID range 0–99 for statically allocated groups, and either 100–499 or 100–999 for groups dynamically allocated by the system in post-installation scripts. These ranges are often specified in /etc/login.defs²⁶ - as shown in the screenshot below.

Lastly, it is important to understand that there are three different types of GID: EGUID (Effective GID), RGID (Real GID) and SGID (Saved GID).

```
Troller # getent group root
root:x:0:
Troller # groupadd trollers
Troller # getent group trollers
trollers:x:1001:
Troller # cat /etc/login.defs | grep ^GID
GID_MIN                      1000
GID_MAX                       60000
```

²³ <https://medium.com/@boutnaru/windows-security-sid-security-identifier-d5a27567d4e5>

²⁴ <https://man7.org/linux/man-pages/man1/getent.1.html>

²⁵ <https://linux.die.net/man/8/groupadd>

²⁶ https://en.wikipedia.org/wiki/Group_identifier

EGID (Effective Group ID)

As with EUID²⁷ we also have EGID (Effective Group ID). It is what is mostly used for determining the permissions of a certain task (process/thread) in a Linux system in the sense of group membership.

Moreover, there are use cases in which the EGID is different from the RGID (Real Group ID) for enabling a non-privileged user to access files which are accessed only by privileged users like root. We can access the EGID using the inside the kernel using the “current_egid” macro²⁸.

Lastly, we can use the “id”²⁹ command line utility for printing the effective group ID. Also, from user mode we can use the “getegid()” system call to retrieve the effective group id of the calling process/task³⁰. There is also a library call with the same name³¹ - as shown in the screenshot below taken using copy.sh³².

```
000d0240 <getegid@eGLIBC_2.0>:
d0240:    f3 0f 1e fb        endbr32
d0244:    b8 ca 00 00 00      mov    $0xca,%eax
d0249:    65 ff 15 10 00 00 00  call   *%gs:0x10
d0250:    c3                  ret
d0251:    66 90              xchg   %ax,%ax
d0253:    66 90              xchg   %ax,%ax
d0255:    66 90              xchg   %ax,%ax
d0257:    66 90              xchg   %ax,%ax
d0259:    66 90              xchg   %ax,%ax
d025b:    66 90              xchg   %ax,%ax
d025d:    66 90              xchg   %ax,%ax
d025f:    90                  nop
```

²⁷ <https://medium.com/@boutnaru/the-linux-security-journey-euid-effective-user-id-65f351532b79>

²⁸ <https://elixir.bootlin.com/linux/v6.9.5/source/include/linux/cred.h#L375>

²⁹ <https://linux.die.net/man/1/id>

³⁰ <https://linux.die.net/man/2/getegid>

³¹ <https://man7.org/linux/man-pages/man3/getegid.3p.html>

³² <https://copy.sh/x86/?profile=archlinux>

Rgid (Real Group ID)

Rgid stands for “Real Group ID”, which is the main group of the user who initiated a specific operation. Thus, we can say that it is basically the GID³³ of the user that started the specific task/process³⁴. Overall, RUID is the “uid” field of the Linux’s “struct cred” data structure³⁵. This information is also included as part of the “Auxiliary Vector”³⁶ in the “AT_UID” entry³⁷.

Moreover, there could be specific syscalls that use only the real uid/group id (and not the effective uid - which I am going to detail about in a future writeup), one example of that is access³⁸.

Lastly, another example of usage is by the “passwd” command line utility³⁹. When executing it gets the permissions of the root user. However, due to the fact the “real uid” is not changed by using a “suid bit”⁴⁰ we can’t change the password of a user which is not us (unless we are the “root” user) - as shown in the screenshot below.

```
[test@localhost ~]$ id  
uid=1000(test) gid=1000(test) groups=1000(test)  
[test@localhost ~]$ passwd test2  
passwd: You may not view or modify password information for test2.  
[test@localhost ~]$ passwd test  
Changing password for test.  
Current password: _
```

³³ <https://medium.com/@boutnaru/the-linux-security-journey-gid-group-identifier-c38e8e25c221>

³⁴ <https://www.geeksforgeeks.org/real-effective-and-saved-userid-in-linux/>

³⁵ <https://elixir.bootlin.com/linux/v6.5.8/source/include/linux/cred.h#L119>

³⁶ <https://medium.com/@boutnaru/linux-the-auxiliary-vector-auxv-cba527871b50>

³⁷ <https://elixir.bootlin.com/linux/v6.5.8/source/include/uapi/linux/auxvec.h#L20>

³⁸ <https://elixir.bootlin.com/linux/v6.5.8/source/fs/open.c#L368>

³⁹ <https://man7.org/linux/man-pages/man1/passwd.1.html>

⁴⁰ <https://medium.com/@boutnaru/linux-security-suid-bit-d4f553e7d99e>

SGID (Saved Group ID)

In this context SGID stands for “Saved Group ID” (and it is different from SGID bit). It is used when we have a task (process/thread) executing with high privilege (such as root, but not limited to that) which needs to do something in an unprivileged manner. Due to the fact, we want to work in a “least privilege” principle⁴¹, we need to use the high privileges only when it is a must.

Thus, we use the SGID in order to save the EGID⁴² and then do the change which causes the task to execute as an unprivileged user. After finishing the operation/s the EGID is taken back from the SGID.

Lastly, we can use the “setresgid” syscall for setting a different value between EGID and SGID⁴³. We can egid=0 if our sgid=0 but we can’t do that if sgid!=0. Thus, SGID uses the same concepts as SGID⁴⁴. SGID is also an attribute saved as part of the “struct cred”⁴⁵ - as shown in the screenshot below.

```
111  struct cred {
112      atomic_long_t usage;
113      kuid_t uid;          /* real UID of the task */
114      kgid_t gid;          /* real GID of the task */
115      kuid_t suid;         /* saved UID of the task */
116      kgid_t sgid;         /* saved GID of the task */ highlighted
117      kuid_t euid;         /* effective UID of the task */
118      kgid_t egid;         /* effective GID of the task */
119      kuid_t fsuid;        /* UID for VFS ops */
120      kgid_t fsgid;        /* GID for VFS ops */
121      unsigned securebits; /* SUID-less security management */
122      kernel_cap_t cap_inheritable; /* caps our children can inherit */
123      kernel_cap_t cap_permitted; /* caps we're permitted */
124      kernel_cap_t cap_effective; /* caps we can actually use */
125      kernel_cap_t cap_bset; /* capability bounding set */
126      kernel_cap_t cap_ambient; /* Ambient capability set */
127 #ifdef CONFIG_KEYS
128      unsigned char jit_keyring; /* default keyring to attach requested
129                                * keys to */
```

⁴¹ <https://www.techtarget.com/searchsecurity/definition/principle-of-least-privilege-POLP>

⁴² <https://medium.com/@boutnaru/the-linux-security-journey-egid-effective-group-id-bda1c56b4995>

⁴³ <https://linux.die.net/man/2/setresgid>

⁴⁴ <https://www.linux.org/threads/linux-ids.10213/>

⁴⁵ <https://github.com/torvalds/linux/blob/master/include/linux/cred.h>

File Permissions

As you probably know there is a basic saying in Linux: “everything is a file”, so it's no surprise that file permissions are core to the security model used by Linux systems. By using them we can define who can access files/directories. In general, we have three levels of permissions: read, write and execute⁴⁶.

Overall, every file/directory in Linux (in case the filesystem supports permissions) has as part of its metadata three categories of ownership: user, group and other. User, is the owner of the file which by default is the one that created it. Group, it is the primary group⁴⁷ of the user when the file/directory was created. Other, which applies to all other users in the system⁴⁸.

Moreover, we can view those permissions using the “-l” switch of the “ls”⁴⁹ command - as shown in the screenshot below. There are also special permissions SUID⁵⁰, SGID⁵¹ and sticky bit⁵² - demonstrated below.

Lastly, we can change the permissions of files (aka file mode bits) using the “chmod” command, this can be done using the form “[ugoa]*([-+=][[rwxXst]*|[ugo]])+”⁵³ - as shown in the example below. By the way, we can also use numeric values for setting the permissions where “read=4”, “write=2” and “execute=1” - example is also shown in the screenshot below. Also, SUID/GUID/Sticky can be defined by adding a fourth number where “SUID=4”, “SGID=2” and “Sticky bit=1” - also shown in the example below.

```
root@localhost:/tmp/files# ls -l
total 2
-rw-r--r-- 1 root root 0 [REDACTED] file1
-rw-r--r-- 1 root root 0 [REDACTED] file2
-rw-r--r-- 1 root root 0 [REDACTED] file3
root@localhost:/tmp/files# chmod u-w file1
root@localhost:/tmp/files# chmod g+x file2
root@localhost:/tmp/files# chmod o-r file3
root@localhost:/tmp/files# ls -l
total 2
-r--r--r-- 1 root root 0 [REDACTED] file1
-rw-r--r-- 1 root root 0 [REDACTED] file2
-rw-r--r-- 1 root root 0 [REDACTED] file3
root@localhost:/tmp/files# chmod 777 file3
root@localhost:/tmp/files# chmod 421 file2
root@localhost:/tmp/files# chmod 000 file1
root@localhost:/tmp/files# ls -l
total 2
----- 1 root root 0 [REDACTED] file1
-r--u--x 1 root root 0 [REDACTED] file2
-rwxrwxrwx 1 root root 0 [REDACTED] file3
root@localhost:/tmp/files# chmod 4777 ./file1
root@localhost:/tmp/files# chmod 2777 ./file2
root@localhost:/tmp/files# chmod 1777 ./file3
root@localhost:/tmp/files# ls -lah
total 2.5K
drwxr-xr-x 2 root root 0 [REDACTED] .
drwxrwxrwt 5 root root 51 [REDACTED]
-rwsrwsrwx 1 root root 0 [REDACTED] file1
-rwxrwsrwx 1 root root 0 [REDACTED] file2
-rwxrwxrwt 1 root root 0 [REDACTED] file3
```

⁴⁶ <https://www.redhat.com/sysadmin/linux-file-permissions-explained>

⁴⁷ <https://medium.com/@boutnaru/the-linux-security-journey-primary-groups-de2b4d6bd27b>

⁴⁸ <https://www.linuxfoundation.org/blog/blog/classic-sysadmin-understanding-linux-file-permissions>

⁴⁹ <https://man7.org/linux/man-pages/man1/ls.1.html>

⁵⁰ <https://medium.com/@boutnaru/linux-security-suid-bit-d4f553e7d99e>

⁵¹ <https://medium.com/@boutnaru/the-linux-security-journey-sgid-bit-set-group-id-c9b82a2ce019>

⁵² <https://medium.com/@boutnaru/linux-security-sticky-bit-ccb0aaaf3c019>

⁵³ <https://linux.die.net/man/1/chmod>

umask (Set File Mode Creation Mask)

When creating a new file/directory the default file mode permissions⁵⁴ are 666 (rw-rw-rw), however those permissions are masked/filtered by the umask (Set File Mode Creation Mask) value. Thus, if we have “umask=0022” the permissions of a newly created file is set to 644 (rw-r--r--). In case of “umask=0077” the permissions of a newly created file is set to 600 (rw-----) and for “umask=0000” we get 666 (rw-rw-rw-) - as shown in the screenshot below.

Overall, “umask” is a system call used for setting the file mode creation mask. This system call always succeeds and the previous value of the mask is returned. “umask” is used by in conjunction with syscalls like “open”⁵⁵ and “mkdir”⁵⁶.

Moreover, as opposed to the “chmod”⁵⁷ syscall which affects the permissions of a specific file/directory “umask” affects every file/directory created by the user. In most Linux distributions the “umask” value are configured in system wide configuration files like: “/etc/profile” or “/etc/bash.bashrc”⁵⁸.

Lastly, based on the shell environment used, “umask” can be a dedicated binary or a builtin command of the shell. There are cases in which “umask” binary is used we can just read the value and not change it, because it will change it for a different process session, so for altering the umask value in those cases the builtin shell command is needed⁵⁹.

```
root@localhost:/tmp/files# umask
0022
root@localhost:/tmp/files# touch troller1
root@localhost:/tmp/files# umask 077
root@localhost:/tmp/files# touch troller2
root@localhost:/tmp/files# umask 000
root@localhost:/tmp/files# touch troller3
root@localhost:/tmp/files# ls -lah
total 2.5K
drwxr-xr-x 2 root root 0
drwxrwxrwt 5 root root 51
-rw-r--r-- 1 root root 0
-rw----- 1 root root 0
-rw-rw-rw- 1 root root 0
2024 troller1
2024 troller2
2024 troller3
```

⁵⁴ <https://medium.com/@boutnaru/the-linux-security-journey-file-permissions-033cb3ce8547>

⁵⁵ <https://man7.org/linux/man-pages/man2/open.2.html>

⁵⁶ <https://man7.org/linux/man-pages/man2/mkdir.2.html>

⁵⁷ <https://man7.org/linux/man-pages/man2/chmod.2.html>

⁵⁸ <https://www.liquidweb.com/kb/what-is-umask-and-how-to-use-it-effectively/>

⁵⁹ <https://docs.oracle.com/cd/E19455-01/806-0624/6j9vek5ja/index.html>

File Permissions are Not Cumulative

As opposed to what we might think, file permissions⁶⁰ under Linux are not cumulative. For example if we have a file that our user only has “read” permission and our primary group⁶¹ has “read and write” permissions we will still just get the “read” permission. Even if any other user has full permissions to the file we won’t be able to alter it - as shown in the screenshot below.

Thus, we can say that the permissions are checked in the following order: user, group and other. In case there are any permissions given to the subject accessing the object (file/directory) the check is stopped and by that file permissions are not cumulative - as demonstrated below.

Lastly, this fact is not relevant for the “root” user due to the fact it has the capability “CAP_DAC_OVERRIDE”⁶², which overrides the read/write/execute file permission check - as also shown below. This is of course relevant for any user holding that capability.

```
[test@localhost test]$ id  
uid=1000(test) gid=1000(test) groups=1000(test)  
[test@localhost test]$ ls -l ./file  
-r--rw-rwx 1 test test 8 Mar 12 2024 ./file  
[test@localhost test]$ cat ./file  
Tr0LLeR  
[test@localhost test]$ echo BlA > ./file  
bash: ./file: Permission denied  
[test@localhost test]$  
  
root@localhost:/tmp/test# id  
uid=0(root) gid=0(root) groups=0(root)  
root@localhost:/tmp/test# ls -la file  
-r--r---- 1 root root 8 Mar 12 2024 file  
root@localhost:/tmp/test# cat ./file  
Tr0LLeR  
root@localhost:/tmp/test# echo BlA > ./file  
root@localhost:/tmp/test# cat file  
BlA  
root@localhost:/tmp/test# chmod 0000 ./file  
root@localhost:/tmp/test# chown test ./file  
root@localhost:/tmp/test# chgrp test ./file  
root@localhost:/tmp/test# ls -lah ./file  
----- 1 test test 4 Mar 12 2024 ./file  
root@localhost:/tmp/test# cat ./file  
BlA  
root@localhost:/tmp/test# echo TeST > ./file  
root@localhost:/tmp/test# cat ./file  
TeST  
root@localhost:/tmp/test# _
```

⁶⁰ <https://medium.com/@boutnaru/the-linux-security-journey-file-permissions-033cb3ce8547>

⁶¹ <https://medium.com/@boutnaru/the-linux-security-journey-primary-groups-de2b4d6bd27b>

⁶² <https://man7.org/linux/man-pages/man7/capabilities.7.html>

Sticky Bit

Beside the ordinary permissions that a file/directory can have in Linux (read, write & execute) we can also assign specific permissions bits that have a special meaning: suid, sgid and sticky bit.

Have you ever asked yourself what the “t” in the output of “ls -l” stands for? (as you can see in the screenshot below taken from copy.sh). As you can see everyone can read and write to “/tmp”, but in the place of “execute” there is a “t” (and not an “x”) - it means “sticky bit”.

The goal of “sticky bit” when setting it on a directory is to allow the removal of files in the directory only by their owner. You can see a full demonstration of that in the image below. As shown even if the file (/tmp/test1_file) has full permissions for everyone it still can’t be deleted by the user test2 (by the way the permissions of the file are not relevant as we will show in a different writeup).

```
root@localhost:~# ls -ld /tmp
drwxrwxrwt 4 root root 117 Feb 20 20:02 /tmp
root@localhost:~# su test2
[test2@localhost root]$ cd /tmp
[test2@localhost tmp]$ ls -lh
total 512
-rwxrwxrwx 1 test1 test1 2 Sep 22 2022 test1_file
[test2@localhost tmp]$ rm -f ./test1_file
rm: cannot remove './test1_file': Operation not permitted
[test2@localhost tmp]$ _
```

SUID Bit (Set User ID)

Beside the ordinary permissions that a file/directory can have in Linux (read, write & execute) we can also assign specific permissions bits that have a special meaning: **suid**, **sgid** and **sticky bit**.

Have you ever asked yourself what the “s” in the output of “ls -l” stands for? As you can see in the screenshot below taken from an Ubuntu 22.04 VM regarding the “passwd” executable.

If we have a file with a “suid bit” it will be executed using the permissions of the owner of the file. It is important to understand that it sets the “euid” but not “ruid” - As you can see in the screenshot below. Information about “euid”, “ruid”, “fsuid” and more are going to be covered as part of the description about “struct cred”⁶³ in the future.

For now all you need to know is that “euid” (Effective UID) is used for most of the permissions checks and “ruid” (Real UID) is the uid (User Identifier) of the user that started the executable. Due to that even though “passwd” runs using root it does not allow changing a password that is not the one that started it (despite root that can change any user's password).

```
Troller $ ls -lah `which passwd`  
-rwsr-xr-x 1 root root 59K Mar 14 2022 /usr/bin/passwd  
Troller $ id | cut -d "(" -f1  
uid=1000  
Troller $ passwd  
Changing password for   
Current password:   
Troller $ ps -a -o comm,euid,ruid,suid,fsuid,egid,rgid,sgid,fsgid  
COMMAND EUID RUID SUID FSUID EGID RGID SGID FSGID  
passwd 0 1000 0 0 1000 1000 1000 1000  
ps 1000 1000 1000 1000 1000 1000 1000 1000  
Troller $
```

⁶³ <https://elixir.bootlin.com/linux/latest/source/include/linux/cred.h#L110>

SGID Bit (Set Group ID)

SGID is a special permission, its meaning is based if it is set on a file or a directory. If it is set on a file it allows the file to be executed with the permissions of the group that owns the file - it is similar to SUID which does the same with the user that owns the file⁶⁴. In case of a directory, if we set the SGID bit, any files created in the directory will have their group ownership set to that of the directory owner⁶⁵.

Thus, this permission is marked with “s” in the location that specifies the “execute” permission. In order to set SGID we can use the “chmod”⁶⁶ command, after doing so the execute indication “x” in the group portion is going to change to “s”.

This can be verified using the “-l” switch of the “ls”⁶⁷ command - as shown in the screenshot below. By the way, if we remove the execute permission the indication is changed from “s” to “S” - as also shown in the screenshot below.

Lastly, as with the normal permissions in which we have the numeric system for setting/removing them by using a 3 numbers, by adding another one we can also specify the other special permissions⁶⁸. For setting the SGID bit we can you a number in the pattern of 2xyz, where x/y/z are 1 or 2 or 4⁶⁹.

```
root@localhost:/tmp/troller# ls -l
total 1
-rwxr-xr-x 1 root root [REDACTED] 2023 troller.log
root@localhost:/tmp/troller# chmod g+s ./troller.log
root@localhost:/tmp/troller# ls -l
total 1
-rwxr-sr-x 1 root root [REDACTED] 2023 troller.log
root@localhost:/tmp/troller# chmod g-x ./troller.log
root@localhost:/tmp/troller# ls -l
total 1
-rwxr-Sr-x 1 root root [REDACTED] 2023 troller.log
```

⁶⁴ <https://medium.com/@boutnaru/linux-security-suid-bit-d4f553e7d99e>

⁶⁵ <https://www.redhat.com/sysadmin/suid-sgid-sticky-bit>

⁶⁶ <https://linux.die.net/man/1/chmod>

⁶⁷ <https://man7.org/linux/man-pages/man1/ls.1.html>

⁶⁸ <https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/how-permissions-chmod-with-numbers-command-explained-777-rwx-unix>

⁶⁹ <https://www.liquidweb.com/kb/how-do-i-set-up-setuid-setgid-and-sticky-bits-on-linux/>

ASLR (Address Space Layout Randomization)

ASLR is a mitigation techniques used to increase the difficulty of running arbitrary code in case of an exploitation of a memory corruption vulnerability such as a buffer overflow (We will go over it in a different writeup but you can read the first article about it from phrack⁷⁰ magazine).

Basically what ASLR does is to randomly select the base address of the executable, it also randomizes the heap, stack and loaded libraries. Thus, in case an attacker can control the flow of execution (such as controlling the instruction pointer), the location of the arbitrary code to execute is unknown.

In order for ASLR to work we need support in the OS (mostly the loader of executables) and in the executable itself, it should be compiled as PIE (position independent code) and have support relocations (in case of absolute addressing used). More on PIE and relocations in future writeups.

There are several limitations in case of ASLR, some of them are general and some of them are relevant for only specific implementations. In that sense different operating systems support ASLR in different manners. For example, in Linux the base address is randomized every call to a syscall from the family of execve() ('man 2 execve'). Thus, if we just use fork() the base address won't change. The screenshot below shows the difference between the address of libc between two executions of the command "ls". We will dive more into specific OS implementations in the future. Also, ASLR is not relevant to other attack types such as "Data Only" (more on them in the future).

You have to remember that there are several ways to bypass ASLR (depending on the OS and other factors). You can search online for those techniques or wait for the next writeup. Also, although there are bypasses, ASLR is a must in my opinion.

Lastly, there is also an ASLR version for the kernel itself, it is usually called KASLR (Kernel Space Layout Randomization).

```
Troller $ ldd `which ls` | grep libc
  libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f1b9cb7d000)
Troller $ ldd `which ls` | grep libc
  libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3f09f39000)
```

⁷⁰ <http://phrack.org/issues/49/14.html>

ASLR in Statically Linked ELF

When compiling code to a statically linked ELF we bake all the code our binary needs from shared libraries inside our own executable⁷¹. The question which arises is how and if it effects the ASLR⁷² posture of the process executing the statically linked binary?

