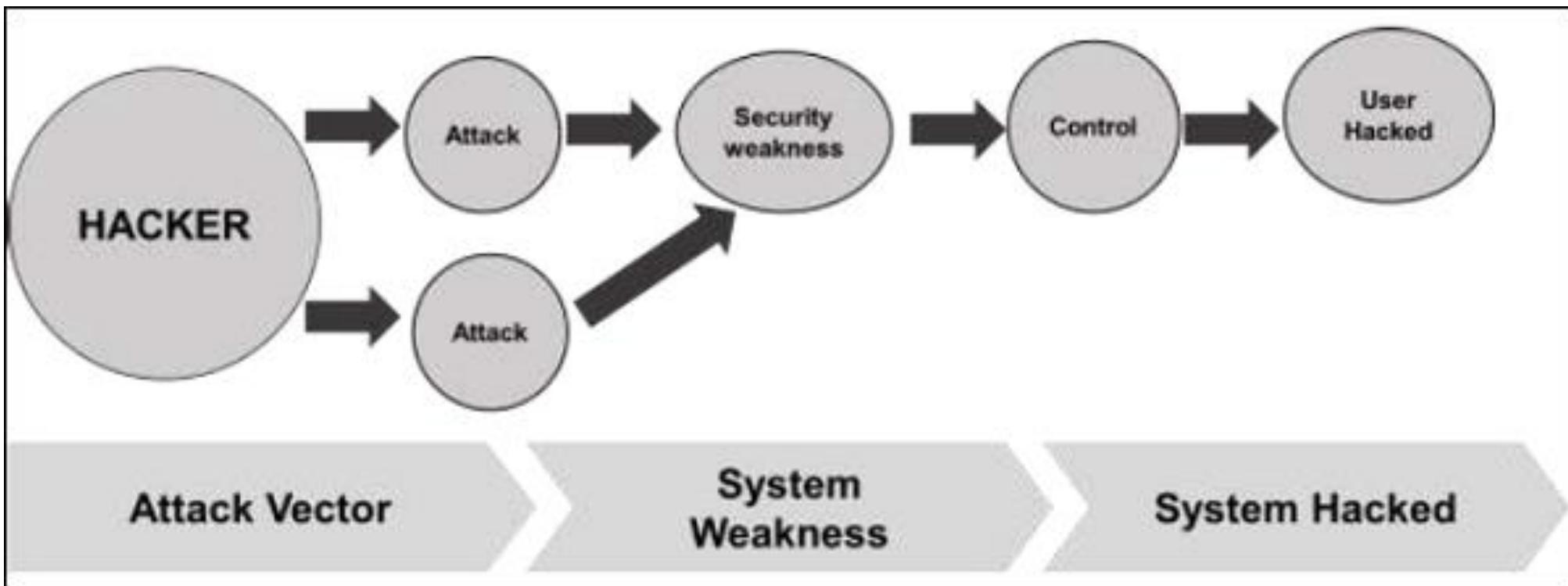


# MODULE 1

## Security Principles

Prof. Kumar Debasis

# Attack model



# Vulnerability

- A **vulnerability** is a weakness which can be exploited by a Threat Actor, such as an attacker, to perform unauthorized actions within a computer system.
- A window of vulnerability is a time frame within which defensive measures are diminished, compromised or lacking.
- Common security vulnerabilities are missing data encryption, outdated security software, improper authorization methods, bugs in software, weak passwords etc.

# Attack vector

- An **Attack vector** is a path or means by which a hacker (or cracker) can gain access to a computer or network server in order to deliver a payload or malicious outcome.
- **Attack vectors** enable hackers to exploit system vulnerabilities, including the human element.
- Common attack vectors include malware, email attachments, web pages, pop-ups, instant messages, text messages and social engineering.

# Attack Surface

- The **attack surface** of a software environment is the sum of the different points where an unauthorized user can try to enter data to or extract data from an environment.
- The **physical attack surface** includes everything related to hardware and physical devices; here we're talking about routers, switches, desktop computers, notebooks, tablets and mobile phones, TVs, printers, USB ports, surveillance cameras, etc.
- Once an attacker has accessed a computing device physically, the intruder will look for **digital attack surfaces** left vulnerable by poor coding, default security setting or poorly-maintained software that has not been updated or patched.
- This digital attack surface, includes software applications, networks, ports, operating system services, web and desktop applications and more. In other words, everything running on the digital side of any company.

# Attack surface categories

## Network Attack Surface

Vulnerabilities over an enterprise network, wide-area network, or the Internet

Included in this category are network protocol vulnerabilities, such as those used for a denial-of-service attack, disruption of communications links, and various forms of intruder attacks

## Software Attack Surface

Vulnerabilities in application, utility, or operating system code

Particular focus is Web server software

## Human Attack Surface

Vulnerabilities created by personnel or outsiders, such as social engineering, human error, and trusted insiders

# What is a software vulnerability?

- In the world of cyber security, **vulnerabilities** are **unintended flaws** found in software programs or operating systems.
- Vulnerabilities can be the result of **improper computer or security configurations** and **programming errors**.
- If left unaddressed, vulnerabilities create security holes that cybercriminals can exploit.

# Malware vs Exploit

- **Malware** is classified according to the payload or malicious action it performs, hence the different categories or types such as viruses, worms, Trojans and botnets, etc.
- As for **Exploits**, we can divide them into different categories according to the vulnerability they exploit to gain access to the system. Among these categories we can mention buffer overflow, Cross Site Scripting (XSS), SQL injection.
- A lot of malicious code makes use of exploits to achieve its goal and take control of a system, but it doesn't all exploit vulnerabilities in the same way.

## When Attackers Target Vulnerabilities



Attacker creates exploits  
to target software  
vulnerability



OR

1. Exploits may arrive via:
  - Attachment to email messages
  - Compromised websites
  - Social networking sites

2. Attacker may directly target vulnerable servers



Users are lured into executing  
the exploit via social  
engineering techniques



OR

Exploits may drop malware  
onto the vulnerable system or  
allow attackers remote  
control

# Why do vulnerabilities pose security risks?

- Hackers write code to target a specific security weakness.
- They package it into malware. The malicious software takes advantage of a vulnerability to compromise a computer system or cause an unintended behavior.
- In most cases, a patch from the software developer can fix this.
- If you're an everyday computer user, a vulnerability can pose serious security risks because malware can infect a computer through otherwise harmless web browsing activities, such as viewing a website, opening a compromised message, or playing infected media.

# How to tell if your system is infected?

- Your computer is slowing down
- Annoying ads are displayed
- Regular Crashes
- Your browser homepage changed without your input
- Your friends say they receive strange messages from you
- Unfamiliar icons are displayed on your desktop
- Programs opening and closing automatically
- You run out of storage space
- Internet traffic suspiciously increases
- Your security solution is disabled
- Unusual error messages are displayed on computer

# What can malwares do?

- Steal your sensitive information
- Restrict access to your files
- Launch attacks on other connected systems
- Corrupt or delete important files
- Provide unauthorized remote access to your system
- Make unwanted modifications to your system
- Unusual system behavior to disrupt daily operations

# Cyber kill chain/Intrusion kill chain/Lockheed martin kill chain



# Why need of a new model – Unified Kill Chain

- Cyber Kill Chain has been widely criticized due to fact that some of the seven attack phases can be bypassed.
- Cyber Kill Chain fails to cover other attack vectors and internal attacks paths.
- The actual scope of today's cyber threats extends far beyond that of the Cyber Kill Chain, and so a need arose to develop a new model.
- The **Unified Kill Chain** was created, combining the Cyber Kill Chain with the ATT&CK matrix.
- MITRE ATT&CK® is a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations.

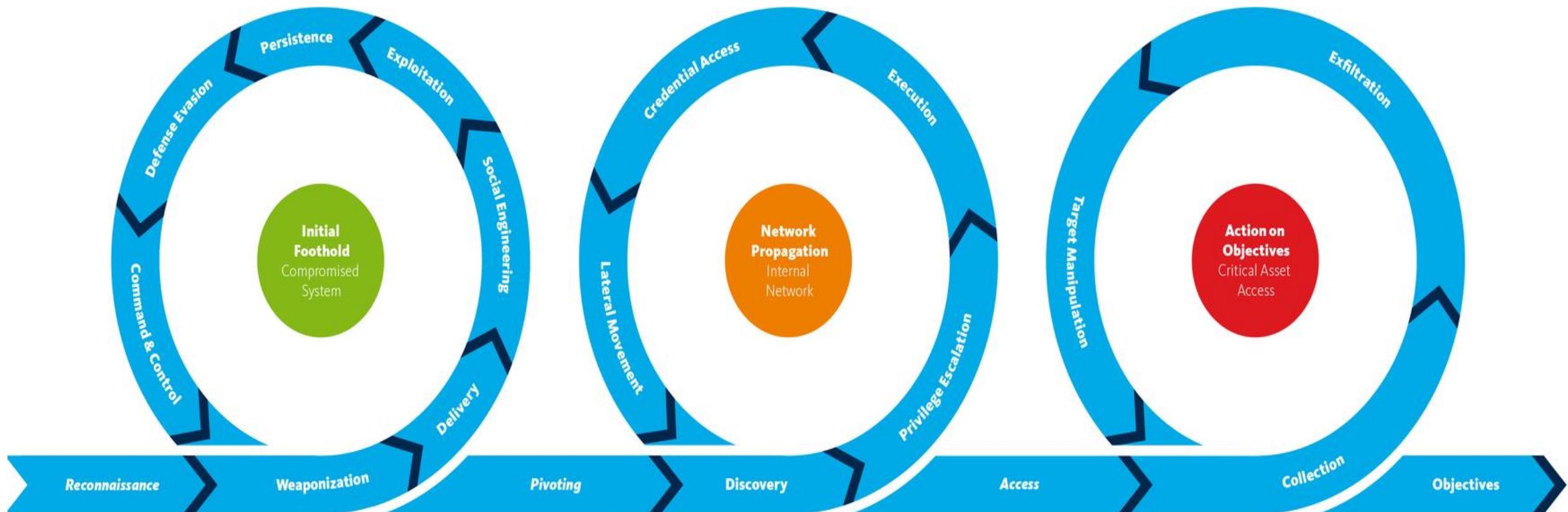
# Why need of a new model – Unified Kill Chain

- The Mitre Corporation is an American not-for-profit organization. It manages federally funded research and development centers supporting several U.S. government agencies.
- ATT&CK stands for Adversarial Tactics, Techniques, and Common Knowledge.
- MITRE started this project in 2013 to document common tactics, techniques, and procedures (abbreviated to TTPs) that advanced persistent threats use against Windows enterprise networks.
- ATT&CK organizes adversarial techniques into a set of tactics to help explain to provide context for each technique. The so-called ATT&CK Matrix available on the MITRE website is also referred to as the Enterprise Matrix.

# Why need of a new model – Unified Kill Chain

- By uniting and extending the Cyber Kill Chain and MITRE's ATT&CK framework, The Unified Kill Chain offers a significant improvement over the scope limitations of the CKC and the time-agnostic nature of ATT&CK.
- The UKC is an ordered arrangement of 18 unique attack phases that may occur in end-to-end cyber attacks, which covers activities that occur outside and within the defended network.
- The unified model can be used to analyze, compare and defend against end-to-end cyber attacks by Advanced Persistent Threats (APTs).

# Unified kill chain



<https://attack.mitre.org/>

# Stuxnet

- **Stuxnet** is an extremely sophisticated computer worm that exploits multiple previously unknown Windows zero-day vulnerabilities to infect computers and spread.
- It will harm only the machine with SCADA/PLC controller
- Specifically, it targets **centrifuges** used to produce the enriched uranium that powers nuclear weapons and reactors.
- Stuxnet was first identified by the infosec community in 2010, but development on it probably began in 2005.

# Stuxnet – Who created it?

- It's now widely accepted that Stuxnet was created by the intelligence agencies of the United States and Israel.
- The classified program to develop the worm was given the code name "Operation Olympic Games"; it was begun under **President George W. Bush** and continued under **President Obama**.

# What is the purpose of Stuxnet

- The U.S. and Israeli governments intended Stuxnet as a tool to **derail**, or at least **delay**, the Iranian program to develop nuclear weapons.
- The **Bush and Obama administrations** believed that if Iran were on the verge of developing atomic weapons, Israel would launch airstrikes against Iranian nuclear facilities in a move that could have set off a **regional war**.
- Operation Olympic Games was seen as a nonviolent alternative.

# Stuxnet Source code and size

- Source code was written in multiple programming platform.
- Average viruses are about 10k bytes in size. Stuxnet was 500 KB (and no graphics).
- It is unusual for a virus to contain one zero-day vulnerability. Stuxnet had 4.
- Stuxnet also acted like a rootkit – hiding its actions and its presence.
- It was the first virus to include code to attack Supervisory Control and Data Acquisition (SCADA) systems.

# Secure Coding

- **Secure coding** is the practice of developing computer software in a way it guards against the accidental introduction of security vulnerabilities – Wiki~\*.
- Secure coding is important for all software — whether you write code that runs on mobile devices, personal computers, servers, or embedded devices.

# Risks of Insecure Software

- An insecure application lets hackers in. They can take direct control of a device — or provide an access path to another device.
- This can result in:
  - **Denial of service to a single user**
  - **Compromised secrets**
  - **Loss of service**
  - **Damage to the systems of thousands of users**
  - **Loss of life**

# Most common security vulnerabilities

- **Buffer overflow**
- **Open source software**
- **Cross-Site Scripting (XSS)**
- **Code injection**

# Implementing best secure coding practices

- Follow OWASP guidelines
- Avoid unbounded write operators in C++
- Implement proper input validation
- Dynamic Application Security Testing (DAST)
- Monitor the security status of your Vendors

# Important Things

- **Embedded Systems:** A heart rate monitor embedded in a wristwatch that can connect to a smart phone to display the heart's status in real time or an accelerometer embedded in shoes to monitor speed, distance traveled and calories burned.
- The **Open Web Application Security Project (OWASP)** is a non-profit organization founded in 2001, with the goal of helping website owners and security experts protect web applications from cyber attacks. OWASP has 32,000 volunteers around the world who perform security assessments and research.

# Software security fundamentals

- What is a Software Security?
  - Software security is the **idea of engineering software** so that it continues to function correctly under malicious attack.
  - Software Security aims to **avoid security vulnerabilities** by addressing security from the early stages of software development life cycle.

# Why Software security?

- Most software systems today contain numerous flaws and bugs that get exploited by attackers.
- New threats emerge everyday.
- Convenience trumps security measures.
- Exponential increase in vulnerabilities in software systems.
- Programmers have a long history of repeating the same security-related mistakes!

# Terminologies

- **Error/Mistake:** A human mistake which causes the flaw in the software is called an Error. Usually Error is introduced in the Development process. For e.g. logical or syntax error or variable name typed incorrectly.
- **Bug/Fault/Defect:** A Defect in a software product reflects its inability or inefficiency to comply with the specified requirements and criteria and, subsequently, prevent the software application from performing the desired and expected work.
- **Failure:** Once the product is deployed and customers find any issues then they call the product as a failed product. After release, if an end user finds an issue then that particular issue is called as failure.

# Terminologies

Error / Mistake	Defect / Bug/ Fault	Failure
Found by Developer	Found by Tester	Found by Customer



# Threat, Vulnerability, & Risk



# Software Security: Three Pillars



Three pillars of software security

1. Risk management framework
2. Touchpoints
3. Knowledge



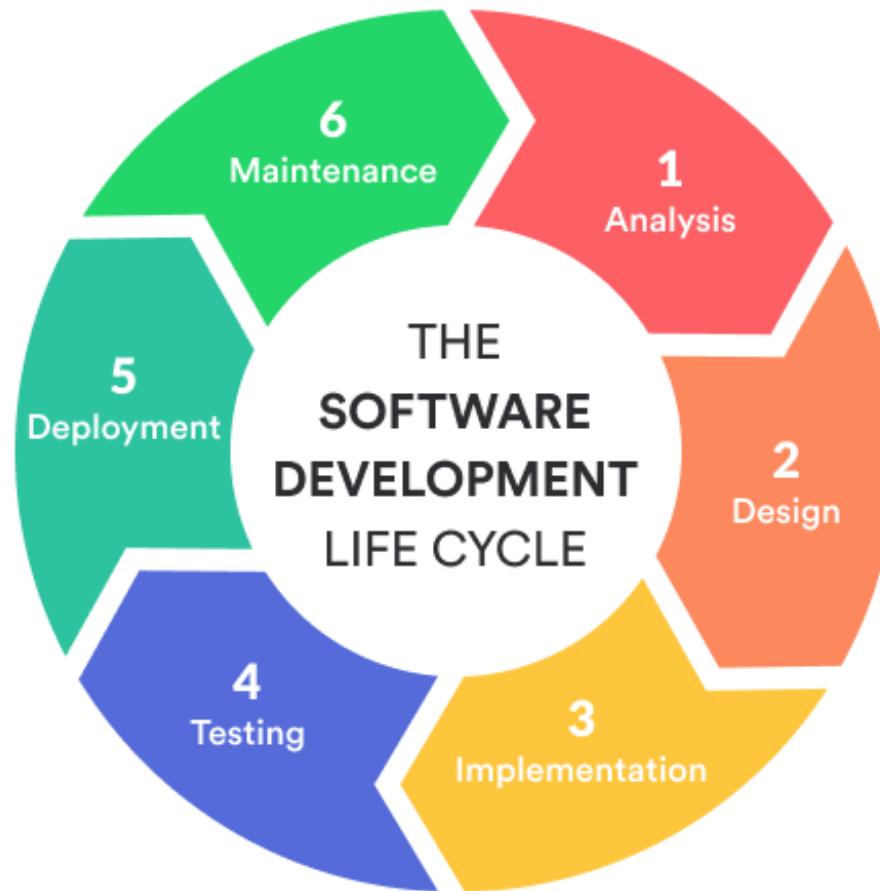
# Software Security: Risk management

- Risk management is an important business practice that helps businesses identify, evaluate, track, and mitigate the risks present in the business environment.
  - **Step 1:** Identify the Risk...
  - **Step 2:** Analyze the Risk...
  - **Step 3:** Evaluate or Rank the Risk...
  - **Step 4:** Treat the Risk...
  - **Step 5:** Monitor and Review the Risk...

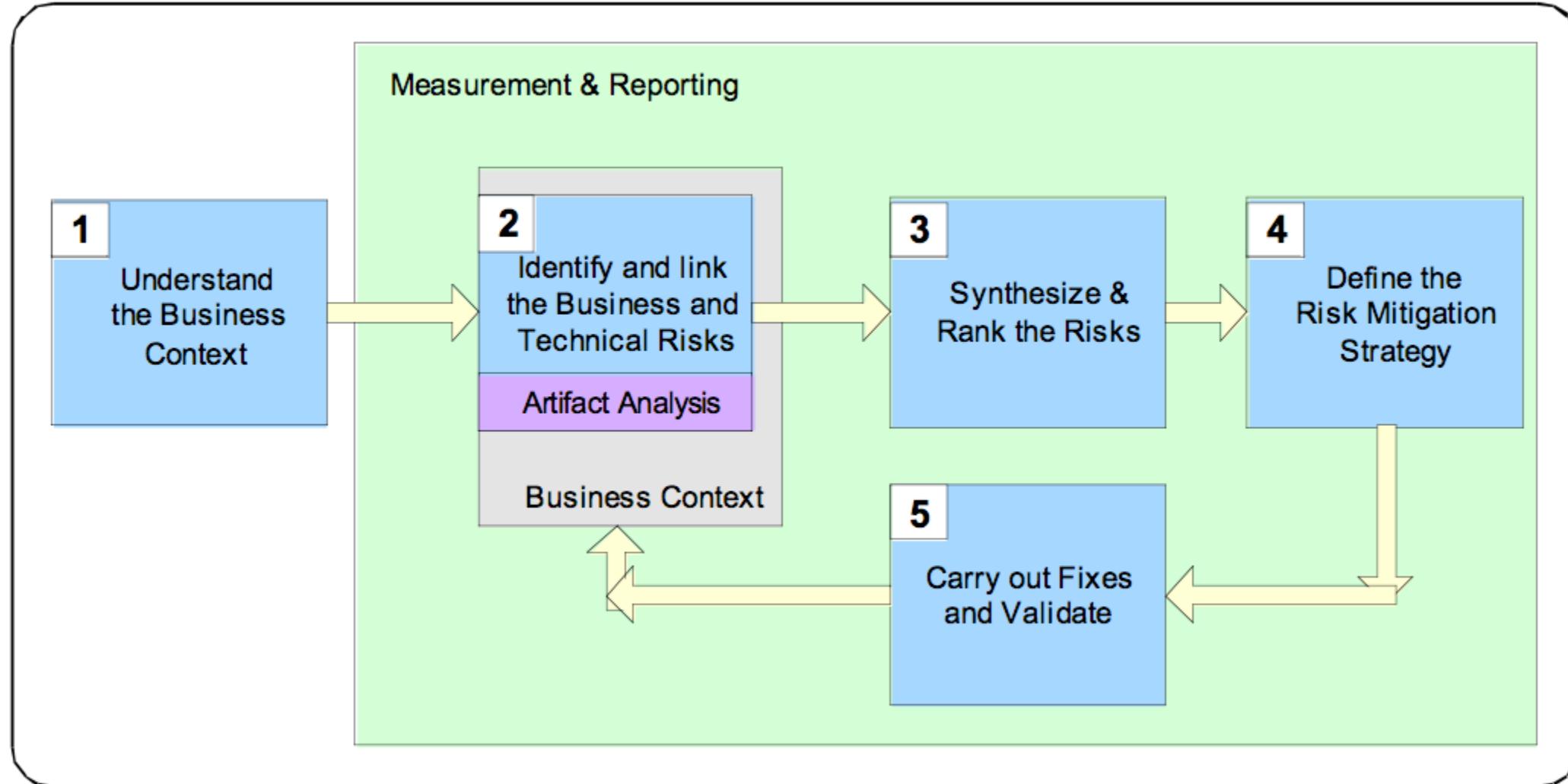
# Software Security: Risk management

- A continuous risk management process is an essential part to software security.
- **Managing software security risk**
  - **Risk management framework**
    - An overall approach to risk management.
    - Allows a consistent and continuous expertise-driven approach to risk management.
    - The goal is to consistently track and handle risks.

# Important Concept



# Risk management framework (RMF)



# RMF: Understanding the Business Context

- A key task of an analyst.
- Extract and describe business goals.
- Set priorities.
- Understanding what risks to care about.

# RMF: Identify the Business and Technical Risks

- Business risks impact business goals
- Identify Business Risks
- Identify Technical Risks and map them to business goals
- Evaluating software artifacts.

(Use Cases, Class diagrams, and other Unified Modeling Language (UML) models, requirements and design documents)

## WEBSITE:

<https://us-cert.cisa.gov/bsi/articles/best-practices/risk-management/risk-management-framework-%28rmf%29>

# RMF: Synthesize and Prioritize Risks

- Prioritize the risks based on the business goals
- Risk metrics:
  - Risk likelihood
  - Risk impact
  - Risk severity

Likelihood	Impact				
	Insignificant	Minor	Moderate	Major	Severe
Almost Certain	M	H	H	E	E
Likely	M	M	H	H	E
Possible	L	M	M	H	E
Unlikely	L	M	M	M	H
Rare	L	L	M	M	H

# RMF: Define the Risk Mitigation Strategy

- Create an efficient strategy for mitigating the risks that takes into account:
  - **Cost**
  - **Implementation time**
  - **Likelihood of success**
  - **Impact**
- Identifying the validation techniques

# RMF: Fix the Problems and Validate the Fixes

- Implementation of mitigation strategy
- Application of validation techniques
- Progress is measured in terms of **completeness against risk mitigation strategy.**

# RMF is a Multilevel Loop

- RMF stages may need to be applied over and over again throughout a project.
- One natural way to apply a cycle of the loop is during each particular software life-cycle phase.
- The RMF loop restarts continuously so that newly arising business and technical risks can be identified and the status of existing risks currently undergoing mitigation can be kept up.

