The Linux Security Journey

Version 5.0
August-2025

By Dr. Shlomi Boutnaru

Created using Craiyon, AI Image Generator

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.

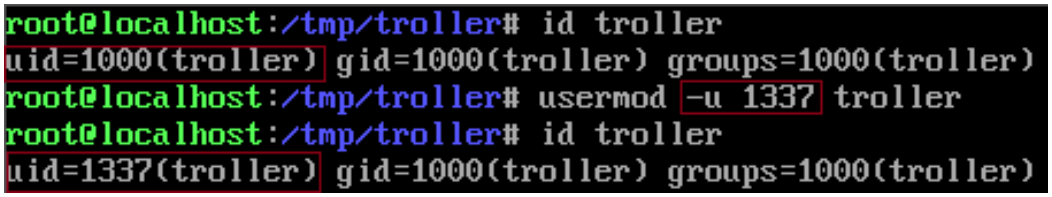
Lets GO!!!!!!

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 SID[1] 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 0[2]. 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).



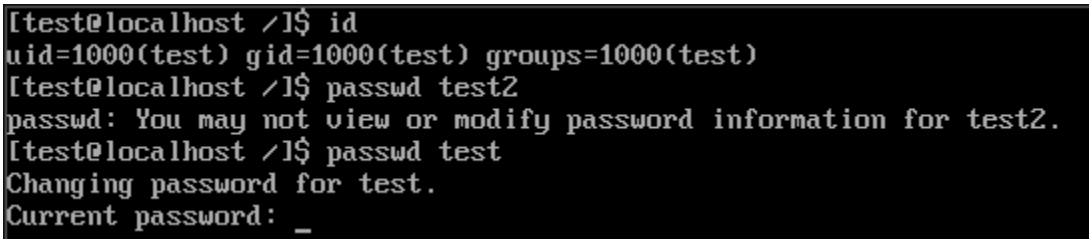
[1] https://medium.com/@boutnaru/windows-security-sid-security-identifier-d5a27567d4e5
[2] https://linuxhandbook.com/uid-linux/
[3] https://linux.die.net/man/8/usermod

RUID (Real User ID)

RUID stands for “Real UserID”, which is the user who initiated a specific operation[4]. Thus, we can say that it is basically the UID[5] of the user that started the specific task/process[6].Overall, RUID is the “uid” field of the Linux’s “struct cred” data structure[7]. This information is also included as part of the “Auxiliary Vector”[8] in the “AT_UID” entry[9].

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 access[10].

Lastly, another example of usage is by the “passwd” command line utility[11]. 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.



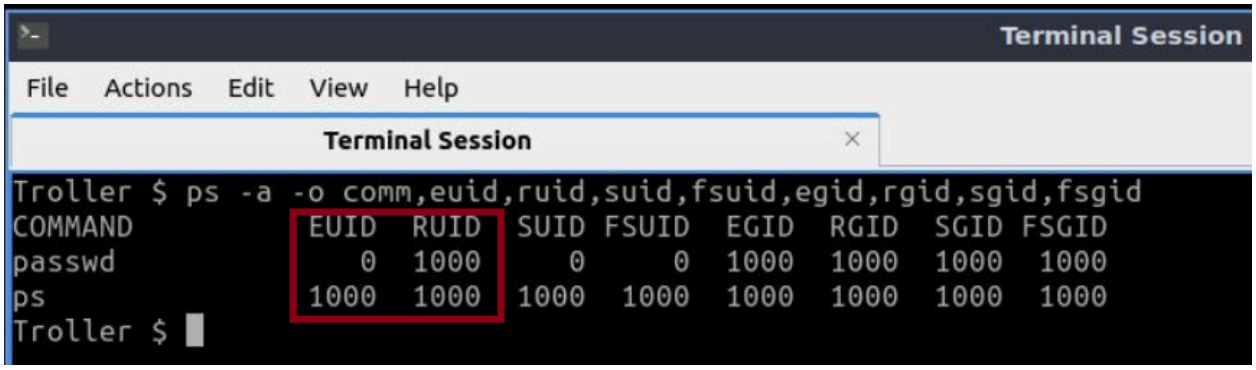
[4] https://linuxhint.com/difference-between-real-effective-user-id-in-linux-os/
[5] https://medium.com/@boutnaru/the-linux-security-journey-uid-user-identifier-2f11bcf90ee8
[6] https://www.geeksforgeeks.org/real-effective-and-saved-userid-in-linux/
[7] https://elixir.bootlin.com/linux/v6.5.8/source/include/linux/cred.h#L119
[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 system[13]. By default, the EUID is equal to the value of RUID, there are also use cases in which those two are different[14].

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 root[15]. 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].



[13] https://linuxhandbook.com/uid-linux/
[14] https://medium.com/@boutnaru/the-linux-security-journey-ruid-real-user-id-b23abcbca9c6
[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 bit[18]). It is used when we have a task (process/thread) execuring 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” principle[19], we need to use the high privileges only when it is a must.

Thus, we use the SUID in order to save the EUID[20] 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 SUID[21].