Thus, as we can see in the screenshot below when linking the binary statically (using “-static”) any time we execute it the addresses of the stack/heap/vdso/vvar memory regions are randomized. However, the memory regions mapped from the binary are not randomized.

In order to fix this we can use “-static-pie” which can load the memory regions mapped for the statically linked binary to randomized addresses without the need of the dynamic linker⁷³. We can also see that in the screenshot below.

```
root@localhost:/tmp/troller# gcc --static ./troller.c -o ./troller
root@localhost:/tmp/troller# file ./troller
./troller: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically linked BuildID[sha1]=ad45a04b2b1505fc27b218c19a1b6c2722bb8f21f, for GNU/Linux 4.4.0, with debug_info, not stripped
root@localhost:/tmp/troller# ./troller &
[1] 1350
root@localhost:/tmp/troller# cat /proc/$(pidof troller)/maps
08048000-08049000 r-p 00000000 00:17 98001    /tmp/troller/troller
08049000-0804a000 r-xp 00001000 00:17 98001    /tmp/troller/troller
0804a000-0804c000 r--p 00065000 00:17 98001    /tmp/troller/troller
080dc000-080df000 r--p 00093000 00:17 98001    /tmp/troller/troller
080d1000-080e1000 rw-p 00096000 00:17 98001    /tmp/troller/troller
080e1000-080e4000 rw-p 00000000 00:00 0
09902000-09924000 rw-p 00000000 00:00 0      [heap]
b7ef4000-b7f01000 r--p 00000000 00:00 0      [vvar]
b7f01000-b7f03000 r-xp 00000000 00:00 0      [vdso]
bfaf97000-bfafb5000 rw-p 00000000 00:00 0      [stack]
root@localhost:/tmp/troller# kill -9 $(pidof troller)
root@localhost:/tmp/troller# ./troller &
[2] 1354
[1] Killed                 ./troller
root@localhost:/tmp/troller# cat /proc/$(pidof troller)/maps
08048000-08049000 r-p 00000000 00:17 98001    /tmp/troller/troller
08049000-0804a000 r-xp 00001000 00:17 98001    /tmp/troller/troller
0804a000-0804c000 r--p 00065000 00:17 98001    /tmp/troller/troller
080dc000-080df000 r--p 00093000 00:17 98001    /tmp/troller/troller
080d1000-080e1000 rw-p 00096000 00:17 98001    /tmp/troller/troller
080e1000-080e4000 rw-p 00000000 00:00 0
08a0a000-08aac000 rw-p 00000000 00:00 0      [heap]
b7f25000-b7f29000 r--p 00000000 00:00 0      [vvar]
b7f29000-b7f2b000 r-xp 00000000 00:00 0      [vdso]
bf8fd000-bf91e000 rw-p 00000000 00:00 0      [stack]
root@localhost:/tmp/troller# gcc --static-pie ./troller.c -o ./troller
root@localhost:/tmp/troller# kill -9 $(pidof troller)
root@localhost:/tmp/troller# ./troller &
[3] 1363
[2] Killed                 ./troller
root@localhost:/tmp/troller# cat /proc/$(pidof troller)/maps | grep troller
b7ea000-b7ea7000 r--p 00000000 00:17 98007    /tmp/troller/troller
b7ea7000-b7fc0000 r-xp 00003000 00:17 98007    /tmp/troller/troller
b7fc0000-b7fd3000 r--p 00066000 00:17 98007    /tmp/troller/troller
b7fd3000-b7f40000 r--p 00098000 00:17 98007    /tmp/troller/troller
b7f40000-b7f42000 rw-p 0009b000 00:17 98007    /tmp/troller/troller
root@localhost:/tmp/troller# _
```

⁷¹ <https://www.ibm.com/docs/en/openlxl-c-and-cpp-aix/17.1.0?topic=cc-dynamic-static-linking>

⁷² <https://medium.com/@boutnaru/security-aslr-address-space-layout-randomization-part-1-overview-3aec7fec01e0>

⁷³ <https://patchwork.ozlabs.org/project/gcc/patch/20170808221841.GA16793@gmail.com/#1758721>

KASLR (Kernel Address Space Layout Randomization)

KA SLR (Kernel Address Space Layout Randomization) is an implementation of ASLR (Address Space Layout Randomization) as part of the Linux kernel⁷⁴. It provides the ability to load the kernel (code/data) to random locations in memory. Thus, protecting against different attacks which are dependent on the knowledge of the kernel addresses⁷⁵. We can see that in the difference between the address of “`__inittext_begin`” as returned from “`/proc/kallsyms`”⁷⁶ vs the address stored in “`System.map`”⁷⁷ - as shown in the screenshot below.

Overall, as of today KSLR is enabled by default however we can deactivate it by adding the “`nokaslr`” as part of the kernel boot parameters passed by the bootloader⁷⁸. The Linux kernel configuration item “`CONFIG_RANDOMIZE_BASE`” is the one responsible for KASLR. Because the feature causes changes in the page tables it is dependent on the CPU architecture. Hence, the configuration is also based on the specific architecture (“`arch/s390/Kconfig`”, “`arch/loongarch/Kconfig`”, “`arch/riscv/Kconfig`”, “`arch/arm64/Kconfig`”, “`arch/x86/Kconfig`” and more), in some architecture it is also dependent on “`CONFIG_RELOCATABLE`”⁷⁹.

Lastly, due to that we can find specific implementations of KASLR for different architectures such as (but not limited to): x86⁸⁰, ARM 64⁸¹ and PowerPC⁸².



```
Troller$ sudo cat /boot/System.map-[REDACTED] | grep "__inittext_begin"
fff[REDACTED]9c0000 T __inittext_begin
Troller$ sudo cat /proc/kallsyms | grep "__inittext_begin"
fff[REDACTED]670000 T __inittext_begin
Troller$
```

⁷⁴ <https://medium.com/@boutnaru/security-aslr-address-space-layout-randomization-part-1-overview-3aec7fec01e0>

⁷⁵ <https://www.ibm.com/docs/en/linux-on-systems?topic=shutdown-kaslr>

⁷⁶ <https://medium.com/@boutnaru/the-linux-concept-journey-proc-kallsyms-kernel-exported-symbols-2bb199f123b9>

⁷⁷ <https://en.wikipedia.org/wiki/System.map>

⁷⁸ https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/virtualization_security_guide/sect-virtualization_security_guide-guest_security-kaslr

⁷⁹ https://cateee.net/lkddb/web-lkddb/RANDOMIZE_BASE.html

⁸⁰ <https://elixir.bootlin.com/linux/v6.15.2/source/arch/x86/mm/kaslr.c#L3>

⁸¹ <https://elixir.bootlin.com/linux/v6.15.2/source/arch/arm64/kernel/kaslr.c#L15>

⁸² https://elixir.bootlin.com/linux/v6.15.2/source/arch/powerpc/mm/nohash/kaslr_booke.c#L353

Non-Executable Memory (NX Bit Support)

NX bit is a mitigation technique used to increase the difficulty of running arbitrary code in case of an exploitation of a memory corruption vulnerability such as a buffer overflow. The goal of NX bit is to separate between memory regions containing code to those containing data. In order for this mitigation to work we need support both on the OS and the CPU⁸³. For example x86 has a single (prevention) NX bit as part of the PTE (Page Table Entry) while ARM64 has both “UXN” (User Never Execute) and “PWN” (Privileged Never Execute) for distinguishing between user-mode (EL0) and kernel mode (EL1\EL2) in this architecture⁸⁴.

Overall, let check out x86 as a reference implementation. First the kernel can check if the CPU (x86 in this case) supports the NX bit security feature⁸⁵. A specific message is emitted based on the x86 CPU support⁸⁶. It can be viewed by: using “journalctl”, using “dmesg” or reading “/var/log/messages” - as shown in the screenshot below⁸⁷. The “set_memory_nx” function is leveraged for setting an already allocated\mapped memory region to non-executable⁸⁸. By the way, in case we don’t yet have a relevant page entry (not still mapped\allocated) we can use “pgprot_nx”⁸⁹.

Lastly, there are multiple implementations of the “set_memory_nx” function for different CPUs such as (but not limited to): ARM⁹⁰, RISC-V⁹¹, PowerPC⁹² and x86⁹³. In case we want to disable the support of NX bit we can pass “noexec=off” or “noexec32=off” as a command line argument to the kernel, this can be done for example using the GRUB menu⁹⁴.

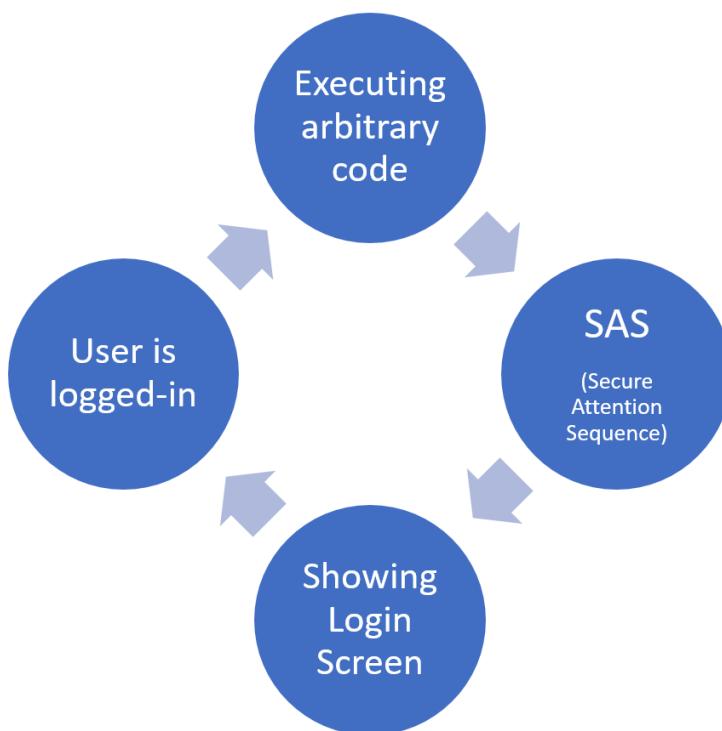
```
# cat /var/log/messages | grep "Execute Disable"
Feb 20 23:20:41 localhost kernel: NX (Execute Disable) protection: active
```

⁸³ <https://medium.com/@boutnaru/security-nx-bit-non-executable-18759fd2802e>
⁸⁴ <https://developer.arm.com/documentation/102376/0200/Permissions/Execution-permissions>
⁸⁵ <https://elixir.bootlin.com/linux/v6.15.8/source/arch/x86/kernel/setup.c#L940>
⁸⁶ <https://elixir.bootlin.com/linux/v6.15.8/source/arch/x86/kernel/setup.c#L824>
⁸⁷ <https://access.redhat.com/solutions/2936741>
⁸⁸ <https://elixir.bootlin.com/linux/v6.15.8/source/arch/x86/mm/init.c#L932>
⁸⁹ https://elixir.bootlin.com/linux/v6.15.8/A/ident/pgprot_nx
⁹⁰ <https://elixir.bootlin.com/linux/v6.15.8/source/arch/arm/mm/pageattr.c#L87>
⁹¹ <https://elixir.bootlin.com/linux/v6.15.8/source/arch/riscv/mm/pageattr.c#L372>
⁹² https://elixir.bootlin.com/linux/v6.15.8/source/arch/powerpc/include/asm/set_memory.h#L25
⁹³ https://elixir.bootlin.com/linux/v6.15.8/source/arch/x86/mm/pat/set_memory.c#L2300
⁹⁴ <https://davidhamann.de/2020/09/09/disable-nx-on-linux/>

SAS (Secure Attention Sequence)

SAS (sometimes called SAK - “Secure Attention Key”) is a special key sequence/combination that triggers the opening of the login screen. The goal of SAS is to make sure that the login screen is not spoofed. Due to the fact that the OS interacts with the hardware it can detect the SAS (think about keyboard interrupts) and then suspend any application and run the login screen⁹⁵ - As shown in the diagram below.

If you want to read about the support of Linux (kernel 2.4.2) for SAS⁹⁶ (SAK in the documentation). This concept is also relevant for other operating systems like Windows⁹⁷.



⁹⁵ https://en.wikipedia.org/wiki/Secure_attention_key

⁹⁶ <https://www.kernel.org/doc/Documentation/SAK.txt>

⁹⁷ <https://security.stackexchange.com/questions/152556/why-does-windows-10-not-have-the-secure-attention-key-as-default>

CryptoAPI (Cryptography Application Programming Interface)

CryptoAPI (Cryptography Application Programming Interface) is a framework as part of Linux which provides cryptographic services. It is leveraged by different security mechanisms such as (but not limited to): IPSec, dm-crypt and cryptoloop⁹⁸. It was introduced as part of kernel version “2.5.45”⁹⁹. By the way, we can read the list of ciphers provided by the kernel CryptoAPI from “/proc/crypto”¹⁰⁰ - as shown in the screenshot below.

Overall, in the kernel lingo all crypto algorithms are called “transformations”, hence a cipher handle variable usually has the name “tfm”¹⁰¹. Moreover, compression transformations are handled in the same way as ciphers¹⁰². Among the core functionality of CryptoAPI we can find: hash functions (like SHA1, SHA256, SHA3), symmetric encryption ciphers (like AES and Blowfish), asymmetric encryption ciphers (like RSA and ECC), digital signatures, authenticated encryption, key agreement protocols, random number generator and more¹⁰³.

Lastly, we can check out the source code of CryptoAPI as part of the Linux kernel located at “/crypto”¹⁰⁴. The relevant header files are stored as part of “/include/linux/crypto.h”¹⁰⁵. Also, the relevant code for kernel modules\drivers are stored in “/drivers/crypto”¹⁰⁶. There could be also architecture specific crypto implementations like (but not limited to) for “arm64”¹⁰⁷ and for “x86”¹⁰⁸.

```
Troller$ cat /proc/crypto | tail -18
name      : rsa
driver    : rsa-generic
module   : kernel
priority  : 100
refcnt   : 1
selftest  : passed
internal  : no
type     : akcipher

name      : dh
driver    : dh-generic
module   : kernel
priority  : 100
refcnt   : 1
selftest  : passed
internal  : no
type     : kpp

Troller$
```

⁹⁸ [https://en.wikipedia.org/wiki/Crypto_API_\(Linux\)](https://en.wikipedia.org/wiki/Crypto_API_(Linux))

⁹⁹ <https://elixir.bootlin.com/linux/v2.5.45/source/crypto>

¹⁰⁰ https://man7.org/linux/man-pages/man5/proc_crypto.5.html

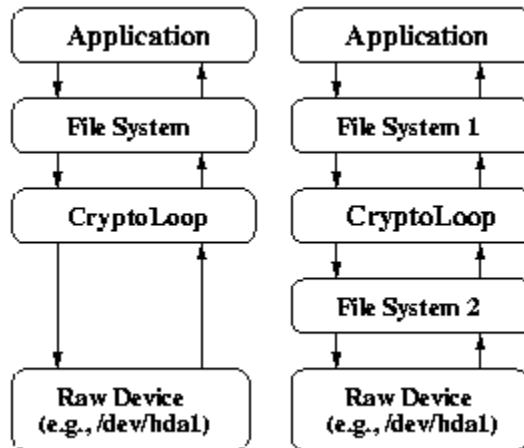
101 <https://elixir.bootlin.com/linux/v6.15.4/A/ident/tfm>102 <https://docs.kernel.org/crypto/intro.html>103 <https://www.kernel.org/doc/html/v4.14/crypto/index.html>104 <https://elixir.bootlin.com/linux/v6.15.4/source/crypto>105 <https://elixir.bootlin.com/linux/v6.15.4/source/include/linux/crypto.h>106 <https://elixir.bootlin.com/linux/v6.15.4/source/drivers/crypto>107 <https://elixir.bootlin.com/linux/v6.15.4/source/arch/arm64/crypto>108 <https://elixir.bootlin.com/linux/v6.15.4/source/arch/x86/crypto>

Cryptoloop

Cryptoloop is a disk encryption module which is implemented as a Linux kernel module. It leverages CryptoAPI that is part of the Linux kernel mainline¹⁰⁹. Cryptoloop has been included as part of the kernel since kernel version “2.5”¹¹⁰. It is important to understand that the functionality of cryptoloop has been incorporated into the device mapper¹¹¹.

Overall, cryptoloop can be leveraged in order to encrypt a file system as part of a partition or a regular file - as shown in the diagram below¹¹². This can be done using a loop device¹¹³. Cryptoloop is thought to be deprecated since kernel version “2.6” and is also vulnerable to different attacks - more on them in future writeups. Thus, for modern Linux versions we should use “dm-crypt” and LUKS¹¹⁴.

Lastly, the kernel support for cryptoloop is controlled using the config parameter “BLK_DEV_CRYPTOLOOP”. It is supported up to kernel version “5.15.86”¹¹⁵. However, it is marked as deprecated since kernel version “5.13.15”¹¹⁶. We can find the relevant config parameter as part of the Linux source code since kernel “2.6”¹¹⁷.



¹⁰⁹ <https://medium.com/@boutnaru/the-linux-security-journey-cryptoapi-cryptography-application-programming-interface-0d026bf4589e>

¹¹⁰ <https://en.wikipedia.org/wiki/Cryptoloop>

¹¹¹ <https://medium.com/@boutnaru/the-linux-concept-journey-dm-device-mapper-e6bc42981893>

¹¹² <https://www.linuxtechtips.com/2013/11/how-to-encrypt-partition-with-cryptoloop.html>

¹¹³ <https://medium.com/@boutnaru/the-linux-concept-journey-loop-device-17caeae3b15c>

¹¹⁴ <https://unix.stackexchange.com/questions/452353/how-does-cryptoloop-work-and-where-can-i-use-it>

¹¹⁵ <https://elixir.bootlin.com/linux/v5.15.186/source/drivers/block/Kconfig#L230>

¹¹⁶ <https://elixir.bootlin.com/linux/v5.13.15/source/drivers/block/Kconfig#L216>

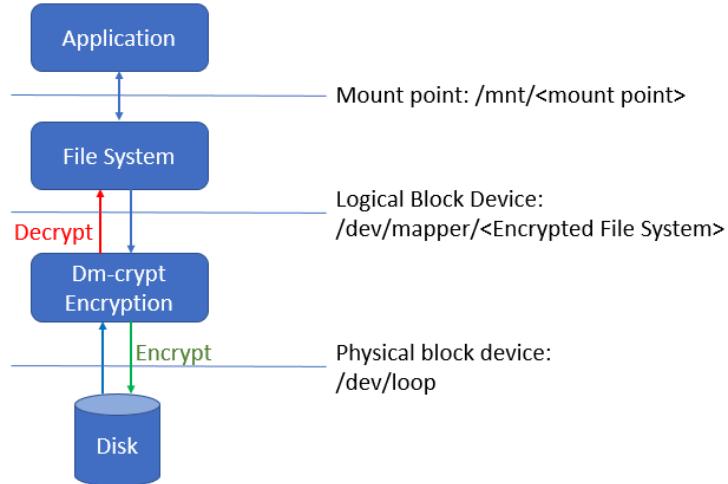
¹¹⁷ <https://elixir.bootlin.com/linux/v2.6.0/source/drivers/block/Kconfig#L251>

dm-crypt (Device Mapper Crypto Target)

dm-crypt (Device Mapper Crypto Target) is Linux's kernel device mapper¹¹⁸ crypto target. We can use it for encrypting whole disks (like removable media), partitions, software RAID, logical volumes and even specific files. Also, it can be stacked on top of other device mapper transformations¹¹⁹.

Overall, dm-crypt allows us to mount encrypted file-systems. After mounting all files are accessible to applications transparently, while the files are encrypted when stored¹²⁰ - as demonstrated in the diagram below. By the way, LUKS (Linux Unified Key Setup) is based on dm-crypt¹²¹ - more on that in future writeups. Also, dm-crypt is the successor of cryptoloop¹²².

Lastly, the kernel support for dm-crypt is controlled using the config parameter “DM_CRYPT”¹²³. It has been included as part of Linux's mainline since kernel version “2.6.4”¹²⁴. The mapping table for a crypt target (used by dm-crypt) includes different properties such as (but not limited to): cipher, key count (multi-key support), chain mode (like cbc and xts) and IV (Initialization Vector) mode¹²⁵.



¹¹⁸ <https://medium.com/@boutnaru/the-linux-concept-journey-dm-device-mapper-e6hc42981893>

¹¹⁹ <https://wiki.archlinux.org/title/Dm-crypt>

¹²⁰ <https://docs.aws.amazon.com/whitepapers/latest/navigating-gdpr-compliance/linux-dm-crypt-infrastructure.html>

¹²¹ https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/5/html/installation_guide/ch29s02

¹²² https://tldp.org/HOWTO/html_single/Cryptoloop-HOWTO/

¹²³ <https://elixir.bootlin.com/linux/v6.15.4/source/drivers/md/Kconfig#L265>

¹²⁴ <https://elixir.bootlin.com/linux/v2.6.4/source/drivers/md/Kconfig>

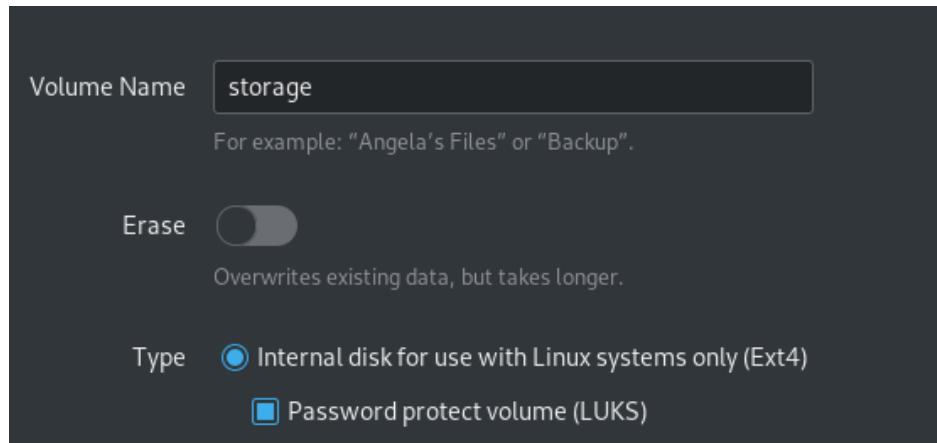
¹²⁵ <https://gulah.com/cryptsetup/cryptsetup/-/wikis/DMCrypt>

LUKS (Linux Unified Key Setup)

LUKS (Linux Unified Key Setup) is basically a disk encryption specification created in 2004 (by Clemens Fruhwirth). It is based on “dm-crypt”¹²⁶ for performing encryption on a block device¹²⁷. LUKS has an unencrypted header (at the start of an encrypted volume) that can hold the cipher type, key size encryption keys and provide the support for multiple encryption keys. This specific header is probably the major difference to just using “dm-crypt” directly due to the support for multiple passphrases¹²⁸.