# Touchpoints

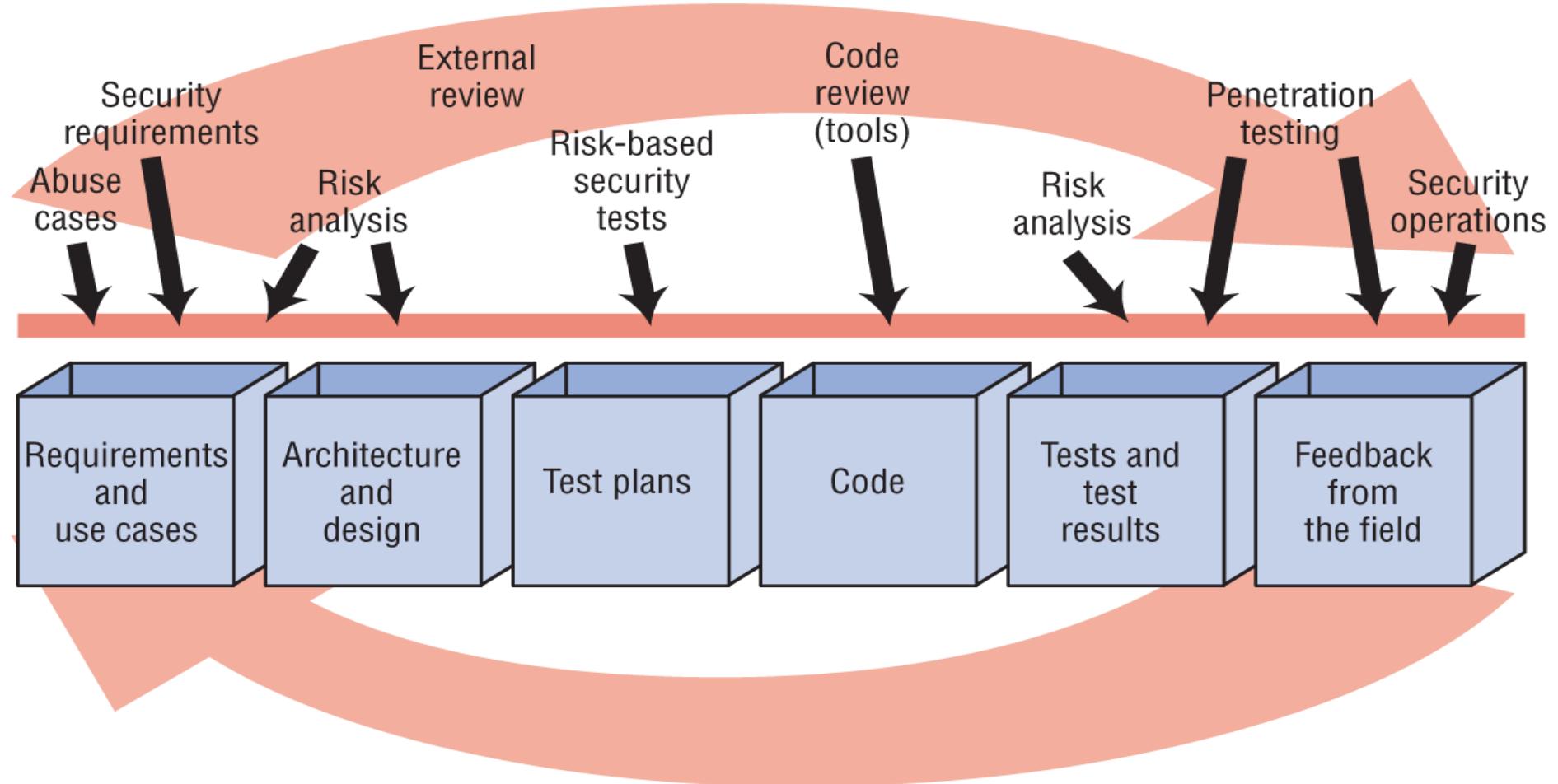
- Software Security Touchpoints specifies one set of touchpoints (best practices).
- These best practices are applied to the various software artifacts produced during software development.
- Touchpoints are a mix of **destructive** and **constructive** activities.
- Destructive activities are about attacks, exploits, and breaking software.
- Constructive activities are about design, defense, and functionality.

# Touchpoints

- **Code Review (Tools)**. Artifact: code. *Constructive activity*.
- **Architectural Risk Analysis**. Artifact: design and specification. *Constructive activity*.
- **Penetration Testing**. Artifact: system in its environment. *Destructive activity*.
- **Risk-Based Security Testing**. Artifact: system. *Mix between destructive and constructive activities*.
- **Abuse Cases**. Artifact: requirements and use cases. *Destructive activity*.
- **Security Requirements**. Artifact: requirements. *Constructive activity*.
- **Security Operations**. Artifact: fielded system. *Constructive activity*.

# Touchpoints

- Security touchpoints are set of security best practices



# Seven touchpoints

1. Code Reviews
  - Artifact: Code
  - Example of risks found: Buffer overflow
2. Architectural Risk Analysis
  - Artifact: Design and specifications.
  - Example of risks found: Design flaws
3. Penetration Testing
  - Artifact: System in its environment
  - Example of risks found: Susceptible to DDoS attack

Website: <https://www.drdobbs.com/the-7-touchpoints-of-secure-software/184415391>  
Website: <https://adriancitu.com/tag/security-touchpoints/>

# Seven touchpoints - Cont...

4. Risk-Based Security Testing
  - Artifact: System
  - Example of risks found: data leakage
5. Abuse cases
  - Artifact: Requirements and use cases
  - Example of risks found: Susceptibility to well-known tampering attack
6. Security Requirements
  - Artifact: Requirements
  - Example of risks found: Explicit description of data protection needs is missing
7. Security Operations
  - Artifact: Fielded system (Deployment system)
  - Example of risks found: Insufficient logging to prosecute a known attacker

# 1. Code Review

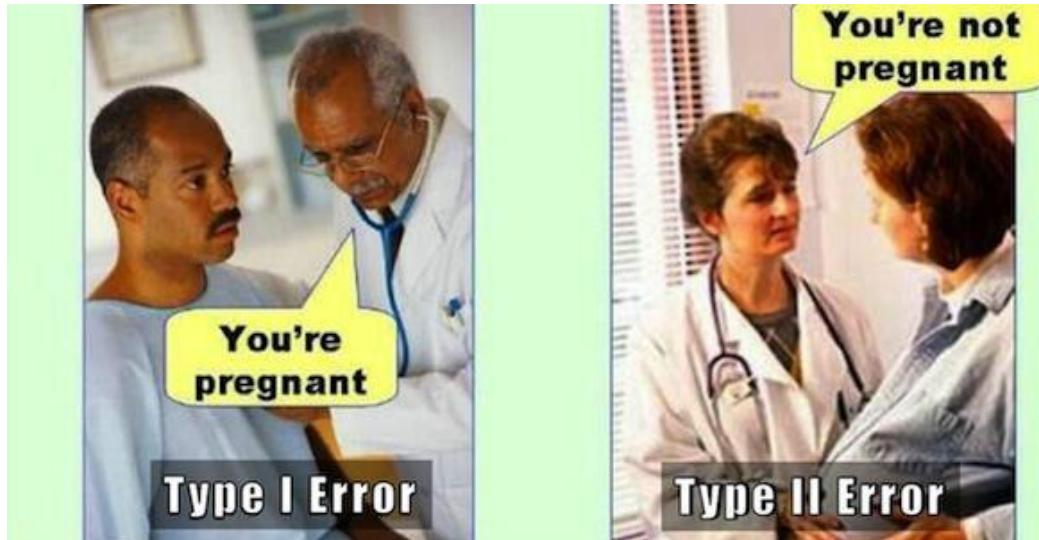
- Source code analysis
  - Static analysis
  - Dynamic analysis
- Focus is on finding and fixing bugs

# Static analysis

- Static analysis, a method of debugging by automatically examining source code before a program is run.
- **How static analysis works?**
  - It's done by analyzing a **set of code** against a set (or multiple sets) of **coding rules**.
  - This type of **analysis** may also be achieved through **manual code reviews**.
  - Using **automated tools** is much more **effective**.
  - Static analysis addresses **weaknesses** in **source code** that might lead to **vulnerabilities**.

# Static analysis

- Tools:
  - Pylint, Coverity, CAST, PEP8 etc.
  - Static analysis suffers from false positives and false negatives.



# Dynamic analysis

- Dynamic analysis
  - Create a dynamic environment which is very similar to the deployment and validate the code.



## 2. Architectural Risk Analysis

- Design flaws
  - **Account for 50% of security problems.**
  - **Can't be detected by staring at code.**
  - **A higher-level understanding** is required.

# Steps in Architectural Risk Analysis

- **Attack resistance analysis** – have as goal to define how the system should behave against known attacks.
- **Ambiguity analysis** – have as goal to discover new types of attacks or risks, so it relies heavily on the experience of the persons performing the analysis.
- **Weakness analysis** – have as goal to understand and assess the impact of external software dependencies.

### 3. Penetration Testing

- Method of evaluating software security by simulating an attack in an outside-in manner
- Testing the system in its final



## 4. Risk-based security tests

- Security testing should start at the feature or component/unit level and should use the items from the architectural risk analysis to identify risks.
- Also the security testing should continue at system level and should be directed at properties of the integrated software system.

# Security Testing approaches

The security testing should involve two approaches:

- **Functional security testing:** testing security mechanism to ensure that their functionality is properly implemented (kind of white hat philosophy).
- **Adversarial security testing:** tests that are simulating the attacker's approach (kind of black hat philosophy).

## 5. Abuse Case

- Misuse and abuse cases describe how users can misuse or exploit weak controls in software features to attack an application.
- A direct attack against business functionalities, which may bring in revenue or provide a positive user experience, can have a tangible business impact.
- Abuse cases can be an effective way to drive security requirements to properly protect these critical business use cases.

## Abuse Case1

- A user misuses the shopping cart by adding a large quantity of products without the intent to purchase

## Abuse Case2

- Denial of service attack with anonymous accounts

## Abuse Case3

- Automated denial of service attacks using botnet or testing tools

# 6. Security Requirements

Must satisfy three criteria:

- **Definition:** Must be explicitly defined at the requirements level.
- **Assumption:** Must take into account the assumption that the system will behave as expected.
- **Satisfaction:** Security requirements must satisfy the security goals.

# 7. Security Operation

- The main idea is that the security operations people and software developers should work together and each category can learn from the other.
- Attacks do occur, regardless of the strength of design and implementation, so monitoring software behavior is an essential defensive technique.
- Knowledge gained by understanding attacks should be cycled back into software development.

# Knowledge

- Involves gathering and sharing security knowledge.
- Software Security Knowledge Lists are:
  - Security Principles
  - Security Guidelines
  - Security Rules and Policies
  - Vulnerabilities
  - Exploits
  - Attack Patterns
  - Historical risks analysis

# Security Principles

- CIA model (Confidentiality, Integrity and Availability)
- Privacy
- Identification and Authentication
  - Passwords
    - Password management
      - **Password generators** (third-party products that can be used to create passwords out of random characters.)
      - **Password checker** (check the passwords for their probability of being guessed)
      - **Honeywords**
      - **Limiting login attempts**
      - **Token devices**
        - Synchronous (time based value used in authentication)
        - Asynchronous (Challenge based)

# Security guidelines

- Physical security
- Perimeter Security
- Physical Access Controls
- Monitoring Physical Access
- Security Operation Centre (System Management)
- System and network security
- Incident Response and Actions
- Protecting Sensitive Information

# Security rules and policies

- Rules for granting, controlling, monitoring, and removal
  - Roles and responsibilities
  - Infrastructure Security
    - Access Control
  - Asset policies
    - BYOD

# Historical risk analysis

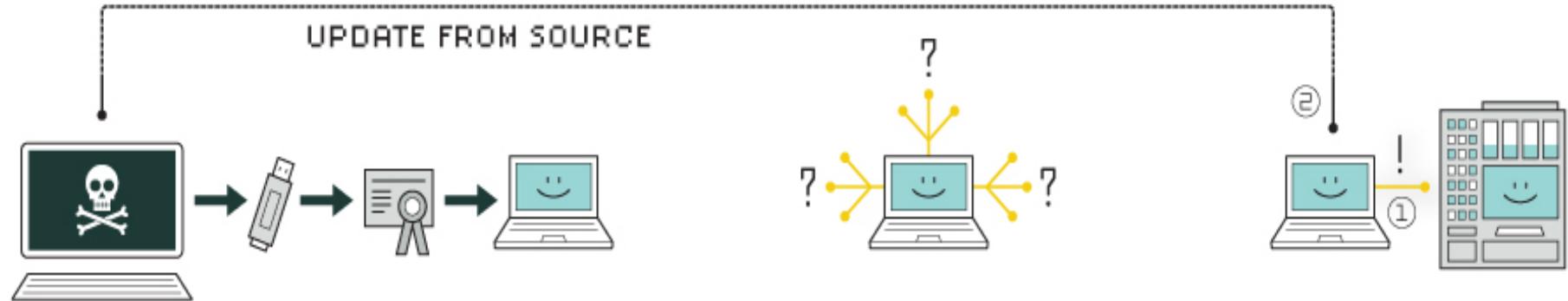
- Based on the obtained knowledge from the historical data, a risk mitigation strategy has to be formed.

# Vulnerabilities and exploits

- Top Deadly vulnerabilities
  - Buffer overflow/overrun
  - Integer overflow
  - Format string attack
  - Command injection/ Shell code injection
  - Failing to handle errors
  - Cross site scripting
  - Failing to Protect Network Traffic
  - Improper/ no use of SSL and TLS
  - Use of Weak Password-Based Systems
  - Improper File Access and access control
  - Information Leakage
  - Failing to Store and Protect Data Securely

# Attack patterns

## Stuxnet



### 1. infection

Stuxnet enters a system via a USB stick and proceeds to infect all machines running Microsoft Windows. By brandishing a digital certificate that seems to show that it comes from a reliable company, the worm is able to evade automated-detection systems.

### 2. search

Stuxnet then checks whether a given machine is part of the targeted industrial control system made by Siemens. Such systems are deployed in Iran to run high-speed centrifuges that help to enrich nuclear fuel.

### 3. update

If the system isn't a target, Stuxnet does nothing; if it is, the worm attempts to access the Internet and download a more recent version of itself.



### 4. compromise

The worm then compromises the target system's logic controllers, exploiting "zero day" vulnerabilities—software weaknesses that haven't been identified by security experts.



### 5. control

In the beginning, Stuxnet spies on the operations of the targeted system. Then it uses the information it has gathered to take control of the centrifuges, making them spin themselves to failure.



### 6. deceive and destroy

Meanwhile, it provides false feedback to outside controllers, ensuring that they won't know what's going wrong until it's too late to do anything about it.

# Open Source vs Closed Source

- **Open source software** refers to the computer software whose source code is open means the general public can access and use.
- **Closed source software** refers to the computer software whose source code is closed means public is not given access to the source code.
- **Link:** <https://www.geeksforgeeks.org/difference-between-open-source-software-and-closed-source-software/>

# Design Principles

- Least Privilege
- Fail-Safe Defaults
- Economy of Mechanism
- Complete Mediation
- Open Design
- Separation Privilege
- Least Common Mechanism
- Psychological Acceptability
- Defense in Depth

# Least privilege

- The principle of least privilege (PoLP) refers to an information security concept in which a user is given the **minimum levels of access** – or permissions – needed to perform his/her job functions.
- The principle of least privilege extends beyond human access. The model can be applied to applications, systems or connected devices that require privileges or permissions to perform a required task.

# Fail-Safe Defaults

- This principle means that the default situation is **lack of access**, and the protection scheme identifies conditions under which access is permitted.
- The alternative, in which mechanisms attempt to identify conditions under which **access should be refused**, presents the wrong psychological base for secure system design.

# Economy of Mechanism

- The principle of economy of mechanism states that security mechanisms should be as simple as possible.
- If the design and implementation is simple, fewer possibilities exist for errors.
- The checking and testing process is less complex, because fewer components and cases need to be tested.

# Complete Mediation

- The principle of **complete mediation** requires that all accesses to objects be checked to ensure they are allowed.
- Whenever a subject attempts to read an object, the operating system should mediate the action.
- First, it determines if the subject can read the object. If so, it provides the resources for the read to occur.

# Open Design

- The Open Design Principle is a concept that the security of a system and its algorithms **should not be dependent on secrecy of its design or implementation.**
- If the strength of a program's security depends on the ignorance of user, a knowledgeable user can defeat the security mechanism.

# Separation Privilege

- The principle of separation of privileges states that a system **should not grant permission based on a single condition.**
- For example, company cheques for over \$75,000 must be signed by two officers of the company. If either does not sign, the cheque is not valid. The two conditions are the signatures of both officers.

# Least Common Mechanism

- The Least Common Mechanism design principle declares that **mechanisms used to access resources should not be shared.**
- **Example:** A Web site provides e-commerce services for a major company. Attackers flood the site with messages and tie up the e-commerce services. Legitimate customers are unable to access the Web site, as a result of which they take their business elsewhere.

# Psychological acceptability

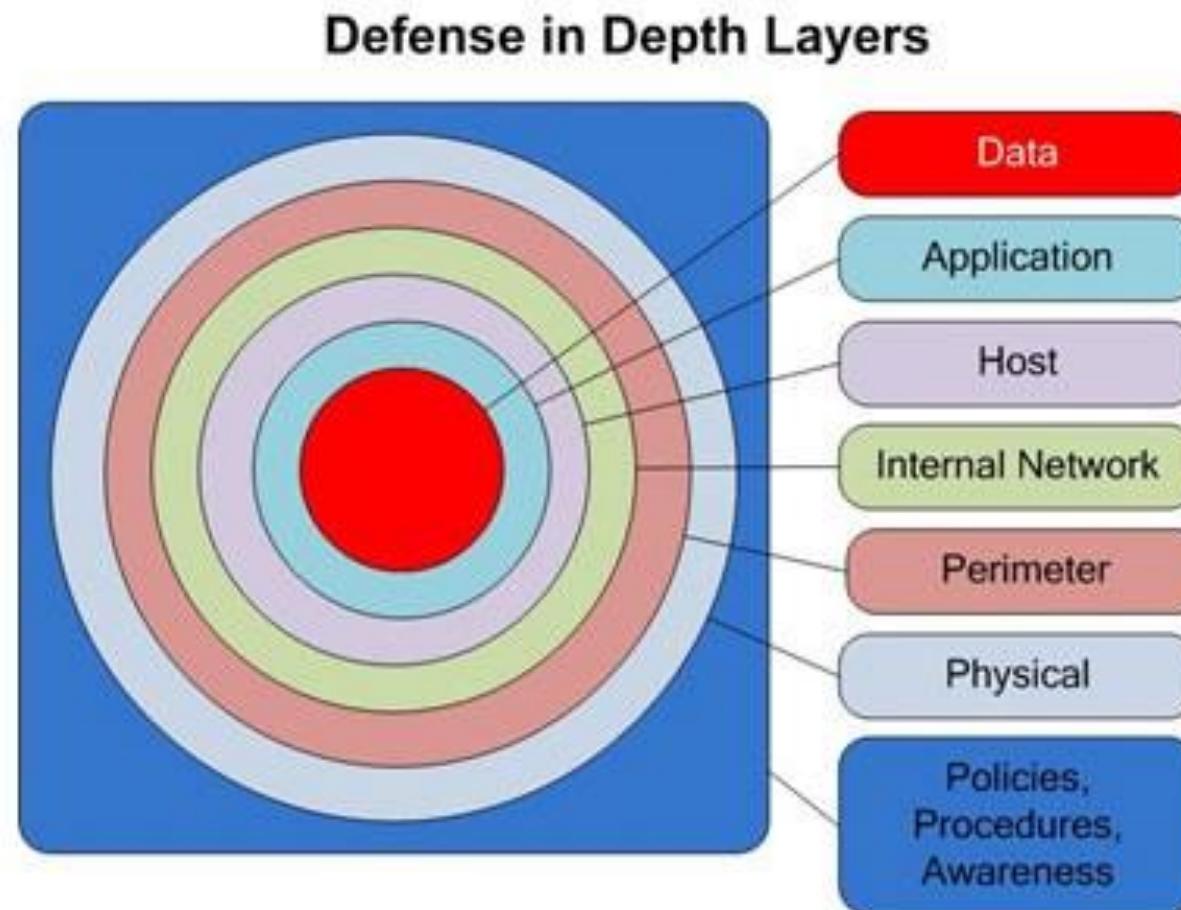
- The principle of **psychological acceptability** states that security mechanisms should not make the resource more difficult to access than if the security mechanisms were not present.
- If security-related software is too complicated to configure, system administrators may unintentionally set up the software in a nonsecure manner.
- Similarly, security-related user programs must be easy to use and must output understandable messages

# Defense in Depth

- Defense in Depth (DiD) is an approach to cybersecurity in which a series of defensive mechanisms are layered in order to protect valuable data and information.
- **Link:**

<https://www.imperva.com/learn/application-security/defense-in-depth/>

# Defense in Depth



# Authentication vs Authorization vs Access Control

- In **authentication** process, identities of the users are verified. Most of the time this verification process includes a username and a password but other methods such as PIN number, fingerprint scan, smart card and such are adapted as well.
- In **authorization** process, it is established if the user (who is already authenticated) is allowed to have access to a resource. In other words, authorization determines what a user is and is not permitted to do.
- In the process of **access control**, the required security for a particular resource is enforced. Once we establish who the user is and what they can access to, we need to actively prevent that user from accessing anything they should not.

<https://www.logsign.com/blog/what-are-appropriate-authentication-authorization-and-access-control-technologies/>

# MODULE 2

## Vulnerabilities

# Bit, nibble, Byte

What is a bit?

What is a nibble?

What is a byte?

What is the difference between b and B?

What is a word in computing?

- A **word** is the **data type (size)** that a processor naturally handles.
- Word = 2 bytes, Dword = 4 bytes, Qword= 8 bytes

UNIT	ABBREVIATION	STORAGE
Bit	Ь	Binary Digit, Single 1 or 0
Nibble	-	4 bits
Byte/Octet	B	8 bits
Kilobyte	KB	1024 bytes
Megabyte	MB	1024 KB
Gigabyte	GB	1024 MB
Terabyte	TB	1024 GB
Petabyte	PB	1024 TB
Exabyte	EB	1024 PB
Zettabyte	ZB	1024 EB
Yottabyte	YB	1024 ZB

Storage units ([www.byte-notes.com](http://www.byte-notes.com))

# Address space

- An **address space** is a **range** of valid **addresses** in memory that **are** available for a program or process.
- That is, it is the memory that a program or process can access.
- The memory can be either physical or virtual and is used for executing instructions and storing data.

# How many bytes is a memory address?

- It's the size of a machine word
  - A word is the natural unit of data used by a particular processor/architectural design.
  - Size of word is **2 bytes**
- **Machine word** is the amount of memory, CPU uses to hold numbers (Memory addresses) (in RAM, cache or internal registers).
  - The architecture of any machine is defined by the Word size.
  - "Word size" refers to the number of bits processed by a computer's CPU in one go
- **32-bit CPU** uses **32 bits (4 bytes)** to hold numbers.
- Memory addresses are numbers too, so on a **32-bit CPU** the memory address consists of **32 bits**. Likewise 64-bit CPU.

# Why memory is needed?

- To get a process executed it must be first placed in the memory. Assigning space to a process in memory is called **memory allocation**.