Lastly, we can use the “setresuid” syscall for setting a different value between EUID and SUID[22] - as shown in the screenshot below. We can see that we can set euid=0 if our suid=0 but we can’t do that if suid!=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
********
Troller#
```

setresuid fails

[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

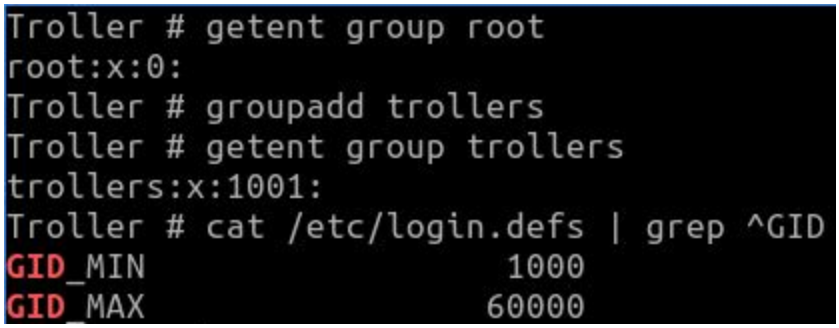
GID (Group Identifier)

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 SID[23] 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 utility[24] - 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.defs[26] - 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).



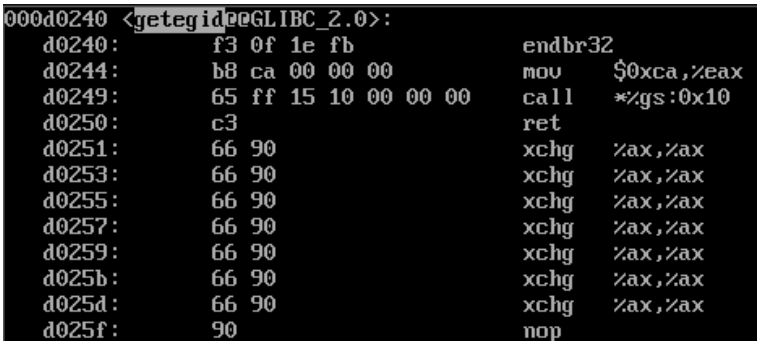
[23] https://medium.com/@boutnaru/windows-security-sid-security-identifier-d5a27567d4e5
[24] https://man7.org/linux/man-pages/man1/getent.1.html
[25] https://linux.die.net/man/8/groupadd
[26] https://en.wikipedia.org/wiki/Group_identifier

EGID (Effective Group ID)

As with EUID[27] 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” macro[28].

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/task[30]. There is also a library call with the same name[31] - as shown in the screenshot below taken using copy.sh[32].



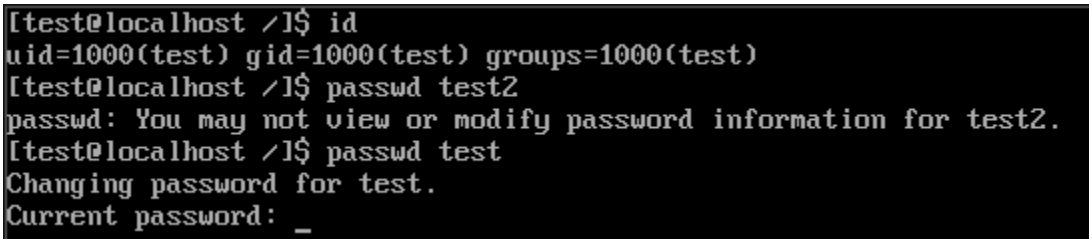
[27] https://medium.com/@boutnaru/the-linux-security-journey-euid-effective-user-id-65f351532b79
[28] https://elixir.bootlin.com/linux/v6.9.5/source/include/linux/cred.h#L375
[29] https://linux.die.net/man/1/id
[30] https://linux.die.net/man/2/getegid
[31] https://man7.org/linux/man-pages/man3/getegid.3p.html
[32] https://copy.sh/v86/?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 GID[33] of the user that started the specific task/process[34]. Overall, RUID is the “uid” field of the Linux’s “struct cred” data structure[35]. This information is also included as part of the “Auxiliary Vector”[36] in the “AT_UID” entry[37].

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 access[38].

Lastly, another example of usage is by the “passwd” command line utility[39]. 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.



[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” principle[41], we need to use the high privileges only when it is a must.

Thus, we use the SGID in order to save the EGID[42] 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 SGID[43]. We can egid=0 if our sgid=0 but we can’t do that if sgid!=0. Thus, SGID uses the same concepts as SGID[44]. 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 */
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

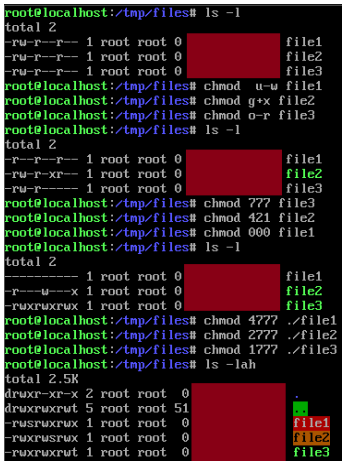
File Permissions

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 execute[46].

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 group[47] of the user when the file/directory was created. Other, which applies to all other users in the system[48].

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 SUID[50], SGID[51] and sticky bit[52] - 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.



[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-ccb0aaf3c019
[53] https://linux.die.net/man/1/chmod

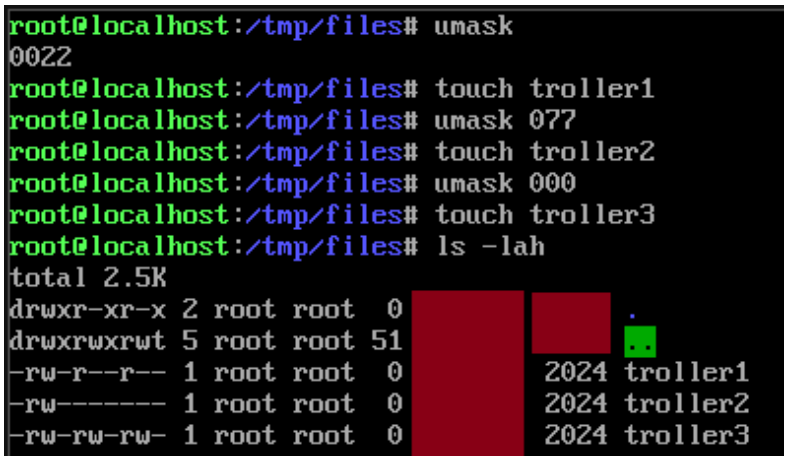
umask (Set File Mode Creation Mask)

When creating a new file/directory the default file mode permissions[54] 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, "umaks" 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 case 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 need[59].



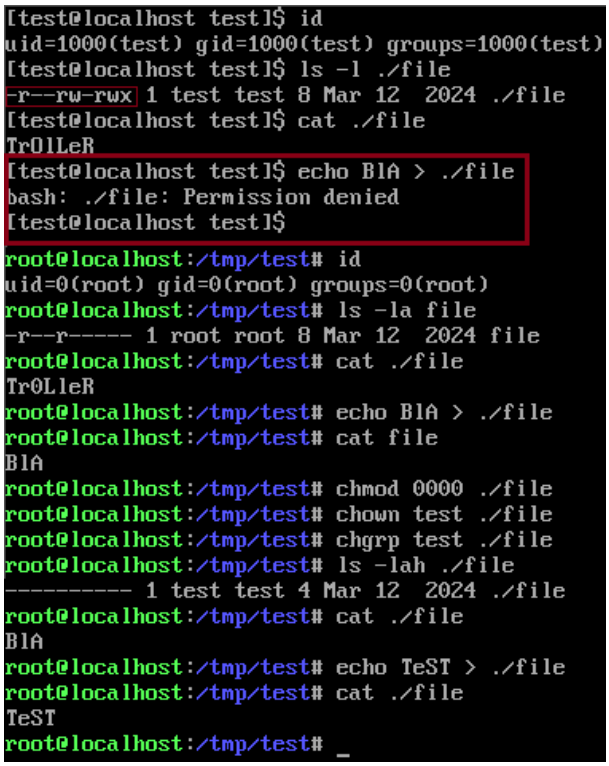
[54] https://medium.com/@boutnaru/the-linux-security-journey-file-permissions-033cb3ce8547
[55] https://man7.org/linux/man-pages/man2/open.2.html
[56] https://man7.org/linux/man-pages/man2/mkdir.2.html
[57] https://man7.org/linux/man-pages/man2/chmod.2.html
[58] https://www.liquidweb.com/kb/what-is-umask-and-how-to-use-it-effectively/
[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 permissions[60] under Linux are not cumulative. For example if we have a file that our user only has "read" permission and our primary group[61] 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.



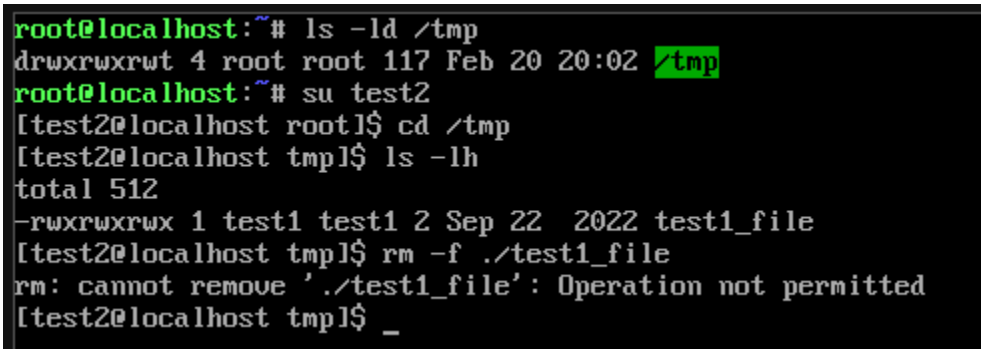
[60] https://medium.com/@boutnaru/the-linux-security-journey-file-permissions-033cb3ce8547
[61] https://medium.com/@boutnaru/the-linux-security-journey-primary-groups-de2b4d6bd27b
[62] https://man7.org/linux/man-pages/man7/capabilities.7.html

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).



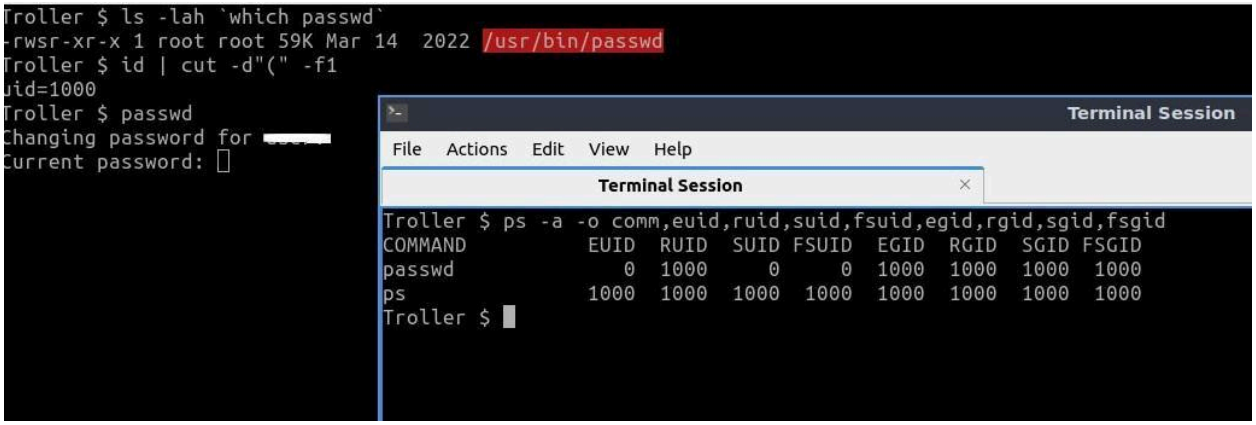
SUID Bit (Set User ID)

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).



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

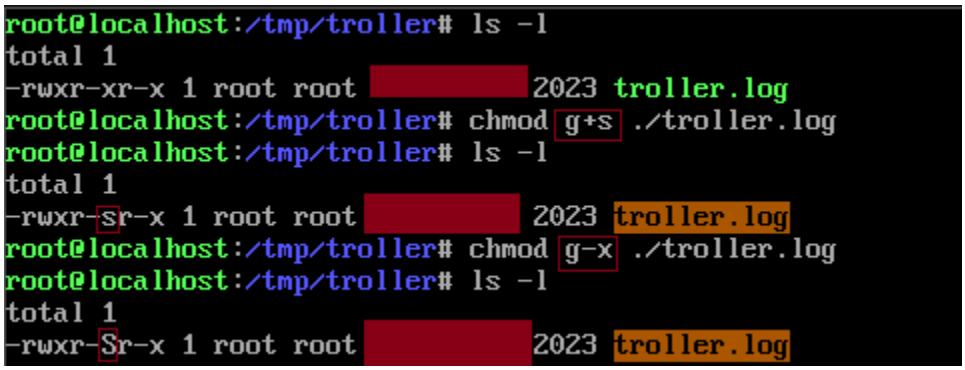
SGID Bit (Set Group ID)

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 tile file - it is similar to SUID which does the same with the user that owns the file[64]. 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 owner[65].

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 permissions[68]. For setting the SGID bit we can you a number in the pattern of 2xyz, where x/y/z are 1 or 2 or 4[69].



[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)

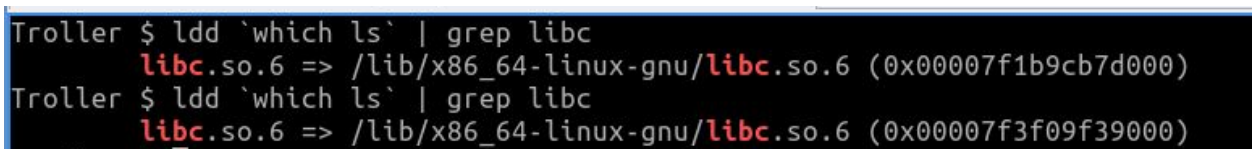
ASLR is a mitigation techniques used to increase the difficulty of running arbitrary code in case of an exploitation of a memory corruption vulnerability such as a buffer overflow (We will go over it in a different writeup but you can read the first article about it from phrack[70] magazine.