Overall, LUKS creates an encrypted container aka “LUKS volume” on a disk partition. The data is encrypted using a symmetric algorithm (like AES), it can be accessed using a passphrase. The encryption is done with a master key (randomly generated when LUKS is initialized). The master key is encrypted using the passphrase and stored in a key slot as part of the header¹²⁹.

Lastly, we can leverage LUKS for different use-cases such as (but not limited to): encrypting file systems (including swap), full disk encryption (FDE), encrypting cloud storage and removable media¹³⁰ - as shown in the screenshot below¹³¹. LUKS supports different formats LUKS1 and LUKS2 each with their own defaults and capabilities¹³².



¹²⁶ <https://medium.com/@boutnaru/the-linux-security-journey-dm-crypt-device-mapper-crypto-target-c86877f6be9b>

¹²⁷ <https://www.redhat.com/en/blog/disk-encryption-luks>

¹²⁸ https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/security_hardening/encrypting-block-devices-using-luks_security-hardening

¹²⁹ <https://www.howtogeek.com/what-is-luks-and-how-does-it-secure-vour-linux-file-system/>

¹³⁰ <https://isecjobs.com/insights/luks-encryption-explained/>

¹³¹ <https://wiki.lunardao.net/luks.html>

¹³² <https://www.linuxconsultant.org/linux-jargon-buster-what-is-luks-encryption/>

Secure Computing Mode (seccomp)

“Secure Computing Mode” (seccomp) is a Linux kernel feature that allows restricting system calls that applications can use, by doing that it reduces their attack surface. With seccomp a process can perform a one-way transition to a “secure mode”, in that mode the process can just run the following syscall: read, write, sigreturn and exit. It was merged into the Linux kernel in version 2.6.12 (released in March 2005).

We can configure seccomp by the libseccomp¹³³, the prctl¹³⁴ system call and/or the seccomp¹³⁵ syscall and/or other CLI tools¹³⁶. Due to its capabilities seccomp is commonly used in sandboxes like in Docker, LXC, systemd’s sandboxing, kubernetes and even within internet browsers (like Google Chrome and Mozilla Firefox) as an additional layer of security. By that, seccomp can help prevent malicious applications from exploiting vulnerabilities or gaining unauthorized access to resources. Also, it can be used to limit an application’s ability to access the network or access the file system¹³⁷.

Since kernel 4.14 there is also a “/proc” interface for seccomp located at “/proc/sys/kernel/seccomp”. There we can find “actions_avail” (a read-only list of seccomp filter return actions supported by the kernel) and “actions_logged” (a read-write list of filter actions that are allowed to be logged)¹³⁸. Also, since kernel 4.14 we can log actions returned by seccomp in the audit log.

In the example shown in the screenshot below we can see how we can block a syscall (mkdir) by passing the relevant parameters to docker which uses seccomp¹³⁹.

```
root@localhost:~# docker run -it --security-opt seccomp=block_mkdir.json ubuntu:18.04 bash
root@ebb5f6b095de:/#
root@ebb5f6b095de:/# cd ~
root@ebb5f6b095de:~#
root@ebb5f6b095de:~# mkdir test
mkdir: cannot create directory 'test': Operation not permitted
root@ebb5f6b095de:~#
```

¹³³ <https://github.com/seccomp/libseccomp>

¹³⁴ <https://man7.org/linux/man-pages/man2/prctl.2.html>

¹³⁵ <https://man7.org/linux/man-pages/man2/seccomp.2.html>

¹³⁶ <https://github.com/david942j/seccomp-tools>

¹³⁷ <https://en.wikipedia.org/wiki/Seccomp>

¹³⁸ <https://man7.org/linux/man-pages/man2/seccomp.2.html>

¹³⁹ https://miro.medium.com/max/1100/0*wz0ilbMy8kr6i30v

Linux Security Modules (LSM)

“Linux Security Modules” (LSM) is a framework which allows the kernel to support various security modules. It was mainly designed to allow implementation of MAC (Mandatory Access Control) with minimal changes to the Linux kernel. For now, you should know that MAC is an organizational-wide security policy that users can’t override (I am going to post about MAC and DAC in more detail separately).

Despite the name containing “Modules” it is not implemented as loadable kernel modules (“.ko” files). The LSM framework is of course optional and needs to be enabled by the CONFIG_SECURITY variable.

If we want to get a list of the running LSMs we just need to read “/sys/kernel/security/lsm” - see the screenshot below (taken from Ubuntu 22.04.01 LTS). It is a comma separated list, at minimum it includes the “capabilities system”. The reason for seeing “capabilities” is due to the fact it was implemented as a “security module”. You can also see the source code for capabilities including “lsm_hooks.h”¹⁴⁰ and thus using different LSM’s enums, macros and functions.

One of the biggest design goals of LSM is to avoid manipulation of the syscall table in order to implement the “security modules”. It is done in order to avoid issues of race conditions and scale problems. Having said that, LSM was not created in order to provide a generic instrumentation/tracing/hooking mechanism for the Linux kernel¹⁴¹.

There are a couple of security features which are implemented as “security modules” like: AppArmor, SELinux, TOMOYO, LoadPin, LandLock and Smack - part of them appear in the screenshot below. A detailed explanation about them will be posted separately.

```
Troller # cat /sys/kernel/security/lsm
lockdown,capability,landlock,yama,apparmorTroller #
```

¹⁴⁰ <https://elixir.bootlin.com/linux/latest/source/security/commoncap.c#L9>

¹⁴¹ <https://www.youtube.com/watch?v=RKBBPsp-TZ0>

LoadPin

After talking about LSM¹⁴² (“Linux Security Modules”) it is time to show different technologies which leverage them. The goal of LoadPin is to ensure that all the files the kernel can load/execute reside on the same filesystem. The idea is that the file system would be based on a read-only device (for example think about a DVD/CD-ROM/any other RO hardware device, it could be also software based (but it has its own drawbacks). You can go over the code of LoadPin¹⁴³.

Among the kernel files artifacts are: kernel modules, firmware, security policies and kexec images (kexec is a syscall which enables loading and booting to a different kernel from the current running one - more on that in a future writeup).

The way it is done is by trapping the kernel’s file reading interface and basically pinning (it for kernel code loading) to the first file system which is used. Thus, any file that is part of a different file system will be rejected - As you can see the screenshot below¹⁴⁴.

One of the use-cases for using LoadPin is to ensure the integrity of kernel code (such as *.ko file) without signing them. It is important to note that I am not recommending using/not using LoadPin, my goal is to explain about it (and leave to the reader what to do with it ;-).

In order to enable LoadPin (which is supported since kernel 4.7) we need to set CONFIG_SECURITY_LOADPIN before building the kernel. If it is enabled we can also toggle it using the kernel command line (“loadpin.enforce=0” or “loadpin.enforce=1”).

```
root@esomimx6s:~# insmod /media/sda1/esom_verify_sys.ko
[ 889.996504] LoadPin: kernel-module denied obj="/media/sda1/esom_verify_sys.ko"
pid=734 cmdline="insmod /media/sda1/esom_verify_sys.ko"
insmod: ERROR: could not insert module /media/sda1/esom_verify_sys.ko: Operation not
permitted
```

¹⁴² <https://medium.com/@boutnaru/linux-security-lsm-linux-security-modules-907bbcf8c8b4>

¹⁴³ <https://elixir.bootlin.com/linux/latest/source/security/loadpin/loadpin.c>

¹⁴⁴ https://www.e-consystems.com/images/LoadPin_not_allow.png

SafeSetID

“SafeSetID” is an LSM that was merged in kernel 5.1. The goal of “SafeSetID” is to restrict the transitions to target UID/GID from current UID/GID based on a system wide whitelist. Thus, it governs the family of syscalls which allows those types of transitions (like seteuid and setegid. In general set*uid/set*gid).

One of the most used use-cases for “SafeSetID” is to enable a non-root application to transition to other untrusted UIDs/GIDs without the need of giving it an uncontrolled CAP_SET{U/G}ID capabilities¹⁴⁵.

It is important to understand that we still grant CAP_SET{U/G}ID capabilities to the non-root application, however “SafeSetID” allows us to restrict its actions. Thus, we can limit the application from entering/creating a new user namespace or setting the uid to 0 (root).

The configuration of “SafeSetID” is done using performing manipulation on files residing on a securityfs mounting point. The relevant files are “safesetid/uid_allowlist_policy” and “safesetid/gid_allowlist_policy” (the format of adding a policy is ‘<UID>:<UID>’ or ‘<GID>:<GID>’). By the way, on my Ubuntu 22.04 VM securityfs is mounted on “/sys/kernel/security”.

We can go over the code of “SafeSetID” as part of the source code of the Linux kernel¹⁴⁶. Moreover, you can see on the image below the source code for creating of the configuration files entries in securityfs¹⁴⁷.

```
uid_policy_file = securityfs_create_file("uid_allowlist_policy", 0600,
                                         policy_dir, NULL, &safesetid_uid_file_fops);
if (IS_ERR(uid_policy_file)) {
    ret = PTR_ERR(uid_policy_file);
    goto error;
}

gid_policy_file = securityfs_create_file("gid_allowlist_policy", 0600,
                                         policy_dir, NULL, &safesetid_gid_file_fops);
if (IS_ERR(gid_policy_file)) {
    ret = PTR_ERR(gid_policy_file);
    goto error;
}
```

¹⁴⁵ <https://medium.com/@boutnaru/linux-security-capabilities-part-1-63c6d2ceb8bf>

¹⁴⁶ <https://elixir.bootlin.com/linux/v6.13.7/source/security/safesetid/lsm.c>

¹⁴⁷ <https://elixir.bootlin.com/linux/latest/source/security/safesetid/securityfs.c#L324>

Yama

“Yama” is an LSM that was merged in kernel 3.4. The goal of “Yama” is to restrict the usage of the “ptrace” syscall (“man 2 ptrace”). Limiting the usage of “ptrace” is one way to isolate between running applications, thus even in case one application is breached it won’t be able to attach to other running applications (like browsers, crypto wallets, encryption apps, remote connection session and more) and read/write to sensitive data or change the flow. In order to include “Yama” the CONFIG_SECURITY_YAMA should be enabled while building the kernel. The configuration of “Yama” could be configured at runtime using “/proc/sys/kernel/yama”, which includes “ptrace_scope” (as shown in the screenshot below).

Based on the kernel documentation¹⁴⁸ “ptrace_scope” can hold 4 different values (0-3). “0” (“classic ptrace permissions”) allows attaching to any running application with the same uid unless it was marked as undumpable. “1” (“restricted ptrace”) allows attaching to running applications based on their relationship, by default only descendants applications could be attached to (you can also change the behavior of the relationship using “prctl” syscall with the option “PR_SET_TRACER”). “2” (“admin-only attach”) allows attaching only from applications holding the “CAP_SYS_PTRACE” capability. “3” (“no attach”) blocks all running applications from attaching to any other running application.

A demonstration of the different values described earlier and their relevant impact on “ptrace” is shown in the screenshot below (just as a reminder “strace” leverages the “ptrace” syscall for attaching running applications). You can go over the code of “Yama” as part of the Linux kernel source code¹⁴⁹.

```
Troller $ cat /proc/sys/kernel/yama/ptrace_scope
1
Troller $ strace -o /tmp/log `which pwd`
/tmp
Troller $ cat /proc/sys/kernel/yama/ptrace_scope
2
Troller $ strace -o /tmp/log `which pwd`
strace: test_ptrace_get_syscall_info: PTRACE_TRACE: Operation not permitted
strace: ptrace(PTRACE_TRACE, ...): Operation not permitted
Troller $ sudo strace -o /tmp/log2 `which pwd`
/tmp
```

¹⁴⁸ <https://www.kernel.org/doc/html/v4.15/admin-guide/LSM/Yama.html>

¹⁴⁹ https://elixir.bootlin.com/linux/v6.13.7/source/security/yama/yama_lsm.c

Keyrings

When writing an application sometimes a need for storing sensitive data elements (like tokens, passwords and cryptographic keys) arises. For that Linux provides “keyrings” which is a data store that allows applications to access data securely without exposing it to other applications/processes/users. Based on the man page “kerings” is an in-kernel key management and retention facility¹⁵⁰. Overall, “keyrings” are used by different types of applications such as authentication servers, web servers and database servers. Examples for those types are: MySQL¹⁵¹.

In order to use “keyrings” we can leverage on of the following syscalls: “add_key()”¹⁵², “request_key()”¹⁵³ or “keyctl()”¹⁵⁴. Each key has several attributes as follows: serial number (ID), type, description (name), payload (data), access rights, expression time and reference count. The types of keys which are supported are: “keyring”, “user”, “logon” and “big_key”¹⁵⁵. They are different libraries/modules in a variety of programming languages that enable programmers to read/write data into/from keyring. An example in Python is shown in the screen below.

Moreover, there are different entries in proc that give us information about the keyrings, we are going to focus only on two. “/proc/keys” which is relevant since kernel 2.6.10, it displays all the keys the reading thread has view permissions. “/proc/key-users” which is also relevant since kernel 2.6.10, that shows various information for each uid that has at least one key on the system¹⁵⁶. Lastly, we can also go over the kernel code that handles keyring¹⁵⁷. Also there is “keyutils” which is a library and a set of utilities that allows access to the in-kernel keyrings facility¹⁵⁸.

```
Troller$ cat keyring_demo.py
import keyring
#Creating a keyring password item
keyring.set_password('Test', 'troller', 'Tr0LeR')
#Retrieving the keyring password item
password = keyring.get_password('Test', 'troller')
print("The password read from the keyring store is : " + str(password))
Troller$ python3 keyring_demo.py
The password read from the keyring store is : Tr0LeR
```

¹⁵⁰ <https://man7.org/linux/man-pages/man7/keyrings.7.html>

¹⁵¹ <https://dev.mysql.com/doc/refman/8.0/en/keyring.html>

¹⁵² https://man7.org/linux/man-pages/man2/add_key.2.html

¹⁵³ https://man7.org/linux/man-pages/man2/request_key.2.html

¹⁵⁴ <https://man7.org/linux/man-pages/man2/keyctl.2.html>

¹⁵⁵ <https://man7.org/linux/man-pages/man7/keyrings.7.html>

¹⁵⁶ <https://man7.org/linux/man-pages/man7/keyrings.7.html>

¹⁵⁷ <https://elixir.bootlin.com/linux/latest/source/security/keys/keyring.c>

¹⁵⁸ <https://man7.org/linux/man-pages/man7/keyutils.7.html>

AppArmor (Application Armor)

“AppArmor” (Application Armor) is an LSM¹⁵⁹ which allows an administrator to create a per-program profile which restricts what an application can do. It basically provides an extension for the DAC (Discretionary Access Control) under Linux by providing MAC (Mandatory Access Control), which constrains a user (subject) can perform on a operating system object¹⁶⁰.

Moreover, one of the differences between “AppArmor” and other MAC systems is that it is path-based. AppArmor’s security model is bound to access control attributes granted to specific programs and not users. It was first seen in Immunix and later included in Linux distributions like Ubuntu¹⁶¹. Thus, profiles contain the lists of access control rules which are loaded and used by “AppArmor”. By default, in Ubuntu those profiles are stored in “/etc/apparmor.d”¹⁶². We can use the “apparmor_status” command to know what profiles are loaded and the current status¹⁶³.

In every profile there are two main types of rules: “path entries” and “capability entries”. “Path Entries” define which files an application can access in the file system. “Capability Entries” define what privileges a confined process is allowed to use. An example of a profile for “/bin/ping” is shown in the screenshot below¹⁶⁴.

Also, “AppArmor” core functionality is part of the Linux mainline since kernel 2.6.3.6¹⁶⁵. You can also go over the source code of “AppArmor” as part of the Linux source code in “/security/apparmor”¹⁶⁶. In order to enable “AppArmor” we need set “CONFIG_SECURITY_APPARMOR=y”, we will still need to install also the usermode tools¹⁶⁷. By the way, “AppArmor” is seen as an alternative to “SELinux”¹⁶⁸.

```
#include <tunables/global>
/bin/ping flags=(complain) {
    #include <abstractions/base>
    #include <abstractions/consoles>
    #include <abstractions/nameservice>

    capability net_raw,
    capability setuid,
    network inet raw,

    /bin/ping mixr,
    /etc/modules.conf r,
}
```

¹⁵⁹ <https://medium.com/@boutmaru/linux-security-lsm-linux-security-modules-907bbcf8c8b4>

¹⁶⁰ <https://en.wikipedia.org/wiki/AppArmor>

¹⁶¹ <https://wiki.ubuntu.com/AppArmor>

¹⁶² <https://linuxhint.com/apparmor-profiles-ubuntu/>

¹⁶³ https://manpages.debian.org/unstable/apparmor/apparmor_status.8.en.html

¹⁶⁴ <https://ubuntu.com/server/docs/security/apparmor>

¹⁶⁵ <https://elixir.bootlin.com/linux/v2.6.36/source/security/apparmor>

¹⁶⁶ <https://elixir.bootlin.com/linux/v6.4.12/source/security/apparmor>

¹⁶⁷ <https://elixir.bootlin.com/linux/v6.4.12/source/security/apparmor/Kconfig#L2>

¹⁶⁸ <https://www.kernel.org/doc/html/v4.15/admin-guide/LSM/apparmor.html>

Primary Groups

Overall, a group is a convenient way to combine users/other groups as one entity in order to manage them as a single unit (such as with permissions). The goal of a primary group is that the operating system can assign it to files/directories that the user is creating¹⁶⁹.

Overall, GID (group identifier) is used in order to uniquely identify the primary group ID that the user belongs to. By the way, we can see it using the “id”¹⁷⁰ command (it is the data which follows “gid=”), or by using the “-gn” switch - as shown in the screenshot below¹⁷¹.

Moreover, we can change it using the “usermod” tool¹⁷², it is important to know that for the change to be visible we need to login again - as shown in the screenshot below. We can also see it as the first group in the output of the “groups”¹⁷³ command - as also shown in the screenshot below. The information about the primary groups is saved as part of “/etc/passwd”¹⁷⁴. Lastly, a user can be part of only one primary group at a time. In parallel the information about the secondary groups is saved in “/etc/group”.

```
Troller $ id
uid=1000(user) gid=1000(user) groups=1000(user),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin)
Troller $ id -gn
user
Troller $ groups
user adm cdrom sudo dip plugdev lpadmin
Troller $ cat /etc/passwd | grep "user:"
user:x:1000:1000:user:/home/user:/bin/bash
Troller $ sudo usermod -g cdrom user
Troller $ id -gn
user
Troller $ sudo login user
Password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux [REDACTED] x86_64)
[REDACTED]
Last login: [REDACTED] Oct [REDACTED] 2023 on pts/3
Troller $ id -gn
cdrom
Troller $
```

¹⁶⁹ <https://www.baeldung.com/linux/primary-vs-secondary-groups>

¹⁷⁰ <https://man7.org/linux/man-pages/man1/id.1.html>

¹⁷¹ <https://unix.stackexchange.com/questions/410367/how-to-get-the-primary-group-of-a-user>

¹⁷² <https://linux.die.net/man/8/usermod>

¹⁷³ <https://man7.org/linux/man-pages/man1/groups.1.html>

¹⁷⁴ <https://man7.org/linux/man-pages/man5/passwd.5.html>

Secondary Group

In general, we can divide the groups in Linux to two main types: primary¹⁷⁵ and secondary. A secondary group is one/more groups which a user is also part of in parallel to the primary group¹⁷⁶.

Thus, when creating a new user with the “useradd”¹⁷⁷ command the user is added to a new primary group which has the same name as the user. In order to create new groups we can use the “groupadd”¹⁷⁸ command - as shown in the screenshot below. When adding users to groups we can use the “gpasswd”¹⁷⁹, those are added as secondary groups- as also shown in the screenshot below.

Lastly, the configuration of secondary groups is stored in “/etc/group”¹⁸⁰. We can also say that secondary groups are those groups which already created users are added¹⁸¹.

```
root@localhost:~# useradd test
root@localhost:~# id test
uid=1000(test) gid=1000(test) groups=1000(test)
root@localhost:~# groupadd test2
root@localhost:~# groupadd test3
root@localhost:~# gpasswd -a test test2
Adding user test to group test2
root@localhost:~# gpasswd -a test test3
Adding user test to group test3
root@localhost:~# id test
uid=1000(test) gid=1000(test) groups=1000(test) 1001(test2),1002(test3)
root@localhost:~# cat /etc/group | grep test2
test2:x:1001:test
root@localhost:~# cat /etc/group | grep test3
test3:x:1002:test
```

¹⁷⁵ <https://medium.com/@boutnaru/the-linux-security-journey-primary-groups-de2b4d6bd27b>

¹⁷⁶ <https://unix.stackexchange.com/questions/605531/primary-vs-secondary-groups-in-linux>

¹⁷⁷ <https://linux.die.net/man/8/useradd>

¹⁷⁸ <https://linux.die.net/man/8/groupadd>

¹⁷⁹ <https://linux.die.net/man/1/gpasswd>

¹⁸⁰ <https://www.baeldung.com/linux/primary-vs-secondary-groups>

¹⁸¹ <https://www.networkworld.com/article/3409781/mastering-user-groups-on-linux.html>

ACL (Access Control Lists)

In general, ACL (Access Control Lists) provides the ability to set permissions to a file/directory in a more granular way than the normal Linux file permissions¹⁸². This can be done without changing the ownership or the granular Linux permissions¹⁸³.