# How memory is handled?

- Let us assume a variable of integer type with value 10.
  - Data structure used for allocating memory?
    - **Stack vs Heap** memory allocation
    - Link: <https://www.guru99.com/stack-vs-heap.html>
  - How memory gets allocated?
    - **Static vs Dynamic** memory allocation
    - Link:  
<https://www.geeksforgeeks.org/difference-between-static-and-dynamic-memory-allocation-in-c/>

# LSB vs MSB

- In binary terms, the **MSB** is the bit that has the greatest effect on the number, and it is the left-most bit.
- In binary terms, the **LSB** is the bit that has the least effect on the number, and it is the right-most bit.
  - Take the example below. For the binary 0011 0101, the value is 53.
  - If you flip the left-most bit from 0 to 1, you have 1011 0101, which gives you 181.
  - Flipping the LSB on the right will give you 0011 0100, which is 52.

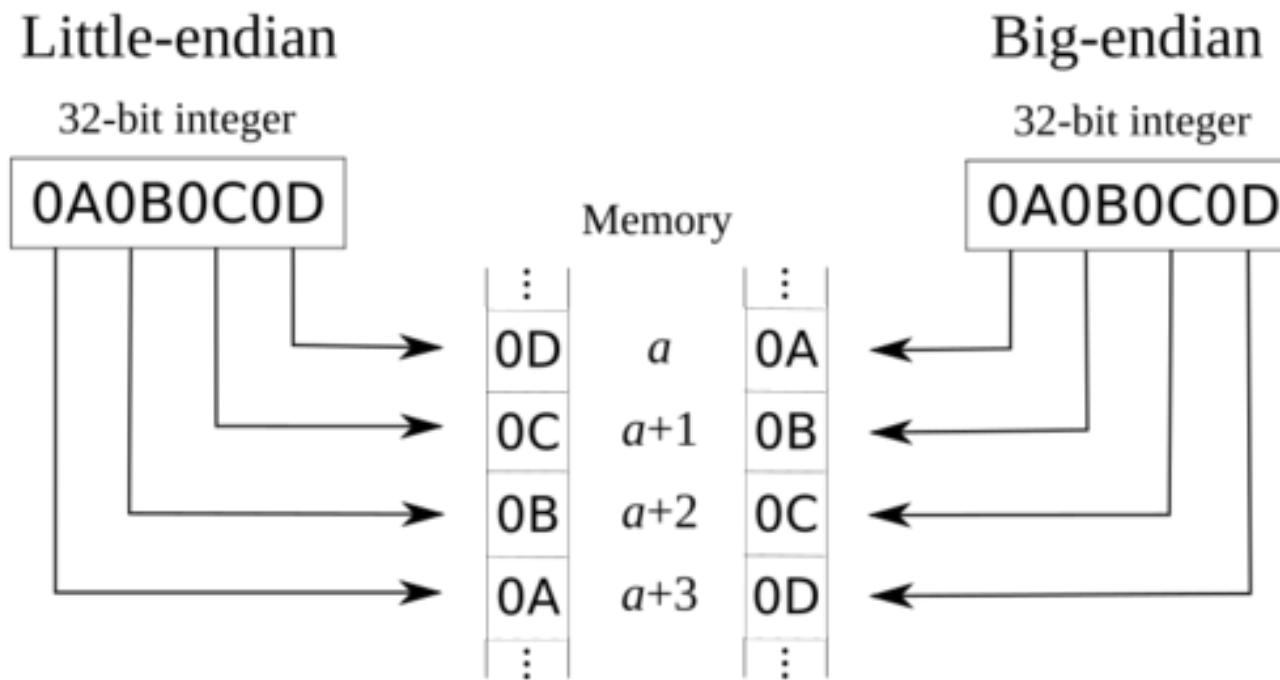
# Big-Endian vs Little-Endian

- **Endianness** is a term that describes the order in which a sequence of bytes are stored in computer memory.
- **Little Endian** – In this scheme, low-order byte is stored on the starting address (A) and high-order byte is stored on the next address (A + 1).
- **Big Endian** – In this scheme, high-order byte is stored on the starting address (A) and low-order byte is stored on the next address (A + 1).

**Link:** <https://www.freecodecamp.org/news/what-is-endianness-big-endian-vs-little-endian/>

# Big-Endian vs Little-Endian

- Suppose integer is stored as 4 bytes, then a variable  $x$  with value  $0x0A0B0C0D$  will be stored as following.

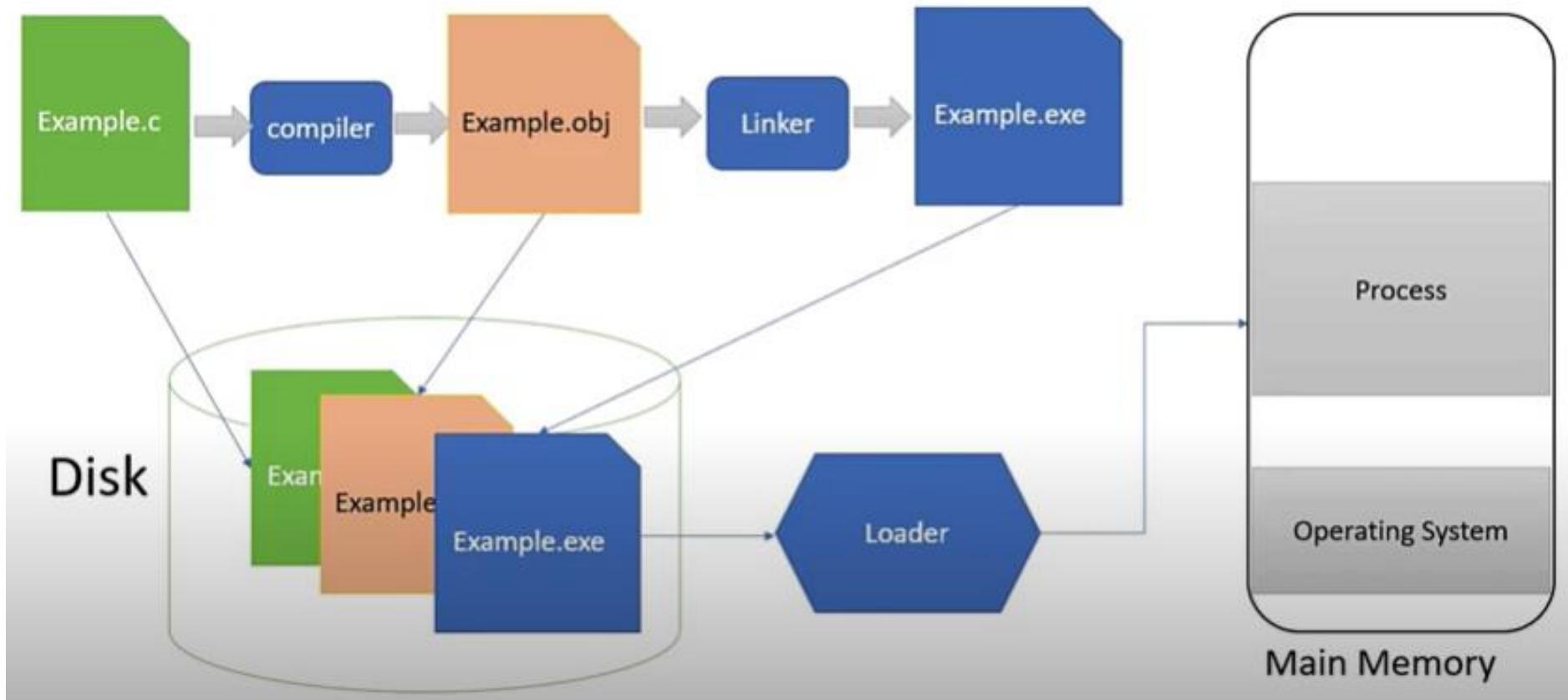


# Big-Endian vs Little-Endian

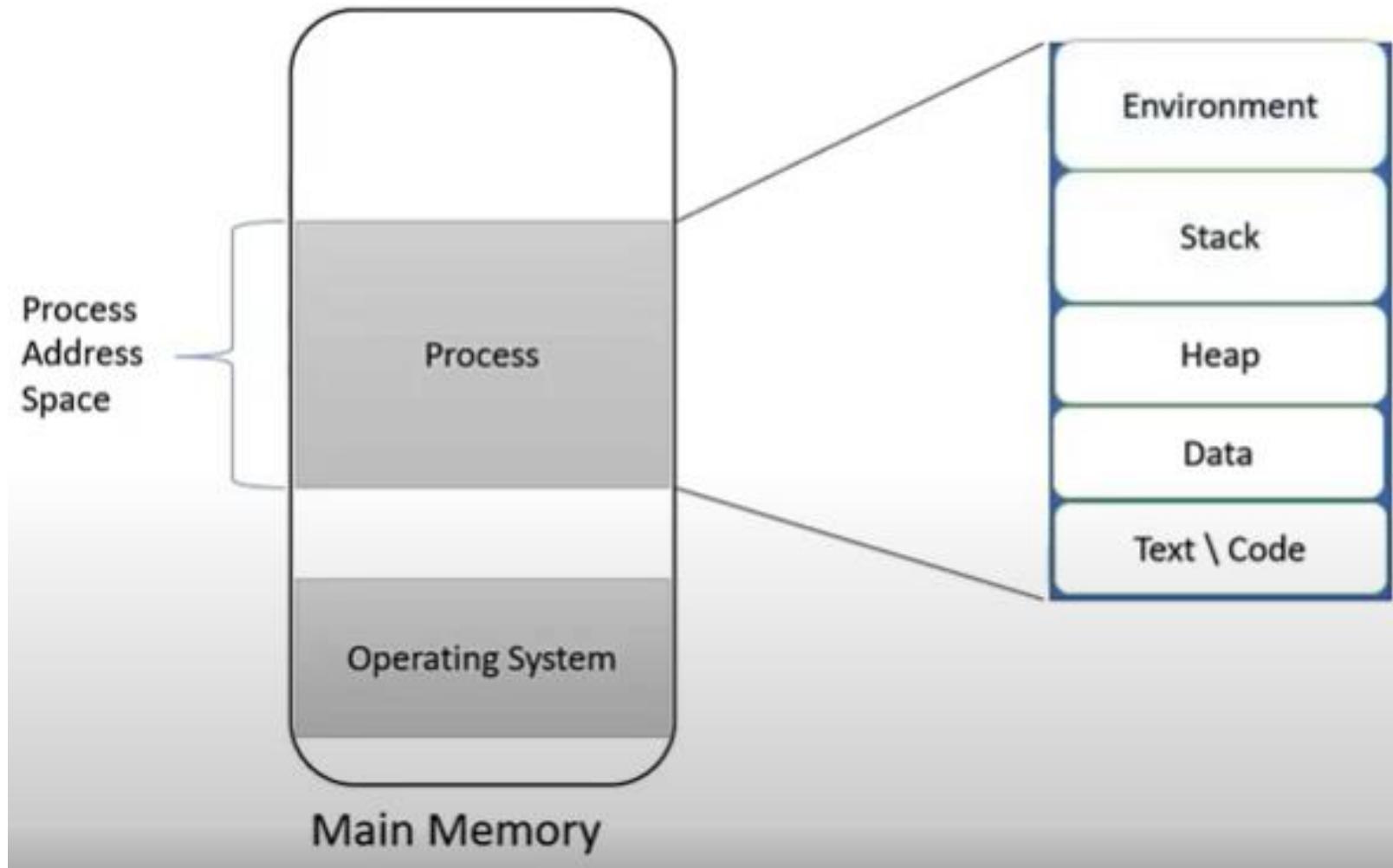
- An important distinction to make about endianness is that it only refers to **how values are stored in memory** and not how we deal with values; for example, 0x12345678 is still 0x12345678.
- If my computer reads bytes from **left to right**, and your computer reads from **right to left**, we're going to have issues when we need to communicate.
- **BE** is the dominant order in any network protocols, and is referred to as network order, for example. On the other hand, most PC's are **LE**.

# Memory Segments of C Program

## Program Compilation, Linking & Execution

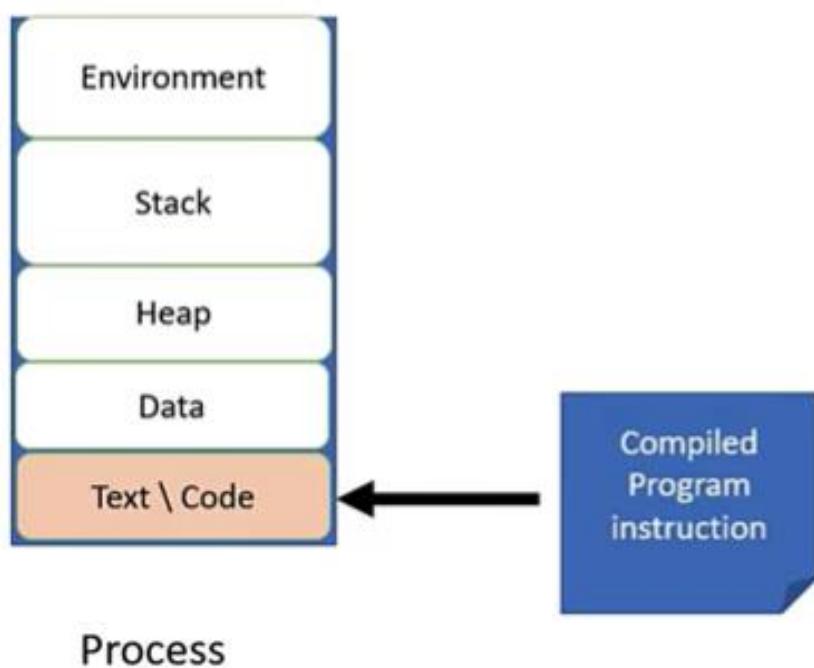


# Memory Segments of C Program



# Memory Segments of C Program

## Text \ Code Segment



It contains program instructions for execution.

It is read only section of program and cannot be changed during execution

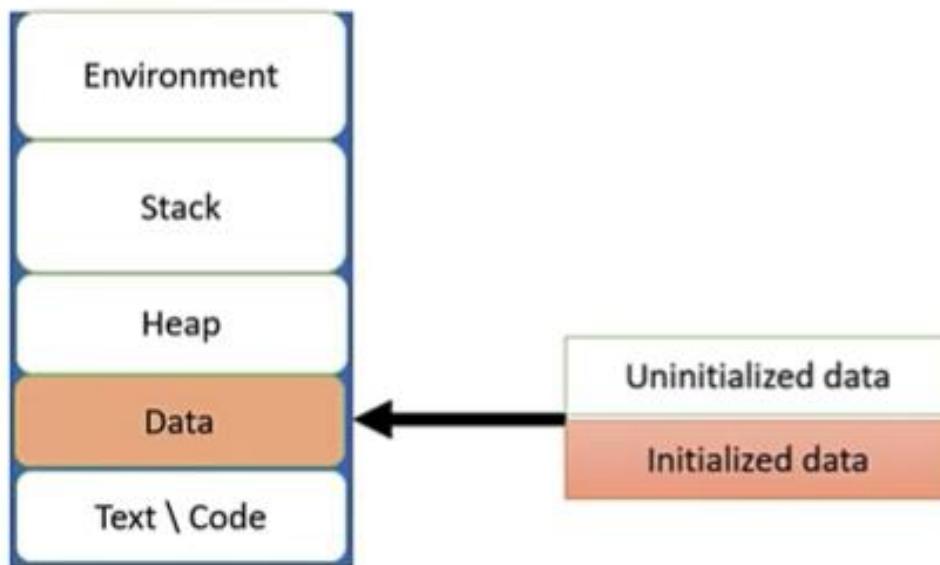
It is placed in lower memory address so its not overwritten by heap or stack

# Important Terms

- A **static variable** is a variable that has been allocated "statically", meaning that its lifetime (or "extent") is the entire run of the program.
- A **global variable** is a variable with global scope, meaning that it is visible (hence accessible) throughout the program, unless shadowed.

# Memory Segments of C Program

## Data Segment

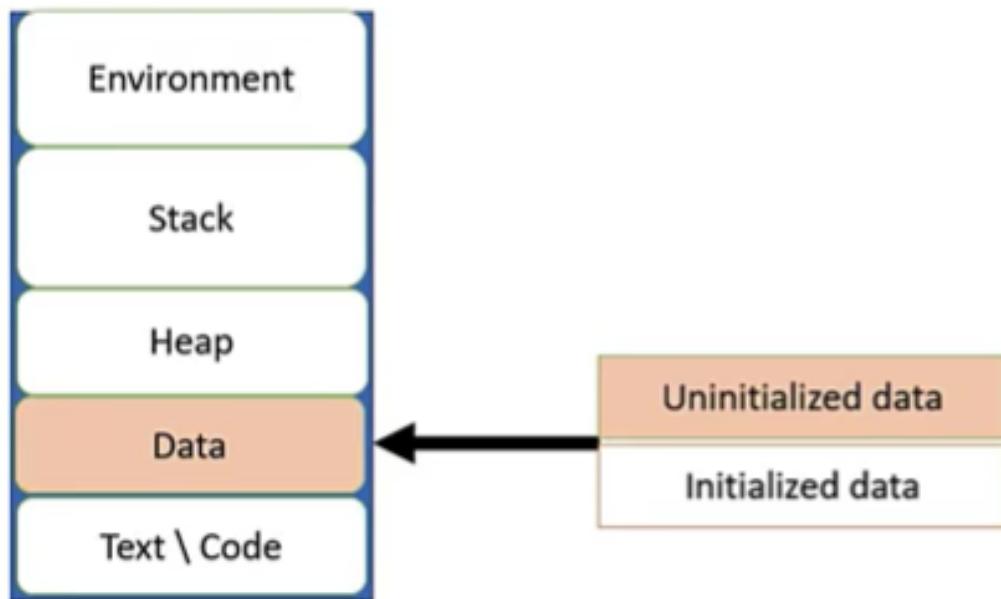


### Initialized data Section

- It contains global & static variables that are explicitly initialized
- Lifetime of variables in this section is throughout the execution of program
- Variables in this section can be changed.
- It is possible to save part of this section as read-only .

# Memory Segments of C Program

## Data Segment...

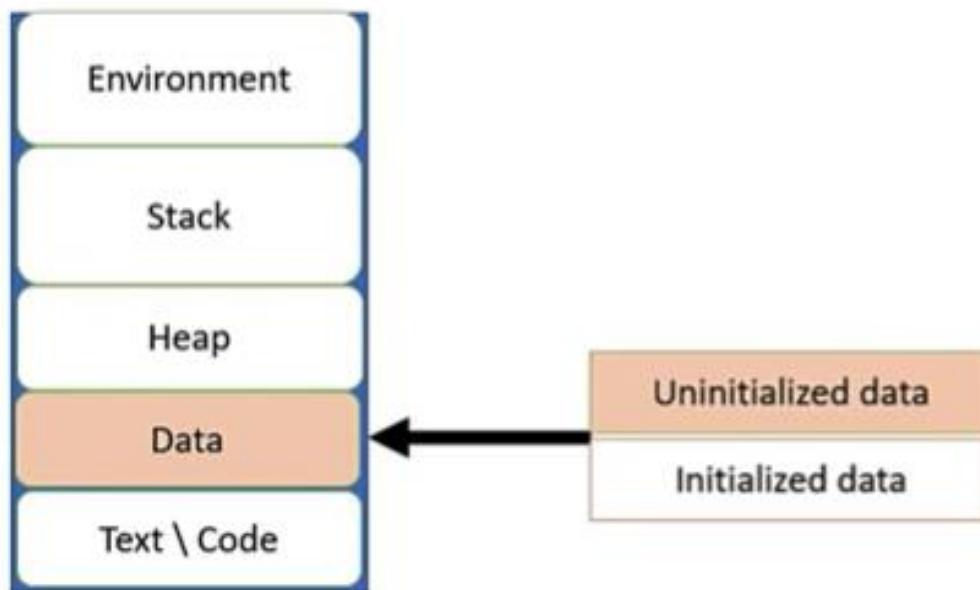


### Uninitialized data section

- It contains global and static data that is not initialized
- Lifetime of these variables are throughout the program execution

# Memory Segments of C Program

## Data Segment...

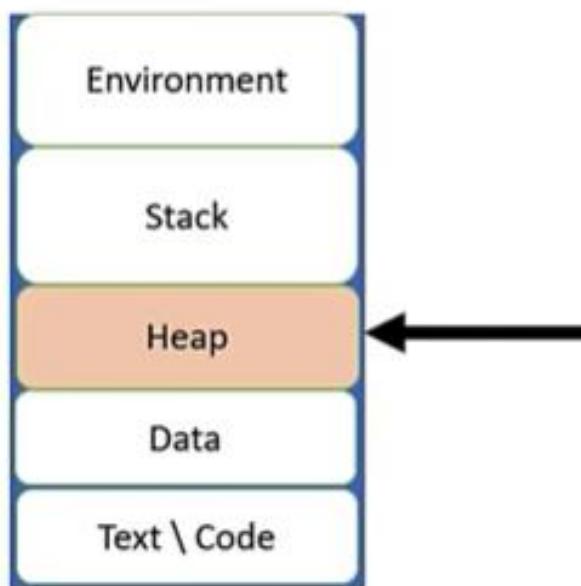


### Example Program

```
const char *str = "Hello World!";
int val;
int main()
{
    static int a = 10;
    static int b;
    return 0;
}
```

# Memory Segments of C Program

## Heap Segment

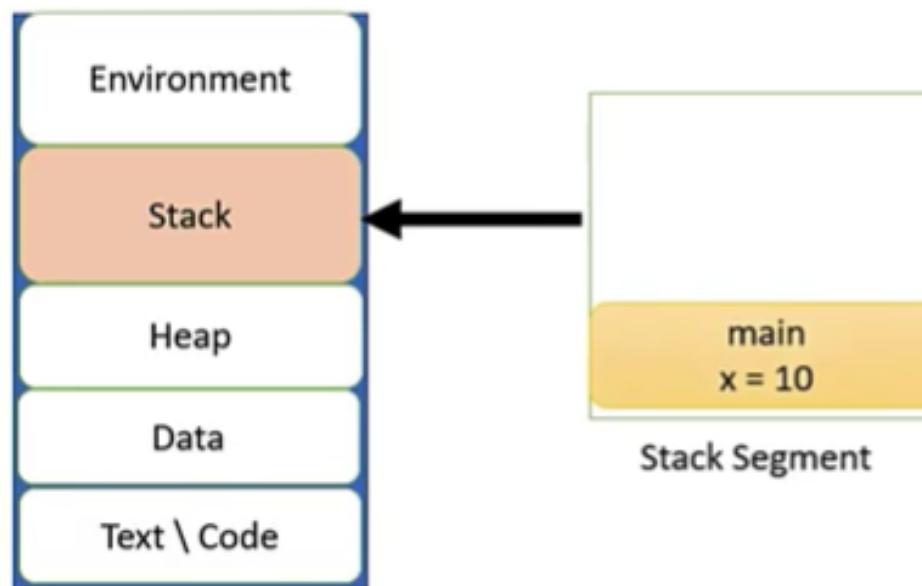


### Heap Segment

- Dynamic memory allocation that happens during program execution is stored in heap section
- Heap memory must be managed by the program statements
- `malloc`, `realloc`, `calloc`, `free` are the functions used to manage heap memory
- If memory not managed properly then it will lead to memory leak problems