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).



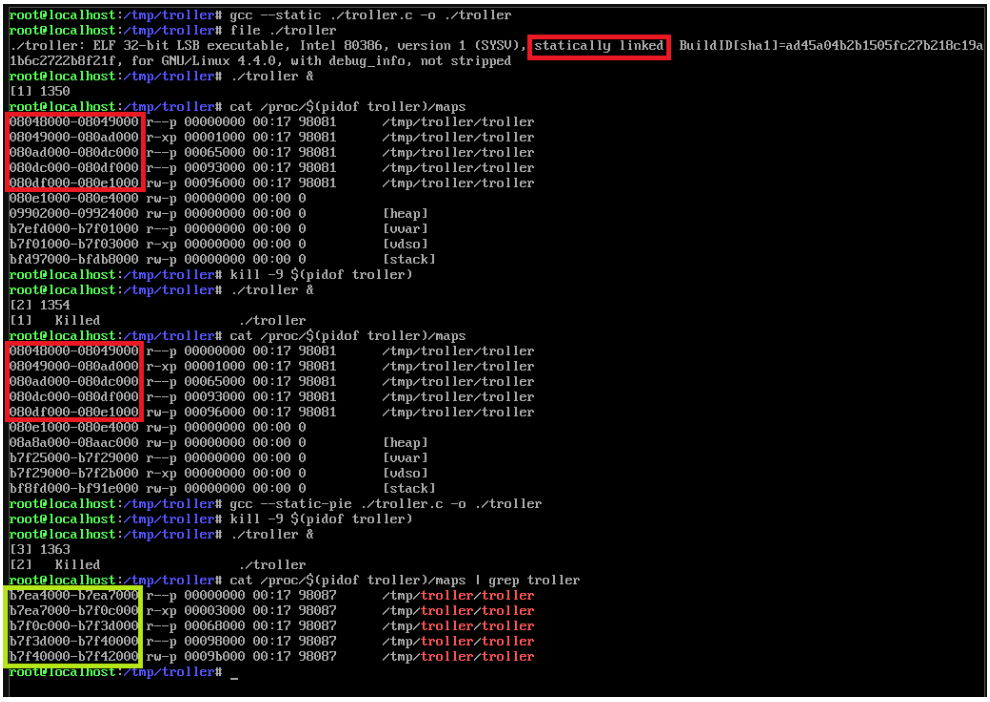
[70] http://phrack.org/issues/49/14.html

ASLR in Statically Linked ELFs

When compiling code to a statically linked ELF we bake all the code our binary needs from shared libraries inside our own executable[71]. The question which arises is how and if it effects the ASLR[72] 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 linker[73]. We can also see that in the screenshot below.



[71] https://www.ibm.com/docs/en/openxl-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 kernel[74]. 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 addresses[75]. 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 bootloader[78]. 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): x86[80], ARM 64[81] and PowerPC[82].

[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 CPU[83]. 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 "PXN" (Privileged Never Execute) for distinguishing between user-mode (EL0) and kernel mode (EL1\EL2) in this architecture[84].

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 feature[85]. A specific message is emitted based on the x86 CPU support[86]. It can be viewed by: using "journalctl", using "dmesg" or reading "/var/log/messages" - as shown in the screenshot below[87]. The "set_memory_nx" function is leveraged for setting an already allocated\mapped memory region to non-executable[88]. 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): ARM[90], RISC-V[91], PowerPC[92] and x86[93]. 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 menu[94].

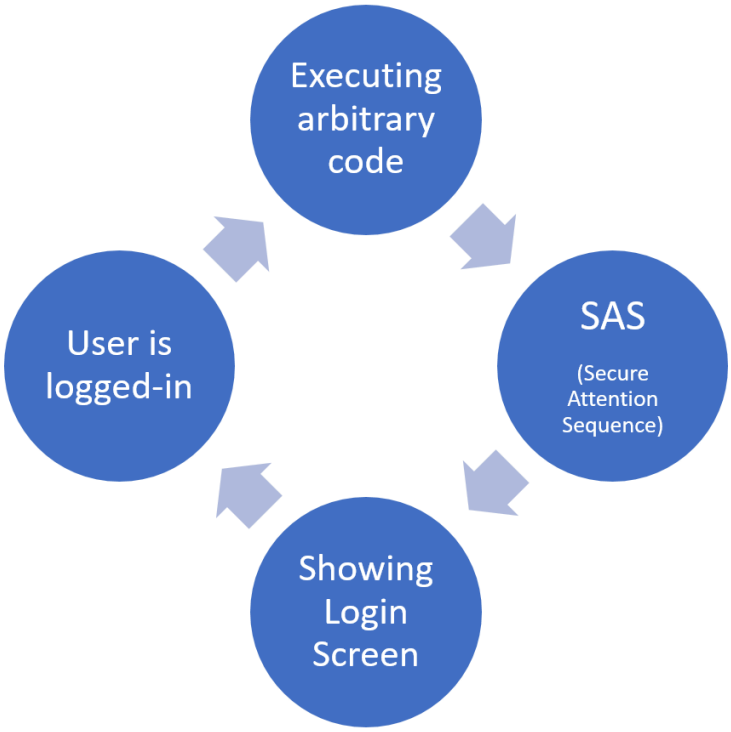
```
# cat /var/log/messages | grep "Execute Disable"
Feb 20 23:20:41 localhost kernel: NX (Execute Disable) protection: active
```

[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 screen[95] - As shown in the diagram below.

If you want to read about the support of Linux (kernel 2.4.2) for SAS[96] (SAK in the documentation). This concept is also relevant for other operating systems like Windows[97].



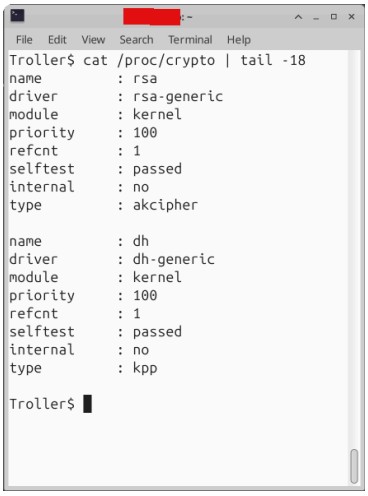
[95] https://en.wikipedia.org/wiki/Secure_attention_key
[96] https://www.kernel.org/doc/Documentation/SAK.txt
[97] https://security.stackexchange.com/questions/152556/why-does-windows-10-not-have-the-secure-attention-key-as-default

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 cryptoloop[98]. 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 ciphers[102]. 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 more[103].

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].



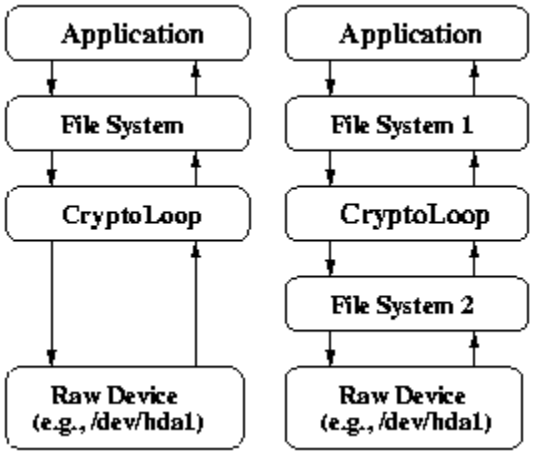
[98] https://en.wikipedia.org/wiki/Crypto_API_(Linux)
[99] https://elixir.bootlin.com/linux/v2.5.45/source/crypto
[100] https://man7.org/linux/man-pages/man5/proc_crypto.5.html
[101] https://elixir.bootlin.com/linux/v6.15.4/A/ident/tfm
[102] https://docs.kernel.org/crypto/intro.html
[103] https://www.kernel.org/doc/html/v4.14/crypto/index.html
[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 mainline[109]. 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 mapper[111].

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 below[112]. This can be done using a loop device[113]. 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 LUKS[114].

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].



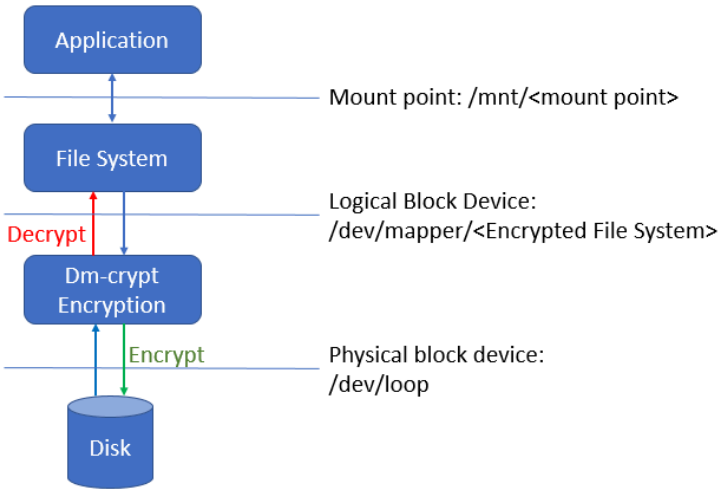
[109] https://medium.com/@boutnaru/the-linux-security-journey-cryptoapi-cryptography-application-programming-interface-0d026bf4589e
[110] https://en.wikipedia.org/wiki/Cryptoloop
[111] https://medium.com/@boutnaru/the-linux-concept-journey-dm-device-mapper-e6bc42981893
[112] https://www.linuxtechtips.com/2013/11/how-to-encrypt-partition-with-cryptoloop.html
[113] https://medium.com/@boutnaru/the-linux-concept-journey-loop-device-17caead3b15c
[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 mapper[118] 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 transformations[119].

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 stored[120] - as demonstrated in the diagram below. By the way, LUKS (Linux Unified Key Setup) is based on dm-crypt[121] - more on that in future writeups. Also, dm-crypt is the successor of cryptoloop[122].

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) mode[125].



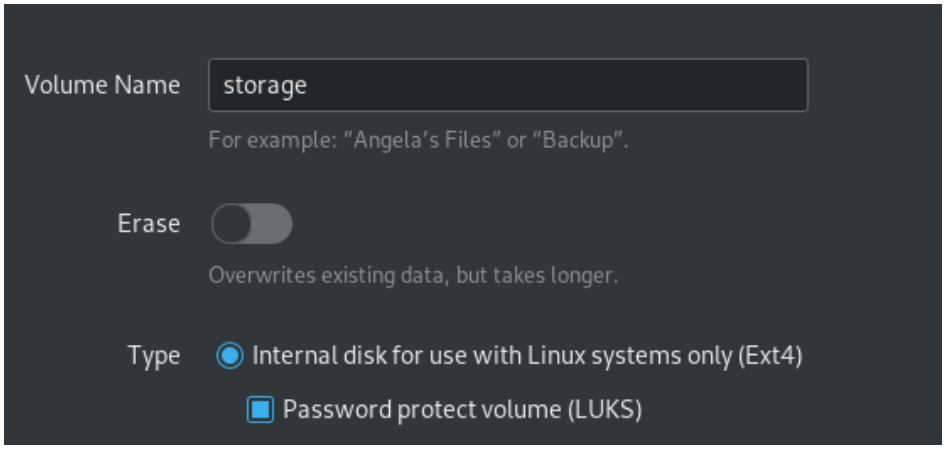
[118] https://medium.com/@boutnaru/the-linux-concept-journey-dm-device-mapper-e6bc42981893
[119] https://wiki.archlinux.org/title/Dm-crypt
[120] https://docs.aws.amazon.com/whitepapers/latest/navigating-gdpr-compliance/linux-dm-crypt-infrastructure.html
[121] https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/5/html/installation_guide/ch29s02
[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://gitlab.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 device[127]. 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 passphrases[128].

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 header[129].

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 media[130] - as shown in the screenshot below[131]. LUKS supports different formats LUKS1 and LUKS2 each with their own defaults and capabilities[132].



[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-your-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 libseccomp[133], the prctl[134] system call and/or the seccomp[135] syscall and/or other CLI tools[136]. 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 system[137] .

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 seccomp[139].