Overall, in order to set or retrieve a file access control list we can use the “getfacl”¹⁸⁴ and the “setfacl”¹⁸⁵ command line utilities - as shown in the screenshot below. In case the filesystem does not support ACL or the feature is not enabled an error of “operation not supported” is returned.

Lastly, to enable ACL we need to mount the filesystem with the “acl” option (for example this can be done using fstab entries)¹⁸⁶. We can also use “tune2fs” for listing the default mounting options of a filesystem, by default in case of btrfs and ext2/3/4 the acl option is enabled¹⁸⁷. In case acl is configured of a file a “+” character appears in the output of “ls -l”¹⁸⁸ - as shown in the screenshot below.

```
Troller $ ls -l troller.txt
---rwx---+ 1 root root 7 Jun 15 02:41 troller.txt
Troller $ id
uid=1001(troller) gid=1001(troller) groups=1001(troller),100(users)
Troller $ cat ./troller.txt
Tr0LeR
Troller $ getfacl ./troller.txt
# file: troller.txt
# owner: root
# group: root
user::---
user:troller:rwx
group::---
mask::rwx
other::---
```

¹⁸² <https://medium.com/@boutnaru/the-linux-security-journey-file-permissions-033cb3ce8547>

¹⁸³ <https://www.redhat.com/sysadmin/linux-access-control-lists>

¹⁸⁴ <https://linux.die.net/man/1/getfacl>

¹⁸⁵ <https://linux.die.net/man/1/setfacl>

¹⁸⁶ https://wiki.archlinux.org/title/Access_Control_Lists

¹⁸⁷ <https://linux.die.net/man/8/tune2fs>

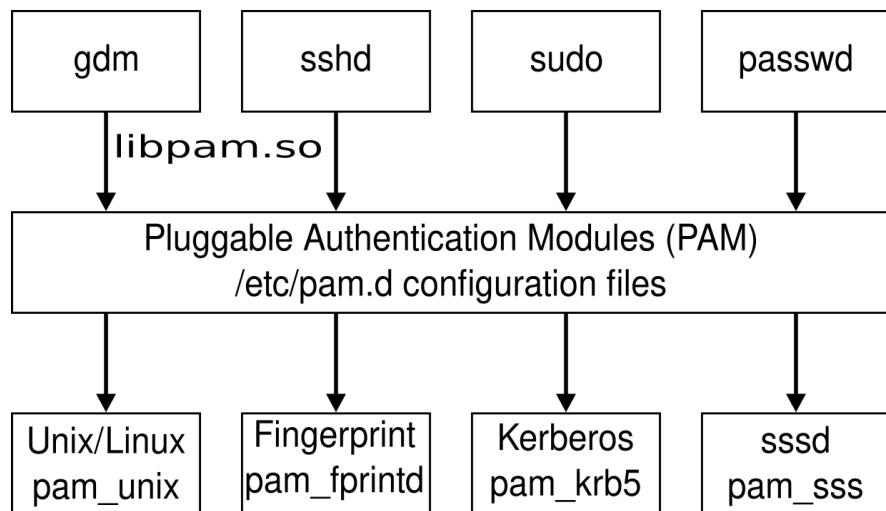
¹⁸⁸ <https://man7.org/linux/man-pages/man1/ls.1.html>

PAM (Pluggable Authentication Module)

The goal of PAM (Pluggable Authentication Module) is to separate the task of authentication for applications (for example login, sshd, ftpd and gdm). This is to avoid the need for every developer to code his own authentication checks. PAM supports local user authentication and also authentication of users defined in a centralized location (for example leveraging the kerberos protocol). A user can enter a username and password/certificate/fingerprint and this information is authenticated using the correct method¹⁸⁹.

Overall, the different applications are linked with the “libpam.so” library - as shown in the screenshot below. The functions in this library provide for PAM¹⁹⁰. An example function is the “pam_authenticate” which is used for authenticating users. In case of successful completion, the function returns PAM_SUCCESS¹⁹¹.

Lastly, the configuration of PAM is stored by default at “/etc/pam.conf”. Also, the configuration of PAM can be stored as the content of the “ /etc/pam.d/” directory. In case the directory exists Linux-PAM ignores the “/etc/pam.conf”¹⁹². We can also go over the code of PAM for more information¹⁹³.



¹⁸⁹ <https://www.redhat.com/sysadmin/pluggable-authentication-modules-pam>

¹⁹⁰ https://docs.oracle.com/cd/E36784_01/html/E36873/libpam-3lib.html

¹⁹¹ https://linux.die.net/man/3/pam_authenticate

¹⁹² https://linux.die.net/man/5/pam_d

¹⁹³ <https://github.com/linux-pam/linux-pam>

Capabilities

Historically, Linux has had two levels of permissions relevant from process: low privilege (effective uid is not 0) and high privilege (effective uid is 0, aka root). Since kernel 2.2 Linux has broken up the high privileges of the root user into smaller distinct units called capabilities. We can assign only selected capabilities for a process/executable without the need to give the full root access. Thus, limiting the risk due to the reduction in the set of privileges granted.

Overall, there are 5 capabilities sets for a task (process or thread): CapEff (Effective Capabilities), CapPrm (Permitted Capabilities), CapInh (Inherited Capabilities), CapBnd (Bounding Set) and CapAmb (Ambient Capabilities Set). Let us go over each of these.

CapEff, this set represents all the capabilities the process is using at a specific moment in time. Those are the privileges which the kernel performs permission checks on.

CapPrm, this set is a superset which acts as a limiting boundary for the effective set. Those capabilities which are not set cannot be enabled in the effective set (they are some edge cases that we are going to talk about in the following write-up).

CapInh, specifies all the capabilities allowed to be inherited from parent to child, after a call to execve syscall (for privilege process/threads).

CapBnd, by using this we can block capabilities that we don't want a process to receive, only those that are in the set will be allowed in the permitted and inherited sets.

CapAmb, applies to non-suid (non privileged) executables that don't have file capabilities (more about it in the next write-up). The goal is to keep capabilities in case of calling the execve syscall family. It is important to know that not all the capabilities in the set can be kept like those which are dropped if they are not in the inheritable/permitted capability set.

In order to see the different sets for a task we can read the file “/proc/[pid]/status” (for a process) or “/proc/[pid]/task/[tid]/status” (for a specific thread) — as shown in the screenshot below (taken from a copy.sh).

There are different capabilities such as: CAP_AUDIT_LOG (gives the ability to write data to the kernel audit log), CAP_CHOWN (gives the ability to change the GIDs and UIDs of files), CAP_DAC_OVERRIDE (bypasses the permissions of files [r/w/x]) and more. To see the list of capabilities and the kernel version which they are relevant for please checkout “man capabilities”.

```
root@localhost:~# cat /proc/$$/status | grep Cap
CapInh: 0000000000000000
CapPrm: 000001ffffffffffff
CapEff: 000001ffffffffffff
CapBnd: 000001ffffffffffff
CapAmb: 0000000000000000
root@localhost:~# cat /proc/$$/task/1250/status | grep Cap
CapInh: 0000000000000000
CapPrm: 000001ffffffffffff
CapEff: 000001ffffffffffff
CapBnd: 000001ffffffffffff
CapAmb: 0000000000000000
root@localhost:~# _
```

chroot (Change Root Directory)

chroot is a Linux system call which allows changing the root directory of a calling process to a specific path. After doing so the directory will be used for the path names beginning with “/”. The changed root directory is inherited to all children of the calling process. By the way, only privileged processes can call “chroot” - root or with “CAP_SYS_CHROOT” in its user namespace¹⁹⁴.

Moreover, there are different use case (which are not just security related) for using chroot like: rebuilding initramfs image, reinstalling a bootloader, upgrading/downgrading a package and more¹⁹⁵. By the way, we can use the “chroot” CLI tool (and not the system call) for preventing access outside the new root directory¹⁹⁶ - as shown in the screenshot below. It is recommended to go over the implementation of the “chroot” syscall¹⁹⁷.

Lastly, we can think about “chroot” as a mitigation/hardening feature (and not a security feature) due to the fact there are specific ways to bypass it¹⁹⁸. We can find it in use when creating sandboxed environments¹⁹⁹ - they are better solutions than just using “chroot” as described in future writeups (namespaces and seccomp as an example). An example for that is “wu-ftpd” which can run in a chrooted environment for anonymous users²⁰⁰.

```
Troller $ ls /tmp/troller/
busybox sh
Troller $ sudo chroot /tmp/troller/ ./sh

BusyBox [REDACTED] (ash) built-in shell (ash)
Enter 'help' for a list of built-in commands.

Troller $ ls
busybox sh
Troller $ pwd
/
```

¹⁹⁴ <https://man7.org/linux/man-pages/man2/chroot.2.html>

¹⁹⁵ <https://wiki.archlinux.org/title/chroot>

¹⁹⁶ <https://linux.die.net/man/1/chroot>

¹⁹⁷ <https://elixir.bootlin.com/linux/v6.5.5/source/fs/open.c#L593>

¹⁹⁸ <https://www.redhat.com/en/blog/chroot-security-feature>

¹⁹⁹ <https://www.lenovo.com/us/en/glossary/what-is-chroot/>

²⁰⁰ <https://www.ariadne.ac.uk/issue/20/unix/>

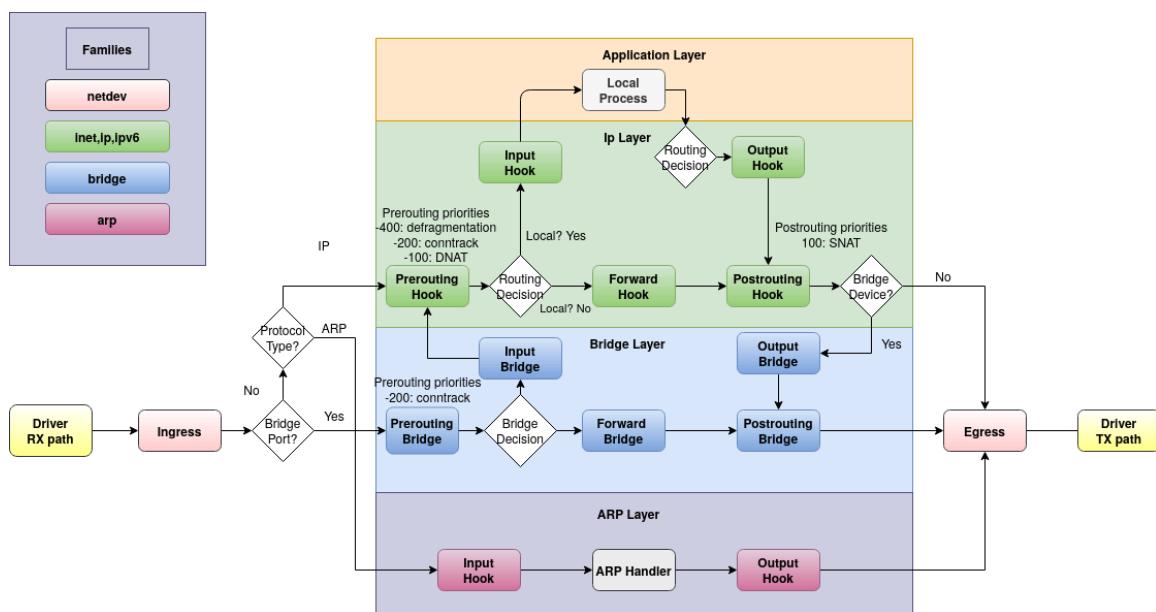
NetFilter

NetFilter is a “Free and Open Source” (FOSS) project that provides packet filtering software for Linux (kernel 2.4 version and later). The main features provided by NetFilter are: stateless packet filtering (IPv4/IPv6), stateful packet filtering (IPv4/IPv6), different kinds of network/port address translations (NAT/PAT), packet logging, userspace packet queuing and other packet mangaling²⁰¹. Thus, NetFilter is used for creating Firewalls (stateless/stateful), NAT based transparent proxies and other packet manipulation technologies.

One of the most important features of NetFilter is “Connection Tracking”. It allows the kernel to keep track of all the sessions/network connections in order to relate all the packets that make up a connection²⁰². We can interface with the connection tracking feature using the “conntrack” CLI tool²⁰³.

Moreover, NetFilter provides “netfilter hooks” which enables using callbacks to provide filtering inside the Linux kernel. There are five different types of “netfilter hooks”: “Pre-Routing”, “Input”, “Forward”, “Output” and “Post-Routing” - as shown in the diagram below²⁰⁴.

Lastly, there are different tools that leverage NetFilter like: iptables, arptables, ebtables and nftables (more on them in future writeups). We can also go over the source code of “NetFilter” as part of the Linux kernel²⁰⁵.



²⁰¹ <https://www.netfilter.org/>

²⁰² <https://en.wikipedia.org/wiki/Netfilter>

²⁰³ <https://manpages.ubuntu.com/manpages/trusty/man8/conntrack.8.html>

²⁰⁴ https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks

²⁰⁵ <https://elixir.bootlin.com/linux/v6.5.5/source/net/netfilter>

Chains (iptables)

In general “iptables” is an administration tool used IPv4/6 packet filtering and NAT²⁰⁶. “iptables” uses a series of rules that are organized into chains, in order to handle network traffic. Overall there are 5 built-in chains: PREROUTING, INPUT, FORWARD, OUTPUT and POSTROUTING. Those chains are based on the NetFilter’s hooks callbacks²⁰⁷. We can also see that in the source code both for IPv4²⁰⁸ and IPv6²⁰⁹.

Moreover, we can also create user defined chains using the following command “sudo iptables -N CHAIN_NAME”. After we created the chain we can add new rules (more on rules in a future writeup) for it by specifying the chain name with the “-A” switch in “iptables” - as shown in the screenshot below. In order to move to another chain we need to use a “Jump Target”, which causes the evaluation to be done on a different chain for additional processing²¹⁰. More on the different targets which are available in a future writeup. Lastly, we also have chains in other networking tools like “ebtables”

```
root@localhost:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source          destination
Chain FORWARD (policy ACCEPT)
target     prot opt source          destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source          destination
root@localhost:~# iptables -N TROLLER
root@localhost:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source          destination
Chain FORWARD (policy ACCEPT)
target     prot opt source          destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source          destination
Chain TROLLER (0 references)
target     prot opt source          destination
root@localhost:~# iptables -A TROLLER -p tcp --dport 2222 -j DROP
root@localhost:~# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source          destination
Chain FORWARD (policy ACCEPT)
target     prot opt source          destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source          destination
Chain TROLLER (0 references)
target     prot opt source          destination
DROP      tcp   --  anywhere           anywhere          tcp dpt:EtherNet-IP-1
root@localhost:~# _
```

²⁰⁶ <https://linux.die.net/man/8/iptables>

²⁰⁷ <https://medium.com/@bouthnaru/the-linux-security-journey-netfilter-90c6cf12ca40>

²⁰⁸ https://elixir.bootlin.com/linux/v6.5.5/source/net/ipv4/netfilter/ip_tables.c#L124

²⁰⁹ https://elixir.bootlin.com/linux/v6.5.5/source/net/ipv6/netfilter/ip6_tables.c#L149

²¹⁰ <https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture#jumping-to-user-defined-chains>

Table Types (iptables)

In general “iptables” is an administration tool used IPv4/6 packet filtering and NAT²¹¹. “iptables” is based on different types of tables: FILTER, RAW, NAT and MANGLE. By the way, there is also the SECURITY table used to set internal SELinux security context on packets. The goal of the tables is to hold rules based on the area of concern we want to evaluate packets²¹².

Overall, the rules on a specific table are organized into “chains”²¹³. We can say that rules are clustered in “tables” based on their goal and “chains” represent the netfilter hooks which are going to trigger the rules. It is time to elaborate on each of the tables.

Filter table is used for controlling if a packet can get to its destination or not. It is the most used type for creating firewalls (which filters packets). A type of a NAT table is used for network address translation rules (think about deciding how to modify the destination/source IP address of a packet). We can use a table of type MANGLE for altering the IP header of a packet in the sense of changing the TTL/Type of service/internal mark for further processing²¹⁴.

Lastly, the RAW type can be used for avoiding the use of the connection tracking capability of “iptables”. For better understanding we can see the relationship between “tables” and “chains” in the illustration below²¹⁵.

Tables ↓	Chains →	PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING
filter			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
nat		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
mangle		<input checked="" type="checkbox"/>				
raw		<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	
security			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

²¹¹ <https://linux.die.net/man/8/iptables>

²¹² <https://www.digitalocean.com/community/tutorials/a-deep-dive-into-iptables-and-netfilter-architecture#relationships-between-chains-and-tables>

²¹³ <https://medium.com/@boutnaru/the-linux-security-journey-iptables-chains-5b31d1eb6b53>

²¹⁴ <https://www.linuxadictos.com/en/iptables-tipos-de-tablas.html>

²¹⁵ <https://oracle-patches.com/en/os/iptables-tutorial-how-it-works-clear-explanation-with-examples>

nftables

nftables is the replacement of the legacy “*tables” tools (iptables/ip6tables/arptables/ebtables). It is the modern Linux kernel packet classification framework. nftables has been available since version 3.13 of the Linux kernel which was released on 2014²¹⁶.

Overall, the creation of nftables is due to different limitations both at the functional and code design level. The core design of nftables is based on a pseudo-virtual machine inspired by BPF - an example is shown in the screenshot below²¹⁷. There is a backward compatibility regarding iptables, thus we can use the specific versions of iptables/iptables utilities that are able to convert iptables rules to nftables bytecode²¹⁸.

Moreover, among the differences between iptables and nftables we can include that: nftables users a new syntax, a single nftables rule can take multiple actions, support for new protocols without the need for a kernel update, no built-in counter per chain/rule, Support for concatenations (since kernel 4.1) tables/chains a fully configurable, simplified dual stack for IPv4/6 administration and better support for dynamic rule set updates²¹⁹.

Lastly, new Linux distributions use nftables as the recommended/default firewalling framework²²⁰. The user-mode command line tool for managing nftables is “nft”. We can summarize that nftables is a project of netfilter providing firewalling/NAT/packet mangling capabilities for Linux²²¹.

```
nft --debug=netlink add rule inet t c ip daddr 10.1.2.3 counter accept
inet t c
[ meta load nfproto => reg 1 ]
[ cmp eq reg 1 0x00000002 ]
[ payload load 4b @ network header + 16 => reg 1 ]
[ cmp eq reg 1 0x0302010a ]
[ counter pkts 0 bytes 0 ]
[ immediate reg 0 accept ]
```

²¹⁶ https://elixir.bootlin.com/linux/v3.13-rc1/source/net/netfilter/nf_tables_core.c
²¹⁷ https://wiki.nftables.org/wiki-nftables/index.php/Ruleset_debug/VM_code_analysis
²¹⁸ https://kernelnewbies.org/Linux_3.13#head-f628a9c41d7ec091f7a62db6a49b8da50659ec88
²¹⁹ https://wiki.nftables.org/wiki-nftables/index.php/What_is_nftables%3F
²²⁰ <https://wiki.debian.org/nftables>
²²¹ <https://netfilter.org/projects/nftables/>

Secure Execution Mode

In general, a binary is executed in “Secure Execution Mode” in case the “AT_SECURE” entry of the auxiliary vector²²² contains a non-zero value. They are different cases that causes this value to be non zero such as: a LSM²²³ has set the value, the “real uid”²²⁴ and the “effective uid”²²⁵ of the task/process differ (and the same for the groups values), a non-root user executed a binary which conferred capabilities to the process²²⁶.

Overall, secure execution mode is a feature of the dynamic linker/loader. In case it is enabled specific environment variables are ignored when executing a binary. Examples of such variables are: “LD_LIBRARY_PATH”, “LD_DEBUG” (unless /etc/suid-debug exists), “LD_DEBUG_OUTPUT”, “LD_DEBUG_WEAK” (since glibc 2.3.4), “LD_ORIGIN_PATH”, “LD_PROFILE” (since glibc 2.2.5), “LD_SHOW_AUXV” (since glibc 2.3.4) and “LD_AUDIT”²²⁷ - as shown in the screenshot below.

Lastly, the goal of the secure execution mode is to block the ability of causing a binary which can be executed with “setuid”/“setgid” to load/executed arbitrary code and thus perform a privilege escalation (due to the fact it is executed by one user but executed with the permissions of another user which case also be root).

```
[troller@localhost troller]$ ls -la --color
total 92
drwxr-xr-x 2 root root 0 2024 .
drwxrwxrwt 5 root root 51 18:35 .
-rwxr-xr-x 1 root root 46556 02:52 normal_id
-rwsr-sr-x 1 root root 46556 2024 suid_id
[troller@localhost troller]$ export LD_DEBUG=statistics
[troller@localhost troller]$ ./normal_id
1353:
1353: runtime linker statistics:
1353:   total startup time in dynamic loader: 870000 cycles
1353:     time needed for relocation: 183999 cycles (21.1%)
1353:       number of relocations: 165
1353:       number of relocations from cache: 9
1353:       number of relative relocations: 28
1353:         time needed to load objects: 368000 cycles (42.2%)
uid=1000(troller) gid=1000(troller) groups=1000(troller)
[troller@localhost troller]$ ./suid_id
uid=1000(troller) gid=1000(troller) euid=0(root) egid=0(root) groups=0(root),1000(troller)
[troller@localhost troller]$
```

²²² <https://medium.com/@boutnaru/linux-the-auxiliary-vector-auxv-cba527871b50>
²²³ <https://medium.com/@boutnaru/linux-security-lsm-linux-security-modules-907bbcf8c8b4>
²²⁴ <https://medium.com/@boutnaru/the-linux-security-journey-ruid-real-user-id-b23abcbca9c6>
²²⁵ <https://medium.com/@boutnaru/the-linux-security-journey-euid-effective-user-id-65f351532b79>
²²⁶ <https://man7.org/linux/man-pages/man8/ld.so.8.html>
²²⁷ <https://manpages.ubuntu.com/manpages/focal/en/man8/ld.so.8.html>

dmesg_restrict

dmesg (Diagnostic Message) is used for printing the message buffer of the kernel²²⁸. Due to the fact it may contain sensitive information about the system we can control which users can read it with “/proc/sys/kernel/dmesg_restrict”²²⁹. An example would be to read kernel addresses from dmesg and thus bypass²³⁰ security protections like KASLR²³¹ (Kernel Address Space Layout Randomization).