# Memory Segments of C Program

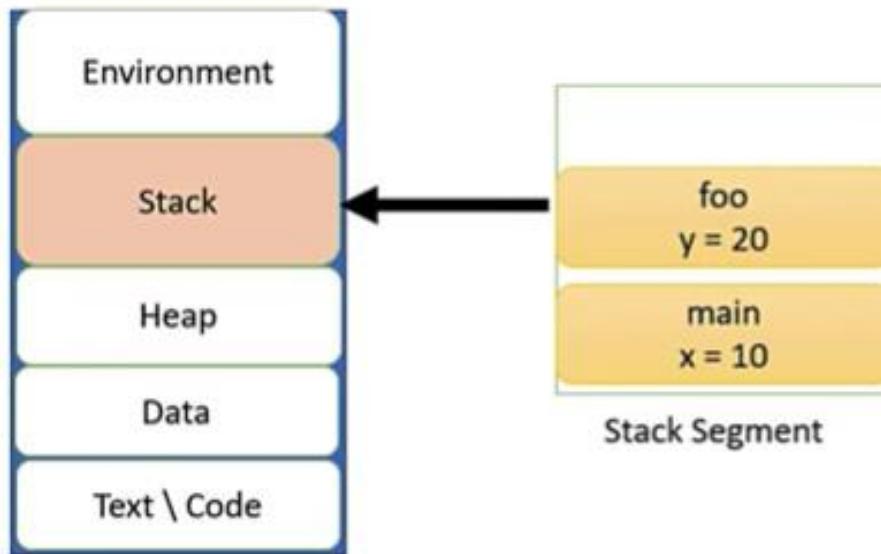
## Stack Segment



```
void foo();  
int main() { ←  
    int x = 10;  
    foo();  
    return 0;  
}  
void foo() {  
    int y = 20;  
}
```

# Memory Segments of C Program

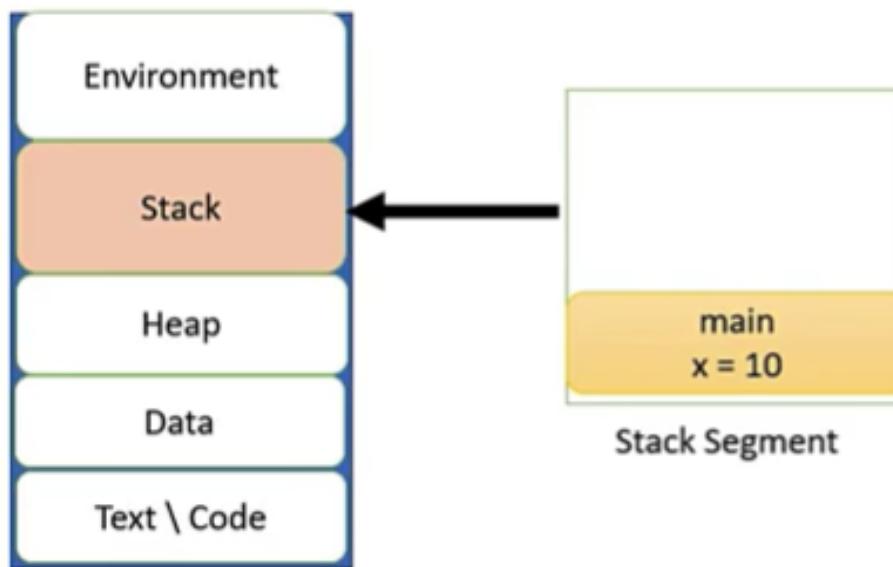
## Stack Segment...



```
void foo();  
  
int main() {  
  
    int x = 10;  
  
    foo();  
  
    return 0;  
}  
  
void foo() {  
  
    int y = 20;  
}
```

# Memory Segments of C Program

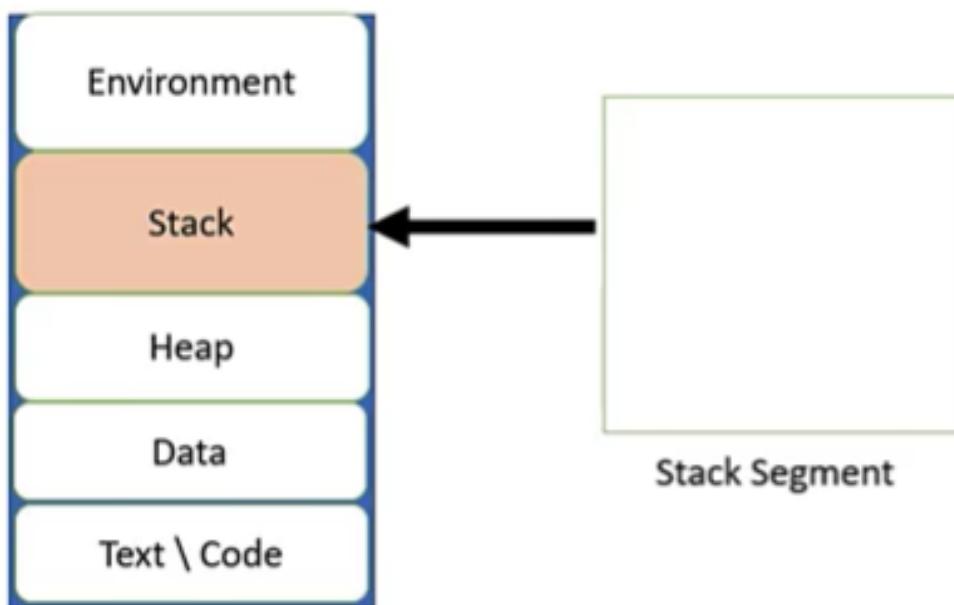
## Stack Segment...



```
void foo();  
  
int main() {  
  
    int x = 10;  
  
    foo(); ←  
  
    return 0;  
}  
  
void foo() {  
  
    int y = 20;  
}
```

# Memory Segments of C Program

## Stack Segment...

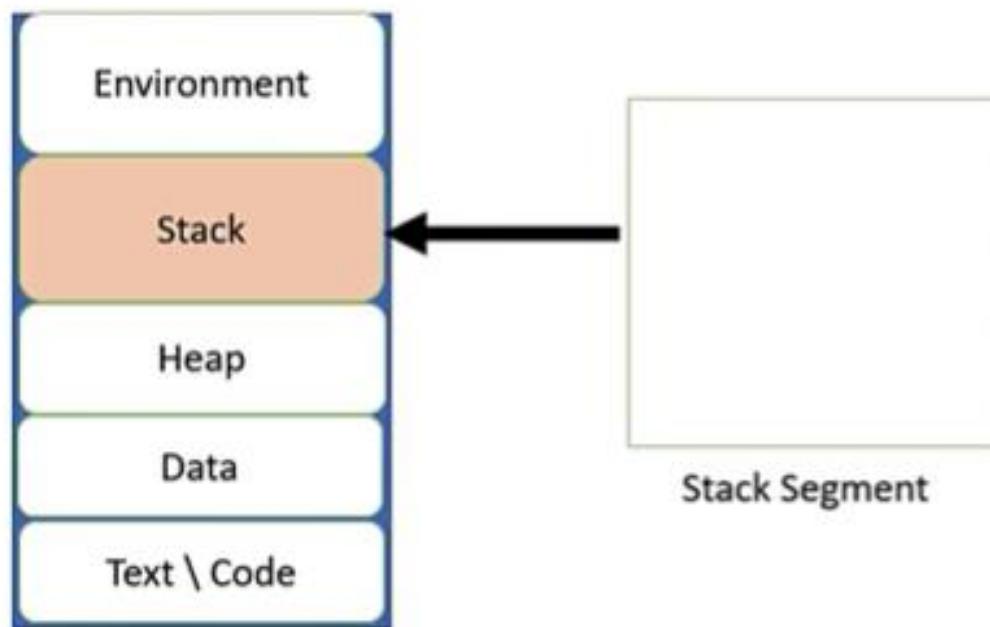


```
void foo();  
  
int main() {  
  
    int x = 10;  
  
    foo();  
  
    return 0;  
}
```

```
void foo() {  
  
    int y = 20;  
}
```

# Memory Segments of C Program

## Stack Segment...

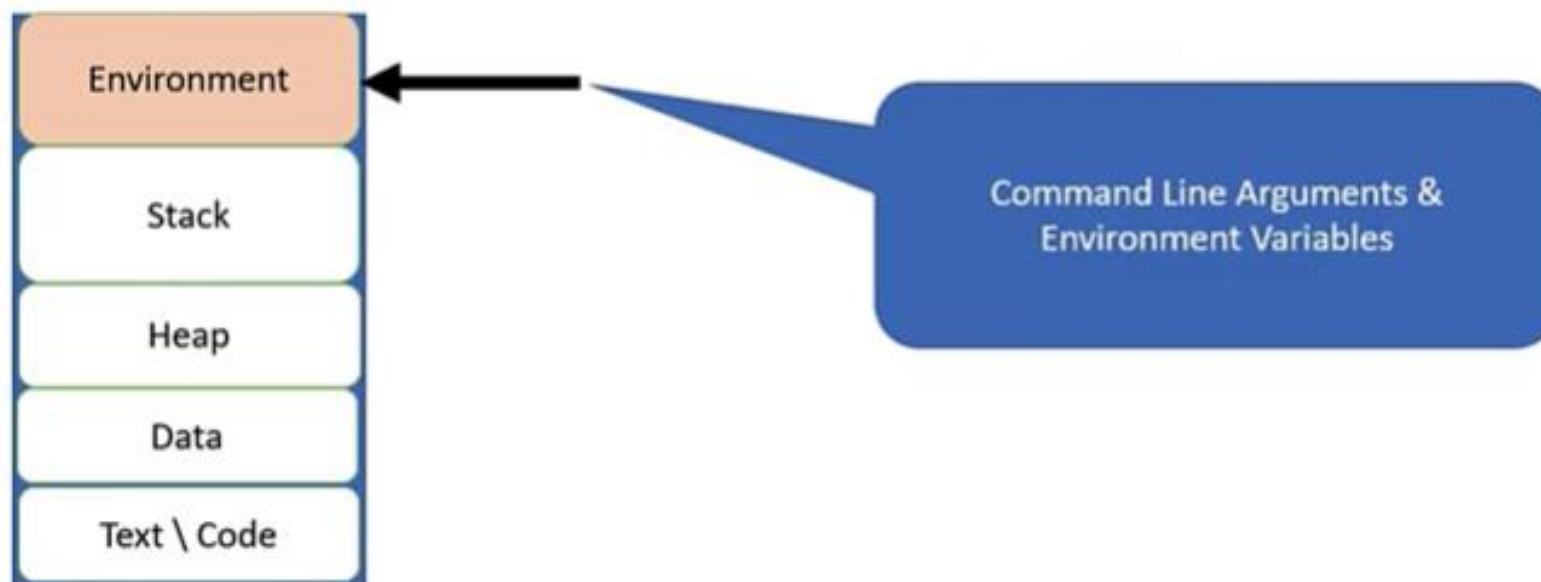


### Stack Section

- It contains activation records \ Stack Frames.
- Stack frames are created for each function call & pushed in stack
- Stack frame is popped from stack when function returns
- Local variables are stored on stack frame

# Memory Segments of C Program

## Command Line Args & Environment Variables



# What is a Buffer?

- Buffer is a **temporary storage area**, usually a block in memory, in which items are placed while waiting to be transferred from an input device or to an output device.
- It is mostly used for moving data between processes within a computer.
- Majority of buffers are implemented in software.

# An example of Buffering

- If you were to print a long document, you would not want your CPU waiting around asking your printer “Are you ready for another paragraph?”
- Instead, the CPU will fill a memory buffer with the document’s data, instruct the printer to print the buffer contents, and go back to its other business.

# Need of Buffering

- It helps in matching speed between two devices, between which the data is transmitted.
- It helps the devices with different data transfer size to get adapted to each other.
- It helps devices to manipulate data before sending or receiving.

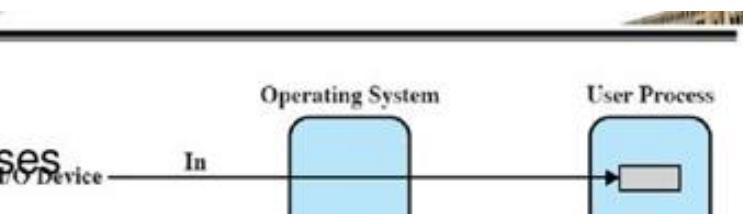
# Types of Buffering (based on capacity)

- **Zero Capacity** – This queue cannot keep any message waiting in it. Thus it has maximum length 0. For this, a sending process must be blocked until the receiving process receives the message. It is also known as no buffering.
- **Bounded Capacity** – This queue has finite length n. Thus it can have n messages waiting in it. If the queue is not full, new message can be placed in the queue, and a sending process is not blocked. It is also known as automatic buffering.
- **Unbounded Capacity** – This queue has infinite length. Thus any number of messages can wait in it. In such a system, a sending process is never blocked.

# I/O Buffering Techniques

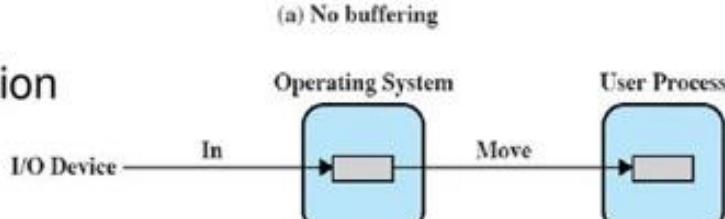
## □ No buffering

- Without a buffer, the OS directly accesses the device when it needs



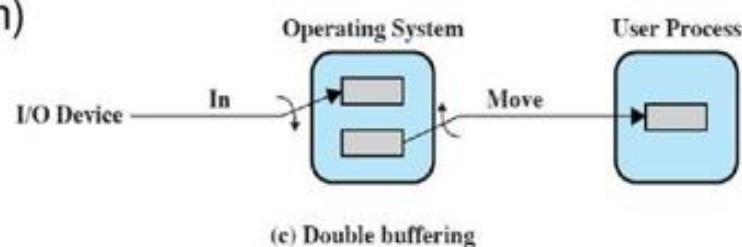
## □ Single buffering

- OS assigns a buffer in the system portion of main memory



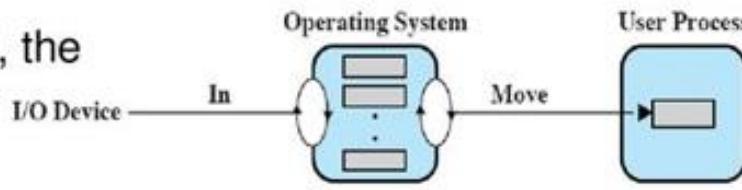
## □ Double buffering

- Use two system buffers
- A process can transfer data to (or from) one buffer while the operating system empties (or fills) the other buffer
- Also known as buffer swapping



## □ Circular buffering

- When more than two buffers are used, the collection of buffers is a circular buffer
- Each individual buffer is one unit in a circular buffer



# Buffering vs Caching

- Buffer memory is used to cope up with the different speed between sender and receiver of the data stream whereas, the cache is a memory which stores the data so that access speed can be fastened for repeatedly used data.
- Buffer always carry the original data to be sent to the receiver. However, cache carries the copy of original data.

# Buffer Overflow Vulnerability

- A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer.
- As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations.
- Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code.

# Buffer Overflow Attack Example

```
char buf1[5];
char buf2[10] = "A1B2C3D4E5";
strcpy(buf1,buf2);
```

- The above 3 lines of code are compiled without any error by the C compiler as there are no syntactical errors.
- But logically we are copying a string of 10 chars into a buffer which can hold only 5 chars. This might be a small typing error on the programmer's side, but results in an attack which can overwrite the data which might have been stored in the memory location next to the space allocated for buf1.

# Types of Buffer Overflow Attacks

## **Stack-based Buffer Overflow**

- The attacker takes advantage of the stack, a part of the memory reserved for the program to store data or addresses. The attacker then partially crashes the stack and forces the program execution to start from a return address of a malicious program address which is actually written by the attacker.
- Link: <https://www.thegeekstuff.com/2013/06/buffer-overflow/>

## **Heap-based Buffer Overflow**

- In Heap-based attack the attacker floods the memory space which is actually reserved for the program. This attacks is not exactly easy as it feels, hence the number of attacks with respect to the heap are very rare.

# What Programming Languages are More Vulnerable?

- C and C++ are two languages that are highly susceptible to buffer overflow attacks, as they don't have built-in safeguards against overwriting or accessing data in their memory. Mac OSX, Windows, and Linux all use code written in C and C++.
- Languages such as PERL, Java, JavaScript, and C# use built-in safety mechanisms that minimize the likelihood of buffer overflow.

# How to protect yourself from buffer overflow attacks?

There are a few options to protect yourself from buffer overflow attacks:

- staying up to date with vendor-issued patches and software updates;
- enabling runtime protections in OSes;
- using address space layout randomization;
- marking areas of memory as either executable or nonexecutable with Data Execution Prevention;
- manually testing for buffer overflows; and
- using a programming language that doesn't allow buffer overflow attacks -- such as Java, Python or .NET -- when possible.

# Important Terms

- **Address space layout randomization** (ASLR) is a memory-protection process for operating systems (OSes) that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory.
- **Data execution prevention** (DEP) is a security feature within operating system that prevents applications from executing code from a non-executable memory location.

# Buffer overflow

- A buffer overflow occurs when the data that is written into the buffer exceeds the allocated space and results in the overwriting of adjacent memory locations.
- Security attacks using buffer overflow are fairly common and most of them seek to modify data in the memory, gain access to confidential data and many more similar exploits.

# Buffer overflow

```
32  
33 if(strcmp(username,"root")==0 && strcmp(password,"pass")==0) {  
34     auth = true;  
35 }  
36 if(auth) {  
37     cout<<"Access Granted" << endl;  
38 }  
39 else {  
40     cout<<"Wrong username or password" << endl;  
41 }  
42 return 0;  
43 }  
44 }
```

```
buffer_overflow.cpp  x  
1 #include <iostream>  
2 #include <string.h>  
3 using namespace std;  
4  
5 int main() {  
6     bool auth = false;  
7     char username[10];  
8     char password[10];  
9  
10    cout<<"Before accepting values" << endl;  
11    cout<<"username addr: "<<&username << endl;  
12    cout<<"username value: "<<username << endl;  
13    cout<<"password addr: "<<&password << endl;  
14    cout<<"password value: "<<password << endl;  
15    cout<<"auth addr: "<<&auth << endl;  
16    cout<<"auth value: "<<auth << endl;  
17  
18    cout<<"Enter Username :";  
19    cin>>username;  
20    cout<<"\nAfter accepting username" << endl;  
21    cout<<"username value: "<<username << endl;  
22    cout<<"password value: "<<password << endl;  
23    cout<<"auth value: "<<auth << endl;  
24  
25    cout<<"Enter Password :";  
26    cin>>password;  
27  
28    cout<<"\nAfter accepting both username and password" << endl;  
29    cout<<"username value: "<<username << endl;  
30    cout<<"password value: "<<password << endl;  
31    cout<<"auth value: "<<auth << endl;  
32 }
```

# Results – Validation : pass

```
File Edit View Search Terminal Help
root@itdsm:~# g++ -g buffer_overflow.cpp -o buffer_overflow
root@itdsm:~# ./buffer_overflow
Before accepting values
username addr: 0x7ffd3c1492b5
username value:
password addr: 0x7ffd3c1492ab
password value: 0IV
auth addr: 0x7ffd3c1492bf
auth value: 0
Enter Username :root

After accepting username
username value: root
password value: 0IV
auth value: 0
Enter Password :pass

After accepting both username and password
username value: root
password value: pass
auth value: 0
Access Granted
```

# Results – Validation : Fail

```
root@itdsm:~# ./buffer_overflow
Before accepting values
username addr: 0x7ffeaf298ec5
username value:
password addr: 0x7ffeaf298ebb
password value: \0
auth addr: 0x7ffeaf298ecf
auth value: 0
Enter Username :random@123

After accepting username
username value: random@123
password value: \0
auth value: 0
Enter Password :morerandom

After accepting both username and password
username value:
password value: morerandom
auth value: 0
Wrong username or password
```

# Results – Validation : Overflow

```
root@itdsm:~# ./buffer_overflow
Before accepting values
username addr: 0x7ffd0ee2c85
username value:
password addr: 0x7ffd0ee2c7b
password value: !V
auth addr: 0x7ffd0ee2c8f
Enter Username :averylongrandomstring

After accepting username
username value: averylongrandomstring
password value: !V
Enter Password :anotheraverylongrandomstring

After accepting both username and password
username value: ylongrandomstring
password value: anotheraverylongrandomstring
Access Granted
Segmentation fault
```

# Results – Validation : Overflow, Example

```
root@itdsm:~# ./buffer_overflow
Before accepting values
username addr: 0x7fff218852d5
username value:
password addr: 0x7fff218852cb
password value: 00U
auth addr: 0x7fff218852df
auth value: 0
Enter Username :uuuuuuuuuuuuuuuuuuuuuuuuuu
After accepting username
username value: uuuuuuuuuuuuuuuuuuuuuuuuu
password value: 00U
auth value: 117
Enter Password :pppppppppppppppppppppppppppp
After accepting both username and password
username value: ppppppppppppppppppppppppppp
password value: ppppppppppppppppppppppppppp
auth value: 112
Access Granted
Segmentation fault
```



# Integer overflow

- An **Integer Overflow** is the condition that occurs when the result of an arithmetic operation, such as multiplication or addition, exceeds the maximum size of the **integer** type used to store it.
- Overflow flag:
  - The **overflow** flag (sometime called V flag) is usually a single bit in a system status register used to indicate when an arithmetic **overflow** has occurred in an operation, indicating that the **signed two's-complement** result would not fit in the number of bits used for the operation

# Two's complement

- This is the most common representation used nowadays for negative integers because it is the **easiest to work with for computers**, but it is also the **hardest to understand for humans**.
- For example **11111111** stands for **-0** in **one's complement** and for **-1** in **two's complement**, and similarly for **10000000** (-127 vs -128).
- In 8-bit two's complement, the value 8 is represented as 00001000 and -8 as 11111000.

# Integer overflow

```
#include <stdint.h>
#include <stdio.h>

void show_info(double d, int16_t n);

int main(){
    double d1 = 32767.0;
    double d2 = 34900.0;
    int16_t n = 0;
    |
    show_info(d1, n);
    show_info(d2, n);

    return(0);
}

void show_info(double d, int16_t n){
    n = (int16_t)d;

    printf("d = %f, n = %d \n", d, n);
}
```

# Integer Overflow

```
integer_overflow.c  x

1 #include<stdio.h>
2
3 int main() {
4     char n1 = 100, n2 = 200, sum;
5     sum = n1+n2;
6     unsigned char un1 = 100, un2 = 200, usum;
7     usum = un1+un2;
8     signed char sn1 = 100, sn2 = 200, ssum;
9     ssum = sn1+sn2;
10    printf("n1: %d n2: %d sum: %d \n", n1,n2,sum);
11    printf("un1: %d un2: %d usum: %d \n", un1,un2,usum);
12    printf("sn1: %d sn2: %d ssum: %d \n", sn1,sn2,ssum);
13
14    return 0;
15 }
16
```

You know Integer overflow cannot be directly exploitable (like buffer overflow)?

- An integer overflow is a different case - you can't exploit the integer overflow to add arbitrary code, and force a change in the flow of an application.
- However, it is possible to overflow an integer to crash an application.

# Format strings vulnerabilities

- Format string exploits can be used to crash a program or to execute harmful code
- The Format String exploit occurs when the submitted data of an input string is evaluated as a command by the application.
- The attacker could execute code, read the stack, or cause a segmentation fault in the running application, causing new behaviors that could compromise the security or the stability of the system.