```
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*wz0ilbMy8kr6t30v

Linux Security Modules (LSM)

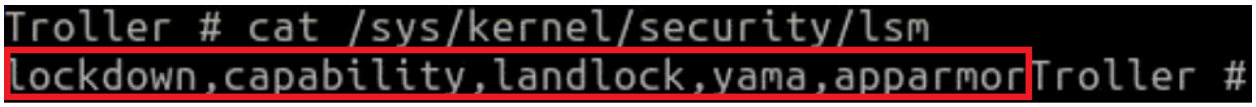
“Linux Security Modules” (LSM) is a framework which allows the kernel to support various security modules. It was mainly designed to allow implementation of MAC (Mandatory Access Control) with minimal changes to the Linux kernel. For now, you should know that MAC is an organizational-wide security policy 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 kernel[141].

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.



[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 LSM[142] (“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 LoadPin[143].

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 below[144].

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 capabilities[145].

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 form 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 kernel[146]. Moreover, you can see on the image below the source code for creating of the configuration files entries in securityfs[147].

```
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 documentation[148] “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 code[149].

```
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_TRACEME: Operation not permitted
strace: ptrace(PTRACE_TRACEME, ...): 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 facility[150]. Overall, “keyrings” are used by different types of applications such as authentication servers, web servers and database servers. Examples for those types are: MySQL[151].

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 system[156]. Lastly, we can also go over the kernel code that handles keyring[157]. Also there is “keyutils” which is a library and a set of utilities that allows access to the in-kernel keyrings facility[158].

```
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 LSM[159] 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 object[160].

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 Ubuntu[161]. 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 status[163].

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 below[164].

Also, "AppArmor" core functionality is part of the Linux mainline since kernel 2.6.3.6[165]. 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 tools[167]. 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,
}
```

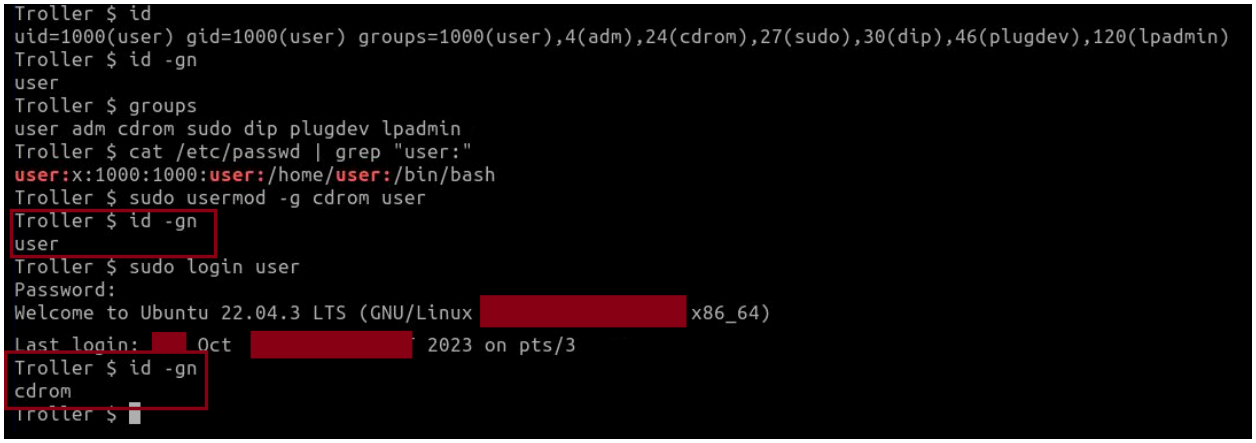
[159] https://medium.com/@boutnaru/linux-security-lsm-linux-security-modules-907bbcf8c8b4
[160] https://en.wikipedia.org/wiki/AppArmor
[161] https://wiki.ubuntu.com/AppArmor
[162] https://linuxhint.com/apparmor-profiles-ubuntu/
[163] https://manpages.debian.org/unstable/apparmor/apparmor_status.8.en.html
[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 creating[169].

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 below[171].

Moreover, we can change it using the "usermod" tool[172], 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".



.

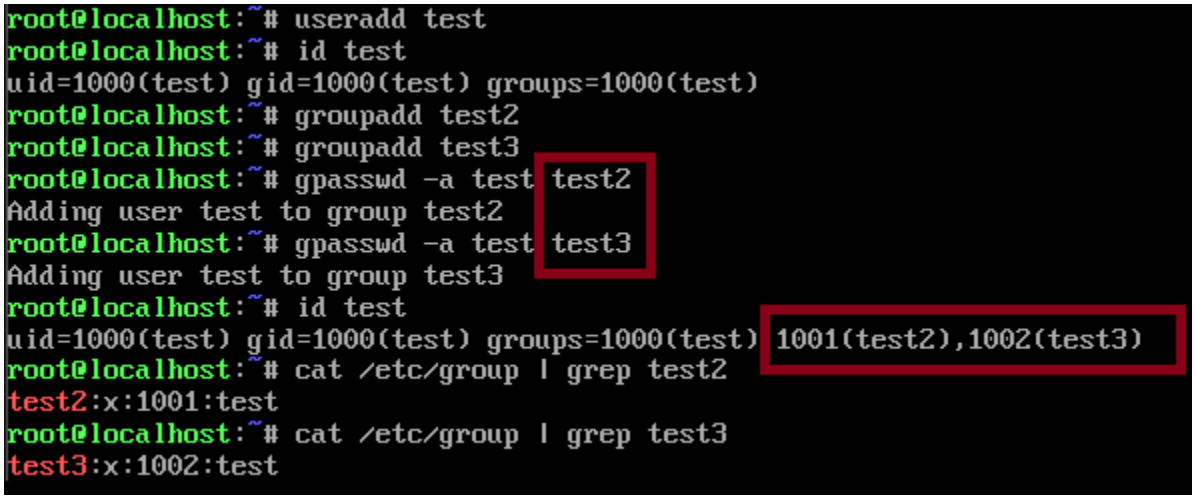
[169] https://www.baeldung.com/linux/primary-vs-secondary-groups
[170] https://man7.org/linux/man-pages/man1/id.1.html
[171] https://unix.stackexchange.com/questions/410367/how-to-get-the-primary-group-of-a-user
[172] https://linux.die.net/man/8/usermod
[173] https://man7.org/linux/man-pages/man1/groups.1.html
[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: primary[175] and secondary. A secondary group is one/more groups which a user is also part of in parallel to the primary group[176].

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 added[181].



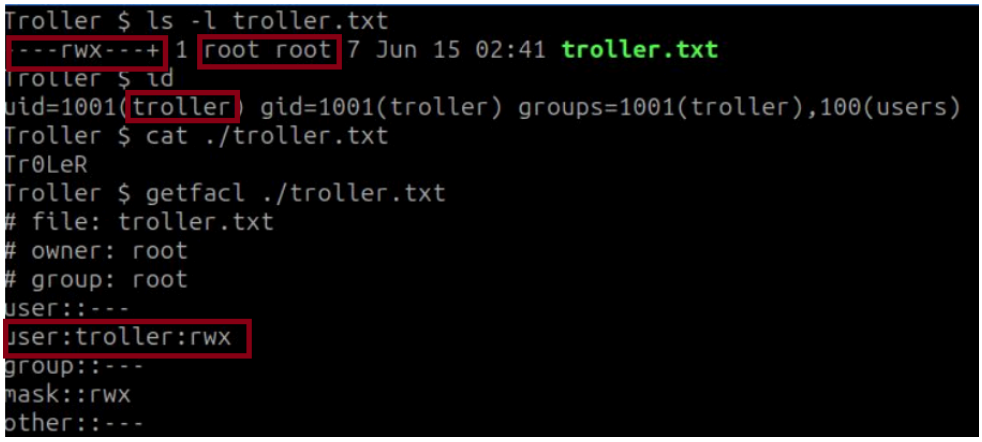
[175] https://medium.com/@boutnaru/the-linux-security-journey-primary-groups-de2b4d6bd27b
[176] https://unix.stackexchange.com/questions/605531/primary-vs-secondary-groups-in-linux
[177] https://linux.die.net/man/8/useradd
[178] https://linux.die.net/man/8/groupadd
[179] https://linux.die.net/man/1/gpasswd
[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 then the normal Linux file permissions[182]. This can be done without changing the ownership or the granular Linux permissions[183].

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 enabled[187]. In case acl is configured of a file a "+" character appears in the output of "ls -l"[188] - as shown in the screenshot below.