Overall, it has two distinct values (zero and one). “dmesg_restrict=0” means there are no restrictions and all users can read information from dmesg. “dmesg_restrict=1” restricts access²³² only to users which have the “CAP_SYSLOG” capability²³³ - as shown in the screenshot below²³⁴.

Lastly, this feature is controlled by “SECURITY_DMESG_RESTRICT”²³⁵. It is relevant since kernel version “2.6.37”²³⁶. The configuration is read into “dmesg_restrict” variable checked in “syslog_action_restricted”²³⁷. We can also check out the source code used to expose “dmesg_restrict” as part of sysctl²³⁸ through procfs²³⁹.

```
linuxopsys@linux:~$ dmesg
dmesg: read kernel buffer failed: Operation not permitted
linuxopsys@linux:~$ sudo sysctl -w kernel.dmesg_restrict=0
kernel.dmesg_restrict = 0
linuxopsys@linux:~$ dmesg
[    0.000000] Linux version 5.13.0-39-generic (buildd@lcy02-amd64-080) (gcc (Ubuntu 9.4.0-1ubuntu1-20.04.1) 9.4.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #44~20.04.1-Ubuntu SMP Thu Mar 24 16:43:35 UTC 2022 (Ubuntu 5.13.0-39.44~20.04.1-generic 5.13.19)
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.13.0-39-generic root=UUID=
```

²²⁸ <https://linux.die.net/man/8/dmesg>

²²⁹ https://sysctl-explorer.net/kernel/dmesg_restrict/

²³⁰ <https://nvd.nist.gov/vuln/detail/CVE-2018-7273>

²³¹ <https://medium.com/@boutnaru/the-linux-security-journey-kaslr-kernel-address-space-layout-randomization-6d5554766fe1>

²³² https://linuxsecurityexpertkb.sysctl/kernel_dmesg_restrict

²³³ <https://medium.com/@boutnaru/linux-security-capabilities-part-1-63c6d2ceb8bf>

²³⁴ <https://linuxopsys.com/topics/dmesg-command-in-linux>

²³⁵ <https://elixir.bootlin.com/linux/v6.15.4/source/security/Kconfig#L10>

²³⁶ <https://elixir.bootlin.com/linux/v2.6.37/source/security/Kconfig#L42>

²³⁷ <https://elixir.bootlin.com/linux/v6.15.4/source/kernel/printk/printk.#L629>

²³⁸ <https://elixir.bootlin.com/linux/v6.15.4/source/kernel/printk/sysctl.c#L63>

²³⁹ <https://medium.com/@boutnaru/the-linux-concept-journey-proofs-proc-filesystem-f6b2e3aa5550>

TCP Wrappers

TCP wrappers can be used (in services which support it) for restricting access based on IP\hostname to specific services²⁴⁰. Thus, we can think about it as an host-based networking ACL (Access Control List) feature. By the way, it has been created in the 90s by “Wietse Venema”²⁴¹, which also created other projects like the “Postfix” email system, SATAN (Security Administrator Tool for Analyzing Networks) and “The Coroner’s Toolkit”²⁴².

Overall, tcp wrappers check the “host access files” (“/etc/hosts.allow” and “/etc/hosts.deny”) for determining if a specific client can connect to a specific service. It is import to understand that rules in “hosts.allow” precedences rules stored in “hosts.deny”. Also, the order of rules is crucial due to the fact the first matching rule is applied. Moreover, in case no rule is found access is granted and there is no cache for the rules which means they are checked each time. Hence, upon changes in the “host access files” affect immediately. Each rule has the following basic format “<daemon list> : <client list> [: <option> : <option> : ...]”²⁴³.

Lastly, probably the most common ways for integrating tcp wrappers is by a service being launched by “tcpd” or compiling the service with libwrap²⁴⁴. For example sshd, apache2, ufw and vsftpd as compiled with libwrap²⁴⁵ - as shown in the output of “ldd”²⁴⁶ in the screenshot below.

```
gacanepa@ubuntu:~$ ldd $(which sshd) | grep libwrap
    libwrap.so.0 => /lib/i386-linux-gnu/libwrap.so.0 (0xb766e000)
gacanepa@ubuntu:~$ ldd $(which apache2) | grep libwrap
gacanepa@ubuntu:~$ ldd $(which ufw) | grep libwrap
gacanepa@ubuntu:~$ ldd $(which vsftpd) | grep libwrap
    libwrap.so.0 => /lib/i386-linux-gnu/libwrap.so.0 (0xb7706000)
gacanepa@ubuntu:~$
```

²⁴⁰ <https://sternumiot.com/iot-blog/linux-security-hardening-19-best-practices-with-linux-commands/>

²⁴¹ https://en.wikipedia.org/wiki/TCP_Wrappers

²⁴² https://en.wikipedia.org/wiki/Wietse_Venema

²⁴³ <https://tinyurl.com/yw8cs2n2>

²⁴⁴ <https://www.debian.org/doc/manuals/securing-debian-manual/tcpwrappers.en.html>

²⁴⁵ <https://fr.linuxgeek.com/article/how-to-secure-network-services-using-tcp-wrappers-in-linux>

²⁴⁶ <https://medium.com/@boutaru/linux-instrumentation-part-4-ldd-888502965a9b>

TCP SYN Cookie Protection

The goal of “TCP SYN Cookie” is to protect against TCP (Transmission Control Protocol) SYN flooding (which can lead to denial of service aka DoS). TCP SYN flooding can be used for taking up all resources of a specific system. This is done by initiating a large number of TCP connections (sending only TCP segments with the SYN flag toggled and not responding to the SYN+ACK response) from a spoofed source IP (Internet Protocol) addresses²⁴⁷

Overall, the way in which TCP SYN cookie solves SYN floods is by using data from the client’s SYN packet and from server-side to calculate a random initial sequence number - as shown below²⁴⁸. Thus, no resources are being allocated after getting a TCP SYN segment (which removes the ability of resource exhaustion). When an ACK is received the acknowledgement number is verified. If everything is correct the connection is established (and resources are allocated) if not the connection is refused and no resources are allocated²⁴⁹. Because the cookie calculation is based on fields from the IP header there is an implementation both for IPv4²⁵⁰ and IPv6²⁵¹. The support of TCP SYN cookie is controlled using the “SYN_COOKIES” configuration parameter²⁵². Which is included (“CONFIG_SYN_COOKIES”) as part of the Linux kernel since kernel version “2.4.0”²⁵³.

Lastly, the cookie generation is done in the “secure_tcp_syn_cookie”²⁵⁴. We can enable\disable the feature by writing “1”\“2”,”0” to “/proc/sys/net/ipv4/tcp_syncookies” or using sysctl²⁵⁵. Also, there are BPF helper function to work with TCP SYN cookies like (but not limited to): “bpf_tcp_gen_syncookie”²⁵⁶ and “bpf_tcp_check_syncookie”²⁵⁷.

```
/*
 * Compute the secure sequence number.
 * The output should be:
 *   HASH(sec1,saddr,sport,daddr,dport,sec1) + sseq + (count * 2^24)
 *   + (HASH(sec2,saddr,sport,daddr,dport,count,sec2) % 2^24).
 * Where sseq is their sequence number and count increases every
 * minute by 1.
 * As an extra hack, we add a small "data" value that encodes the
 * MSS into the second hash value.
 */
```

²⁴⁷ https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_data_zbf/configuration/xe-16/sec-data-zbf-xe-16-book/conf-fw-tcp-syn-cookie.pdf

²⁴⁸ <https://elixir.bootlin.com/linux/v6.15.5/source/net/ipv4/synccookies.c#L88>

²⁴⁹ <https://www.geeksforgeeks.org/computer-networks-how-syn-cookies-are-used-to-preventing-syn-flood-attack/>

²⁵⁰ <https://elixir.bootlin.com/linux/v6.15.5/source/net/ipv4/synccookies.c>

²⁵¹ <https://elixir.bootlin.com/linux/v6.15.5/source/net/ipv6/synccookies.c>

²⁵² <https://elixir.bootlin.com/linux/v6.15.5/source/net/ipv4/Kconfig#L268>

²⁵³ https://elixir.bootlin.com/linux/2.4.0/source/net/ipv4/tcp_ip4.c#L1287

²⁵⁴ https://elixir.bootlin.com/linux/v6.15.5/A/ident/secure_tcp_syn_cookie

²⁵⁵ https://www.tenable.com/audits/items/CIS_Rocky_Linux_8_v1.0.0_L1_Server.audit.95e3320517071e79c94501bed716202c

²⁵⁶ <https://elixir.bootlin.com/linux/v6.15.5/source/net/core/filter.c#L7514>

²⁵⁷ <https://elixir.bootlin.com/linux/v6.15.5/source/net/core/filter.c#L7441>

UFW (Uncomplicated Firewall)

UFW (Uncomplicated Firewall) is a firewall configuration tool for easing netfilter²⁵⁸ based firewall configuration. Thus, it provides a user-friendly way for creating an IPv4\IPv6 host-based firewall. UFW is the default tool for the job in case of Ubuntu, which is by default disabled. In case we enable ufw (“sudo ufw enable”) it uses a default set of rules (profile) that should be fine for the average home user (or so the developers believe). All 'incoming' is denied, with some exceptions to make things easier for home users²⁵⁹.

Overall, UFW is available by default as part of Ubuntu since version 8.04 LTS. Also, it is included as part of Debian since version 10. For easier usage there is also “Gufw” (GUI for Uncomplicated Firewall) which is built using Python, GTK and of course ufw²⁶⁰ - as shown in the screenshot below.

Lastly, “ufw” provides different capabilities such as (but not limited to): default incoming policy (allow/deny), IPv6 support, logging, per rule logging, rsyslog support, rate limiting and filtering by interface. We can checkout the configuration files located at “/etc/ufw/*”²⁶¹.



²⁵⁸ <https://medium.com/@boutnaru/the-linux-security-journey-netfilter-90c6cf12ca40>

²⁵⁹ <https://help.ubuntu.com/community/UFW>

²⁶⁰ https://en.wikipedia.org/wiki/Uncomplicated_Firewall

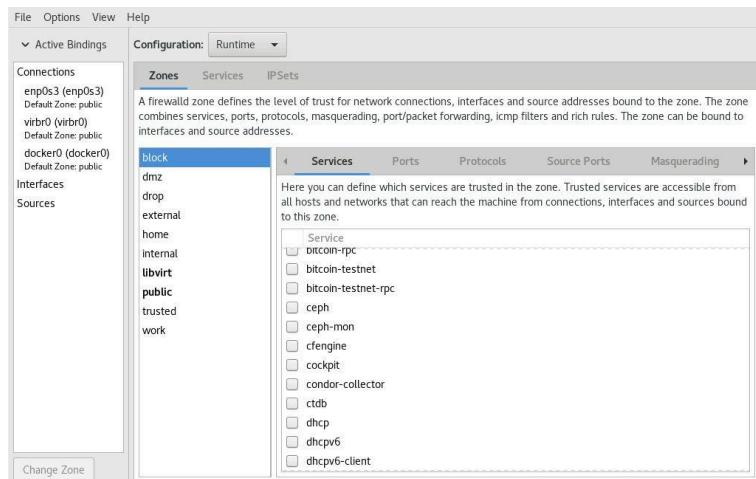
²⁶¹ <https://wiki.ubuntu.com/UncomplicatedFirewall>

Firewalld (Firewall Daemon)

Firewalld (Firewall Daemon) is a firewall management tool for Linux based systems. Hence, basically “firewalld” acts as a front-end for netfilter²⁶². As with UFW (Uncomplicated Firewall) it is also written in Python. firewlld is included and enabled by default as part of different Linux distributions such as: CentOS (from version 7 and above), Fedora (from version 18), OpenSuse, RedHat Enterprise Linux (7 and above) and EndeavourOS. It can also be installed by others from their package repositories like in Debian or Ubuntu²⁶³.

Overall, firewalld has a variety of features including (but not limited to): IPv4\IPv6 NAT (Network Address Table) support, integration with Puppet, logging of denied packets, firewall zones, timed firewall rules²⁶⁴ and complete D-BUS²⁶⁵ API. firewalld provides different levels of security based on a concept called connection zones. A zone is associated with at least one network interface (like eth0)²⁶⁶.

Lastly, there are multiple zones which are defined by default: “drop”, “dmz”, “external”, “home”, “trusted” and more - as shown in the screenshot below²⁶⁷. The screenshot is taken from “firewall-config” which is a GUI interface for accessing firewalld as opposed to “firewall-cmd” which is a CLI client of firewalld²⁶⁸. By the way, there is also “firewall-applet” which is a tray applet application for firewalld²⁶⁹. We can also checkout the source code of firewalld hosted on GitHub²⁷⁰.



²⁶² <https://medium.com/@boutnaru/the-linux-security-journey-netfilter-90c6cf12ca40>

²⁶³ <https://en.wikipedia.org/wiki/Firewalld>

²⁶⁴ <https://firewalld.org/>

²⁶⁵ <https://medium.com/@boutnaru/the-linux-concept-journey-d-bus-desktop-bus-dd8c69ade019>

²⁶⁶ <https://www.redhat.com/en/blog/beginners-guide-firewalld>

²⁶⁷ <https://www.how2shout.com/how-to/how-to-install-firewalld-graphical-user-interface-on-linux.html>

²⁶⁸ <https://firewalld.org/documentation/man-pages/firewall-cmd.html>

²⁶⁹ <https://firewalld.org/documentation/utilities/firewall-applet.html>

²⁷⁰ <https://github.com/firewalld/firewalld>

su (Substitute User)

“su” (Substitute User) is a utility as part of the Linux ecosystem which is used for running commands with the privileges of another user (by default the root user). This command allows us to switch to a specific account that we want in the current login session even if the user is not allowed to login using SSH/using GUI display manager²⁷¹.

Overall, when switching to another user (su [USERNAME]) we need to know the password of that target user. However, if we have root privileges we can switch to whatever user we want without the need of knowing the target user’s password - as shown in the screenshot below. The substitution is done by setting the user id with the “setuid”\“setuid32” syscall²⁷².

Lastly, we can also execute a specific command as a different user using the following pattern “su - [USERNAME] -c [COMMAND]”. For enhanced security we should restrict su access, add additional settings with PAM (Pluggable Authentication Modules) and configure logging and monitoring both locally and remotely²⁷³.

```
root@localhost:~# whoami
root
root@localhost:~# useradd troller
root@localhost:~# su troller
[troller@localhost root]$ whoami
troller
[troller@localhost root]$ exit
exit
root@localhost:~# whoami
root
root@localhost:~# su troller
[troller@localhost root]$ su root
Password:
```

²⁷¹ <https://linuxize.com/post/su-command-in-linux/>

²⁷² <https://linux.die.net/man/2/setuid32>

²⁷³ <https://labex.io/tutorials/nmap-how-to-defend-against-su-command-attacks-420285>

OpenSSH

OpenSSH is an open source implementation of the SSH (Secure Shell) protocol. It is an open-source implementation based on the free version by Tatu Ylonen and further developed by the OpenBSD team and the user community²⁷⁴. OpenSSH is a suite of tools including: remote operations (ssh, scp and sftp), key management (ssd-add, ssh-keygen, ssh-keysign and ssh-keyscan) and the service (sshd, sftp-server and ssh-agent). Thus, the main goal is to keep communication secret between client and server - as written in the banner from the official website shown below²⁷⁵.

Overall, OpenSSH is used for remote sign-in by leveraging the SSH (Secure Shell Protocol). Its main security functionality is to encrypt all traffic between client and server. Thus, it eliminates eavesdropping, connection hijacking and other security threats. It is important to know that it is not supported only by Linux but also by Windows since “Windows server 2019” and “Windows 10” (build 1809)²⁷⁶.

Lastly, OpenSSH provides different features such as (but not limited to): tunneling capabilities with the abilities like multiplexing connections and encryption, an ad hoc SOCKS proxy server and even remote file system mounting with sshfs²⁷⁷ and X11 forwarding²⁷⁸. Also, OpenSSH is supported on various Unix based operating like (but not limited to): AIX, HP-UX, Irix, Linux, NetXT, SCO Solaris, macOS and Cygwin²⁷⁹.



²⁷⁴ <https://www.ssh.com/academy/ssh/openssh>

²⁷⁵ <https://www.openssh.com/>

²⁷⁶ https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse?tabs=gui&pivots=windows-server-2025

²⁷⁷ <https://en.wikipedia.org/wiki/OpenSSH>

²⁷⁸ <https://www.openssh.com/features.html>

²⁷⁹ <https://www.openssh.com/portable.html>

Disable Kernel Modules

In case an LKM aka “Loadable Kernel Module”²⁸⁰ is loaded it can basically execute any code in kernel mode. Thus, the disable kernel module is a security feature that helps in hardening the system against attempts of loading malicious kernel modules like rootkits²⁸¹. It is important to understand that once enabled, modules can’t be either loaded or unloaded²⁸².

Overall, the configuration of this security feature is saved into the “modules_disabled” variable²⁸³. Thus, beside checking for the “CAP_SYS_MODULE” capability when trying to unload a kernel module²⁸⁴ or when trying to load a kernel module²⁸⁵ the “modules_disabled” is also checked.

Lastly, We can enable\disable this feature by writing “1” to “/proc/sys/kernel/modules_disabled” (“echo 1 > /proc/sys/kernel/modules_disabled”) or using sysctl (“sysctl kernel.modules_disabled = 1”). In case the feature is enabled when we try to load a kernel module with “insmod”²⁸⁶ the operation will fail²⁸⁷ - as shown in the screenshot below. By the way, the same goes when trying to remove a module using for example “rmmod”²⁸⁸. Remember we can use “modprobe” for performing both operations²⁸⁹.

```
echo 1 > /proc/sys/kernel/modules_disabled
# insmod xor.ko
insmod: ERROR: could not insert module xor.ko: Operation not permitted
```

²⁸⁰ <https://medium.com/@boulnaru/the-linux-concept-journey-loadable-kernel-module-lkm-5eaa4db346a1>

²⁸¹ https://dfir.ch/posts/today_i_learned_lkm_kernel_modules_disabled/

²⁸² https://sysctl-explorer.net/kernel/modules_disabled/

²⁸³ <https://elixir.bootlin.com/linux/v6.15.5/source/kernel/module/main.c#L129>

²⁸⁴ <https://elixir.bootlin.com/linux/v6.15.5/source/kernel/module/main.c#L732>

²⁸⁵ <https://elixir.bootlin.com/linux/v6.15.5/source/kernel/module/main.c#L3047>

²⁸⁶ <https://man7.org/linux/man-pages/man8/insmod.8.html>

²⁸⁷ <https://linux-audit.com/kernel/increase-kernel-integrity-with-disabled-linux-kernel-modules-loading/>

²⁸⁸ <https://linux.die.net/man/8/rmmod>

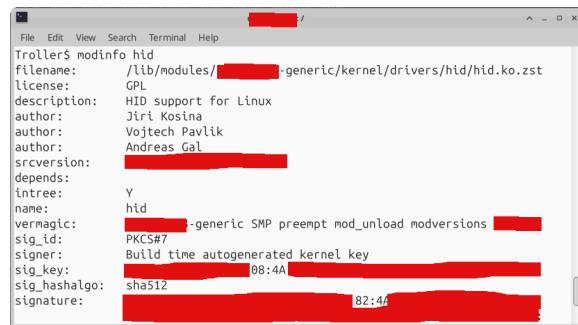
²⁸⁹ <https://linux.die.net/man/8/modprobe>

Kernel Module Signing

The Linux kernel provides the ability for cryptographically signing kernel modules during their installation. Thus, when they are being loaded the signature is validated. By doing so we increase the kernel security due to the fact that unsigned kernel modules\signed modules with an invalid key(s) are blocked from loading. We can leverage different hashing algorithms as part of the signing process like: SHA-1,SH-224, SHA-256, SHA-384 and SHA-512. Also, the public key for singing is handled using X.509 ITU-T standard certificates²⁹⁰. Based on the kernel configuration modules can be signed using a RSA key which is controlled by “CONFIG_MODULE_SIG_KEY_TYPE_RSA”²⁹¹ or using an elliptic curve key controlled by “CONFIG_MODULE_SIG_KEY_TYPE_ECDSA”²⁹². By the way, in case a kernel module is signed we can check out different attributes such as: the signature, hashing algorithm used, the signing key, the name of the signer and more using the “modinfo”²⁹³ utility - as shown below.

Overall, the main structure related to module singing is “struct module_signature”²⁹⁴ (“module signature information block”). It contains: signer’s name, key identifier, signature data and information block²⁹⁵. It is leveraged in the kernel in different places (not limited to): code for signing a module file²⁹⁶, verifying the kernel signature during “kexec_file_load”²⁹⁷ and in “mod_verify_sig”²⁹⁸ which is used for verifying the signature of a module.

Lastly, the general flow is that the “init_module_from_file” function calls “load_module”²⁹⁹. Than the “load_module” (used for allocating and loading the module) function calls the “module_sig_check” which does the signature check³⁰⁰. “module_sig_check” calls “mod_verify_sig”³⁰¹. Based on the return value from “mod_verify_sig” the “module_sig_check” function created the appropriate error message³⁰² and emits the appropriate log entry³⁰³.