# Format string

- A Format String is an ASCIIZ string that contains **text and format parameters**.
  - **ASCIIZ** means that the string is terminated by the \0 (ASCII code 0) NUL character.
  - A simple formatted string
    - `printf("my name is:%s\n","sibi");`
  - Output will be sibi

# Format string: Two utility

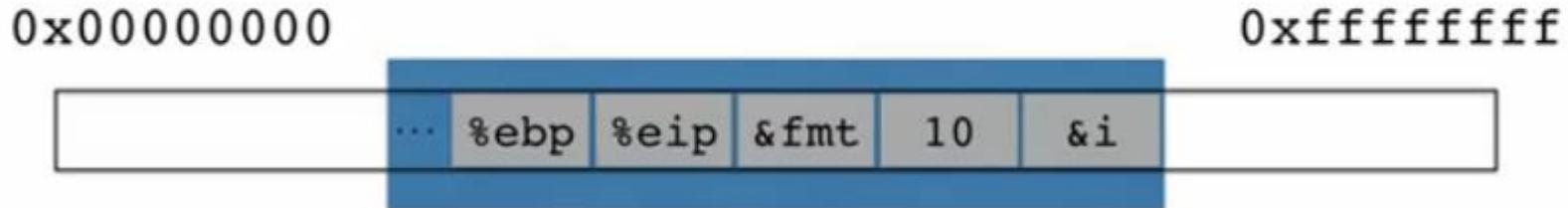
- Format function
  - Fprint, printf, sprint...
- Format specifiers
  - %p,%d,%c,%u,%x,%s.... (some are passed as values and some are passed as reference)
  - For example: %d are passed as value whereas %p, %s is passed as reference
  - Note:

# Format specifiers

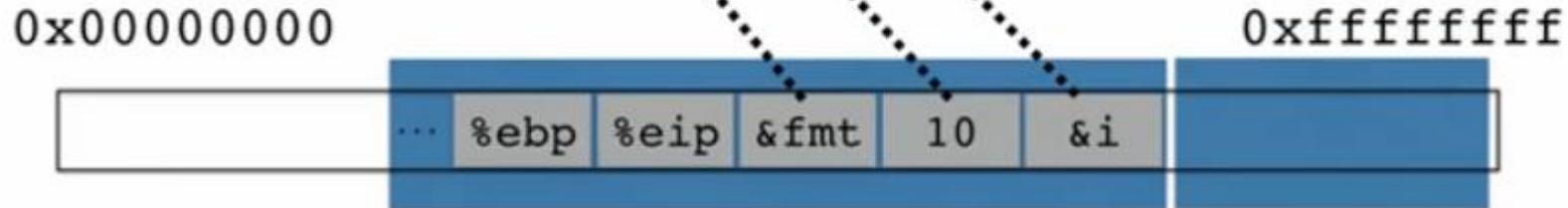
- Format string uses format specifiers to indicate how our data should be formatted.
- `printf("my name is:%s and time is : %d \n", a, b);`

# Format string: Printf- Stack

```
int i = 10;  
printf("%d %p\n", i, &i);
```



```
int i = 10;  
printf("%d %p\n", i, &i);
```



What happens to the stack when a format string is specified with no corresponding variable (format specifier) on stack??!!

- It will start to pop data off the stack from where the variables should have been located.
- For example
  - `printf("my name is: %s and time is : %d \n", a, b);`
  - This works fine
  - What about this?
  - `Printf(argv[1],b);`
- Note: The items the **program returns are values and addresses saved on the stack.**

# Note!!!

- Exploits corresponding to this vulnerability will not work on modern system due to NX (No execute), ASLR, and modern kernel security mechanisms.

# *Load order Attacks*

- The Load order attack is also known as DLL preloading attack or DLL hijacking.
- A Closed code operating system such as windows and its associated applications does not require the full path of dlls to get loaded in runtime.

When the entry path of dlls for an application is not defined.,

- Windows tries to locate the dll in possible ways as mentioned below:
- 1. Initially, **Windows searches for the particular dll in memory**, if it is already loaded in memory, the application makes use of the loaded dll.
- 2. Secondly, **Windows searches for the particular dll in knownDLL lists** (WindowsNT utility used to cache all common dlls). The list of commonly known dlls is available in the following registry entry:
  - "HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\SessionManager\KnownDLLs"
- 3. Finally, if the application requires to **load the dependent dll**, which is not listed in Step2 and Step 3, then windows tries to locate the particular dll by searching present application directory or pre-determined **System (64/32/16) directories** or directories listed in **SYSTEM PATH or Windows directory**.

# Set vs Setx

- **Set** used to set the current environment variables.
- **Setx** provides the only command-line or programmatic way to directly and permanently set system and user environment values.

16/32/64

- **System**
- **SysWOW64**
- **System32**

# How this Load order feature in windows is vulnerable? - Answer

- It is possible for an attacker to load the malicious dll with the name of legitimate dlls, making windows to load the malicious dll in memory for execution.

# *Search Path based attacks*

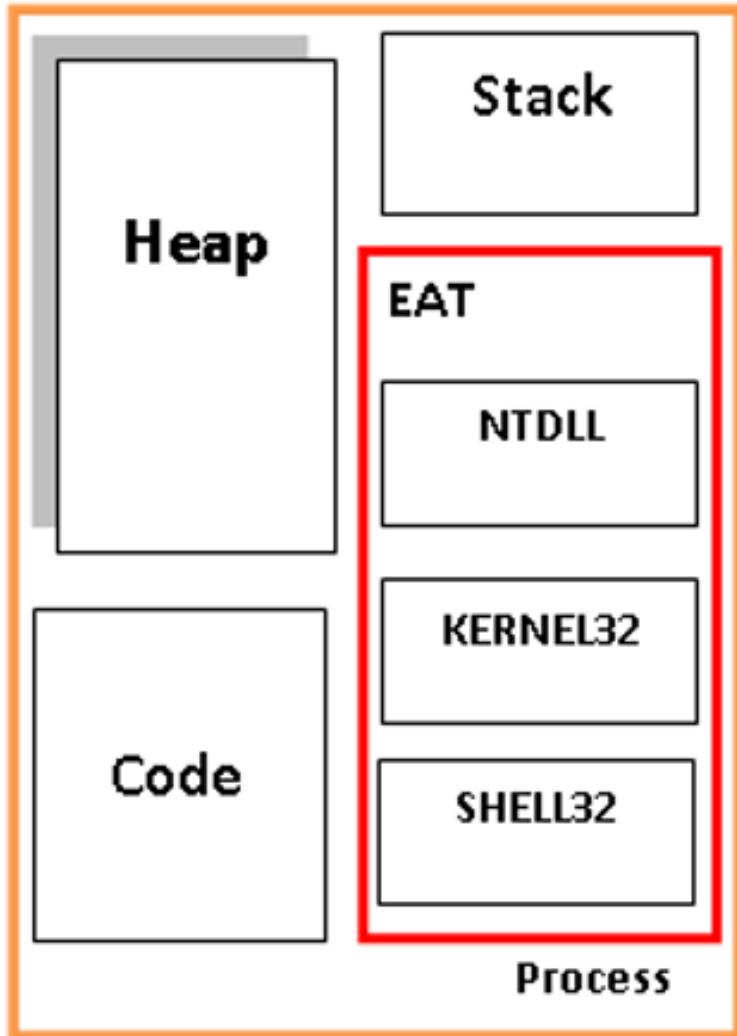
- In general, application uses **Search Path function** to locate a particular dll and loads the result returned by Search Path function in run time.
- There is a huge chance for locating and loading wrong dlls.
- Malwares and malware writers make use of this functionality to dynamically load malicious dlls.



# Search order and Load path

Parameter	Registry Entry	Value	Results Returned (Directory)	Technique used
REG_DWORD	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SessionManager\SafeProcessSearchMode	1	System Path	Search Path
REG_DWORD	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SessionManager\SafeProcessSearchMode	0	Present Working directory	Search Path
REG_DWORD	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SessionManager\SafedllSearchMode	0	Present Working directory	Load order
REG_DWORD	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SessionManager\SafedllSearchMode	1	Present Working directory	Load order

# Process on load – Memory and execution



- Kernel32.dll is the **32-bit dynamic link library** found in the **Windows operating system kernel**. It handles **memory management, input/output operations, and interrupts**.
  - When Windows boots up, kernel32.dll is loaded **into a protected memory space** so other applications do not take that space over.
- Ntdll.dll is mostly concerned with **system tasks** and it includes a number of **kernel-mode functions** which enables the "Windows Application Programming Interface (API)"
- Shell32.dll is a **dynamic link library** that **controls certain API functions of the Windows Shell**.

# Export Address Table

- **Any module under execution** keeps the addresses of the imported functions/variables in a table called Export Address table (EAT).
- The PE loader looks for this table to load an executable and its associated files.

# Code Injection

- **Code injection** is the exploitation of a **computer bug** that is caused by **processing invalid data**.
  - **Injection** is used by an attacker to introduce (or "inject") **code** into a vulnerable computer program and change the course of **execution**.



# **MODULE 3**

Security Validation

# Considered Input

- Input consideration completely depends on the security practices.
- **Input in services/applications** differ from each other based on their **security policies**
- **For example.** **Gmail, yahoo** and **Microsoft** uses their own input and its validation.

# Regular expression – windows os

OneDrive > VIT > Secure Coding >

Name	Status	Date modified	Type	Size	
New folder	✓	2/19/2019 9:12 AM	File folder		
Assignment	✓	12/29/2018 9:44 AM	Microsoft Word Doc...	17 KB	
Decipher	A file name can't contain any of the following characters: \\ : * ? " < >		12/13/2018 10:07 AM	Microsoft PowerPoint...	65 KB
Lab ex	✓	6/29/2018 4:11 PM	Microsoft Word Doc...	16 KB	
Lab experiments_Secure coding	✓	6/29/2018 3:55 PM	Microsoft Word Doc...	238 KB	
Realtime case study	✓	1/11/2019 3:49 PM	Microsoft Word Doc...	20 KB	
SC_CAT1_Security principles	✓	12/21/2018 3:47 PM	Microsoft PowerPoint...	3,665 KB	
SC_CAT1_Vulnerabilities	✓	1/12/2019 9:06 AM	Microsoft PowerPoint...	1,377 KB	
SC_CAT2_Application_hardening_techniques	✓	2/16/2019 10:06 AM	Microsoft PowerPoint...	2,358 KB	
SC_CAT2_Security Validation	⟳	2/19/2019 9:01 AM	Microsoft PowerPoint...	308 KB	
SC_lab	✓	2/14/2019 2:22 PM	Microsoft PowerPoint...	270 KB	
SC_model_question_paper	✓	1/18/2019 3:06 PM	Microsoft Word Doc...	58 KB	
Secure coding	✓	2/18/2019 12:55 PM	Microsoft Word Doc...	245 KB	
user_input	✓	6/21/2018 10:35 AM	CPP File	1 KB	

# Why Input validation?

- Input validation is performed to ensure only **properly formed data** is entering the **workflow in an information system**.
- Data from all **potentially trusted/untrusted sources** should be subject to **input validation**.
- Input validation can be implemented using **any programming technique** that allows **effective enforcement of syntactic and semantic correctness**.

# Input Validation

- Input validation strategy
  - Syntactical
  - Semantical
- **Syntactic** validation should **enforce correct syntax of structured fields** (e.g. date, currency symbol).
- **Semantic** validation should enforce **correctness** of the **values in the structured fields** (e.g. start date is before end date, price is within expected range).

# Examples of weak passwords

- Any word that can be found in a dictionary, in any language (e.g., airplane or aeroplano).
- A dictionary word with some letters simply replaced by numbers (e.g., a1rplan3 or aer0plan0).
- A repeated character or a series of characters (e.g., AAAAA or 12345).
- A keyboard series of characters (e.g., qwerty or poiuy).
- Personal information (e.g., birthdays, names of pets or friends, Social Security number, addresses).
- Anything that's written down and stored somewhere near your computer.

# Tips for keeping your password secure

- Change it regularly—once every three to six months.
- Change it if you have the slightest suspicion that the password has become known by a human or a machine.
- Never use it for other websites.
- Avoid typing it on computers that you do not trust; for example, in an Internet café.
- Never save it for a web form on a computer that you do not control or that is used by more than one person.
- Never tell it to anyone.
- Never write it down.

# Characteristics of strong passwords

- At least 8 characters—the more characters, the better
- A mixture of both uppercase and lowercase letters
- A mixture of letters and numbers
- Inclusion of at least one special character, e.g., ! @ # ? ]

# Weak Random Numbers

- Random number generation is important for lotteries, games and security.
- In cryptography, randomness is important because it removes any reasoning and therefore any predictability.
- An attacker is usually trying to attain information on a system, when this information is randomly generated there are no clues as to what it maybe and therefore no open opportunities to attack the system.
- However, computers are deterministic machines, and as such are unable to produce true randomness.

# Weak Random Numbers

- This means that numbers generated come from a finite pool and will ultimately at some point repeat and follow some pattern.
- All that would be needed for an attacker to attain the pseudo-random numbers used in encryption is the algorithm used for generating the numbers and the initial input passed to that algorithm (also called "seed").

Webpage: <https://www.commonlounge.com/discussion/481152258acb4003a5903d2fc1bc425f>

# Improper use of cryptography

**Q:** If encryption is so unbreakable, why do businesses and governments keep getting hacked?

**A:** The REAL problem with encryption:  
you're doing it wrong!

# Improper use of cryptography

- Assuming your developers are security experts
- Believing that regulatory compliance means you're secure
- Relying on cloud providers to secure your data
- Relying on low-level encryption
- Using the wrong cipher modes and algorithms
- Getting key management wrong

Website: <https://www.crypteron.com/blog/the-real-problem-with-encryption/#:~:text=Mistake%20%235%3A%20Using%20the%20wrong,Wikipedia%20list%20of%20cryptographic%20algorithms.&text=Using%20AES%20ECB%20mode%20for,the%20entire%20encryption%20process%20itself>

# Client side vs Server side validation

- In client-side validation method, all the input validations are carried out on the client side i.e., on the user's end.
- In server-side validation, all the input validations are carried out on the server side.
- Client-side validation is faster than server-side because the validation takes place on client side (on browser) and no communication needs to be established between client and server.
- Server-side validation is done on the web server. The server validates the input and sends the response back to the client.

# Client side vs Server side validation

- Client-side validation provides less security to your applications as anyone can bypass your client and enter some dangerous or malicious data which can crash your server.
- If you use server side validation, you are making your application more secure.

**NOTE:** One should always prefer server side validation as it is more secured and reliable and from performance point of view its effect is negligible.

# Blacklist vs whitelist validation

- One of the principal advantages of blacklisting lies in the simplicity of its principle: **You identify everything bad** that you don't want getting into or operating on your system, **exclude it from access**, then allow the free flow of everything else.
- Application whitelisting turns the blacklist logic on its head: You draw up a **list of acceptable entities** (software applications, email addresses, users, processes, devices, etc.) that are allowed access to a system or network, and block everything else. It's based on a "**zero trust**" principle which essentially denies all, and allows only what's necessary.

Webpage: <https://blog.finjan.com/blacklisting-vs-whitelisting-understanding-the-security-benefits-of-each/>

# Common String manipulation error

- Unbounded string copies
- Null-termination errors
- Truncation
- Off-by-one errors
- Improper data refinement

# Unbounded string copies

- Unbounded string copies occur when data is copied from an unbounded source to a fixed length character array.
- It is easy to make errors when copying and concatenating strings because **standard functions do not know the size of the destination buffer.**
- Webpage: <https://www.informit.com/articles/article.aspx?p=430402&seqNum=2>

# Off-by-one errors

- These types of errors often occur in C and other languages where arrays start numbering with 0 instead of one.
- Youtube Link: <https://www.youtube.com/watch?v=rSVD7u4z-9U>

# Null-termination errors

- '\0' will always be there to terminate strings placed in the destination buffer
- If there is **no null character** among the first n character of src, the string placed in dest will not be null-terminated.
- This may cause segmentation fault
- Finally, **non-terminated string in C/C++** is a **time-bomb** just waiting to **destroy code**.

# Truncation

- **Restriction in the number of bytes** are often recommended to mitigate against **buffer overflow vulnerabilities**.
- **Strings that exceed the specified limits** are **truncated**.
- Truncation results in a loss of data.

# MODULE 4

Application Hardening  
techniques

# Application Hardening

- Application hardening is the act of applying levels of security in order to protect **applications** from IP theft, misuse, vulnerability exploitation, tampering or even repackaging by people with ill intentions.

# Hardware enabled Security

Cyber-attacks are moving down the computing stack, traversing from software to hardware, threatening devices in homes, cars, businesses, networks, and cloud.

The legacy model of software protecting software can't keep up with advancing threats against digital security, safety, and privacy.

Intel builds hardware-enabled security capabilities directly into our silicon to help protect every layer of the compute stack (hardware, firmware, operating systems, applications, networks, and the cloud).

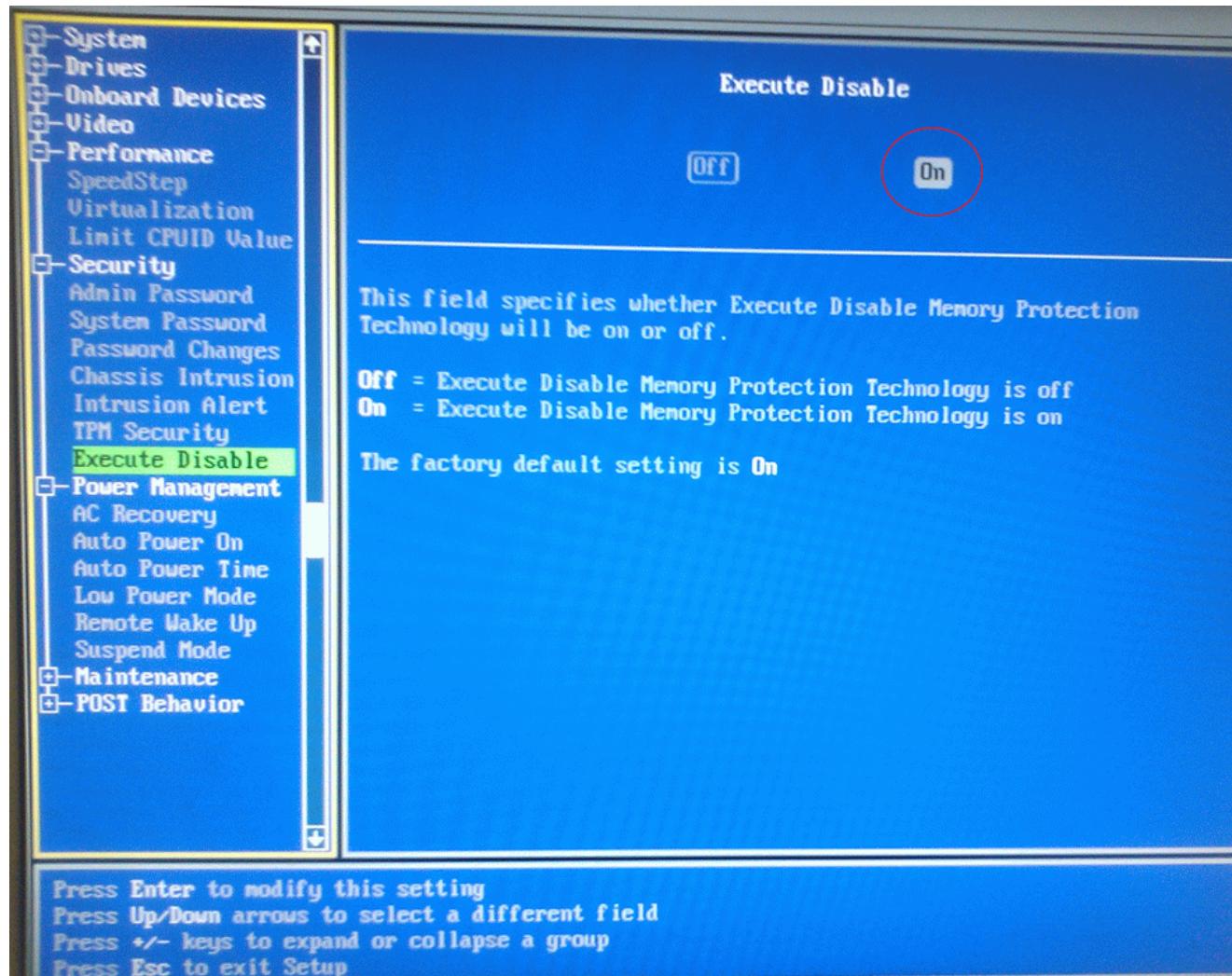
# Data Execution Prevention (DEP)

- DEP is a **security feature** that can **help prevent damage** to your computer from viruses and other security threats.

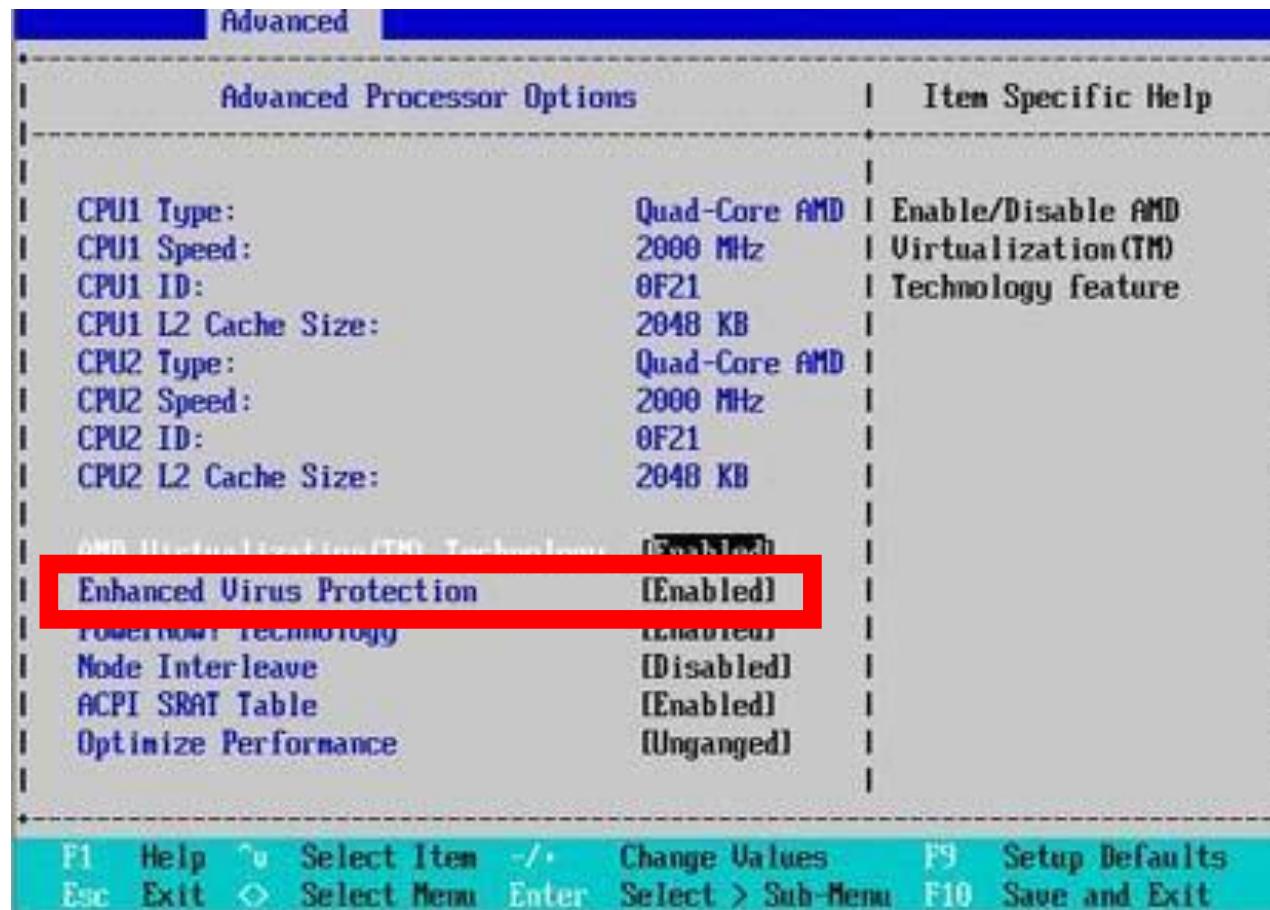
Or in other words

- Data Execution Prevention (DEP) is a **system-level memory protection** feature that is built into the operating system

# Execute Disable – Intel Processor



# Enhanced Virus Protection - AMD



# Why we need DEP?

- **Harmful programs** can try to **attack OS** by attempting to run (execute) code from **system memory locations** reserved for **OS system program and other authorized programs**.
  - These types of attacks can harm your programs and files.

