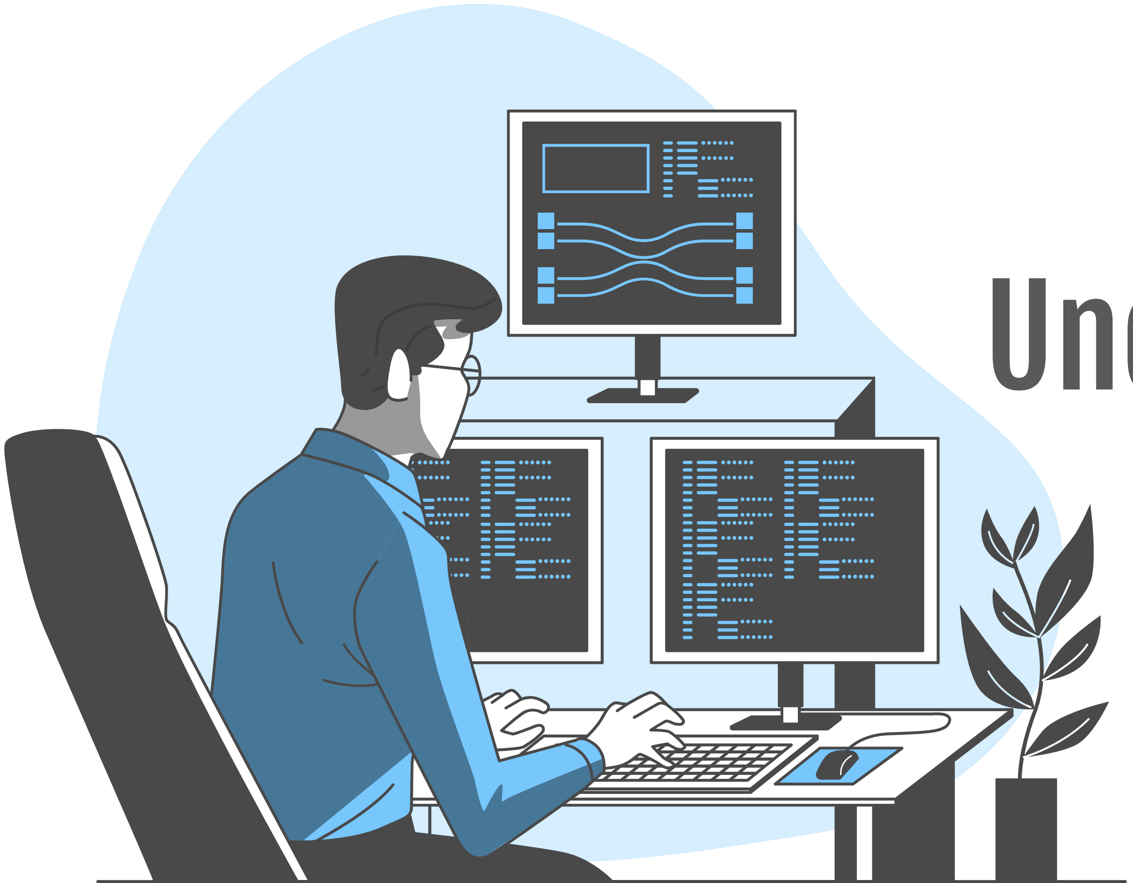


Understanding Capabilities

But Why?

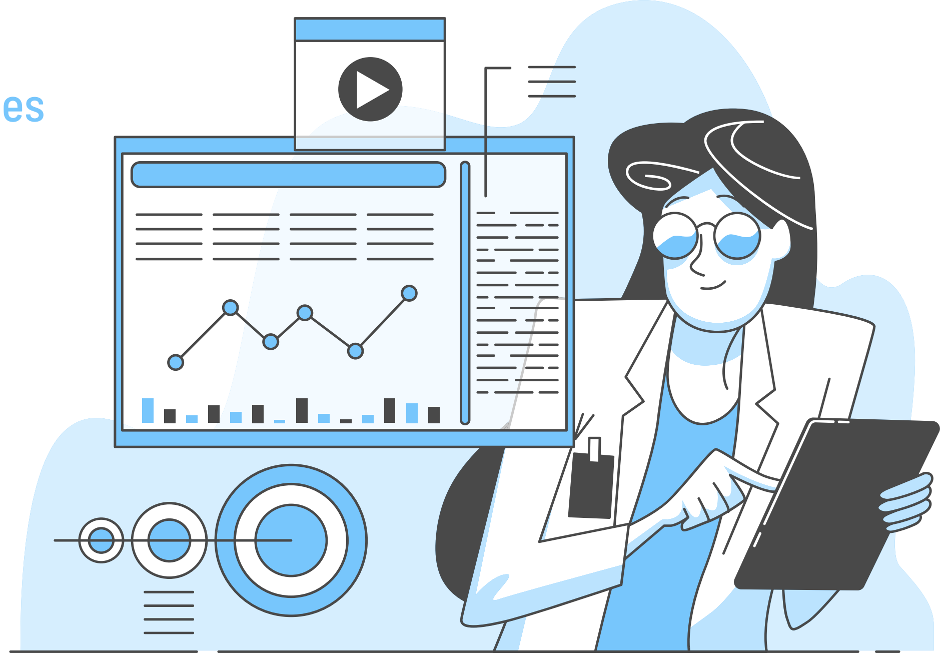
Table of Contents

01 Understanding SUID

02 Understanding Capabilities

03 Understanding BPF

04 Understanding Seccomp



Let’s understand suid first

Why suid ?