²⁹⁰ <https://www.kernel.org/doc/html/v4.19/admin-guide/module-signing.html>

²⁹¹ <https://elixir.bootlin.com/linux/v6.15.6/source/certs/Kconfig#L25>

²⁹² <https://elixir.bootlin.com/linux/v6.15.6/source/certs/Kconfig#L30>

²⁹³ <https://linux.die.net/man/8/modinfo>

²⁹⁴ https://elixir.bootlin.com/linux/v6.15.6/source/include/linux/module_signature.h#L33

²⁹⁵ https://elixir.bootlin.com/linux/v6.15.6/source/include/linux/module_signature.h#L24

²⁹⁶ <https://elixir.bootlin.com/linux/v6.15.6/source/scripts/sign-file.c#L222>

²⁹⁷ https://elixir.bootlin.com/linux/v6.15.6/source/arch/s390/kernel/machine_kexec_file.c#L28

²⁹⁸ <https://elixir.bootlin.com/linux/v6.15.6/source/kernel/module/signing.c#L45>

²⁹⁹ <https://elixir.bootlin.com/linux/v6.15.6/source/kernel/module/main.c#L3601>

³⁰⁰ <https://elixir.bootlin.com/linux/v6.15.6/source/kernel/module/main.c#L3275>

³⁰¹ <https://elixir.bootlin.com/linux/v6.15.6/source/kernel/module/signing.c#L87>

³⁰² <https://elixir.bootlin.com/linux/v6.15.6/source/kernel/module/signing.c#L99>

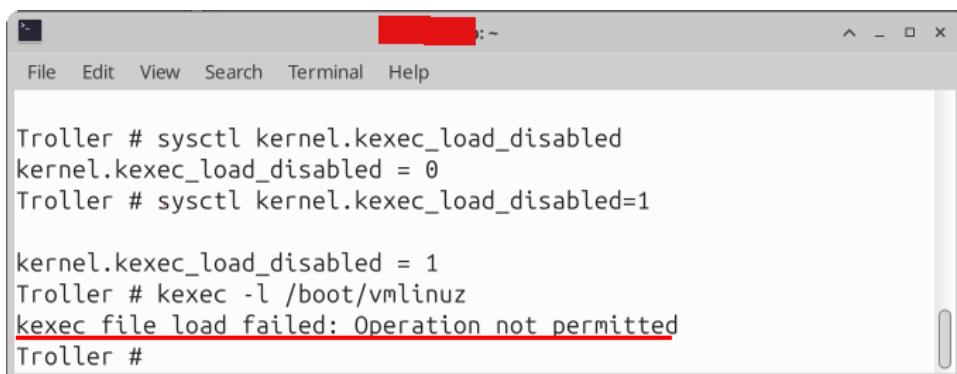
³⁰³ <https://elixir.bootlin.com/linux/v6.15.6/source/kernel/module/signing.c#L120>

Disable Kexec (Disable Kernel Execution)

When rebooting in some Linux distributions³⁰⁴ the kernel is loaded directly (without going over the firmware and the bootloader again), this is due to the use of the kexec³⁰⁵. Because kexec can be leveraged to bypass security mitigations\mechanisms like secure boot³⁰⁶ there is the ability to disable it.

Overall, the check of if kexec is disabled is done as part of the “kexec_load_permitted” boolean function³⁰⁷. It is called by both “kexec_load_check”³⁰⁸ as part of the “kexec_load” syscall call flow and part of the “kexec_load_file” syscall³⁰⁹.

Lastly, we can do it by setting “/proc/sys/kernel/kexec_load_disabled” to “1”³¹⁰, setting the “LOAD_EXEC=false” as part of “/etc/default/kexec”, using “sudo sysctl kernel.kexec_load_disabled=1”³¹¹. The specific variable in the kernel which is affected by those configurations is “kexec_load_disabled”³¹². By the way, kexec can be also blocked by the Linux kernel Lockdown feature³¹³.



The screenshot shows a terminal window with a gray header bar containing the title 'Troller' and a redacted address bar. The menu bar includes 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The main terminal area contains the following text:

```
Troller # sysctl kernel.kexec_load_disabled  
kernel.kexec_load_disabled = 0  
Troller # sysctl kernel.kexec_load_disabled=1  
  
kernel.kexec_load_disabled = 1  
Troller # kexec -l /boot/vmlinuz  
kexec file load failed: Operation not permitted  
Troller #
```

³⁰⁴ <https://medium.com/@boutmaru/the-linux-concept-journey-linux-distribution-2dfc68aaa4f4>

³⁰⁵ <https://medium.com/@boutmaru/the-linux-concept-journey-kexec-kernel-execute-e5050fa085ea>

³⁰⁶ <https://mig59.dreamwidth.org/28746.html>

³⁰⁷ https://elixir.bootlin.com/linux/v6.15.7/source/kernel/kexec_core.c#L971

³⁰⁸ <https://elixir.bootlin.com/linux/v6.15.7/source/kernel/kexec.c#L210>

³⁰⁹ https://elixir.bootlin.com/linux/v6.15.7/source/kernel/kexec_file.c#L342

³¹⁰ https://sysctl-explorer.net/kernel/kexec_load_disabled/

³¹¹ <https://www.maketecheasier.com/secure-linux-server/>

³¹² https://elixir.bootlin.com/linux/v6.15.7/source/kernel/kexec_core.c#L898

³¹³ <https://elixir.bootlin.com/linux/v6.15.7/source/kernel/kexec.c#L222>

USB Guard (Universal Serial Bus Guard)

“USB Guard” (Universal Serial Bus Guard) is a software framework which aids in protecting a Linux based computer system against rogue USB devices. This is done by allowing\preventing USB devices based on device attributes. USB guard provides a rule language for writing USB device policies³¹⁴. For enforcing the policy it leverages the USB device authorization which is implemented as part of the Linux kernel³¹⁵.

Overall, the rule policies can have different actions like: allow (authorize the device), block (deauthorize the device) and reject (remove the device from the system). The rules can be based on: the device ID (“`vendor_id:product_id`”), device attributes (like name, serial number, port ID) and conditions³¹⁶.

Lastly, the rule set is stored in “`/etc/usbguard/rules.conf`”. For ease of use we can create a rule-set based on the currently connected devices using the following command: “`usbguard generate-policy > /etc/usbguard/rules.conf`” while running with root privileges³¹⁷. For more information we can check the source code of USB guard as part of its GitHub repository³¹⁸.

Allow a specific Yubikey device to be connected via a specific port. Reject everything else on that port.

```
allow 1050:0011 name "Yubico Yubikey II" serial "0001234567" via-port "1-2" hash "044b5e168d40ee0245478  
reject via-port "1-2"
```

³¹⁴ <https://usbguard.github.io/>

³¹⁵ <https://www.kernel.org/doc/Documentation/usb/authorization.txt>

³¹⁶ <https://usbguard.github.io/documentation/rule-language.html>

³¹⁷ <https://wiki.archlinux.org/title/USBGuard>

³¹⁸ <https://github.com/USBGuard/usbguard>

USB Auth (Universal Serial Bus Authorization)

USB Auth (Universal Serial Bus Authorization) is a software framework which aids in protecting a Linux based computer system against rogue USB devices. Hence, it allows locking down the usage of USB devices. By default, when a USB device is connected it is automatically configured and accessible by users. Using this feature root users can authorize specific devices only³¹⁹.

Overall, USB authorization is a fundamental dependency required by “USB Guard”³²⁰ in order to work correctly. The configuration is done from user-mode by leveraging “/sys” which of type sysfs³²¹. For example we can authorize a device using the following command “echo 1 > /sys/bus/usb/devices/{DEVICE}/authorized”, where the “DEVICE” represents the device we want to authorize. In case we want to de-authorize a device we just write “0”³²² - as shown below.

Lastly, the relevant function as part of sysfs implementation used to perform the authorize\de-authorize is “authorized_store”³²³. This function calls the relevant function in the USB hub core driver implementation based on the requested operation. The function “usb_authorize_device”³²⁴ is called for authorization and the “usb_deauthorize_device” function³²⁵ is called for de-authorization.

Set new devices connected to hostX to be deauthorized by default (ie: lock down):

```
$ echo 0 > /sys/bus/usb/devices/usbX/authorized_default
```

Remove the lock down:

```
$ echo 1 > /sys/bus/usb/devices/usbX/authorized_default
```

³¹⁹ <https://www.kernel.org/doc/Documentation/usb/authorization.txt>

³²⁰ <https://medium.com/@boutnaru/the-linux-security-journey-usb-guard-universal-serial-bus-guard-66e1a45ac901>

³²¹ <https://medium.com/@boutnaru/the-linux-concept-journey-sysfs-filesystem-for-exporting-kernel-objects-ea8c0172b1cf>

³²² <https://docs.kernel.org/usb/authorization.html>

³²³ <https://elixir.bootlin.com/linux/v6.15.8/source/drivers/usb/core/sysfs.c#L739>

³²⁴ <https://elixir.bootlin.com/linux/v6.15.8/source/drivers/usb/core/hub.c#L2766>

³²⁵ <https://elixir.bootlin.com/linux/v6.15.8/source/drivers/usb/core/hub.c#L2751>

sudo (SuperUser Do)

sudo (SuperUser Do) is a command line part of Linux which can be used for temporarily elevating privileges. By doing so we can allow users to perform administrative tasks without logging as the root user³²⁶. Also, with sudo we can execute commands as a specific user (not only root) by leveraging the “-u” argument and providing the name of the target username³²⁷ - as shown in the screenshot below.

Overall, “sudo” is similar to “su” command line utility³²⁸. However, there are differences between them such as: “sudo” asks for our password while “su” asks for the password of the target user whom we are switching to, for using “sudo” we need to have a relevant entry in “/etc/sudoers” for the command we want to execute and “sudo” allows us to issue commands as another user without changing your identity³²⁹.

Lastly, in order to edit the sudoers files it is recommended to use “visudo” which opens a text editor (like normal) that validates the syntax of the file upon saving³³⁰. sudo was also added as part of the Windows operating system³³¹.

```
root@localhost:~# whoami
root
root@localhost:~# useradd troller
root@localhost:~# sudo -u troller whoami
troller
root@localhost:~# sudo -i -u troller
[troller@localhost root]$ whoami
troller
[troller@localhost root]$ exit
logout
root@localhost:~#
```

³²⁶ <https://phoenixnap.com/kb/linux-sudo>

³²⁷ <https://unix.stackexchange.com/questions/176997/sudo-as-another-user-with-their-environment>

³²⁸ <https://medium.com/@boutnaru/the-linux-security-journey-su-substitute-user-4c50cf6df034>

³²⁹ <https://www.redhat.com/en/blog/difference-between-sudo-su>

³³⁰ <https://www.digitalocean.com/community/tutorials/how-to-edit-the-sudoers-file>

³³¹ <https://learn.microsoft.com/en-us/windows/sudo/>

Authentication Logs

Authentication logs can be used for viewing different security and access-related events in Linux. On different Linux distributions³³². For example on Ubuntu\Debian systems the logs are stored on “/var/log/auth.log” while on Fedora\RedHat\CentOS the logs are located at “/var/log/secure”³³³.

Overall, we can find in the authentication logs information about login attempts (like using SSH), sudo commands and more³³⁴ - as shown in the screenshot below³³⁵. By the way, we can also write directly to the authentication logs using the “logger” command line utility³³⁶. This can be done for example using the following command “logger -p auth.info ‘Tr0Ller Message’”.

Lastly, we can leverage the logs to check for authentication failures, for successful\failed SSH connections, usage of an illegal user\user which does not exists and more security related events³³⁷. We can also watch this information using “journalctl” and even send it using the syslog protocol.

```
guestuser@guestuser-VirtualBox: ~
guestuser@guestuser-VirtualBox: $ sudo tail -f /var/log/auth.log
[sudo] password for guestuser:
Jun 13 19:54:33 guestuser-VirtualBox gpasswd[6285]: user workuser added by root
to group plugdev
Jun 13 19:54:33 guestuser-VirtualBox gpasswd[6292]: user workuser added by root
to group sambashare
Jun 13 19:54:33 guestuser-VirtualBox gpasswd[6299]: user workuser added by root
to group lxd
Jun 13 19:54:33 guestuser-VirtualBox accounts-daemon: request by system-bus-name
::1.164 [gnome-control-center pid:6136 uid:1000]: set password and hint of user
'workuser' (1001)
Jun 13 19:54:33 guestuser-VirtualBox usermod[6305]: change user 'workuser' passw
ord
Jun 13 19:55:29 guestuser-VirtualBox sudo: questuser : TTY=pts/0 ; PWD=/home/gue
stuser ; USER=root ; COMMAND=/usr/bin/tail -f /var/log/auth.log
Jun 13 19:55:29 guestuser-VirtualBox sudo: pam_unix(sudo:session): session opene
d for user root(uid=0) by (uid=1000)
Jun 13 19:56:38 guestuser-VirtualBox sudo: pam_unix(sudo:session): session close
d for user root
Jun 13 19:56:57 guestuser-VirtualBox sudo: guestuser : TTY=pts/0 ; PWD=/home/gue
stuser ; USER=root ; COMMAND=/usr/bin/tail -f /var/log/auth.log
Jun 13 19:56:57 guestuser-VirtualBox sudo: pam_unix(sudo:session): session opene
d for user root(uid=0) by (uid=1000)
```

run this command to view auth.log history of the user

³³² <https://medium.com/@boutnaru/the-linux-concept-journey-linux-distribution-2dfc68aaa4f4>

³³³ <https://betterstack.com/community/guides/logging/monitoring-linux-auth-logs/>

³³⁴ <https://www.netsuron.com/articles/top-5-linux-log-file-groups-in-var-log>

³³⁵ <https://linuxier.com/how-to-check-login-history-in-linux/>

³³⁶ <https://linux.die.net/man/1/logger>

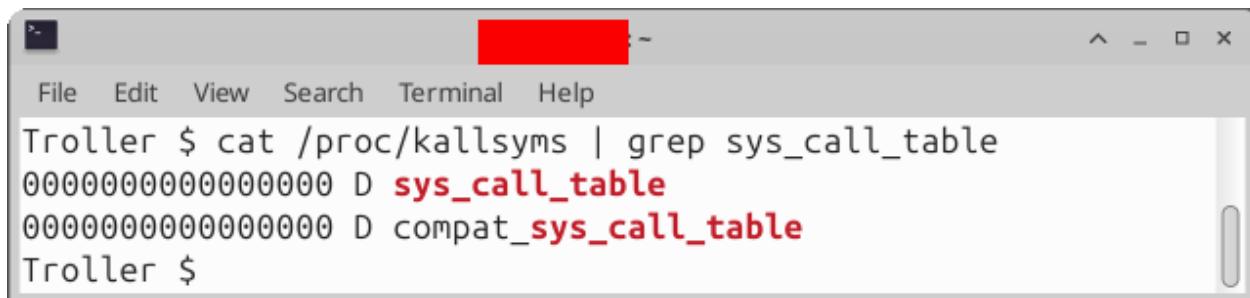
³³⁷ <https://github.com/munin-monitoring/contrib/blob/master/plugins/system/auth>

Reduced Access to Syscalls

System calls (syscalls) provide an interface between user-mode and kernel mode code³³⁸. Due to security protections we can't modify the code of a syscall from user-mode, we can just select the syscall we want to execute and pass parameters to it based on an ABI³³⁹. However, even this small attack surface can still lead to security vulnerabilities that circumvent Linux security measures³⁴⁰.

Overall, we can eliminate many syscalls on 64-bit systems by building the kernel with the “CONFIG_COMPAT” configuration item disabled³⁴¹. Enabling this configuration allows the system to handle syscalls from ELF binaries which are not 64-bit (for example x86 32 bit). This is relevant for different CPU architectures like (but not limited to): PowerPC, RISC-V, S390, Sparc and Mips³⁴². When enabled the symbol “compat_sys_call_table” is defined³⁴³ which is parallel to the symbol of the standard syscall table “sys_call_table”³⁴⁴. We can check this out by reading the output of “kallsyms”³⁴⁵ - as shown in the screenshot below.

Lastly, we can also use seccomp for restricting system calls that applications can use³⁴⁶. Those types of mitigations are also part of other operating systems like the “win32k.sys system call filtering” on Windows³⁴⁷.



```
Troller $ cat /proc/kallsyms | grep sys_call_table
0000000000000000 D sys_call_table
0000000000000000 D compat_sys_call_table
Troller $
```

³³⁸ <https://medium.com/@boutnaru/the-linux-concept-journey-syscalls-system-calls-efcd7703e072>

³³⁹ <https://man7.org/linux/man-pages/man2/syscall.2.html>

³⁴⁰ <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=syscall+linux>

³⁴¹ <https://www.kernel.org/doc/html/v4.19/security/self-protection.html>

³⁴² <https://eateee.net/lkddb/web-lkddb/COMPAT.html>

³⁴³ <https://elixir.bootlin.com/linux/v6.15.8/source/arch/arm64/include/asm/syscall.h#L17>

³⁴⁴ <https://elixir.bootlin.com/linux/v6.15.8/source/arch/arm64/include/asm/syscall.h#L14>

³⁴⁵ <https://medium.com/@boutnaru/the-linux-concept-journey-proc-kallsyms-kernel-exported-symbols-2bb199f123b9>

³⁴⁶ <https://medium.com/@boutnaru/linux-security-secure-computing-mode-seccomp-2314c3e77a76>

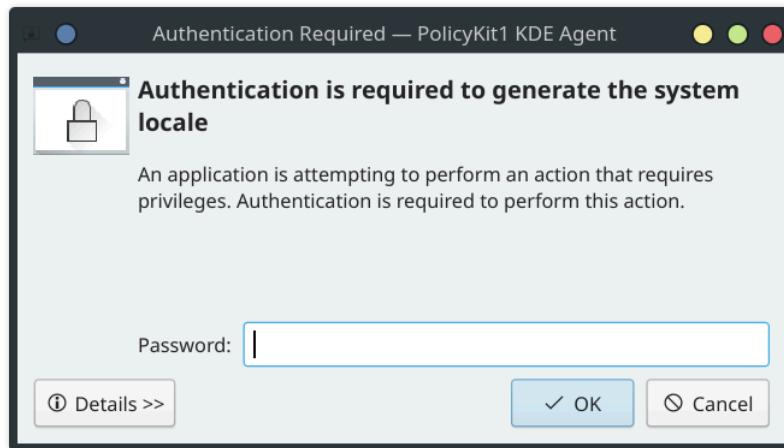
³⁴⁷ <https://github.com/mith-hff/win32k-mitigation>

PolKit (Authorization Manager)

PolKit (Authorization Manager) is used for handling policies which allow unprivileged processes to speak to privileged processes. Thus, it basically a framework for centralizing the decision making with respect to granting access to privileged operations for unprivileged applications³⁴⁸.

Overall, it is used to be called “PolicyKit” due to providing the ability to handle authorization based on policy. We can also go over its GitHub code repository for more information³⁴⁹. In case polkit applications (apps using polkit authority as a decider component) a “*.policy” file should be created at “/usr/share/polkit-1/actions” and communication with the polkit authority should be done at runtime. The communication could be based on D-BUS\libpolkit-gobject-1\pkcheck command³⁵⁰.

Lastly, there are also “PolKit Authentication Agents” that are provided by desktop environments - as shown in the screenshot below³⁵¹. Hence, for each user session there is authority and an authentication agent³⁵². There are several binaries as part of PolKit: “polkit” which is the authorization manager³⁵³, “polkitd” which is the system daemon³⁵⁴, “pkcheck” that verifies if a process is authorized³⁵⁵, “pkaction” which gets details about a registered action³⁵⁶, “pkexec” that executes command as another user³⁵⁷ and “pkttyagent” that is a textual authentication helper³⁵⁸. There are also the polkit rules which are stored in “/usr/share/polkit-1/rules.d”.



³⁴⁸ <https://wiki.archlinux.org/title/Polkit>

³⁴⁹ <https://github.com/polkit-org/polkit>

³⁵⁰ <https://polkit.pages.freedesktop.org/polkit/polkit-apps.html>

³⁵¹ <https://www.hanyoung.uk/blog/polkit-explained/>

³⁵² <https://documentation.suse.com/zh-cn/sles/12-SP5/html/SLES-all/cha-security-policykit.html>

³⁵³ <https://polkit.pages.freedesktop.org/polkit/polkit.8.html>

³⁵⁴ <https://polkit.pages.freedesktop.org/polkit/polkitd.8.html>

³⁵⁵ <https://polkit.pages.freedesktop.org/polkit/pkcheck.1.html>

³⁵⁶ <https://polkit.pages.freedesktop.org/polkit/pkaction.1.html>

³⁵⁷ <https://polkit.pages.freedesktop.org/polkit/pkexec.1.html>

³⁵⁸ <https://polkit.pages.freedesktop.org/polkit/pkttyagent.1.html>

KSPP (Kernel Self Protection Project)

KSPP (Kernel Self Protection Project) is a project which has a goal of adding security features into the Linux kernel³⁵⁹. This is opposed to Grsecurity which is implemented as a series of modifications to the kernel source code³⁶⁰. The goal is to protect the kernel rather than the kernel protecting user-space³⁶¹. For better understanding I suggest going over KSPP's GitHub tracker³⁶². By the way, the image below was created using "Google Gemini".

Overall, KSPP can be used to protect the Linux kernel by removing entire classes of bugs, blocking security flaw exploitation methods, actively detecting attack attempts and more. The holy grail of a self-protection mechanism is that it is effective, works by default (no need for opt-in), has no performance impact, does not affect kernel debugging and if fully tested³⁶³.

Lastly, among the features of KSPP we can find: attack surface reduction (like strict kernel memory permissions, reduced access to syscalls and restricting access to kernel modules), memory integrity (like heap memory integrity, counter integrity and size), probabilistic defenses (like canaries and KASLR) and preventing information expositors (like destination tracking and memory initialization)³⁶⁴.



³⁵⁹ <https://medium.com/@boutnaru/the-linux-concept-journey-linux-kernel-0b06457569e8>

³⁶⁰ <https://security.stackexchange.com/questions/176390/what-is-kspp-kernel-self-protection-project>

³⁶¹ <https://outflux.net/slides/2021/lss/kspp.pdf>

³⁶² <https://github.com/KSPP/linux/issues>

³⁶³ <https://www.kernel.org/doc/Documentation/security/self-protection.txt>

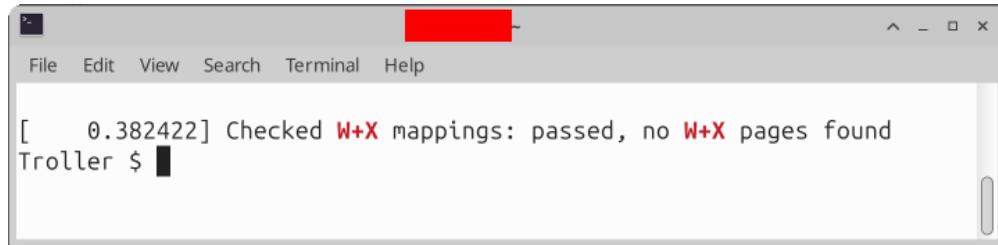
³⁶⁴ <https://docs.kernel.org/security/self-protection.html>

Debug WX

“Debug WX” is a build configuration (“CONFIG_DEBUG_WX”) of the Linux kernel³⁶⁵. It is used to generate warnings in case any memory mapping marked as W+X (write & execute) is identified during the boot process. It is important to understand that even if the check fails the kernel can still load. But it makes the exploitation of other unfixed kernel bugs easier³⁶⁶.