# Fileless malware

- Unlike attacks carried out using traditional malware, fileless malware attacks don't **entail attackers installing software on a victim's machine.**
- Instead, tools that are **built-in to Windows** are **hijacked by adversaries** and used to carry out attacks.
- Essentially, **Windows is turned against itself.**

# Data Execution Prevention

- **Data Execution Prevention (DEP)** is a security feature that can help prevent damage to your computer from viruses and other security threats.
- Harmful programs can try to attack Windows by attempting to run code from system memory locations reserved for Windows and other authorized programs.
- DEP can help protect your computer by monitoring your programs to make sure that they use system memory safely. If DEP notices a program on your computer using memory incorrectly, it closes the program and notifies you.

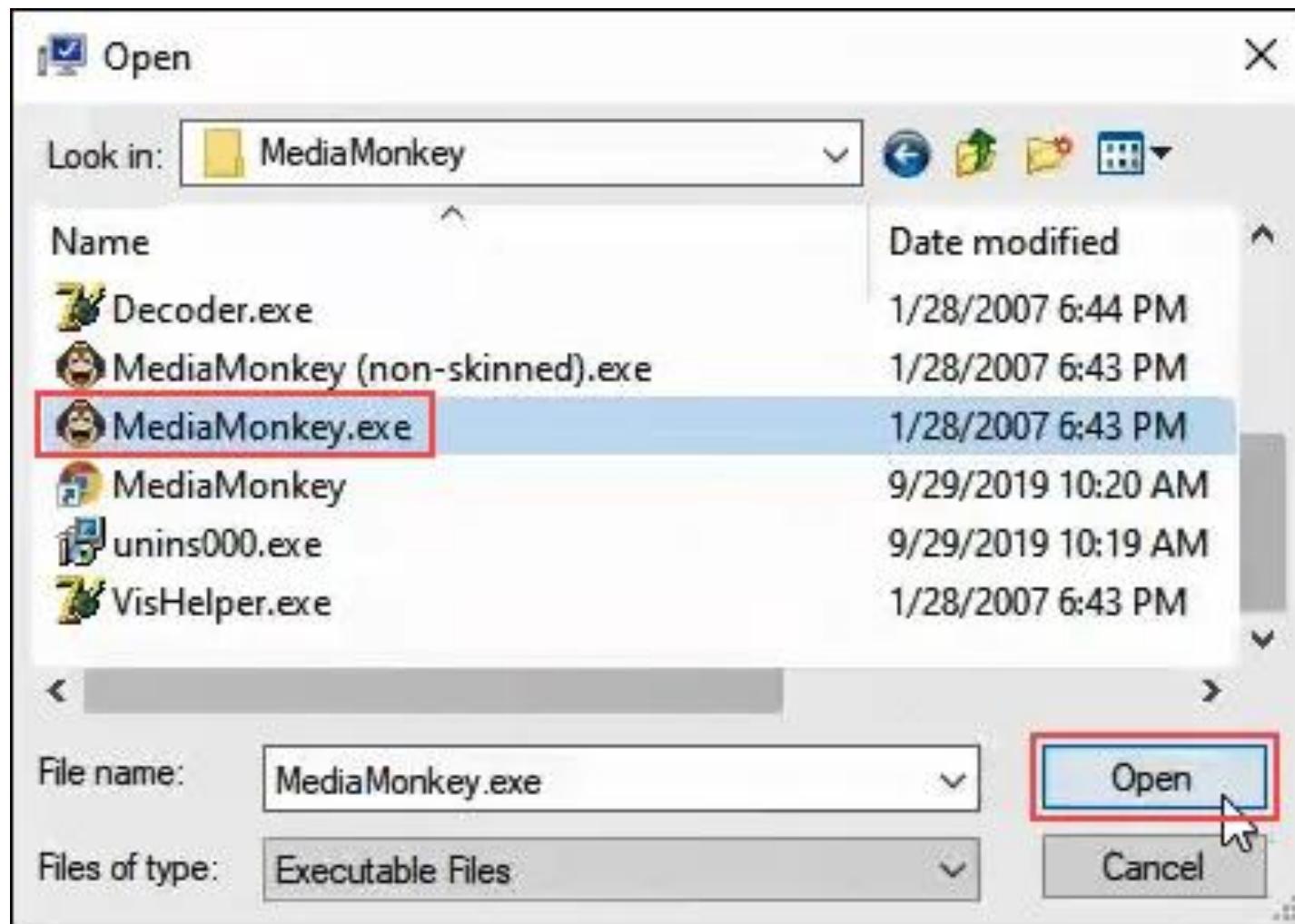
# How To Configure Or Turn Off DEP

- In Windows 10, DEP defaults to the setting **Turn on DEP for essential Windows programs and services only**. Most of the time, this is sufficient. It means that the majority of your programs will be ignored by DEP.
- If you want to turn off DEP for an individual program, follow the instructions given in the following pages.

# How To Configure Or Turn Off DEP

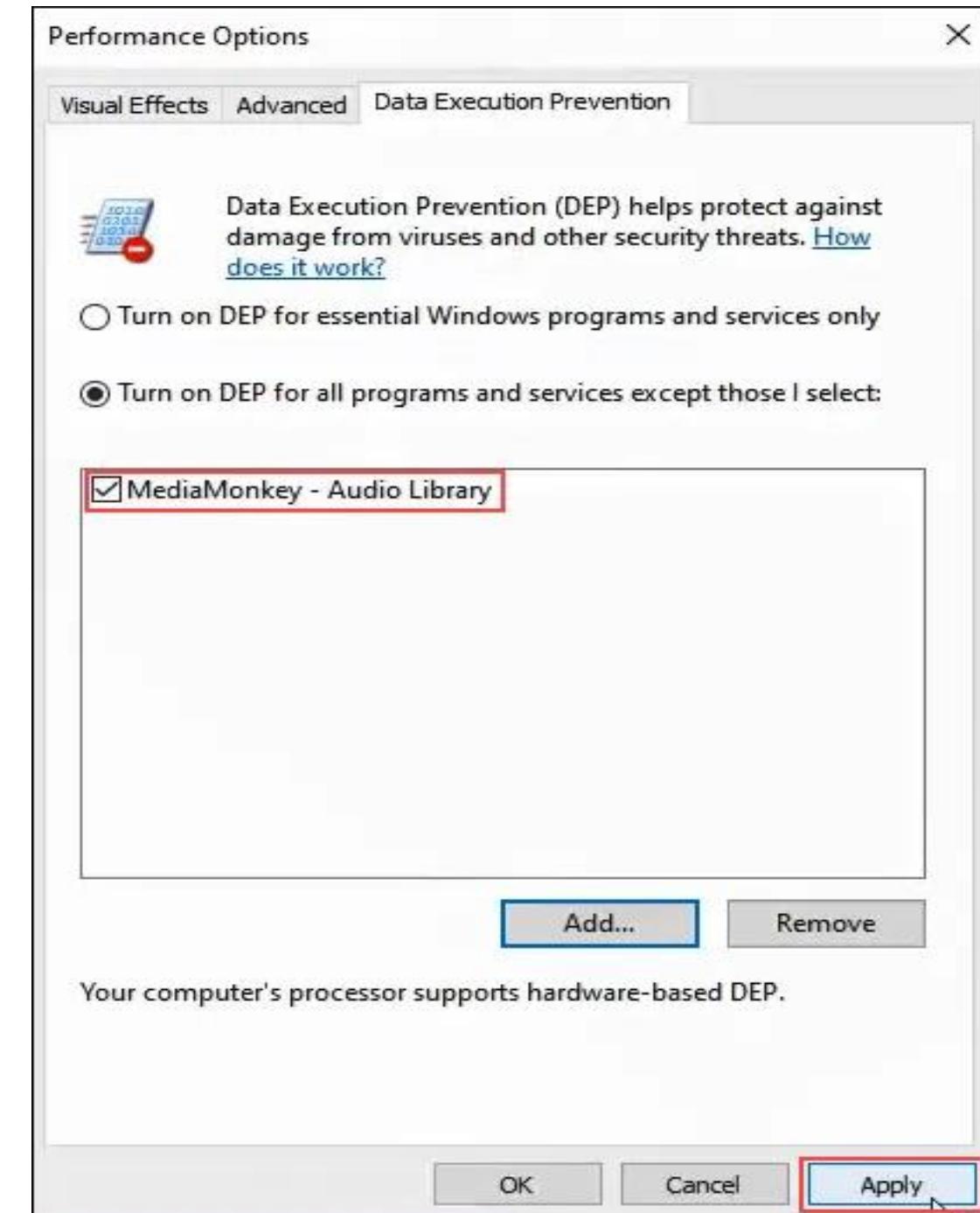
1. Open **System** by clicking the Start button, right-clicking **Computer**, and then clicking **Properties**.
2. Click **Advanced system settings**. If you're prompted for an administrator password or confirmation, type the password or provide confirmation.
3. Under **Performance**, click **Settings**.
4. Click the **Data Execution Prevention** tab, and then click **Turn on DEP for all programs and services except those I select**.
5. Click on **Add**.
6. Navigate to the executable for the program that we'd like to add as an exception. It will most likely be in **C:/Program Files (x86)**
7. In this example, **MediaMonkey**, an old music player utility, is added. Click on the **.exe** file once we find it and click on **Open**.

# How To Configure Or Turn Off DEP



# How To Configure Or Turn Off DEP

8. In Performance Options, click on **Apply**. Now, MediaMonkey will run outside of DEP protection while all others will run within DEP protection.



# Turn DEP Completely Off

- If you want to turn DEP completely off, you should do that as part of troubleshooting an issue. DEP is there for your protection.
- Since it's something that isn't advised, there isn't a good point-and-click way to do it. Let's look at how we can turn off DEP.
  1. Open the **Command window** as Administrator. Do this by typing **cmd** in the program search field near the Start menu.
  2. Enter the command **bcdedit.exe /set {current} nx AlwaysOff** and press enter.
  3. **Restart** the computer.
  4. DEP will now be completely, and permanently off.

# Turn DEP On For Everything

- To turn DEP on for absolutely everything, the process and command is like above.
  1. Open the **Command window** as Administrator.
  2. Enter the command **bcdedit.exe /set {current} nx AlwaysOn**.
  3. **Restart** the computer.
  4. DEP will be turned on and all programs monitored.

**NOTE:** *After turning DEP to being always on or always off, it CANNOT be changed via the Data Execution Prevention tab in system settings.*

# Set DEP Back To Default Behavior

- To set DEP behavior back to default and make it manageable again via system settings, do the following.
  1. Open the **Command window** as Administrator.
  2. Enter the command **bcdedit.exe /set {current} nx OptIn**.
  3. **Restart** the computer.
  4. Now the radio buttons in the DEP tab in systems settings are accessible again.

# DEP Settings

- If you turn off DEP for a specific program, it might become vulnerable to attack. A successful attack could then spread to other programs on your computer, to your contacts, and could damage your personal files.
- If you suspect that a program does not run correctly when DEP is turned on, check for a DEP-compatible version or update from the software publisher before you change any DEP settings.

# How DEP prevents the exploitation?

- DEP enables the system to mark one or more pages of memory as **non-executable**.
- Marking memory regions as non-executable means that the arbitrary code cannot run from that region of memory, which makes it harder for the exploitation of buffer overruns.
- If an application attempts to run code from a protected page, the application receives an exception with the status code **STATUS\_ACCESS\_VIOLATION**. If the exception is not handled, the calling process is terminated.

# How DEP prevents the exploitation?

- If your application must run code from a memory page, it must allocate and set the proper virtual memory protection attributes.
- The allocated memory must be marked with any one of the protection attributes:
  - **PAGE\_EXECUTE**
  - **PAGE\_EXECUTE\_READ**
  - **PAGE\_EXECUTE\_READWRITE**
  - **PAGE\_EXECUTE\_WRITECOPY**

# Memory protection attributes

Constants	Values	Description
PAGE_EXECUTE	0x10	Enables execute access to the committed region of pages.
PAGE_EXECUTE_READ	0x20	Enables execute or read-only access to the committed region of pages.
PAGE_EXECUTE_READWRITE	0x40	Enables execute, read-only, or read/write access to the committed region of pages.
PAGE_EXECUTE_WRITECOPY	0x80	Enables execute, read-only, or copy-on-write access

NOTE: Please visit <https://docs.microsoft.com/en-us/windows/win32/memory/memory-protection-constants> for more information.

# How memory protection is carried out?

- In general, Memory that belongs to a process is implicitly protected by its **private virtual address space**.
- When a processor reads or writes to a memory location, it uses a virtual address.
- As part of the read or write operation, the processor translates the virtual address to a physical address.

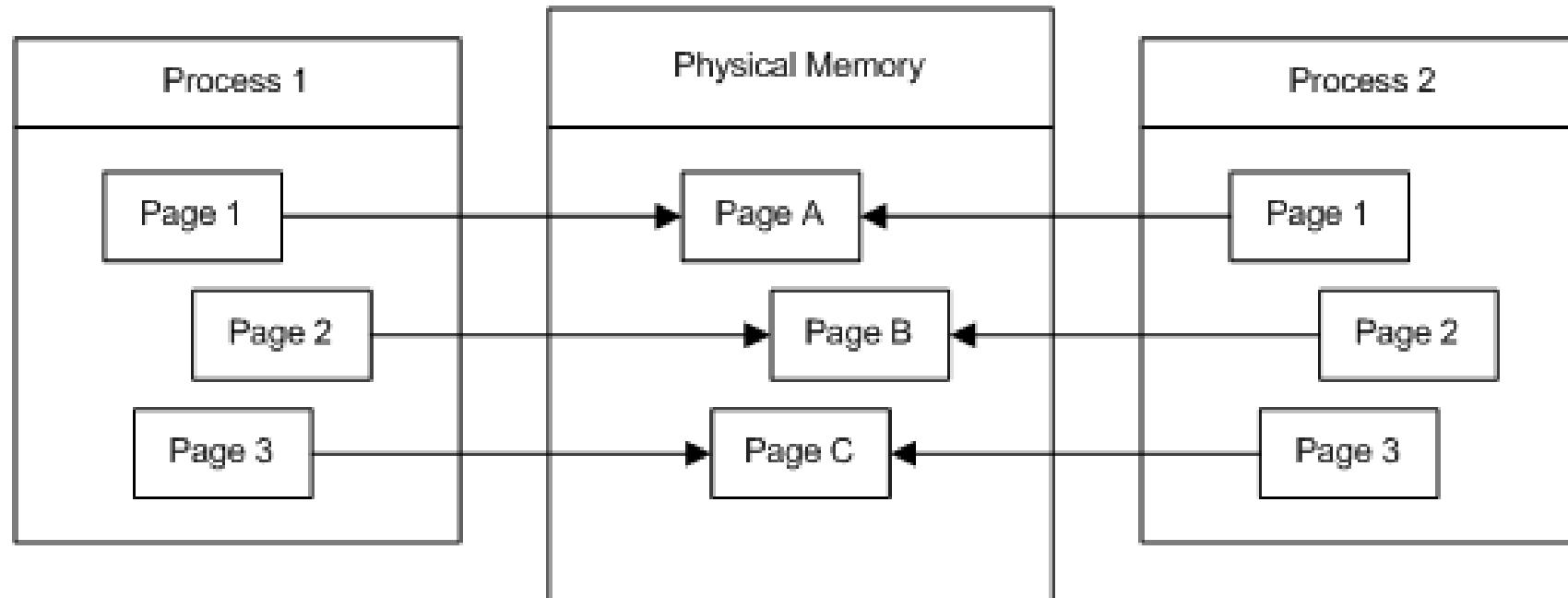
**NOTE:** For more information on virtual address, please visit the following link.  
<https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/virtual-address-spaces>

# Memory protection – COW Protection

- **Copy-on-write (COW) protection** is an optimization that allows multiple processes to map their virtual address spaces such that they share a physical page until one of the processes modifies the page.
- This is part of a technique called **lazy evaluation**, which allows the system to conserve physical memory and time by not performing an operation until absolutely necessary.
- Note: **The implementation of this protection varies with the processor.**

# How COW protection works?

- Suppose two processes load pages from the same DLL into their virtual memory spaces.
- These virtual memory pages are mapped to the same physical memory pages for both processes.
- As long as neither process writes to these pages, they can map to and share the same physical pages.

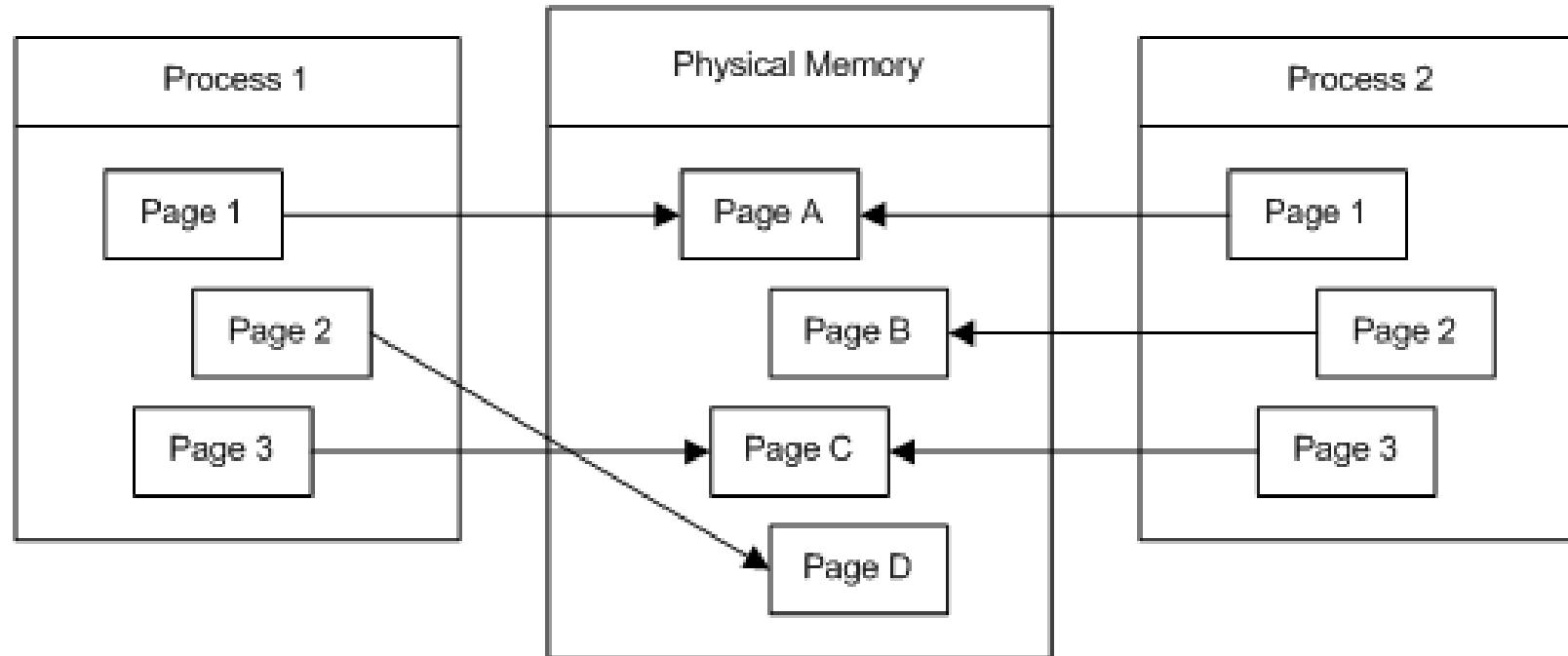


# IMPORTANT CONCEPT

- A dynamic link library (DLL) is a collection of small programs that can be loaded when needed by larger programs and used at the same time.
- The small program lets the larger program communicate with a specific device, such as a printer or scanner. It is often packaged as a DLL program, which is usually referred to as a DLL file. DLL files that support specific device operation are known as device drivers.
- The advantage of DLL files is space is saved in random access memory (RAM) because the files don't get loaded into RAM together with the main program. When a DLL file is needed, it is loaded and run.
- For example, as long as a user is editing a document in Microsoft Word, the printer DLL file does not need to be loaded into RAM. If the user decides to print the document, the Word application causes the printer DLL file to be loaded and run.

# How COW protection works?

- If Process 1 writes to one of these pages, the contents of the physical page are copied to another physical page and the virtual memory map is updated for Process 1.
- Both processes now have their own instance of the page in physical memory.
- Therefore, it is not possible for one process to write to a shared physical page and for the other process to see the changes.



# DEP protection levels

- Hardware DEP or Hardware enforced DEP
- Software DEP or Software enforced DEP

# Hardware enforced DEP

- Hardware-enforced DEP marks all memory locations in a process as **non-executable** unless the location explicitly contains executable code.
- A class of attacks exists and tries to insert and run code from non-executable memory locations.
- DEP helps prevent these attacks by intercepting them and raising an exception.

# How hardware DEP works?

- Hardware-enforced DEP relies on processor hardware to mark memory with **memory protection attributes** that indicates that the **code** should not be executed from that memory region.
- DEP functions on a **per-virtual memory page basis**, and DEP typically **changes a bit** in the **page table entry (PTE)** to **mark the memory page**.
  - DEP marks memory pages as non-executable by **marking the XD or NX bit** in the **Page Table Entry**.
  - This mechanism is **difficult to bypass** as it is implemented at the **hardware level**.

**NOTE:** For information on PTE, please visit the link: <https://www.gatevidyalay.com/page-table-paging-in-operating-system/>

# Software-enforced DEP

- Software-enforced DEP can help block programs that take advantage of exception-handling mechanisms in Windows.

**NOTE:** For more information on Hardware and Software DEP, please visit the link:  
*<http://www.heelpbook.net/2016/data-execution-prevention-dep-and-verify-status/>*

# DEP configuration

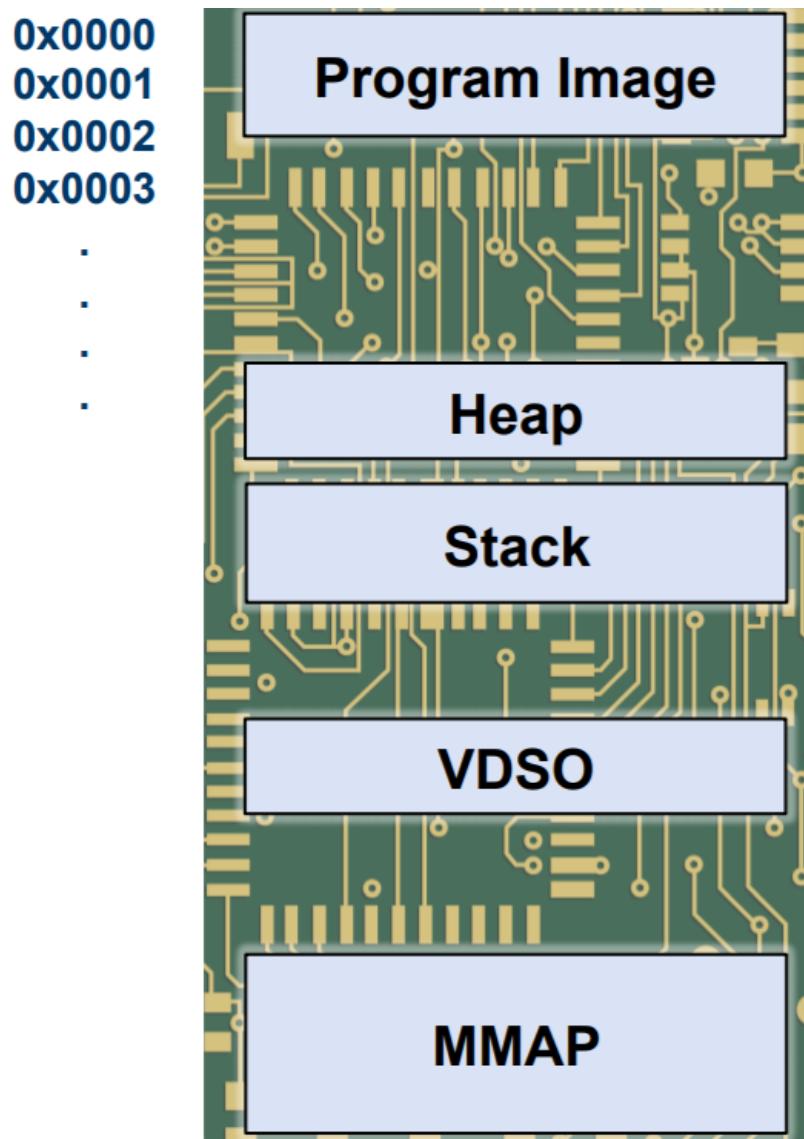
- DEP is configured at **system boot** according to the **no-execute page protection policy setting** in the boot configuration data.