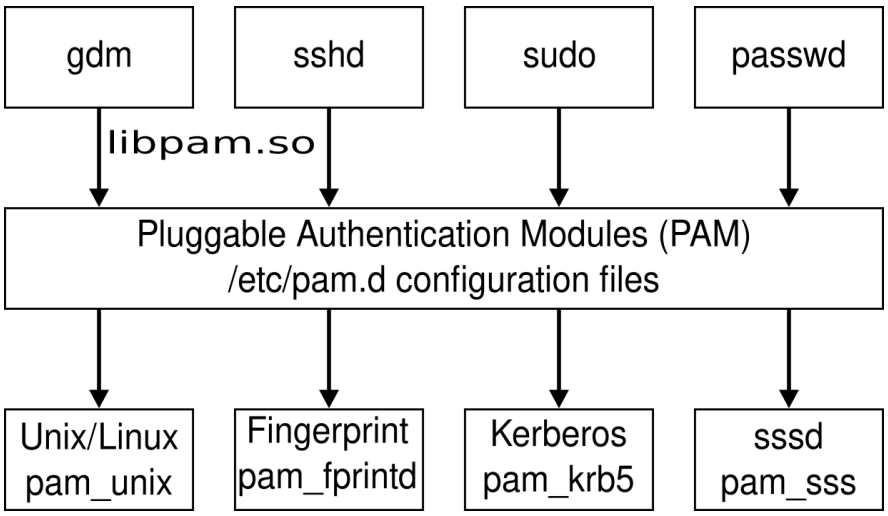
[182] https://medium.com/@boutnaru/the-linux-security-journey-file-permissions-033cb3ce8547
[183] https://www.redhat.com/sysadmin/linux-access-control-lists
[184] https://linux.die.net/man/1/getfacl
[185] https://linux.die.net/man/1/setfacl
[186] https://wiki.archlinux.org/title/Access_Control_Lists
[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 method[189].

Overall, the different applications are linked with the "libpam.so" library - as shown in the screenshot below. The functions in this library provide for PAM[190]. An example function is the "pam_authenticate" which is used for authenticating users. In case of successful completion, the function returns PAM_SUCCESS[191].

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 information[193].



[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

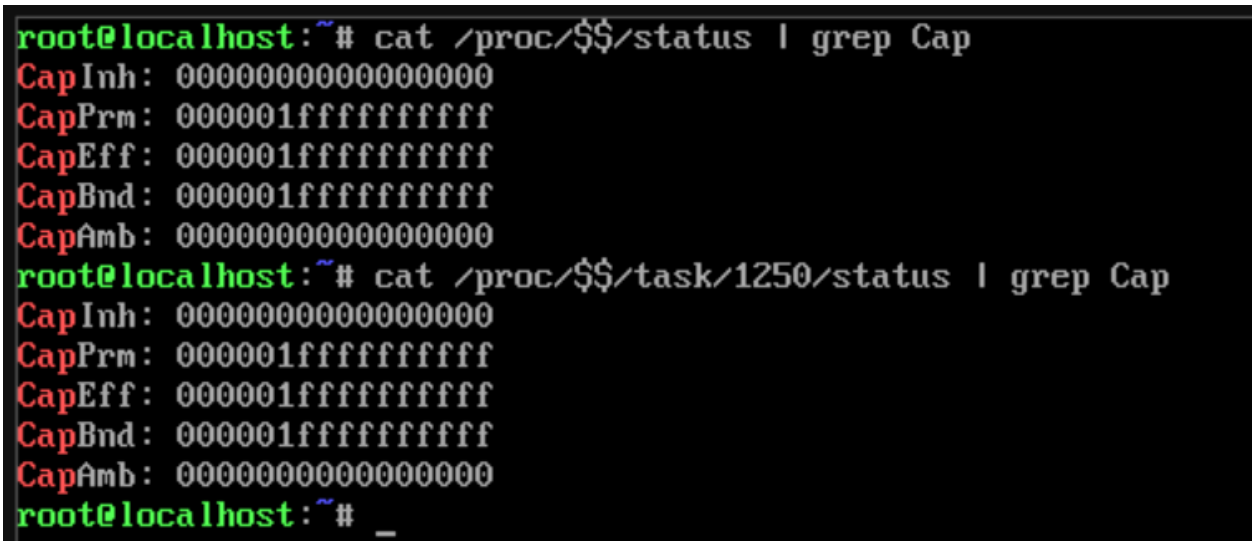
Historically, Linux has had two levels of permissions relevant from process: low privilege (effective uid is not 0) and high privilege (effective uid is 0, aka root). Since kernel 2.2 Linux has broken up the high privileges of the root user into smaller distinct units called capabilities.We can assign 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".

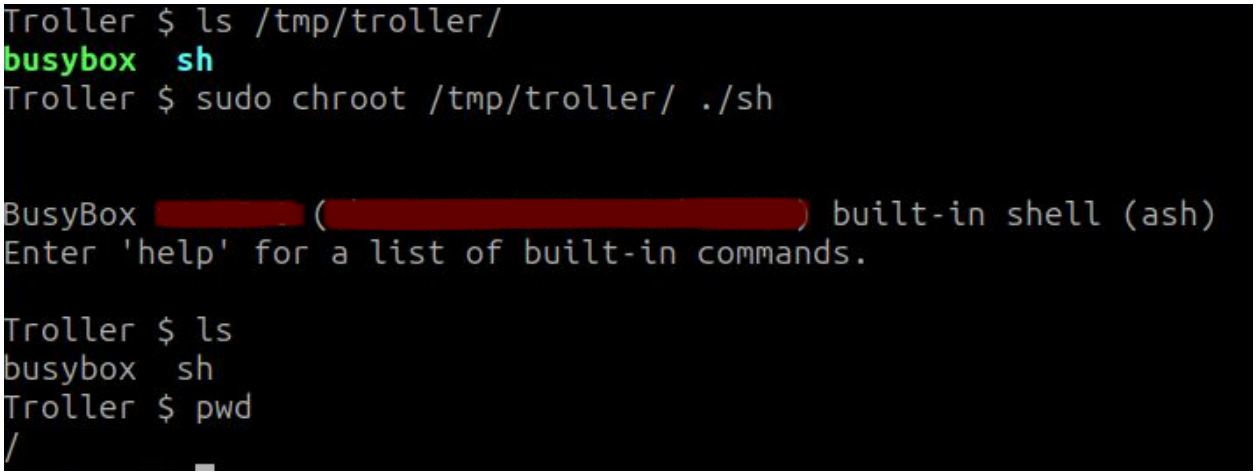

root@localhost:~# cat /proc/$$/status | grep Cap
CapInh: 0000000000000000
CapPrm: 000001ffffffffff
CapEff: 000001ffffffffff
CapBnd: 000001ffffffffff
CapAmb: 0000000000000000
root@localhost:~# cat /proc/$$/task/1250/status | grep Cap
CapInh: 0000000000000000
CapPrm: 000001ffffffffff
CapEff: 000001ffffffffff
CapBnd: 000001ffffffffff
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 namespace[194].

Moreover, there are different use case (which are not joust security related) for using chroot like: rebuilding initramfs image, reinstalling a bootloader, upgrading/downgrading a package and more[195]. By the way, we can use the "chroot" CLI tool (and not the system call) for preventing access outside the new root directory[196] - as shown in the screenshot below. It is recommended to go over the implementation of the "chroot" syscall[197].

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 it[198]. We can find it in use when creating sandboxed environments[199] - 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 users[200].



[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/

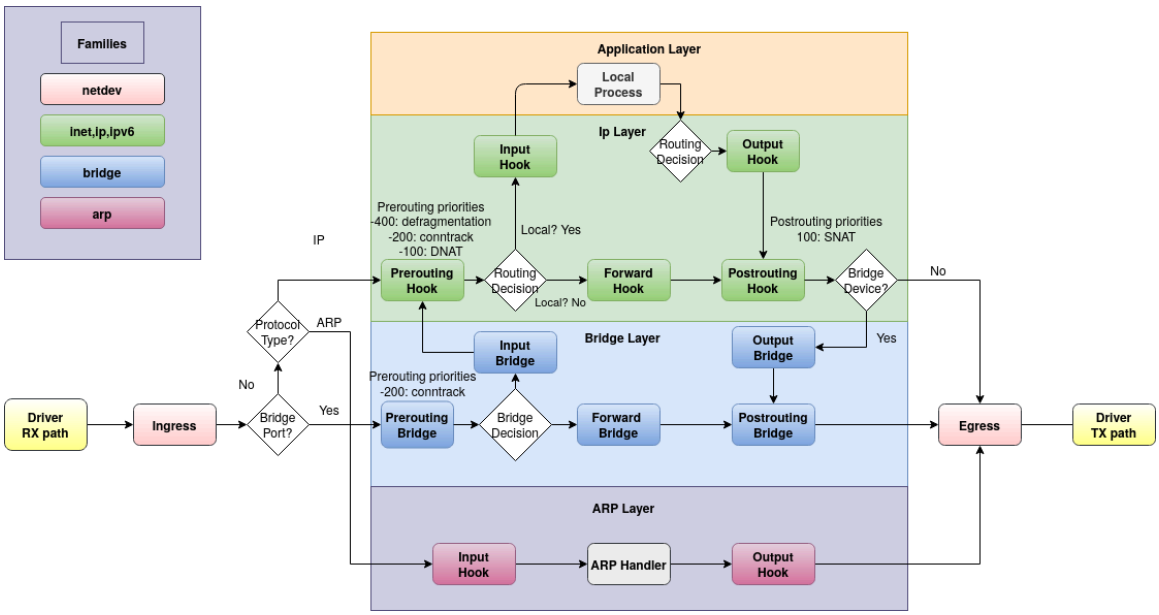
NetFilter

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 mangaling[201]. 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 connection[202]. We can interface with the connection tracking feature using the “conntrack” CLI tool[203].

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 below[204].

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 kernel[205].