Overall, In case the feature is enabled based on the result of the check a message is written to the kernel ring buffer - as shown in the screenshot below. This feature is supported for different CPU architectures such as (but not limited to): x86³⁶⁷, RISC-V³⁶⁸, ARM64³⁶⁹ and PowerPC³⁷⁰.

Lastly, this feature is included as part of the kernel configuration since version 5.8 of the Linux kernel³⁷¹. Also, due to the fact it is only a one time check there is no memory\runtime effect once the kernel has booted. Thus, it is useful for discovering cases where the kernel is leaving “W+X” mappings after applying NX³⁷².



³⁶⁵ <https://medium.com/@boutharu/the-linux-concept-journey-linux-kernel-0b06457569e8>

³⁶⁶ <https://elixir.bootlin.com/linux/v6.16.3/source/mm/Kconfig.debug#L193>

³⁶⁷ https://elixir.bootlin.com/linux/v6.16.3/source/arch/x86/mm/dump_pagetables.c#L462

³⁶⁸ <https://elixir.bootlin.com/linux/v6.16.3/source/arch/riscv/mm/ptdump.c#L401>

³⁶⁹ <https://elixir.bootlin.com/linux/v6.16.3/source/arch/arm64/mm/ptdump.c#L358>

³⁷⁰ <https://elixir.bootlin.com/linux/v6.16.3/source/arch/powerpc/mm/ptdump/ptdump.c#L393>

³⁷¹ <https://elixir.bootlin.com/linux/v5.8/source/mm/Kconfig.debug>

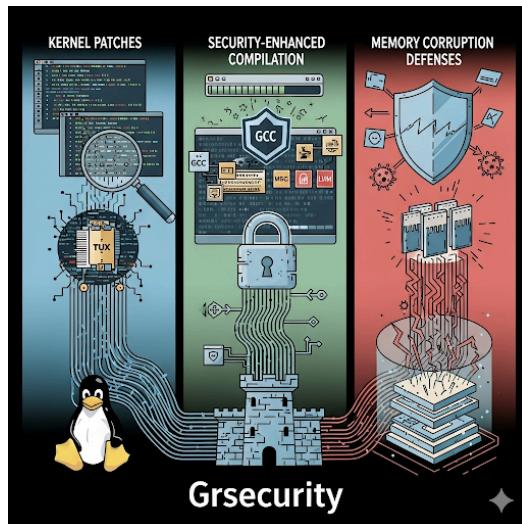
³⁷² https://cateee.net/lkddb/web-lkddb/DEBUG_WX.html

Grsecurity

Grsecurity provides a full suite of synergistic defenses. This includes mechanisms from security-enhanced compilation and to world-class memory corruption defenses to access control. The defenses can be clustered into the following categories: memory corruption defenses, filesystem hardening, RBAC (Role Based Access Control), GCC plugins and miscellaneous protections³⁷³. By the way, the image below was created using “Google Gemini”.

Overall, Grsecurity is a set of patches for the Linux kernel³⁷⁴ for increasing its security³⁷⁵. Since April 2017 (kernel version 4.9) Grsecurity’s patches are available only to paying customers³⁷⁶. Hence, Grsecurity provides out-of-tree Linux kernel patches focused on security³⁷⁷. Grsecurity is developed and maintained by “Open Source Security Inc”³⁷⁸.

Lastly, there is an open-source alternative Grsecurity named OpenPax³⁷⁹. Among the different capabilities provided by Grsecurity are (but not limited to): extending the system’s logging facilities, kernel stack isolation, industry-leading ASLR, compiler based kernel heap hardening, prevent direct user-land access by the kernel, random padding between thread stacks and bounds checks on kernel copies to/from user-mode³⁸⁰.



³⁷³ <https://grsecurity.net/features>

³⁷⁴ <https://medium.com/@boutmaru/the-linux-concept-journey-linux-kernel-0b06457569e8>

³⁷⁵ <https://wiki.debian.org/grsecurity>

³⁷⁶ <https://encyclopedia.pub/entry/37420>

³⁷⁷ <https://www.phoronix.com/news/Edera-OpenPaX-Announced>

³⁷⁸ <https://opensrcsec.com/>

³⁷⁹ <https://github.com/edera-dev/linux-openpax>

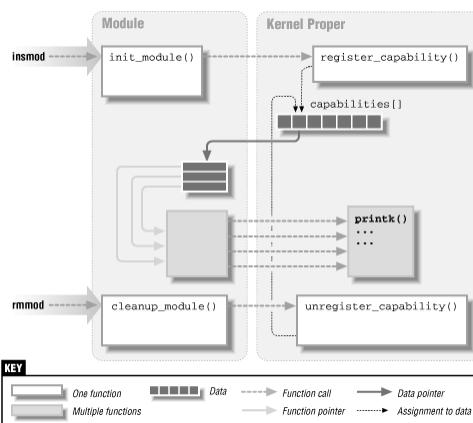
³⁸⁰ https://grsecurity.net/featureset/memory_corruption

Kernel Modules Strict RWX (Read-Write-Execute) Protection

The Linux configuration provides the “CONFIG_STRICT_MODULE_RWX” configuration item. It is used for setting the data section of a loadable kernel module³⁸¹ as NX³⁸² and the code section (text) as read-only³⁸³. This can be done due to the fact LKMs have their own memory regions holding different type of information - as shown below³⁸⁴. It is relevant in the source code of the Linux kernel since version “4.11”³⁸⁵.

Overall, it is part of the “Attack Surface Reduction” capabilities in the category of strict kernel memory permissions of the “Kernel Self Protection Project”³⁸⁶. By the way, most of the architectures have “CONFIG_STRICT_MODULE_RWX” enabled by default³⁸⁷ in combination with “CONFIG_STRICT_KERNEL_RWX” - more on the last one in future writeups.

Lastly, the memory of permissions for the kernel module pages are set by set of “module_enable_*” functions implemented as part of “kernel/module/strict_rwx.c”³⁸⁸. For example “module_enable_text_rox” for setting the code pages as read only + execute³⁸⁹, “module_enable_rodata_ro” for setting non-modifiable data pages as read only³⁹⁰. These family of functions is based on the “set_memory_*” API can be used to change various attributes of a virtual address range³⁹¹, that can have different implementations for different CPU architectures³⁹².



³⁸¹ <https://medium.com/@boutnaru/the-linux-concept-journey-loadable-kernel-module-lkm-5eaa4db346a1>

³⁸² <https://medium.com/@boutnaru/security-nx-bit-non-executable-18759fd2802e>

³⁸³ https://cateee.net/lkddb/web-lkddb/STRICT_MODULE_RWX.html

³⁸⁴ <https://www.nullsecurity.net/write-your-own-linux-kernel-module-with.html>

³⁸⁵ https://elixir.bootlin.com/linux/v4.11/A/ident/CONFIG_STRICT_MODULE_RWX

³⁸⁶ <https://medium.com/@boutnaru/the-linux-security-journey-kspp-kernel-self-protection-project-9250528f3a11>

³⁸⁷ <https://www.kernel.org/doc/html/v4.19/security/self-protection.html>

³⁸⁸ https://elixir.bootlin.com/linux/v6.16.3/source/kernel/module/strict_rwx.c

³⁸⁹ https://elixir.bootlin.com/linux/v6.16.3/source/kernel/module/strict_rwx.c#L33

³⁹⁰ https://elixir.bootlin.com/linux/v6.16.3/source/kernel/module/strict_rwx.c#L55

³⁹¹ https://elixir.bootlin.com/linux/v6.16.3/source/arch/x86/include/asm/set_memory.h#L13

³⁹² https://elixir.bootlin.com/linux/v6.16.3/C/ident/set_memory_rox

Kernel Strict RWX (Read-Write-Execute) Protection

The Linux configuration provides the “CONFIG_STRICT_KERNEL_RWX” configuration item. It is used for setting the data section of the Linux kernel³⁹³ as NX³⁹⁴ and the code (and read-only) section (text) as read-only³⁹⁵. It is relevant in the source code of the Linux kernel since version “4.11”³⁹⁶.

Overall, it is part of the “Attack Surface Reduction” capabilities in the category of strict kernel memory permissions of the “Kernel Self Protection Project”³⁹⁷. By the way, most of the architectures have “CONFIG_STRICT_KERNEL_RWX” enabled by default³⁹⁸ in combination with “CONFIG_STRICT_MODULE_RWX”³⁹⁹.

Lastly, if the feature is enabled the “mark_rod़ata_ro” function is called⁴⁰⁰. The implementation of the functions is architecture dependent⁴⁰¹ like for: ARM64⁴⁰², RISC-V⁴⁰³ and x86-64⁴⁰⁴. When the protection is applied an information message can be written to the kernel ring buffer⁴⁰⁵ - as shown below⁴⁰⁶.

```
# grep -i read-only /var/log/messages
messages.2:Jan 4 17:20:44 localhost kernel: Write protecting the kernel read-only data: 527k
messages.3:Dec 27 17:05:11 localhost kernel: Write protecting the kernel read-only data: 527k
```

³⁹³ <https://medium.com/@boutnaru/the-linux-concept-journey-linux-kernel-0b06457569e8>

³⁹⁴ <https://medium.com/@boutnaru/security-nx-bit-non-executable-18759fd2802e>

³⁹⁵ https://cateee.net/lkddb/web-lkddb/STRICT_KERNEL_RWX.html

³⁹⁶ https://elixir.bootlin.com/linux/v4.11/A/ident/CONFIG_STRICT_KERNEL_RWX

³⁹⁷ <https://medium.com/@boutnaru/the-linux-security-journey-kspp-kernel-self-protection-project-9250528f3a11>

³⁹⁸ <https://www.kernel.org/doc/html/v4.19/security/self-protection.html>

³⁹⁹ <https://medium.com/@boutnaru/the-linux-security-journey-kernel-modules-strict-rwx-read-write-execute-protection-1b01fedc97ce>

⁴⁰⁰ <https://elixir.bootlin.com/linux/v6.16.4/source/init/main.c#L1448>

⁴⁰¹ https://elixir.bootlin.com/linux/v6.16.4/C/ident/mark_rod़ata_ro

⁴⁰² <https://elixir.bootlin.com/linux/v6.16.4/source/arch/arm64/mm/mmu.c#L707>

⁴⁰³ <https://elixir.bootlin.com/linux/v6.16.4/source/arch/riscv/mm/init.c#L800>

⁴⁰⁴ https://elixir.bootlin.com/linux/v6.16.4/source/arch/x86/mm/init_64.c#L1381

⁴⁰⁵ https://elixir.bootlin.com/linux/v6.16.4/source/arch/x86/mm/init_64.c#L1390

⁴⁰⁶ <https://access.redhat.com/solutions/328213>

Hardened User Copy

“Hardened User Copy” is a security feature provided by the Linux kernel⁴⁰⁷. It is used for hardening memory copies between kernel and user space. This means that when “CONFIG_HARDENED_USERCOPY” is enabled when copying memory from/to the kernel checks are performed that will reject memory regions which have certain attributes such as (but not limited to): larger than the specified head object, span multiple separately allocated pages, not on the process stack and part of the kernel text (code) pages⁴⁰⁸.

Overall, the main goal of this feature is to kill an entire class of heap overflow exploits and/or similar memory exposures⁴⁰⁹. Those checks are performed by the “_check_object_size” function⁴¹⁰ - as shown in the screenshot below. Which is called by the “check_object_size” function in “include/linux/ucopysize.h”⁴¹¹. Those functions are defined in case “CONFIG_HARDENED_USERCOPY” is enabled⁴¹². “check_object_size” is called from “check_copy_size”⁴¹³.

Lastly, “check_copy_size” is called both from “copy_from_user”⁴¹⁴ and “copy_to_user”⁴¹⁵ which are used for copying arbitrary amounts of data to and from userspace respectively⁴¹⁶. For further information regarding the implementations of “CONFIG_HARDENED_USERCOPY*” we can check “mm/usercopy.c” as part of the Linux kernel source code⁴¹⁷. By the way, the code is based on “PAX_USERCOPY”⁴¹⁸ from Grsecurity⁴¹⁹.

```
/* Check for invalid addresses. */
check_bogus_address((const unsigned long)ptr, n, to_user);

/* Check for bad stack object. */
switch (check_stack_object(ptr, n)) {
    case NOT_STACK:
        /* Object is not touching the current process stack. */
        break;
    case GOOD_FRAME:
    case GOOD_STACK:
        /*
         * Object is either in the correct frame (when it
         * is possible to check) or just generally on the
         * process stack (when frame checking not available).
         */
        return;
    default:
        usercopy_abort("process stack", NULL, to_user,
#ifdef CONFIG_ARCH_HAS_CURRENT_STACK_POINTER
            IS_ENABLED(CONFIG_STACK_GROWSUP) ?
                ptr - (void *)current_stack_pointer :
                (void *)current_stack_pointer - ptr,
#else
                0,
#endif
        );
}
/* Check for bad heap object. */
check_heap_object(ptr, n, to_user);

/* Check for object in kernel to avoid text exposure. */
check_kernel_text_object((const unsigned long)ptr, n, to_user);
```

407 <https://medium.com/@boutnaru/the-linux-concept-journey-linux-kernel-0b06457569e8>

408 https://cateee.net/lkdb/web-lkdb/HARDENED_USERCOPY.html

409 <https://www.timesys.com/security/securing-your-linux-configuration-kernel-hardening/>

410 <https://elixir.bootlin.com/linux/v6.16.4/source/mm/usercopy.c#L215>

411 <https://elixir.bootlin.com/linux/v6.16.4/source/include/linux/ucopysize.h#L22>

412 <https://elixir.bootlin.com/linux/v6.16.4/source/include/linux/ucopysize.h#L26>

413 <https://elixir.bootlin.com/linux/v6.16.4/source/include/linux/ucopysize.h#L59>

414 <https://elixir.bootlin.com/linux/v6.16.4/source/include/linux/uaccess.h#L207>

415 <https://elixir.bootlin.com/linux/v6.16.4/source/include/linux/uaccess.h#L219>

416 <https://www.kernel.org/doc/html/v4.13/kernel-hacking/hacking.html>

417 <https://elixir.bootlin.com/linux/v6.16.4/source/mm/usercopy.c#L196>

418 <https://grsecurity.net/news>

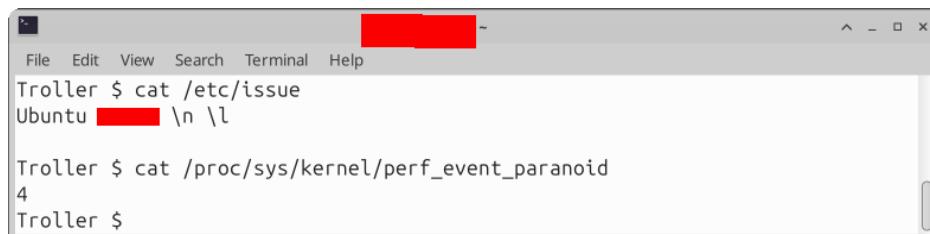
419 <https://medium.com/@boutnaru/the-linux-security-journey-grsecurity-5d2592277693>

perf_event_paranoid

“perf_event_paranoid” is a sysctl⁴²⁰ based configuration which defines how “paranoid” the kernel should be in regards to performance monitoring⁴²¹. Thus, it controls the use of system performance events (CPU\kernel profiling\tracepoints\etc) by unprivileged users which are not root\without the “CAP_SYS_ADMIN” capability⁴²².

Overall, there are four distinct values relevant for the “perf_event_paranoid” configuration: “-1”, “0”, “1” and “2” as detailed next. ”-1” allows use of almost all events by all users (not paranoid at all). “0” prevents ftrace\raw tracepoints access by unprivileged users. “1” prevents CPU event access by unprivileged users. “2” prevents kernel profiling by unprivileged users⁴²³. By default the value is “2”⁴²⁴. In case of Debian based Linux distributions⁴²⁵ there is an addition on-top of that detailed later.

Lastly, the relevant function checking for each permission level are: “perf_allow_tracepoint”⁴²⁶, “perf_allow_cpu”⁴²⁷ and “perf_allow_kernel”⁴²⁸. The specific configuration is exposed as part of “/proc/sys/kernel/perf_event_paranoid”⁴²⁹. By the way, in case of Debian based distributions there is also the value of “4” which stands for preventing all usage of perf events by unprivileged users. Also, this is the default value on those distributions⁴³⁰ - as shown in the screenshot below.



The screenshot shows a terminal window with a redacted title bar. The menu bar includes File, Edit, View, Search, Terminal, and Help. The command Troller \$ cat /etc/issue is run, followed by Troller \$ cat /proc/sys/kernel/perf_event_paranoid, which outputs the value 4. The prompt Troller \$ appears again at the bottom.

⁴²⁰ <https://elixir.bootlin.com/linux/v6.16.5/source/kernel/events/core.c#L550>

⁴²¹ https://linuxsecurity.expert/kb/sysctl/kernel_perf_event_paranoid/

⁴²² <https://askubuntu.com/questions/1400874/what-does-perf-paranoia-level-four-do>

⁴²³ <https://www.kernel.org/doc/Documentation/sysctl/kernel.txt>

⁴²⁴ <https://elixir.bootlin.com/linux/v6.16.5/source/kernel/events/core.c#L467>

⁴²⁵ <https://medium.com/@boutinaru/the-linux-concept-journey-linux-distribution-2dfc68aaa4f4>

⁴²⁶ https://elixir.bootlin.com/linux/v6.16.5/source/include/linux/perf_event.h#L1787

⁴²⁷ https://elixir.bootlin.com/linux/v6.16.5/source/include/linux/perf_event.h#L1779

⁴²⁸ https://elixir.bootlin.com/linux/v6.16.5/source/include/linux/perf_event.h#L1777

⁴²⁹ https://sysctl-explorer.net/kernel/perf_event_paranoid/

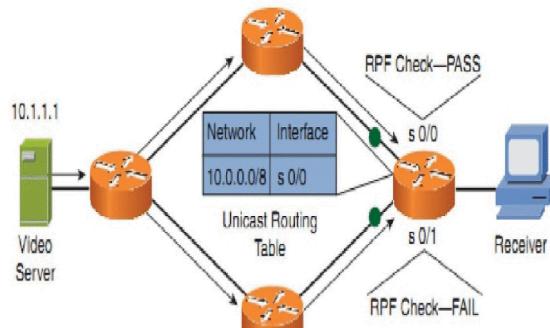
⁴³⁰ <https://askubuntu.com/questions/1400874/what-does-perf-paranoia-level-four-do>

IPv4 RPF (Reverse Path Filtering)

IPv4 RPF (Reverse Path Filtering) is a security feature of the Linux operating system. It is designed for preventing IP address spoofing. Thus, if the reply to this packet wouldn't go out the interface this packet came in, then this is a bogus packet and should be ignored⁴³¹.

Overall, we can use “/proc/sys/net/ipv4/conf/interface/rp_filter” to configure RPF. It can have one of the following values: “0” (no source validation), “1” (strict mode as defined in RFC3704, each packet is tested and if the incoming interface is not the best reverse path the packet check will fail) and “2” (loose mode as defined in RFC3704, each packet is tested and if the source address is not reachable by any interface the packet check will fail)⁴³² - as shown in the diagram below⁴³³. By the way we can set this configuration also using the “sysctl” CL utility or as part of the configuration files in “/etc/sysctl.d”⁴³⁴.

Lastly, the reverse lookup is done by the “rpfilter_lookup_reverse” function⁴³⁵. It is important to understand that RPF could be problematic in case of multicast communication. This is due to the fact the source multicast address won’t be part of the subnet like the incoming interface. This affects different protocols such as (but not limited to): IGMP and PTPv2⁴³⁶.



⁴³¹ <https://tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.kernel.rpf.html>

⁴³² https://sysctl-explorer.net/net/ipv4_rp_filter/

⁴³³ <https://www.internetworks.in/2019/11/what-is-reverse-path-forwarding-rpf.html>

⁴³⁴ https://docs.datadoghq.com/security/default_rules/def-000-zql/

⁴³⁵ https://elixir.bootlin.com/linux/v6.16.5/source/net/ipv4/netfilter/ipt_rpfilter.c#L33

⁴³⁶ <https://tinyurl.com/263wf3z>

Reset Attack Mitigation

In case a Linux based system is rebooted/shutdown the content of RAM (Random Access Memory) which is volatile is not immediately lost. By the way, without an electric charge the data will begin to decay. Hence, during this period there is a short timeframe that can allow an attacker to extract sensitive information from memory (like encryption keys). This can be done if an attacker can turn off/reboot the platform and quickly turn it back on to boot into a program that dumps the contents of memory⁴³⁷.

Overall, we can enable “CONFIG_RESET_ATTACK_MITIGATION” (“Reset Memory Attack Mitigation”) when building the Linux kernel. This basically clears the contents of RAM after a reboot using the “TCG Platform Reset Attack Mitigation” specification⁴³⁸. The mitigation is performed by the “efi_enable_reset_attack_mitigation” function. It sends a request to the firmware for clearing the RAM on next reboot before proceeding with boot⁴³⁹.

Lastly, “CONFIG_RESET_ATTACK_MITIGATION” is supported since kernel version 4.14⁴⁴⁰. Hence, also the “efi_enable_reset_attack_mitigation” function is relevant since kernel version 4.14⁴⁴¹. The security mitigation is based on the “MemoryOverwriteRequestControl” EFI variable⁴⁴² - as shown in the screenshot below.