Configuration	Description
OptIn	This setting is the default configuration. On systems with processors that can implement hardware-enforced DEP, DEP is enabled by default for limited system binaries and programs
OptOut	DEP is enabled by default for all processes. You can manually create a list of specific programs that do not have DEP applied by using the System dialog box in Control Panel.
AlwaysOn	This setting provides full DEP coverage for the whole system. All processes always run with DEP applied. The exceptions list to exempt specific programs from DEP protection is not available.
Always Off	This setting does not provide any DEP coverage for any part of the system, regardless of hardware DEP support.

# Programming Considerations

- An application can use the **VirtualAlloc** function to allocate executable memory with the appropriate memory protection options.
- It is suggested that an application set, at a minimum, the **PAGE\_EXECUTE** memory protection option.
- After the executable code is generated, it is recommended that the application set memory protections to disallow write access to the allocated memory.
- Applications can disallow write access to allocated memory by using the **VirtualProtect** function.
- Disallowing write access ensures maximum protection for executable regions of process address space.

# User memory layout



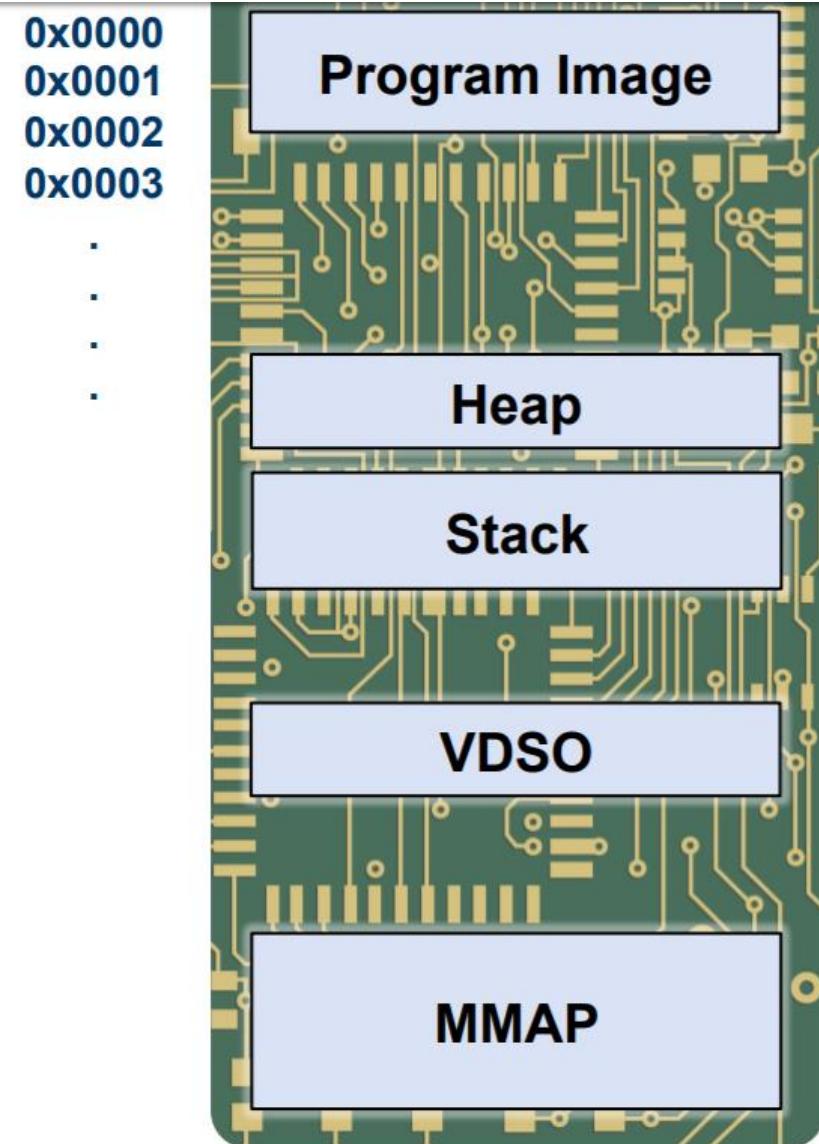
**Multiple sections required to run a program:**

- Code to run (“**Program Image**”)
- Variables used in execution (“**Heap**” and “**Stack**”)
- Kernel functions (“**VDSO**”)
- Libraries (“**MMAP**”)

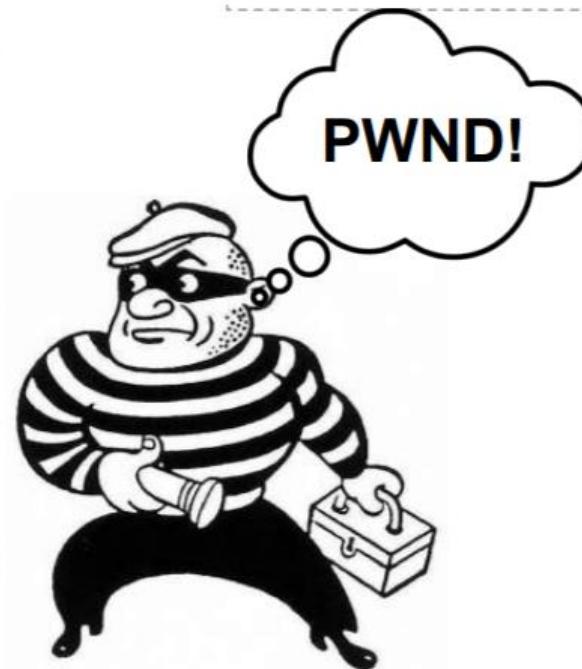
MMAP: Memory Map

VDSO: Virtual Dynamically-linked Shared Objects

# Problem with user memory layout

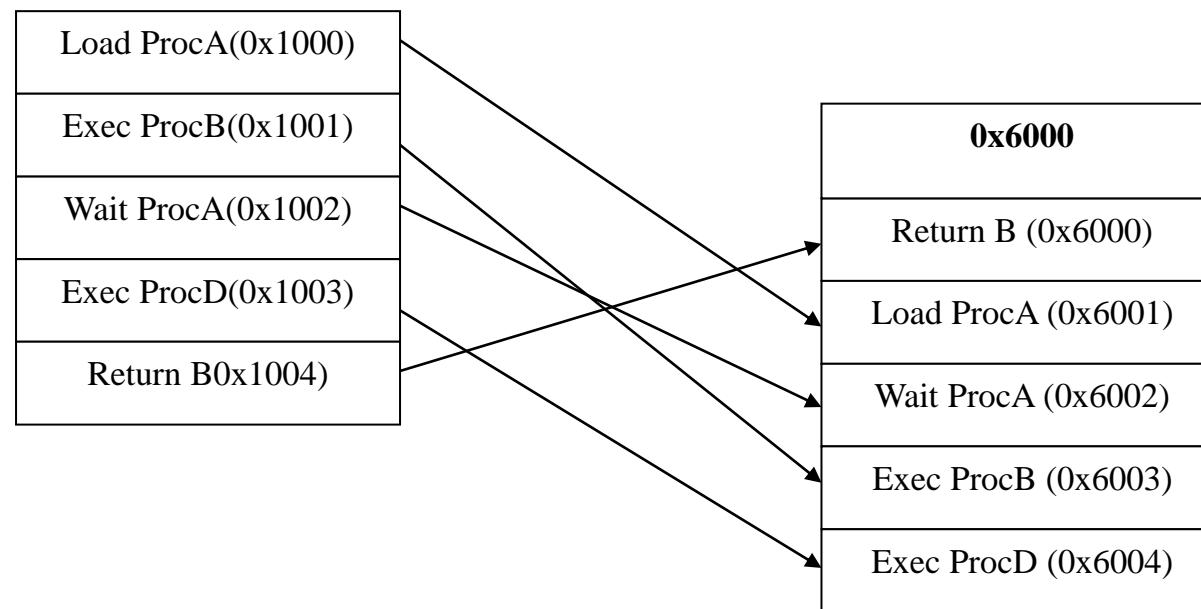


- In a static layout a variety of attacks are possible since an adversary can trivially know the location of objects in memory



# ASLR – Address Space Layout Randomization

- Address space layout randomization (ASLR) is a **memory-protection process** for operating systems (OSes) that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory.



# ASLR

- The success of many cyberattacks, particularly zero-day exploits, relies on the hacker's **ability to know or guess the position of processes and functions in memory**.
- ASLR aimed at minimizing the **predictability** in the **loading address of system modules** as well as **user programs**.
- ASLR uses a **global offset** and **randomizes** the **base address** of the PE image of programs and default system heaps, queues etc., every time the computer is rebooted or every time a program execution is initiated.
- Hence, ASLR is able to put address space targets in unpredictable locations.
- If an attacker attempts to exploit an **incorrect address space location**, the target application will crash, stopping the attack and alerting the system.

# How ASLR works

- Buffer overflows **require an attacker to know** where each part of the program is located in memory.
  - Figuring this out is usually a difficult process of trial and error.
  - After determining the memory location, they must craft a payload and find a suitable place to inject it.
- **If the attacker does not know where their target code is located,** it can be difficult or impossible to exploit it.

# How ASLR works ...

- ASLR works alongside virtual memory management to randomize the locations of different parts of the program in memory.
- Every time the program is run, **components (including the stack, heap, and libraries)** are moved to a different address in virtual memory space.
- Attackers can no longer **know** where their **target** is through trial and error, because the address will be different every time.

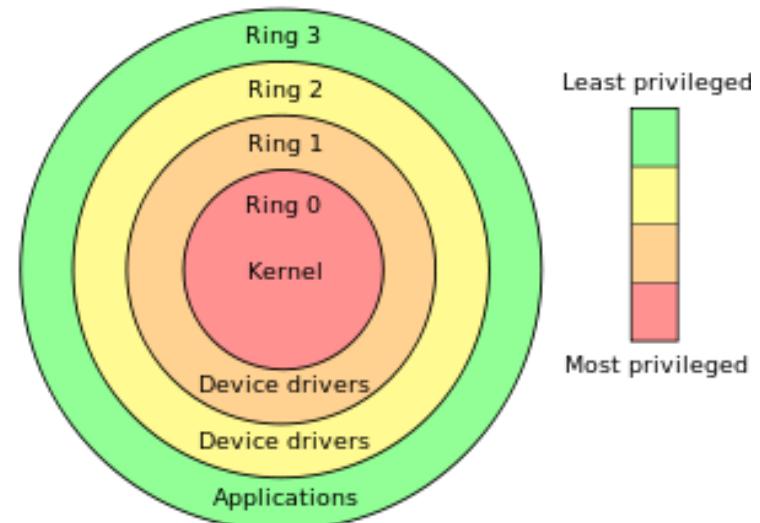
**Note:** Generally, applications need to be compiled with ASLR support, but this is becoming the default

# How ASLR randomizes the location

- ASLR randomizes the location of executables and DLLs in memory, as well as the stack and heaps.
- When an executable is loaded into memory,
- **Step 1:** Windows gets the processor's timestamp counter (TSC),
- **Step 2:** Shifts TSC counter by four places,
- **Step 3:** Performs division mod 254 and then adds 1.  
ensures that the value can never be 0, so that the executable w(by adding 1, memory manager ill never load at the address in the PE header)
- **Step 4:** Obtained value in Step 3 is then multiplied by 64KB (default stack size), and the executable image is loaded at this offset.
- **Results:** This means that there are 256 possible locations for the executable in 32 bit systems.

# Time-Stamp Counter (TSC)

- 64 bit specific register.
- Used to Increment the CPU clock cycle when the CPU is powered on, starting from 0.
- It won't overflow for at least 10 years.
- Programs running in ring level 3 can also access PTC using **rdtsc**



# How applications incorporate ASLR feature in executable image

- Windows incorporated ASLR feature in VS 2010.
- Every executable image must indicate its **inclusion in ASLR** by **setting the DYNAMICBASE flag**.

# What happens if the attacker finds the initial base address of any application image?

- Even, if you are compromised and in next boot or start up, the infection won't work.
- **Code injection attacks** fail due to randomness in Address Space Layout (ASL).
- Process hooking fails due to **complete randomness**.
- The layout of the memory changes at every reboot, the attacker might find it difficult to utilize any security vulnerability present in the programs

# Randomization in Address Space Layout

- Top-down
- Bottom-up
- Based

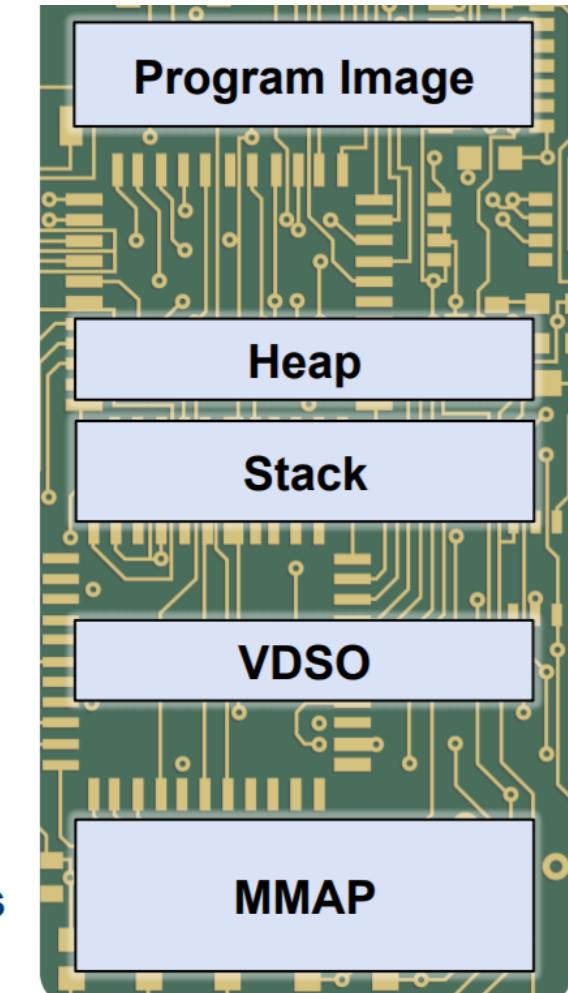
# Virtual memory allocation

- Virtual memory allocations can happen in three ways:
  - **top-down, bottom-up and based.**
- The top-down approach searches for an **address** at the **top of the address space**.
- Bottom-up approach searches and allocates **address** at the **bottom of the address space**.
- Based approach allocates address with a **specified base address** being mentioned explicitly.

- **Problem with normal ASLR**
  - Windows incorporated ASLR feature in VS 2010.
  - Every executable image must indicate its inclusion in ASLR by setting the **DYNAMICBASE** flag.
- **Why Mandatory ASLR**
  - If an image does not set the DYNAMICBASE flag, then it might not be relocated by the Windows kernel when loaded into the address space.
  - Hence some images are loaded at predictable address making them more vulnerable.

# Types of ASLR

- Force or Mandatory ASLR (System Wide)
- Bottom-up ASLR (Per process basis)
- Entropy ASLR (ASLR can use the entire 64-bit address space)
  - Static ASLR
  - Process Independent Executable ASLR



# Mandatory ASLR – Present Windows

- when an image forces ASLR, then its address will be randomized irrespective of the **DYNAMICBASE** flag.
- This however, might lead to compatibility issues while relocating images which do not have support for ASLR.