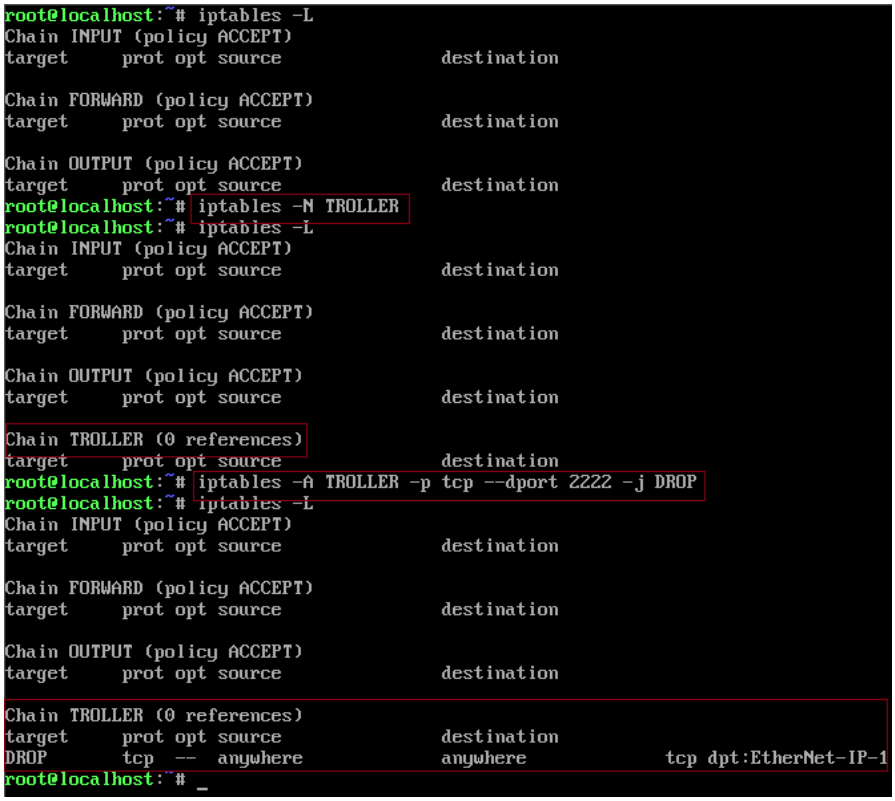


[201] https://www.netfilter.org/
[202] https://en.wikipedia.org/wiki/Netfilter
[203] https://manpages.ubuntu.com/manpages/trusty/man8/conntrack.8.html
[204] https://wiki.nftables.org/wiki-nftables/index.php/Netfilter_hooks
[205] https://elixir.bootlin.com/linux/v6.5.5/source/net/netfilter

Chains (iptables)

In general “iptables” is an administration tool used IPv4/6 packet filtering and NAT[206]. “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 callbacks[207]. We can also see that in the source code both for IPv4[208] and IPv6[209].

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 processing[210]. More on the different targets which are available in a future writeup. Lastly, we also have chains in other networking tools like “ebtables”



[206] https://linux.die.net/man/8/iptables
[207] https://medium.com/@boutnaru/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 NAT[211]. “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 packets[212].

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 processing[214].

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 below[215].

Chains ➡ / Tables ⬇	PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING
filter		☑	☑	☑	
nat	☑	☑		☑	☑
mangle	☑	☑	☑	☑	☑
raw	☑			☑	
security		☑	☑	☑	

[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

nftabes is the replacement of the legacy "*tables" tools (iptables/ip6tables/arptabels/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 2014[216].

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 below[217]. 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 bytecode[218].

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 updates[219].

Lastly, new Linux distributions use nftables as the recommended/default firewalling framework[220]. The user-mode command line tool for managing nftables is "nft". We can summarize that ntfables is a project of netfilter providing firewalling/NAT/packet mangling capabilities for Linux[221].

```
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 ]
```

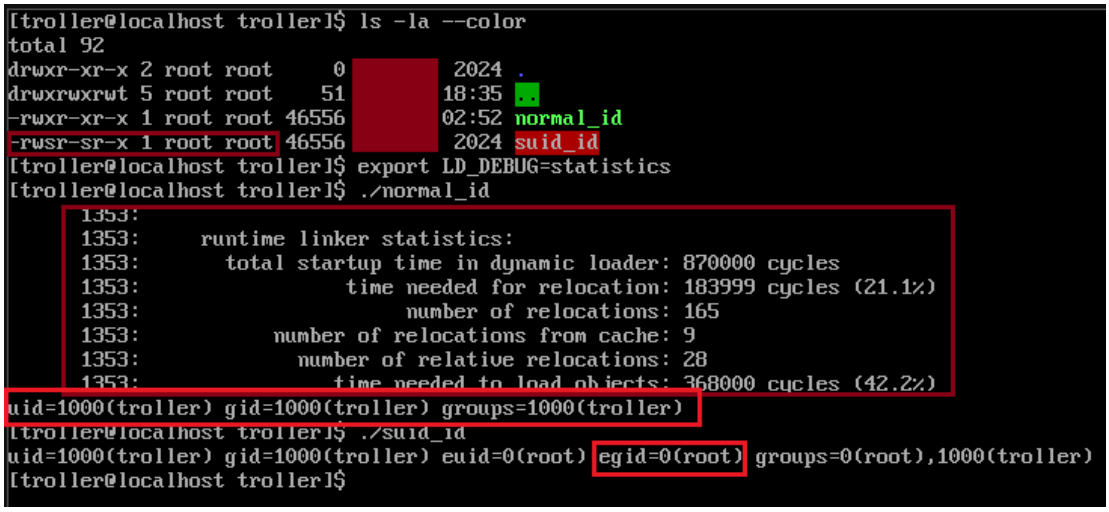
[216] https://elixir.bootlin.com/linux/v3.13-rc1/source/net/netfilter/nf_tables_core.c
[217] https://wiki.nftables.org/wiki-nftables/index.php/Ruleset_debug/VM_code_analysis
[218] https://kernelnewbies.org/Linux_3.13#head-f628a9c41d7ec091f7a62db6a49b8da50659ec88
[219] https://wiki.nftables.org/wiki-nftables/index.php/What_is_nftables%3F
[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 vector[222] contains a non-zero value. They are different cases that causes this value to be non zero such as: a LSM[223] 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 process[226].

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).



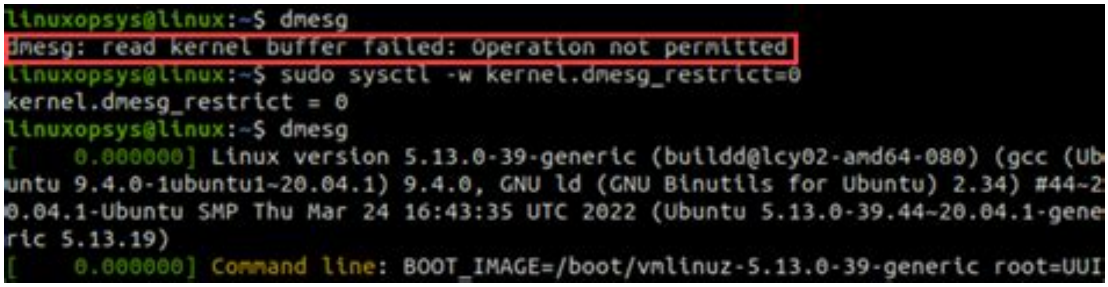
[222] https://medium.com/@boutnaru/linux-the-auxiliary-vector-auxv-cba527871b50
[223] https://medium.com/@boutnaru/linux-security-lsm-linux-security-modules-907bbcf8c8b4
[224] https://medium.com/@boutnaru/the-linux-security-journey-ruid-real-user-id-b23abcbca9c6
[225] https://medium.com/@boutnaru/the-linux-security-journey-euid-effective-user-id-65f351532b79
[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 kernel[228]. 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 bypass[230] security protections like KASLR[231] (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 access[232] only to users which have the "CAP_SYSLOG" capability[233] - as shown in the screenshot below[234].

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 sysctl[238] through procfs[239].



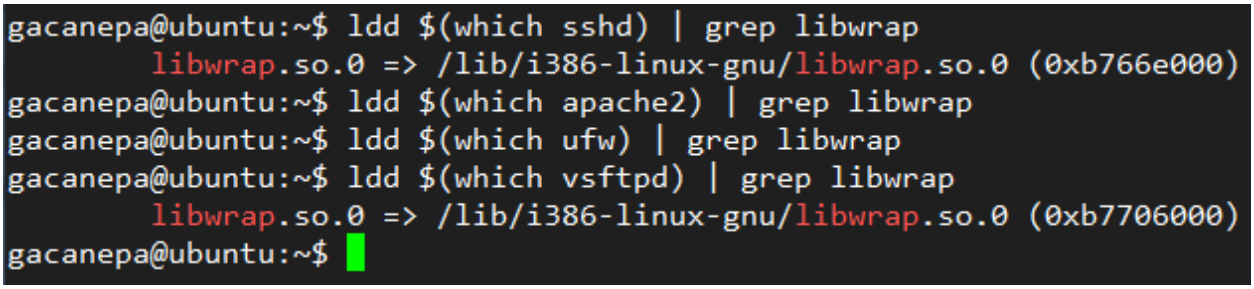
[228] https://linux.die.net/man/8/dmesg
[229] https://sysctl-explorer.net/kernel/dmesg_restrict/
[230] https://nvd.nist.gov/vuln/detail/CVE-2018-7273
[231] https://medium.com/@boutnaru/the-linux-security-journey-kaslr-kernel-address-space-layout-randomization-6d5554766fe1
[232] https://linuxsecurity.expert/kb/sysctl/kernel_dmesg_restrict/
[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.c#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-procfs-proc-filesystem-f6b2e3aa5550

TCP Wrappers

TCP wrappers can be used (in services which support it) for restricting access based on IP\hostname to specific services[240]. 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. Ecah 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 libwrap[244]. For example sshd, apache2, ufw and vsftpd as compiled with libwrap[245] - as shown in the output of "ldd"[246] in the screenshot below.