5.1 UEFI Variable

5.1.1 The MemoryOverwriteRequestControl

Start of informative comment:

The purpose of the MemoryOverwriteRequestControl UEFI variable is to give users (e.g., OS, loader) the ability to indicate to the platform that secrets are present in memory and that the platform firmware must clear memory upon a restart.

The OS loader should not create the variable. Rather, the firmware is required to create it and must support the semantics described here.

End of informative comment.

⁴³⁷ <https://trustedcomputinggroup.org/resource/pc-client-work-group-platform-reset-attack-mitigation-specification/>

⁴³⁸ <https://elixir.bootlin.com/linux/v6.16.3/source/drivers/firmware/efi/Kconfig#L174>

⁴³⁹ <https://elixir.bootlin.com/linux/v6.16.5/source/drivers/firmware/efi/libstub/tpm.c#L29>

⁴⁴⁰ https://elixir.bootlin.com/linux/v4.14/A/ident/CONFIG_RESET_ATTACK_MITIGATION

⁴⁴¹ https://elixir.bootlin.com/linux/v4.14/A/ident/efi_enable_reset_attack_mitigation

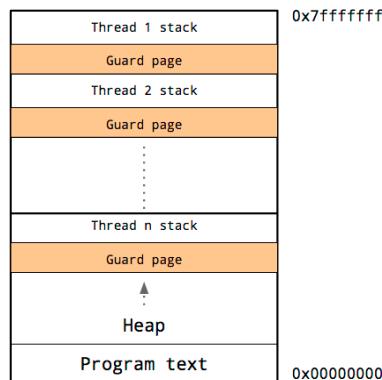
⁴⁴² <https://trustedcomputinggroup.org/wp-content/uploads/Platform-Reset-Attack-Mitigation-Specification.pdf>

Guard Pages

“Guard Pages” are non-writable memory pages placed at the end of a thread’s\task stack - as shown in the diagram below⁴⁴³. If the stack attempts to grow into this guard page a segmentation fault is triggered. This stops the process and thus prevents stack overflows⁴⁴⁴.

Overall, despite the use of guard pages a technique called “Stack Clash” was discovered by Qualys’ researchers. It can be leveraged to bypass the guard page by causing the stack to "clash" with another memory region (for example memory-mapped file or the heap) as they grow toward each other. It exploited the way stacks and other memory regions expand, allowing them to skip over the guard page and overwrite a neighboring memory region⁴⁴⁵.

Lastly, for dealing with “Stack Clash” the “CONFIG_VMAP_STACK” could be used⁴⁴⁶ - more on that in future writeups. By the way, the stack guard gap is controlled using the “stack_guard_gap” variable⁴⁴⁷.



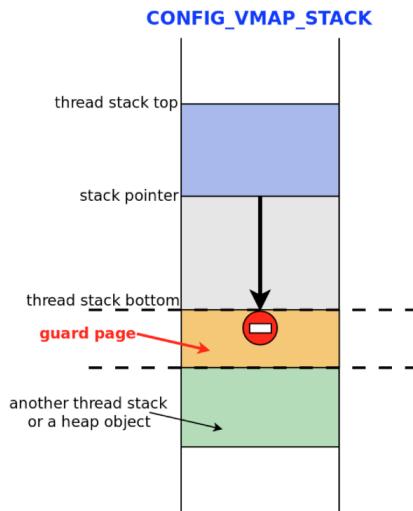
⁴⁴³ <https://dave.cheney.net/2015/08/08/performance-without-the-event-loop>
⁴⁴⁴ <https://blog.k3170makan.com/2020/11/linux-kernel-exploitation-0x1-smashing.html>
⁴⁴⁵ <https://www.qualys.com/2017/06/19/stack-clash/stack-clash.txt>
⁴⁴⁶ https://cateee.net/lkddb/web-lkddb/VMAP_STACK.html
⁴⁴⁷ <https://elixir.bootlin.com/linux/v6.16.6/source/mm/mmap.c#L945>

Use Virtually Mapped Stack with Guard Pages

“Use Virtually Mapped Stack” is a security feature of the Linux operating system. Enabling it (CONFIG_VMAP_STACK) causes the use of virtually-mapped kernel stacks with guard pages. Hence, in case of kernel stack overflows they are caught immediately rather than causing difficult-to-diagnose corruption⁴⁴⁸.

Overall, guard pages are non-writable memory pages placed at the end of a process's stack. If the stack attempts to grow into this guard page a segmentation fault is triggered. However, using guard pages naively is susceptible to “Stack Clash”⁴⁴⁹. Thus, “VMAP_STACK” allows mapping stack addresses to the range of virtual memory addresses used by “vmalloc”. Because “vmalloc” ranges are physically non-contiguous it means wrapping guard pages around stack memory is easy. Also, due to “vmalloc” guard pages became bigger so they are harder to skip over⁴⁵⁰ - as shown below⁴⁵¹.

Lastly, this feature is relevant since kernel version 4.9⁴⁵². By the way, for using this with KASAN (Kernel Address Sanitizer) modes, the architecture must support backing virtual mappings with real shadow memory and “KASAN_VMALLOC” must be enabled⁴⁵³ - more on those in future writeups. There are also specific implementation for different CPU architectures to enable the support for “CONFIG_VMAP_STACK”⁴⁵⁴.



⁴⁴⁸ https://cateee.net/lkddb/web-lkddb/VMAP_STACK.html

⁴⁴⁹ <https://medium.com/@boutnaru/the-linux-security-journey-v-guard-pages-47b04586a68e>

⁴⁵⁰ <https://blog.k3170makan.com/2020/11/linux-kernel-exploitation-0x1-smashing.html>

⁴⁵¹ <https://www.pvsm.ru/informatsionnaya-bezopasnost/294153>

⁴⁵² https://elixir.bootlin.com/linux/v4.9/A/ident/CONFIG_VMAP_STACK

⁴⁵³ <https://elixir.bootlin.com/linux/v6.16.6/source/arch/Kconfig#L1442>

⁴⁵⁴ https://elixir.bootlin.com/linux/v6.16.6/A/ident/CONFIG_VMAP_STACK

LKRG (Linux Kernel Runtime Guard)

LKRG (Linux Kernel Runtime Guard) is an LKM⁴⁵⁵ which performs kernel integrity checks and vulnerability exploit detection. It supports different CPU architectures: x86-64, 32-bit x86, AArch64 (ARM64), and 32-bit ARM CPU. Due to its nature as an LKM it can be loaded on top of a wide range of mainline and distros' kernels⁴⁵⁶.

Overall, LKRG provides system integrity by detecting unsupported kernel modifications and with the default response of panic the kernel. This is done by calculating hash of guarded regions such as (but not limited to): CPU metadata, the kernel “.text” section (syscalls table/IRQ handlers\etc), kernel “.rodata”, kernel exception table, dynamic loaded LKMs, critical global variables (selinux_enabled\Cr4.wr\SMEP\SMAP\etc) - as shown in the screenshot below⁴⁵⁷. In regards to anti-exploitation the goal is to detect specific data corruption in the kernel as an indicator of a kernel exploitation process. This is done using pCFI (Poor man's Control Flow Integrity) and monitoring attributes of critical process and system attributes⁴⁵⁸.

Lastly, among the security benefits of LKRG are: identifying sandboxes\namespaces\containers, escapes, usermodehelper blocking, identifying ROP (Return Oriented Programming) and identifying stack pivoting attacks⁴⁵⁹. For more information I suggest going over LKRG’s GitHub repository⁴⁶⁰.

```
Feb 13 21:41:21 ubuntu kernel: [ 1759 .563938] [p_lkrg] <p_get_cpus> online[1] possible[64] present[2] active[1] nr_cpu_ids[64]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .564385] [p_lkrg] <p_dump_IDT_MSR> CPU:[0] IDT => base[0xffffffffffff578000] size[0xffff] hash[0x2fa81891]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .565180] [p_lkrg] Reading IDT 1 to verify data:
Feb 13 21:41:21 ubuntu kernel: [ 1759 .565194] [p_lkrg] cr_low[0x1720]&sel[0x10]&none[0x3]&flags[0xe]&off_midi[0x0e1f]&off_high[0x1fffff]&padding[0x0]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .566456] [p_lkrg] <p_read_msr[0]> MSR arg[0x174] value[10]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .566930] [p_lkrg] <p_read_msr[0]> MSR arg[0x175] value[0]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .567409] [p_lkrg] <p_read_msr[0]> MSR arg[0x176] value[0]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .567888] [p_lkrg] <p_read_msr[0]> MSR arg[0x175] value[0]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .568337] [p_lkrg] MSR IA32_SYSENTER_ESP offset 0x175 on CPU:[0] is not set!
Feb 13 21:41:21 ubuntu kernel: [ 1759 .568338] [p_lkrg] <p_read_msr[0]> MSR arg[0x176] value[ffffffff817f1d60]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .568793] [p_lkrg] <p_read_msr[0]> MSR arg[0x277] value[0]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .569217] [p_lkrg] <p_read_msr[0]> MSR arg[0x277] value[0]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .569620] [p_lkrg] <p_read_msr[0]> MSR arg[0x277] value[0]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .569970] [p_lkrg] MSR IA32_CR_PAT offset 0x277 on CPU:[0] is not set!
Feb 13 21:41:21 ubuntu kernel: [ 1759 .569977] [p_lkrg] <p_read_msr[0]> MSR arg[0x1b] value[fee00900]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .569978] [p_lkrg] <p_read_msr[0]> MSR arg[0x00000080] value[d01]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .570365] [p_lkrg] <p_read_msr[0]> MSR arg[0x00000081] value[2300100000000]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .570737] [p_lkrg] <p_read_msr[0]> MSR arg[0x00000082] value[ffffffff817ef990]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .571074] [p_lkrg] <p_read_msr[0]> MSR arg[0x00000083] value[ffffffff817f1f60]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .571391] [p_lkrg] <p_read_msr[0]> MSR arg[0x00000084] value[47700]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .571706] [p_lkrg] <p_read_msr[0]> MSR arg[0x00000084] value[47700]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .572024] [p_lkrg] <hash from CPU data> Hash for cpu id 0 total_hash[0x36b9890b]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .572344] [p_lkrg] <hash from CPU data> Offline cpu id 1 total_hash[0x36b9890b]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .572670] [p_lkrg] Hash from CPUs metadata => [0x36b9890b]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .572675] [p_lkrg] Hash from kernel exception table => [0x31f4af10]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .580095] [p_lkrg] Hash from _stext memory block => [0x6dd1f6aa]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .581818] [p_lkrg] Hash from _rodata memory block => [0xe01d361]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .581820] [p_lkrg] Hash from IOMMU table => [0xffffffff]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .581823] [p_lkrg] Hash from 'module list' => [0x3195e007]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .581828] [p_lkrg] Hash from 'module kobj(s)' => [0xb1549775]
Feb 13 21:41:21 ubuntu kernel: [ 1759 .581828] [p_lkrg] System is clean!
```

⁴⁵⁵ <https://medium.com/@boutnaru/the-linux-concept-journey-loadable-kernel-module-lkm-5eaa4db346a1>

⁴⁵⁶ <https://lkrg.org/>

⁴⁵⁷ https://openwall.info/wiki/p_lkrg/Examples

⁴⁵⁸ <https://www.openwall.com/presentations/OSTconf2020-LKRG-In-A-Nutshell/OSTconf2020-LKRG-In-A-Nutshell.pdf>

⁴⁵⁹ <https://www.openwall.com/presentations/NullconBerlin2025-LKRG/NullconBerlin2025-LKRG.pdf>

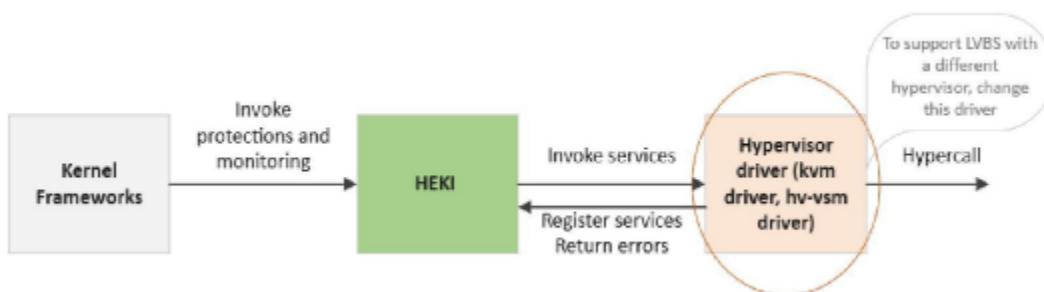
⁴⁶⁰ <https://github.com/lkre-org/lkrg>

LVBS (Linux Virtualization Based Security)

LVBS (Linux Virtualization Based Security) leverages virtualization in order to provide enhanced security on Linux based systems. This is done by using a hypervisor as a security boundary. Among the benefits of using LVBS (but not limited to): preventing bypass of guest security policies\mechanisms, supporting TEE (Trusted Execution Environment) features like executable code integrity\key management. It is an open-source architecture which is not dependent on: TEE implementation, ISA, Hypervisor and more⁴⁶¹.

Overall, LVBS is inspired by Windows' VBS (Virtualization Based Security) implementation⁴⁶². It is composed of the secure kernel (VTL1) which is initialized and verified alongside the guest kernel (VTL0). It implements core security features like (but not limited to): control register pinning, memory protections, and module authentication⁴⁶³. Hypervisor-Enforced Kernel Integrity (Heki) is a PoC (proof-of-concept) that implements new KVM features and a new API to protect guest VMs⁴⁶⁴. Thus, it implements an agnostic abstraction layer of kernel integrity - as shown in the diagram below⁴⁶⁵.

Lastly, the threat model of LVBS is to protect the kernel against userspace attacks exploiting kernel vulnerabilities. It is dependent on hardware capabilities: SLAT (Second Level Address Translation) and MBEC (Mode Based Execution Control). For more information I suggest going over LVBS's GitHub repository as part of Heki. The repository includes the Linux source tree with changes to support LVBS⁴⁶⁶. LVBS can't live alone and for protecting against persistent attacks complementary security mechanisms should be used for example: kernel module signing\IMA\IPE\etc⁴⁶⁷.



⁴⁶¹ https://lpc.events/event/17/contributions/1515/attachments/1353/2717/LPC_2023_LVBS.pdf

⁴⁶² <https://medium.com/@boutnaru/the-windows-security-journey-vbs-virtual-based-security-d4d7b1f60475>

⁴⁶³ <https://www.socallinuxexpo.org/scale/22x/presentations/linux-virtualization-based-security-lvbs/>

⁴⁶⁴ <https://github.com/heki-linux>

⁴⁶⁵ <https://kernel-recipes.org/en/2025/schedule/linux-virtualization-based-security-why-what-and-how/>

⁴⁶⁶ <https://github.com/heki-linux/lvbs-linux/>

⁴⁶⁷ <https://lore.kernel.org/all/20230505152046.6575-1-mic@digikod.net/>

Blacklisting Kernel Modules

Because Linux supports the capability of extending the kernel functionality by loading a loadable kernel module aka “LKM”⁴⁶⁸ it can be exploited for loading malicious code such as rootkits. We can disable the loading\unloading of LKMs if we want⁴⁶⁹. However, there are cases in which it is not feasible so different Linux distributions⁴⁷⁰ like Ubuntu allows us to block specific kernel modules from loading⁴⁷¹.

Overall, the module blacklisting is a feature of the “modprobe” utility⁴⁷² which is used for ensuring that the kernel cannot automatically load certain modules. For blacklisting a module we need to add a line in the following format: “blacklist <module_name>” into a file at “/etc/modprobe.d/”⁴⁷³.

Lastly, although you can modify the default “/etc/modprobe.d/blacklist.conf” file, the preferred and cleaner approach is to create a separate file like “/etc/modprobe.d/<module_name>.conf” containing only the configuration for that specific kernel module⁴⁷⁴ - as shown in the screenshot below. It is important to understand that to enhance security we should also have other security features such as kernel module signing⁴⁷⁵.



```
Troller $ ls
alsa-base.conf      blacklist-firewire.conf    blacklist-oss.conf
blacklist-ath_pci.conf  blacklist-framebuffer.conf  blacklist-rare-network.conf
blacklist.conf       blacklist-modem.conf        iwlwifi.conf
Troller $
```

⁴⁶⁸ <https://medium.com/@boutnaru/the-linux-concept-journey-loadable-kernel-module-lkm-5eaa4db346a1>

⁴⁶⁹ <https://medium.com/@boutnaru/the-linux-security-journey-disable-kernel-modules-0bd20e881675>

⁴⁷⁰ <https://medium.com/@boutnaru/the-linux-concept-journey-linux-distribution-2dfc68aaa4f4>

⁴⁷¹ <https://askubuntu.com/questions/110341/how-to-blacklist-kernel-modules>

⁴⁷² <https://man7.org/linux/man-pages/man5/modprobe.d.5.html>

⁴⁷³ https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/6/html/deployment_guide/blacklisting_a_module

⁴⁷⁴ <https://wiki.debian.org/KernelModuleBlacklisting>

⁴⁷⁵ <https://medium.com/@boutnaru/the-linux-security-journey-kernel-module-signing-fe269232248e>

SecurityFS (Security File System)

SecurityFS is a virtual filesystem which is intended to be used by security kernel modules⁴⁷⁶. It is used by the LSMs (Linux Security Modules) for storing policies and other data artifacts. By default it is mounted at the following location “/sys/kernel/security”⁴⁷⁷. The files created by the security modules should be removed when the module is unloaded⁴⁷⁸.

Overall, securityfs has a very simple API: “securityfs_create_file” (for creating a file), “securityfs_create_dir” (for creating a directory), “securityfs_create_symlink” (for creating a symbolic link) and “securityfs_remove” (for removing a file\directory). It is dependent on enabling “CONFIG_SECURITYFS” kernel configuration⁴⁷⁹.

Lastly, securityfs is used by different Linux security features like: AppArmor, IMA, SafeSetID, TOMOYO and TPM⁴⁸⁰. By default securityfs is mounted with the specific flags including (but not limited to): “rw”, “nosuid” and “noexec” - as shown in the diagram below. For more information I suggest going over the implementation of the securityfs’ functions⁴⁸¹.



```
File Edit View Search Terminal Help
Troller $ cat /proc/filesystems | grep securityfs
nodev   securityfs
Troller $ mount | grep securityfs
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
Troller $
```

⁴⁷⁶ <https://medium.com/@boutnaru/linux-security-lsm-linux-security-modules-907bbcf8c8b4>

⁴⁷⁷ <https://www.linux.org/threads/pipefs-sockfs-debugfs-and-securityfs.9638/>

⁴⁷⁸ <https://lwn.net/Articles/153366/>

⁴⁷⁹ <https://elixir.bootlin.com/linux/v6.17.1/source/include/linux/security.h#L2217>

⁴⁸⁰ <https://elixir.bootlin.com/linux/v6.17.1/source/security/Kconfig#L97>

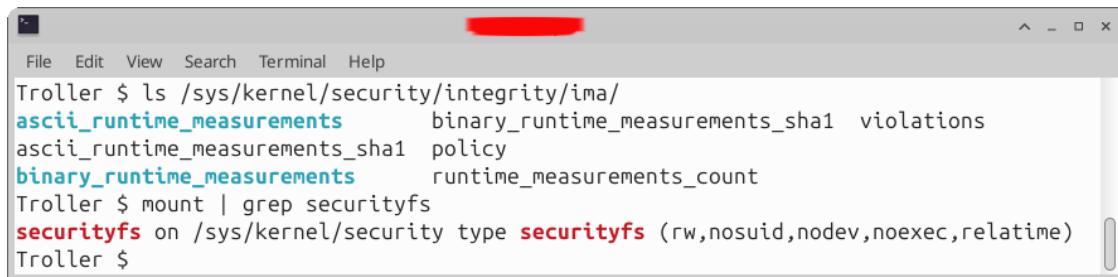
⁴⁸¹ <https://elixir.bootlin.com/linux/v6.17.1/source/security/inode.#L69>

IMA (Integrity Measurement Architecture)

The goal of IMA (Integrity Measurement Architecture) is to detect if files have been altered (accidentally/maliciously) by a user remotely/locally. It is done by appraising the file's measurement versus a “well known good” stored as an extended attribute (thus enforcing the file integrity). It provides five main integrity functions: “Collect”, “Store”, “Attest”, “Apprise”, “Protect” and “Audit”⁴⁸².

Overall, “Collect” is used for measuring a file before accessing it. “Store” is used to add the measurement computed to the kernel list, in case we have a TPM (Trusted Platform Module) the PCR is extended. “Attest” uses the TPM (if it exists) to sign the IAM PCR. “Apprise” is used to enforce a local validation of measurements using the values in the extended attributes of files. “Protect” is used for securing the file’s extended attributes against off-line attacks. “Audit” is used for auditing the file hashes⁴⁸³.

Lastly, IAM is included in the Linux kernel after kernel version 2.6.36. We can check if IMA is enabled by checking if the “/sys/kernel/security/integrity/ima” exists, it is stored on a securityfs based filesystem - as shown in the screenshot below. By the way, the kernel configurations which are relevant for IAM are: “CONFIG_INTEGRITY”, “CONFIG_IMA”, “CONFIG_AUDIT” and “CONFIG_IMA_AUDIT”⁴⁸⁴. For more information I suggest going over IAM’s source code as part of the Linux kernel⁴⁸⁵.



```
File Edit View Search Terminal Help
Troller $ ls /sys/kernel/security/integrity/ima/
ascii_runtime_measurements      binary_runtime_measurements_sha1  violations
ascii_runtime_measurements_sha1  policy
binary_runtime_measurements      runtime_measurements_count
Troller $ mount | grep securityfs
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
Troller $
```

⁴⁸² <https://sourceforge.net/p/linux-ima/wiki/Home/>

⁴⁸³ <https://www.linux-magazine.com/Issues/2021/250/Integrity-Measurement-Architecture>

⁴⁸⁴ <https://github.com/FiveDirections/IMA>

⁴⁸⁵ <https://elixir.bootlin.com/linux/v6.17.1/source/security/integrity/ima>