- Imagine a situation where a script needs to be executed by a set of users but only owner of that script has rights to execute that script in a particular way.

- By default Linux applications and programs runs with the same exact permissions of the user who executes it.

We need a solution for this situation

Why suid ?

- If you are the owner of an executable file, with the help of SUID permission set, other users will be running the executable with your permission and not theirs.
- This elevation of privileges is not permanent. It’s a temporary elevation only when the program is being executed.
- Example :- Being able to change password of user without being root

01 Demonstration

Changing Password

Drawbacks of suid

All root privileges Or none

Should we trust privileges escalation to process code itself?

Do We Need All Privileges

Do every process require all root possibilities to perform one privileged action?

No!

SUID is like
All or nothing.
Deal with it.

Why not suid ?

Usually it makes sense to allow a trusted binary to use root permissions to execute. The unfortunate thing with software is that it may contain bugs. So even the smallest mistake with a setuid binary may result in total compromise.

LETS UNDERSTAND CAPABILITIES NOW

WHY CAPABILITIES ?

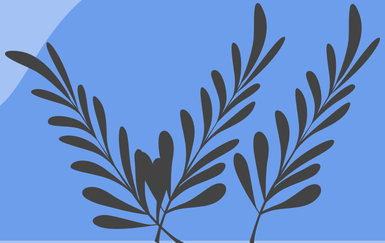
Because it is what it is xD

Is root the most powerful user ?

Let’s break the myth, The way I describe it is that most people think of root as being all-powerful.
This isn't the whole picture, the root user with all capabilities is all-powerful.

HOLD A MINUTE !

LET ME EXPLAIN



02

Demonstration

Capabilities Overview

CAPABILITIES

Linux capabilities provide a subset of the available root privileges to a process.
This effectively breaks up root privileges into smaller and distinctive units.
Each of these units can then be independently be granted to processes.

This way the full set of privileges is reduced and decreasing the risks of exploitation.

38 Capabilties Total

DOCKER PROVIDES THESE BY DEFAULT

chown, dac_override, fowner, fsetid, kill, setgid, setuid, setpcap, net_bind_service, net_raw, sys_chroot, mknod, audit_write, setfcap

03 Demonstration

Capabilities

Berkersly Packet Filter (BPF)

Not So Typical !

BERKERSLY PACKET FILTER

BPF is especially suited to writing network programs and it's possible to write programs that attach to a network socket to filter traffic, to classify traffic, and to run network classifier actions. It's even possible to modify the settings of an established network socket with an eBPF program.

BPF USE CASES

01 Filtering Packets

02 DDOS Protection

03 Monitoring Agents

04 IDS (Intrusion Detection System)

WHY SECCOMP ?

Introduction of seccomp

First Introduced in 2004, as a way for a process to transition itself to a kernel-enforced mode where only the read, write, exit, and sigreturn system calls are possible.

This process was needed to constrain this untrusted binary code so it could only read input, make calculations, and write its results. It should permit nothing else.

Introduction of seccomp

With seccomp, a management process could load an untrusted shared object file, open files for input and output, and enter seccomp mode before calling the untrusted shared object's entry point. The constrained process now cannot open any new files, change directories, fork new processes, spawn threads, exec new programs, or anything except read and write to its open files and exit gracefully.

Seccomp filter mode allows a process to install Berkeley Packet Filter (BPF) byte-code. Once installed, this BPF program can prevent the calling process, or any descendants, from making any system calls.

04

Demonstration

Seccomp By Program

05 Demonstration

Seccomp By Docker

Why not just use LSM's ?

LSM's uses Mandatory Access Control to protect the objects :- file, indoes, task structures, IPC structures etc.

LSM's attach context information to the objects, which it checks against a previously loaded policy.

Security Administrator with the required permissions can only modify the policies.

Unprivileged users can't change the policy

Why not just use LSM's ?

The kernel checks seccomp filters so early in the system call handling sequence, they reduce the amount of code that an attacker can search for exploitable flaws. This reduces the kernels attack surface from these processes.
LSMs, because they hook more deeply in the system call sequence, do not reduce the attack surface to the same extent that seccomp filters do. Specifically, the kernel runs the hooks after it has mapped system call parameters to internal objects, and this code may all contain flaws either now or after future modifications.
Again, seccomp reduces this attack surface.

Conclusion

Mandatory Access Control policies implemented via LSMs are the tool of choice when you seek to create a fine-grained security policy globally for the system. Seccomp filters are the tool of choice for an unprivileged process to drop its ability to make certain system calls, and can be an important component of more general process sandboxing techniques like Linux containers.

THANKS