[240] https://sternumiot.com/iot-blog/linux-security-hardrining-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.ilinuxgeek.com/article/how-to-secure-network-services-using-tcp-wrappers-in-linux
[246] https://medium.com/@boutnaru/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 fooding 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) addresses[247]

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 below[248]. 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 allocated[249]. Because the cookie calculation is based on fields from the IP header there is an implementation both for IPv4[250] and IPv6[251]. The support of TCP SYN cookie is controlled using the "SYN_COOKIES" configuration parameter[252]. 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 sysctl[255]. 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.
 */
```

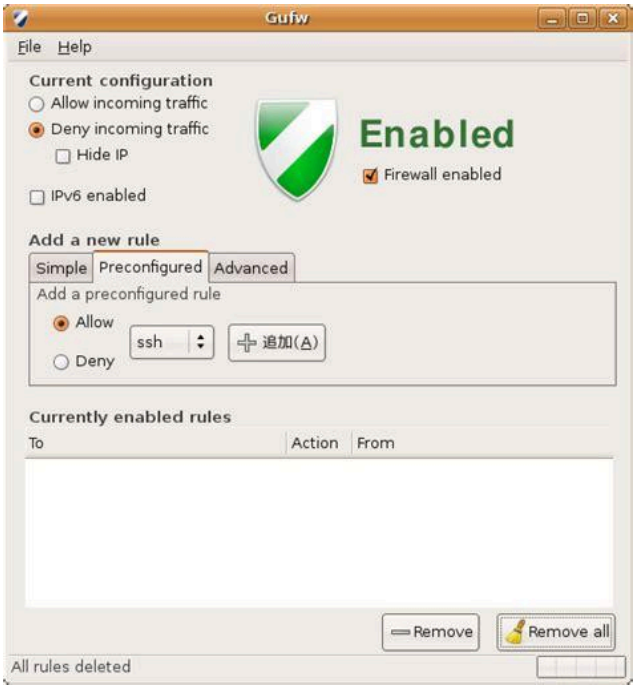
[247] https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_data_zbf/configuration/xe-16/sec-data-zbf-xe-16-book/conf-fw-tcp-syn-cookie.pdf
[248] https://elixir.bootlin.com/linux/v6.15.5/source/net/ipv4/syncookies.c#L88
[249] https://www.geeksforgeeks.org/computer-networks/how-syn-cookies-are-used-to-preventing-syn-flood-attack/
[250] https://elixir.bootlin.com/linux/v6.15.5/source/net/ipv4/syncookies.c
[251] https://elixir.bootlin.com/linux/v6.15.5/source/net/ipv6/syncookies.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_ipv4.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 netfilter[258] 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 users[259].

Overall, UFW is available by default as part of Ubuntu since version 8.04 TLS. 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 ufw[260] - 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].



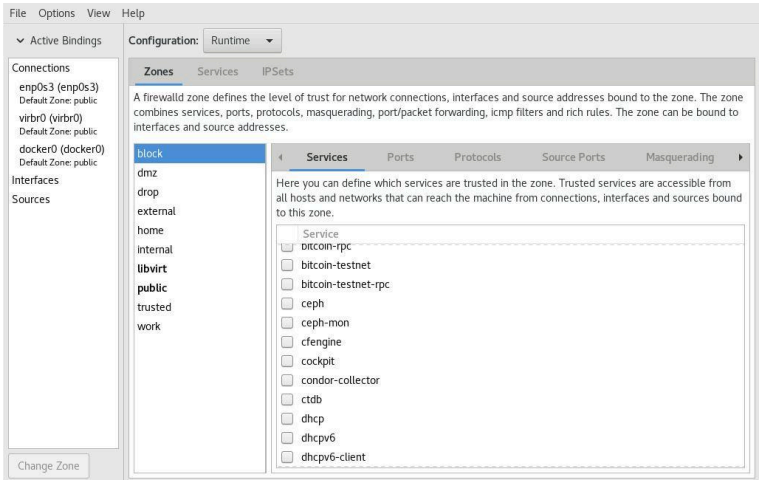
[258] https://medium.com/@boutnaru/the-linux-security-journey-netfilter-90c6cf12ca40
[259] https://help.ubuntu.com/community/UFW
[260] https://en.wikipedia.org/wiki/Uncomplicated_Firewall
[261] https://wiki.ubuntu.com/UncomplicatedFirewall

Firewalld (Firewall Daemon)

Firewalld (Firewall Daemon) is a firewall management tool for Linux based systems. Hence, basically “firewalld” acts as a front-end for netfilter[262]. 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 Ubuntu[263].

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 rules[264] and complete D-BUS[265] 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 below[267]. 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 firewalld[268]. By the way, there is also “firewall-applet” which is a tray applet application for firewalld[269]. We can also checkout the source code of firewalld hosted on GitHub[270].



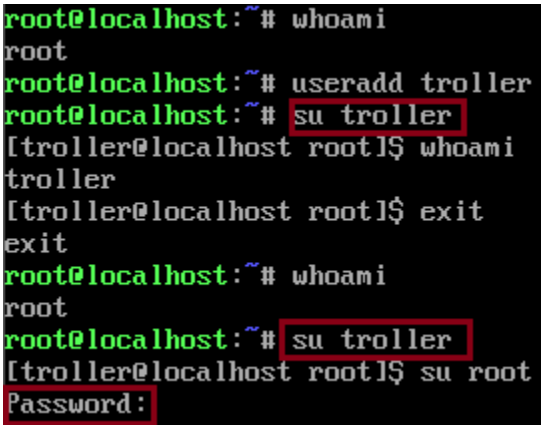
[262] https://medium.com/@boutnaru/the-linux-security-journey-netfilter-90c6cf12ca40
[263] https://en.wikipedia.org/wiki/Firewalld
[264] https://firewalld.org/
[265] https://medium.com/@boutnaru/the-linux-concept-journey-d-bus-desktop-bus-dd8c69ade019
[266] https://www.redhat.com/en/blog/beginners-guide-firewalld
[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 manager[271].

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” syscall[272].

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 remotely[273].



[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 community[274]. 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 below[275].

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 sshfs[277] and X11 forwarding[278]. Also, OpenSSH is supported on various Unix based operating like (but not limited to): AIX, HP-UX, Irix, Linux, NetXT, SCO Solaris, macOS and Cygwin[279].

[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 rootkits[281]. It is important to understand that once enabled, modules can't be either loaded or unloaded[282].

Overall, the configuration of this security feature is saved into the "modules_disabled" variable[283]. Thus, beside checking for the "CAP_SYS_MODULE" capability when trying to unload a kernel module[284] or when trying to load a kernel module[285] 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 fail[287] - 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 operations[289].

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

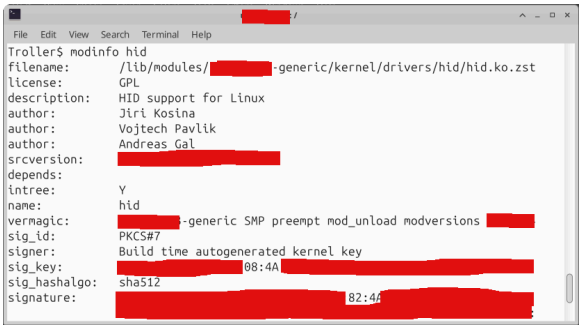
[280] https://medium.com/@boutnaru/the-linux-concept-journey-loadable-kernel-module-lkm-5eaa4db346a1
[281] https://dfir.ch/posts/today_i_learned_lkm_kernel.modules_disabled/
[282] https://sysctl-explorer.net/kernel/modules_disabled/
[283] https://elixir.bootlin.com/linux/v6.15.5/source/kernel/module/main.c#L129
[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 certificates[290]. 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 block[295]. It is leveraged in the kernel in different places (not limited to): code for signing a module file[296], 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 check[300]. "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 message[302] and emits the appropriate log entry[303].



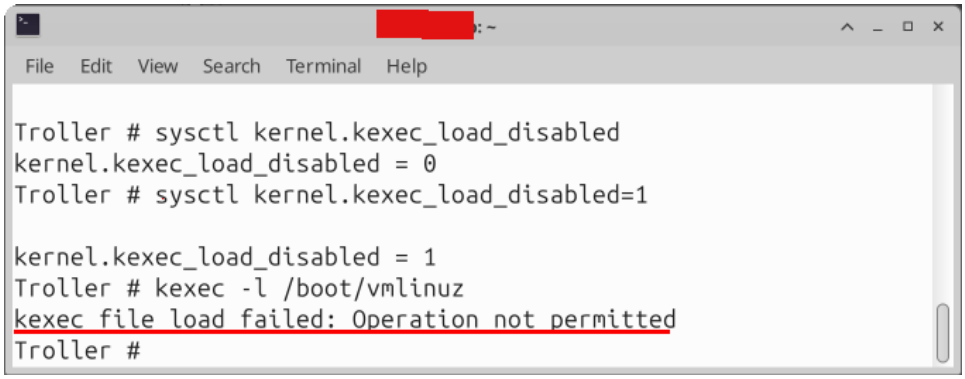
[290] https://www.kernel.org/doc/html/v4.19/admin-guide/module-signing.html
[291] https://elixir.bootlin.com/linux/v6.15.6/source/certs/Kconfig#L25
[292] https://elixir.bootlin.com/linux/v6.15.6/source/certs/Kconfig#L30
[293] https://linux.die.net/man/8/modinfo
[294] https://elixir.bootlin.com/linux/v6.15.6/source/include/linux/module_signature.h#L33
[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 distributions[304] the kernel is loaded directly (without going over the firmware and the bootloader again), this is due to the use of the kexec[305]. Because kexec can be leveraged to bypass security mitigations\mechanisms like secure boot[306] 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 function[307]. 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" syscall[309].

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 feature[313].



[304] https://medium.com/@boutnaru/the-linux-concept-journey-linux-distribution-2dfc68aaa4f4
[305] https://medium.com/@boutnaru/the-linux-concept-journey-kexec-kernel-execute-e5050fa085ea
[306] https://mjg59.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 policies[314]. For enforcing the policy it leverages the USB device authorization which is implemented as part of the Linux kernel[315].

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 conditions[316].

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 privileges[317]. For more information we can check the source code of USB guard as part of its GitHub repository[318].

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 "044b5e168d40ee024547
reject via-port "1-2"
```

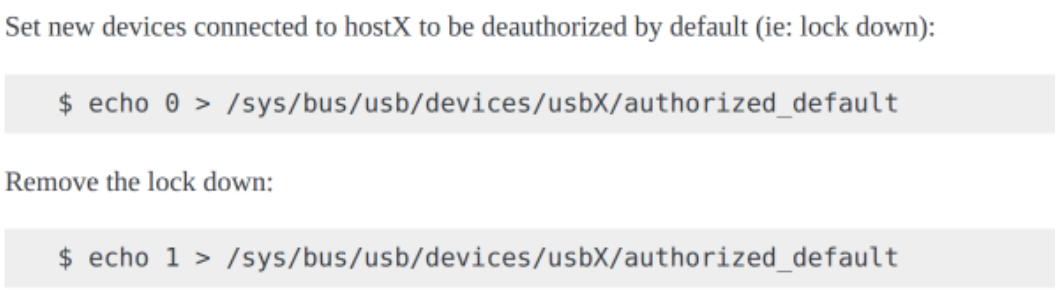
[314] https://usbguard.github.io/
[315] https://www.kernel.org/doc/Documentation/usb/authorization.txt
[316] https://usbguard.github.io/documentation/rule-language.html
[317] https://wiki.archlinux.org/title/USBGuard
[318] https://github.com/USBGuard/usbguard

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 only[319].

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 sysfs[321]. 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" function[325] is called for de-authorization.



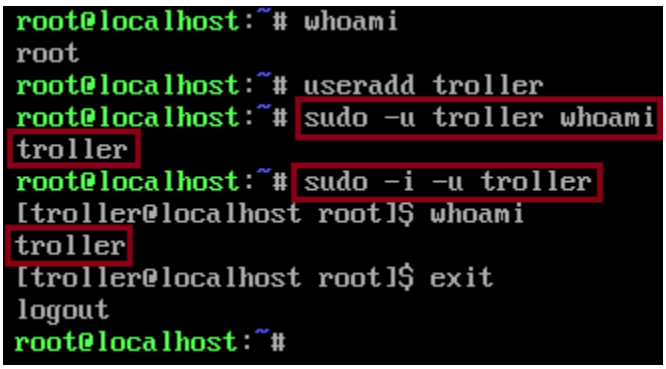
[319] https://www.kernel.org/doc/Documentation/usb/authorization.txt
[320] https://medium.com/@boutnaru/the-linux-security-journey-usb-guard-universal-serial-bus-guard-66e1a45ac901
[321] https://medium.com/@boutnaru/the-linux-concept-journey-sysfs-filesystem-for-exporting-kernel-objects-ea8c0172b1cf
[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 user[326]. 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 username[327] - as shown in the screenshot below.

Overall, "sudo" is similar to "su" command line utility[328]. 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 identity[329].

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 saving[330]. sudo was also added as part of the Windows operating system[331].



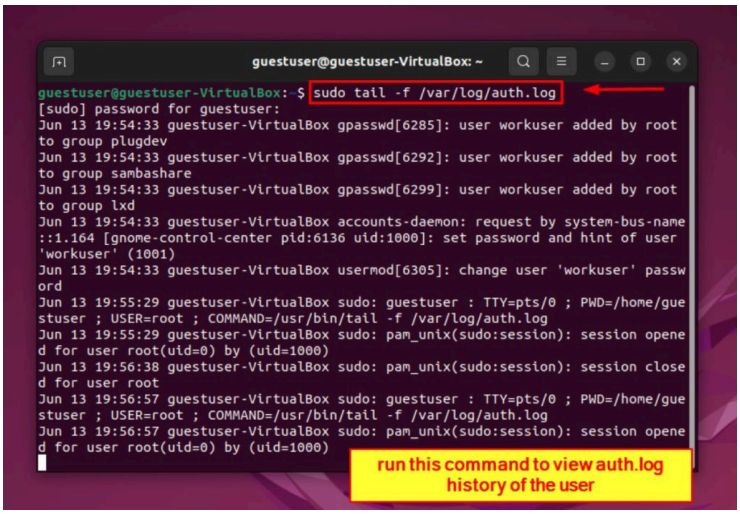
[326] https://phoenixnap.com/kb/linux-sudo
[327] https://unix.stackexchange.com/questions/176997/sudo-as-another-user-with-their-environment
[328] https://medium.com/@boutnaru/the-linux-security-journey-su-substitute-user-4c50cf6df034
[329] https://www.redhat.com/en/blog/difference-between-sudo-su
[330] 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 distributions[332]. 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 more[334] - as shown in the screenshot below[335]. By the way, we can also write directly to the authentication logs using the "logger" command line utility[336]. 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 events[337]. We can also watch this information using "journalctl" and even send it using the syslog protocol.



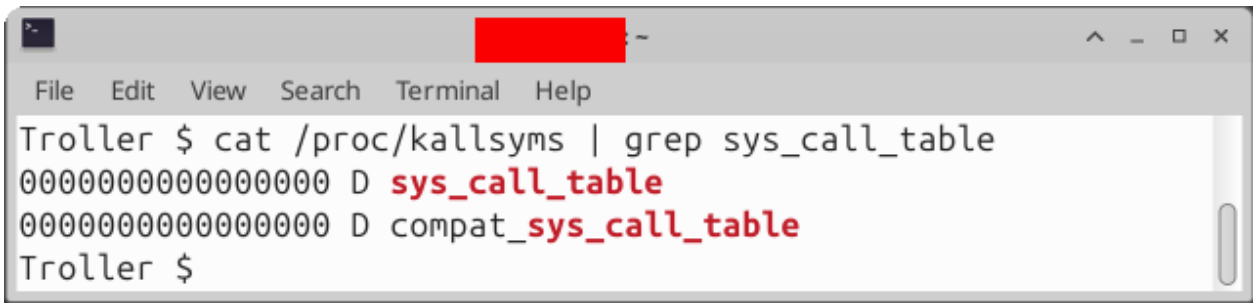
[332] https://medium.com/@boutnaru/the-linux-concept-journey-linux-distribution-2dfc68aaa4f4
[333] https://betterstack.com/community/guides/logging/monitoring-linux-auth-logs/
[334] https://www.netsurion.com/articles/top-5-linux-log-file-groups-in-var-log
[335] https://linuxier.com/how-to-check-login-history-in-linux/
[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 code[338]. 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 ABI[339]. However, even this small attack surface can still lead to security vulnerabilities that circumvent Linux security measures[340].

Overall, we can eliminate many syscalls on 64-bit systems by building the kernel with the "CONFIG_COMPAT" configuration item disabled[341]. 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 Mips[342]. When enabled the symbol "compat_sys_call_table" is defined[343] 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 use[346]. Those types of mitigations are also part of other operating systems like the "win32k.sys system call filtering" on Windows[347].



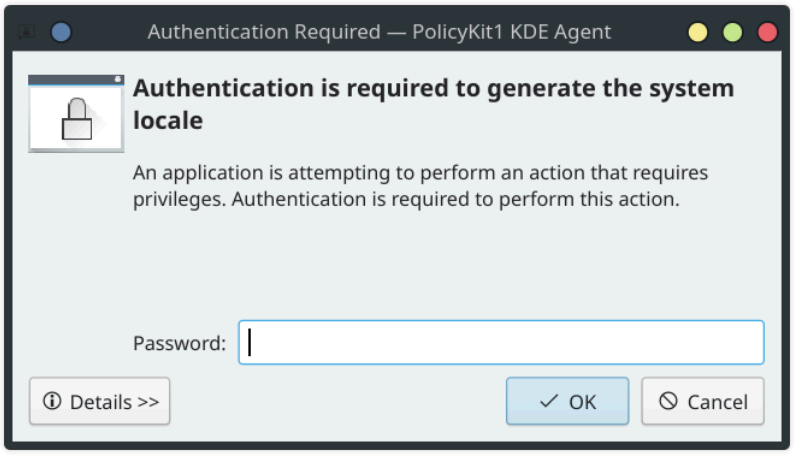
[338] https://medium.com/@boutnaru/the-linux-concept-journey-syscalls-system-calls-efcd7703e072
[339] https://man7.org/linux/man-pages/man2/syscall.2.html
[340] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=syscall+linux
[341] https://www.kernel.org/doc/html/v4.19/security/self-protection.html
[342] https://cateee.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/mtth-bfft/win32k-mitigation

PolKit (Authorization Manager)

PolKit (Authorization Manager) is used for handling policies which allow unprivileged processes to speak to privileged processes. Thus, is it basically a framework for centralizing the decision making with respect to granting access to privileged operations for unprivileged applications[348].

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 information[349]. 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 command[350].

Lastly, there are also “PolKit Authentication Agents” that are provided by desktop environments - as shown in the screenshot below[351]. Hence, for each user session there is authority and an authentication agent[352]. There are several binaries as part of PolKit: “polkit” which is the authorization manager[353], “polkitd” which is the system daemon[354], “pkcheck” that verifies if a process is authorized[355], “pkaction” which gets details about a registered action[356], “pkexec” that executes command as another user[357] and “pkttyagent” that is a textual authentication helper[358]. 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 kernel[359]. This is opposed to Grsecurity which is implemented as a series of modifications to the kernel source code[360]. The goal is to protect the kernel rather than the kernel protecting user-space[361]. For better understanding I suggest going over KSPP's GitHub tracker[362]. 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 tested[363].

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 exposers (like destination tracking and memory initialization)[364].

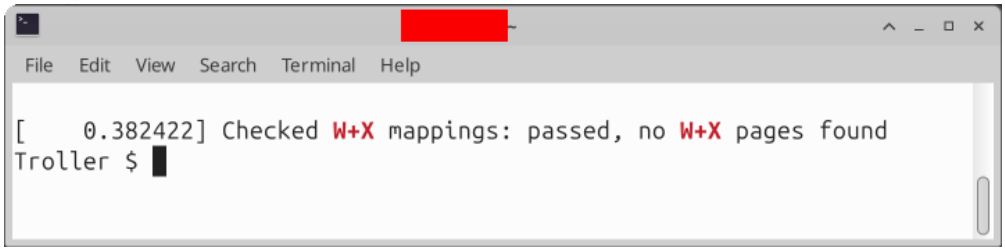
[359] https://medium.com/@boutnaru/the-linux-concept-journey-linux-kernel-0b06457569e8
[360] https://security.stackexchange.com/questions/176390/what-is-kspp-kernel-self-protection-project
[361] https://outflux.net/slides/2021/lss/kspp.pdf
[362] https://github.com/KSPP/linux/issues
[363] https://www.kernel.org/doc/Documentation/security/self-protection.txt
[364] https://docs.kernel.org/security/self-protection.html

Debug WX

"Debug WX" is a build configuration ("CONFIG_DEBUG_WX") of the Linux kernel[365]. 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 easier[366].

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): x86[367], RISC-V[368], ARM64[369] and PowerPC[370].

Lastly, this feature is included as part of the kernel configuration since version 5.8 of the Linux kernel[371]. 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 NX[372].



[365] https://medium.com/@boutnaru/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