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\kernel]
"MitigationOptions"=hex:00,01,01,00,00,00,00,00,00,00,00,00,00,00,00,00
```

# ASLR in real time

- DYNAMICBASE
- MANDATORYASLR
- HIGHENTROPYVA
- All the configuration of ASLR happens when linking and generating executable
- Configuration property → linker → commandline → HIGHENTROPYVA
- Configuration property → linker → commandline → DYNAMICBASE

# Structured Exception Handler Overwrite Protection

- The purpose of the **SEHOP mitigation** is to prevent an attacker from being able to make use of the **Structured Exception Handler (SEH) overwrite** exploitation technique.
- SEHOP validates all the exceptions generated in the computer.

# Exception

- An exception is an event that occurs during the execution of a program.
- Requires the **execution of code outside** the **normal flow of control**.
- **Exception is a type of run time error.**

try

{ Run this code }

except

{ Run this code if an exception occurs }

else

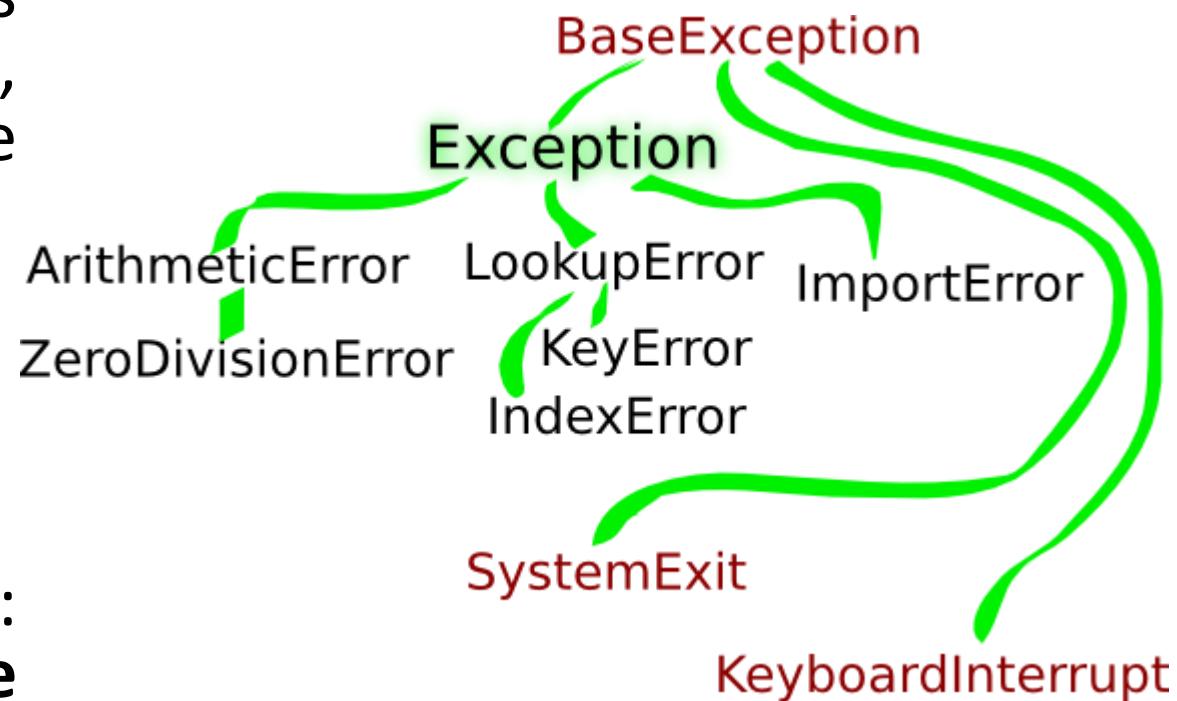
{ Run this code if no exception occurs }

finally

{ Always run this code }

# Exception Handling

- An *exception* is an event that occurs during the execution of a program, and requires the execution of code outside the normal flow of control.
- Completely depends on runtime.
- There are two kinds of exceptions: **hardware exceptions and software exceptions**.



# *Structured exception handling*

- *Structured exception handling* is a mechanism for **handling both hardware and software exception**.
- Structured exception handling enables you to have **complete control over the handling of exceptions**, provides **support for debuggers**, and is **usable across all programming languages** and machines.
- Finally, **your code** will handle hardware and software exceptions in same way.

# *Structured exception handling*

- Structured exception handling is **made available primarily through compiler support**.
- SEH Consists of two different phases namely: **structured exception handling** and **termination handling**.
- The **structured exception handling** and **termination handling** mechanisms are integral parts of the system; they enable the system to be robust.
  - You can use these mechanisms to create consistently robust and reliable application.

# *Structured Exception Handling*

- **`_try`** keyword that **identifies** a **guarded body of code**.
- **`_except`** keyword that **identifies** an **exception handler**.
- **`_finally`** keyword that **identifies** a **termination handler**.

# SEH control flow

0006FF70	0006FF38	
0006FF74	7C80E14F	RETURN to kernel32.7C80E14F from ntdll.RtlAnsiStringToUnicodeString
0006FF78	0006FFE0	Pointer to next SEH record
0006FF7C	7C839AD8	SE handler
0006FF80	7C80E0F8	kernel32.7C80E0F8
0006FF84	00000000	
0006FF88	0006FF9C	
0006FF8C	7C801D72	RETURN to kernel32.7C801D72 from kernel32.LoadLibraryExW
0006FF90	7FFDFC00	UNICODE "HookSwitchHookEnabledEvent"
0006FF94	00000000	
0006FF98	00000000	
0006FF9C	0006FFB8	
0006FFA0	7C801DA8	RETURN to kernel32.7C801DA8 from kernel32.LoadLibraryExA
0006FFA4	00081F01	ASCII "C:\Program Files\Gogago\YouTube Video Downloader\MDIEEx.dll"
0006FFA8	00000000	
0006FFAC	00000000	
0006FFB0	00081F01	ASCII "C:\Program Files\Gogago\YouTube Video Downloader\MDIEEx.dll"
0006FFB4	7FFD9000	
0006FFB8	0006FFC4	
0006FFBC	004100B4	RETURN to LOADDLL.004100B4 from <JMP.&KERNEL32.LoadLibraryA>
0006FFC0	00081F01	ASCII "C:\Program Files\Gogago\YouTube Video Downloader\MDIEEx.dll"
0006FFC4	7C817077	RETURN to kernel32.7C817077
0006FFC8	7C920228	ntdll.7C920228
0006FFCC	FFFFFFFF	
0006FFD0	7FFD9000	
0006FFD4	00000640	
0006FFD8	0006FFC8	
0006FFDC	89819DA8	
0006FFE0	FFFFFFFF	End of SEH chain
0006FFE4	7C839AD8	SE handler
0006FFE8	7C817080	kernel32.7C817080
0006FFEC	00000000	
0006FFF0	00000000	
0006FFF4	00000000	
0006FFF8	00410070	LOADDLL.<ModuleEntryPoint>
0006FFFC	00000000	

# SEH architecture

## Exception Registration Structure

Exception handler 1

⋮

Exception handler n

# Termination Handling

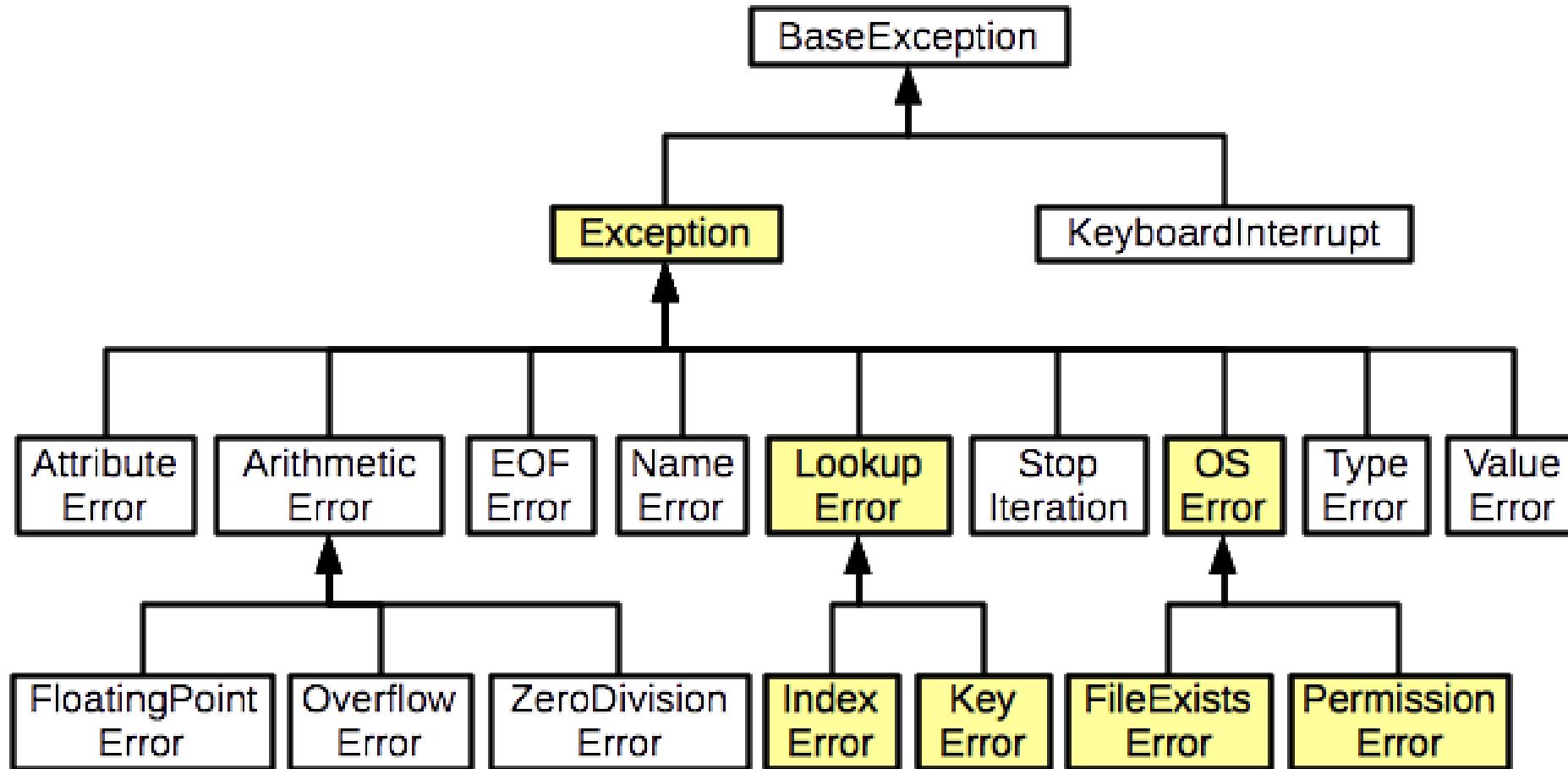
- A ***termination handler*** ensures that a **specific block of code** is executed whenever **flow of control** leaves a particular **guarded body of code**.
- A termination handler consists of the following elements.
  - A **guarded body of code**.
  - A **block of termination code** to be executed when the flow of control leaves the guarded body.
  - **try** and **finally**

- If your application program is trying to access some invalid memory, an exception is raised with some status code. What type of exception is this.?
- Assume that you are giving invalid parameter to your application, now an exception is raised. What type of exception is this. ?

# Types of exceptions

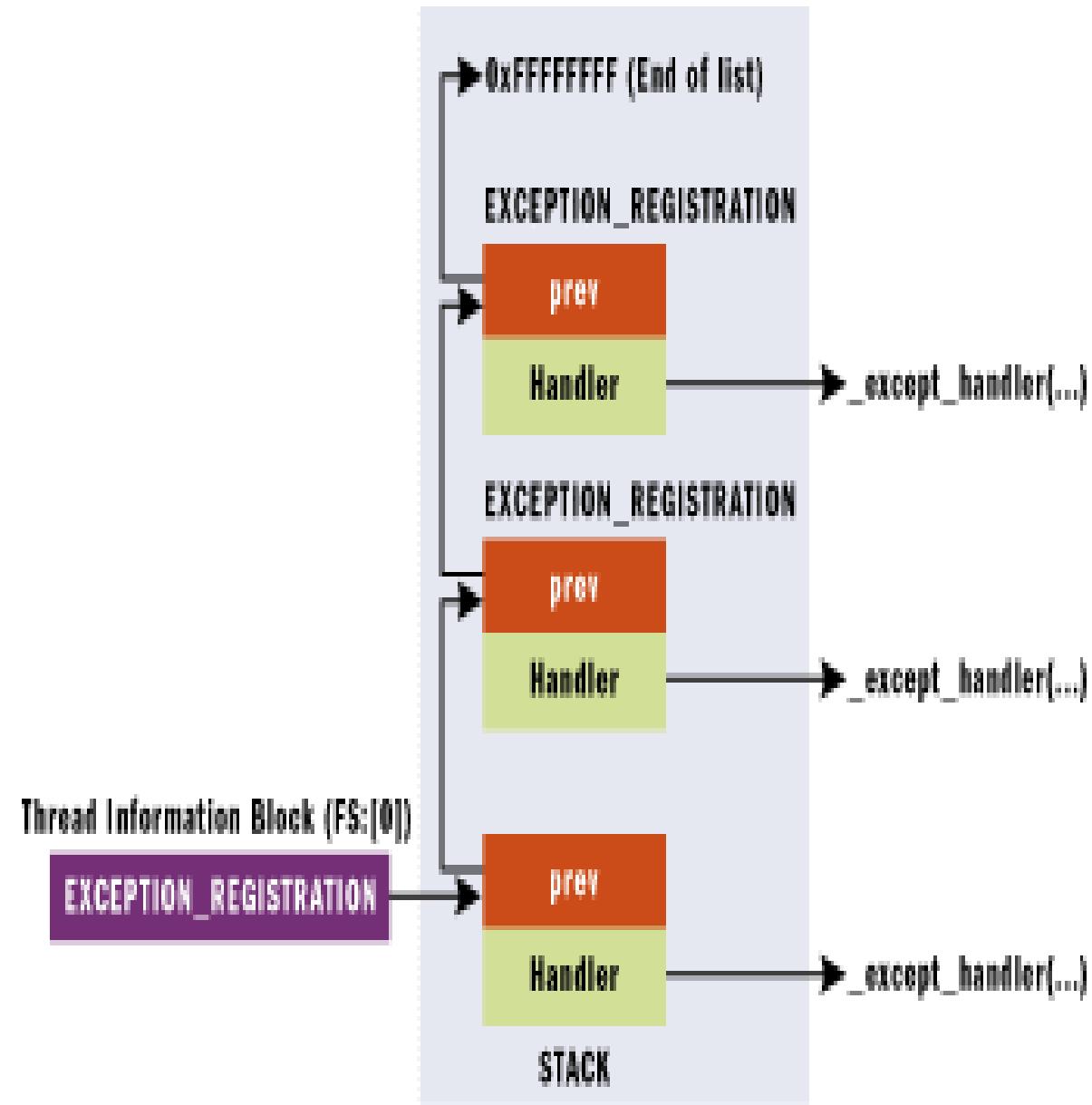
- *Hardware exceptions* are initiated by the CPU.
  - For example; they can result from the **execution of certain instruction sequences**, such as **division by zero** or an **attempt to access an invalid memory address**.
- *Software exceptions* are initiated explicitly by applications or the operating system.
  - For example, the system can detect when an invalid parameter value is specified or given by the user.

# Classification - Exceptions



# How SEH works - Short

- The **exception handlers** are **linked** to each other
- They form a **linked list chain** on the stack, and sit relatively close to the **bottom** of the stack.
- When an **exception** occurs, Windows retrieves the head of the SEH chain walks through the list and tries to find the suitable handler to close the application properly



# How SEH works?

- Whenever an exception occurs,
- **Step1:** The processor stops execution of instructions and transfers control to the operating system(OS).
- **Step2:** Exception in a thread context and **searches** for the **corresponding Exception Handler routine** from the **Exception Registration (ER) structure**.
- **Step3:** An ER structure has **two pointers**: one pointer points to **another ER structure** and another pointer points to an **exception handler routine**.
- **Step4:** The **Exception Registration structures** are constructed as elements of a **linked list**.
- **Step5:** The system walks through the linked list once, till it finds the corresponding **exception routine**.

# How SEH works? – cont'd

- If it does not find a routine, then it calls the default exception handler that is built along with the creation of every thread.
- Once an exception handler is found out, then the system walks through the ER list once again to **unwind (stack)** and look for the previously handled exception routines.

**(How many of you know about stack unwinding)**

- All the previously opened exception routines are terminated using a termination handler until the corresponding exception routine moves up the stack.

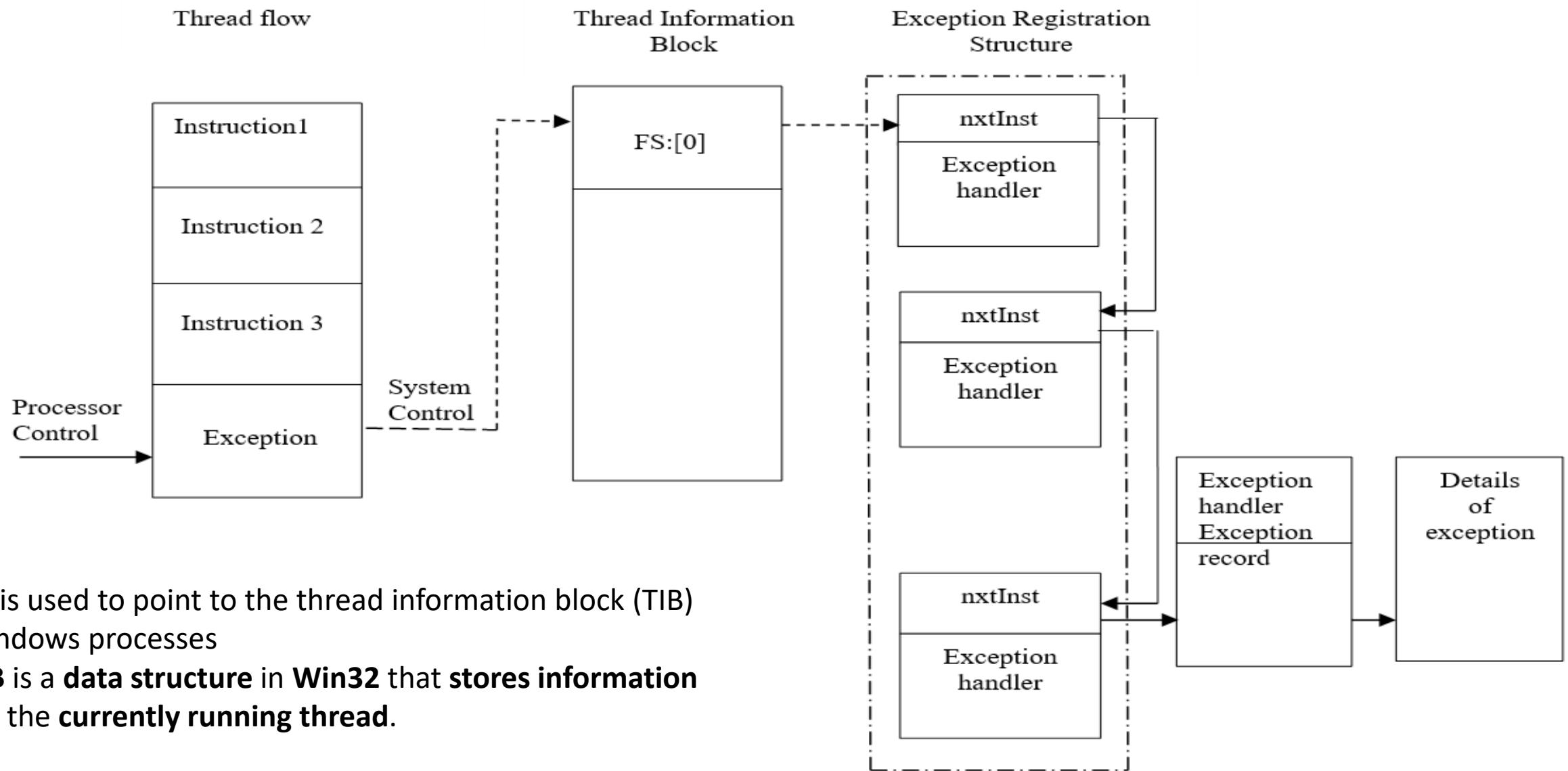
# Stack unwinding

- The process of **removing function entries from function call stack** at run time is called **Stack Unwinding**.

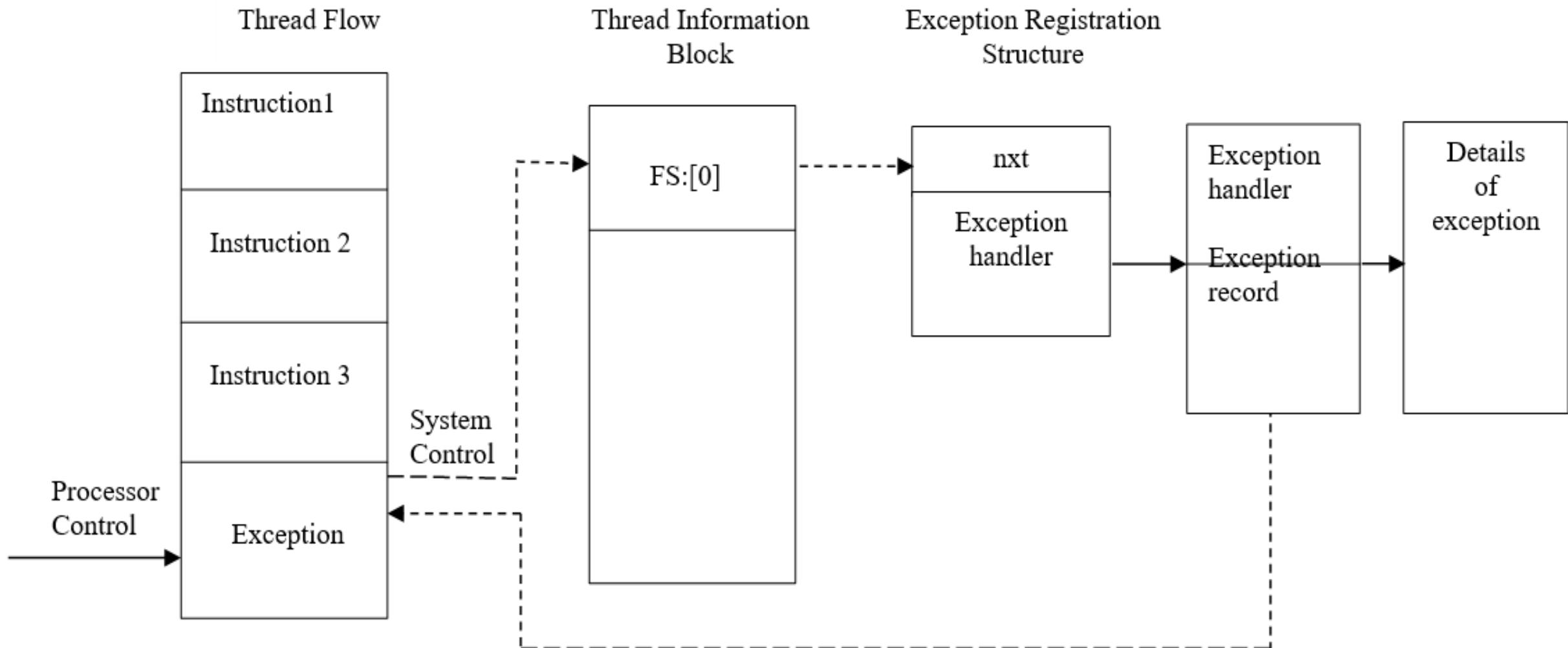
(a **call stack** is a stack data structure that stores information about the active subroutines of a computer program)

- **Stack Unwinding** is generally related to **Exception Handling**.
- When an **exception occurs**, the function **call stack** is **linearly searched** for the **exception handler**, and all the entries before the function with exception handler are removed from the function call stack.
- So exception handling involves **Stack Unwinding**, if exception is not handled in same function/control block.

# SEH before unwinding



# SEH after unwinding



# Problem with SEH

- **Structured Exception Handling overwrites** and it aims at exploiting the vulnerability of the **Structured Exception Handling** technique.
- The attacker modifies the **pointer field** in the **Exception Handler Routine** to **point to any arbitrary code** or **redirect the flow of control back** to the buffer by using stack based buffer overflow technique.
- The attacker might try to **modify both the pointers** of the **ER** so that the **system while walking through the list of ERs**, might **execute any code from any arbitrary location** pointed to by the **corrupted Exception Registration structure**.

Offset: @\$scopeip		Previous	Next	Virtus: 015fd044
No prior disassembly possible				
61616161 ??	???			015fd044 61616161
61616162 ??	???			015fd048 61616161
61616163 ??	???			015fd04c 61616161
61616164 ??	???			015fd050 61616161
61616165 ??	??			015fd054 22616161
61616166 ??	???			015fd058 73256300
61616167 ??	???			015fd05c 015fd220
61616168 ??	???			015fd060 003cbe1c
61616169 ??	???			015fd064 003c51d8
6161616a ??	???			015fd068 003cd6c8
6161616b ??	???			015fd06c 015fd2a8
6161616c ??	???			015fd070 00000002
6161616d ??	???			015fd074 ffffffff
6161616e ??	???			015fd078 003cff98
6161616f ??	???			015fd07c 003cb550
61616170 ??	???			015fd080 00000001
61616171 ??	???			015fd084 00000000
61616172 ??	???			015fd088 003ce108
61616173 ??	???			015fd08c 003ce108
61616174 ??	???			015fd090 003ce128
61616175 ??	???			015fd094 003cffb8
61616176 ??	???			015fd098 00000000
61616177 ??	???			015fd09c 003cd9b8
61616178 ??	???			015fd0a0 00000000
61616179 ??	???			015fd0a4 003cfef8
6161617a ??	???			015fd0a8 003ce108
6161617b ??	???			015fd0ac 00000000
6161617c ??	???			015fd0b0 ffffffff
6161617d ??	???			015fd0b4 00000000
6161617e ??	???			015fd0b8 00000000
6161617f ??	???			015fd0bc 00000000
61616180 ??	???			015fd0c0 015fd00c
61616181 ??	???			015fd0c4 00000000
				015fd0c8 015fd220
				015fd0cc 00000000

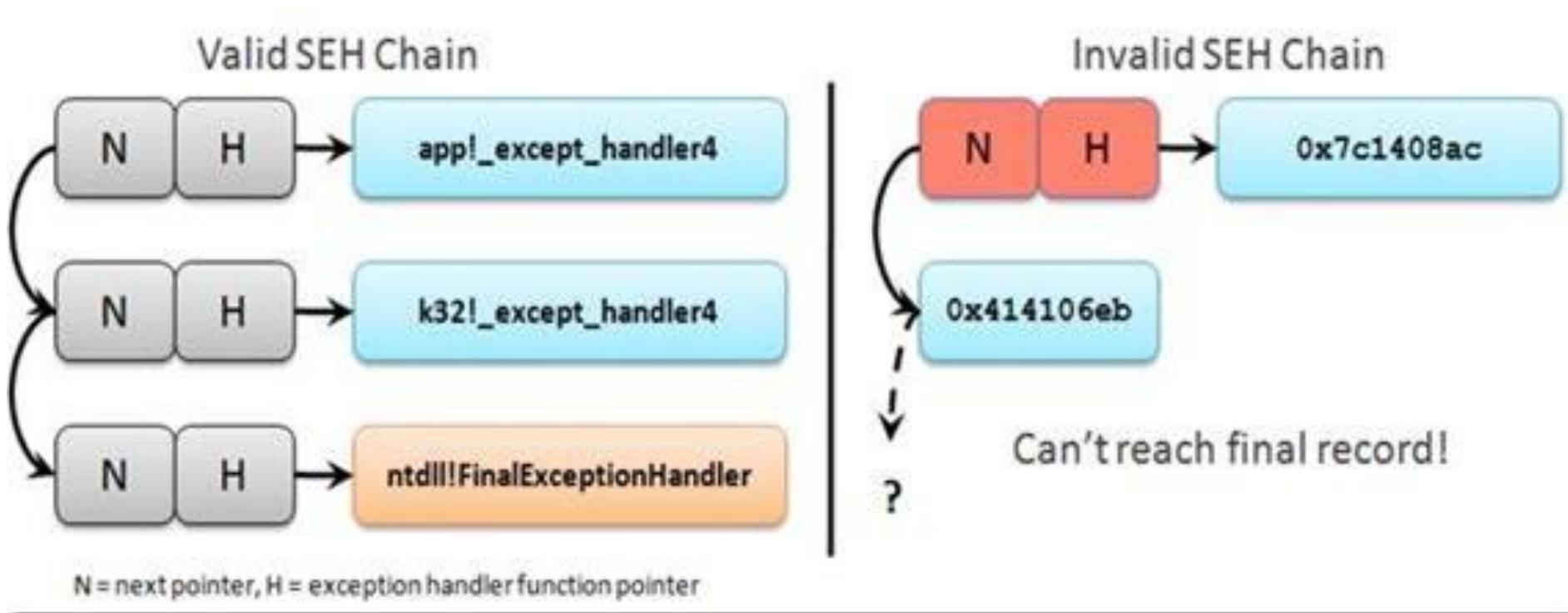
EIP point to **0x61616161**,  
so we can control the **flow** of the program

Pointer to the **Next** record and **SEH handler** was overwritten

# SEH overwrite protection (SEHOP)

- Structured Exception Handling Overwrite Protection technique aims at **preventing such overwrites** made to the **pointers of ER fields**. It works as follows:
  - A **symbolic record (/SAFESEH)** is **added** at the **end of the list of exception handlers** to a thread at the time of creation and is **registered for future reference**.
  - When an exception is returned, this list is **cross checked** with the **thread's updated list** each time.
  - If an Exception Registration structure has been corrupted, then the changes are reflected in the symbolic record which, when compared with the original record returns a mismatch and the process will be safely terminated.

# SEHOP



# Trusted Platform Module

- What is trusted platform module?
- What are the uses of TPM?
- What is TPM 2.0?
- Find the TPM types of keys, give brief description about each key?
- How remote attestation works? Or How unforgeable key gets generated?

# Trusted Platform Module

- **Hardware-based** security.
- A **crypto-processor** implemented in a chip.
- Support to use along aside with other **security systems** like **Firewall, software, smart card, biometric systems**.
- Installed on **motherboard** and is used in almost all PCs.
- **Firmware** supported

# Use of TPM

- Secure generation of cryptographic key
- Protection of secure keys
- Hardware based pseudo number generation
- Hardware authentication
- Sealed storage (password, encryption keys and digital certificates)
- Remote attestation → unforgeable key

# Some old hacks

- Physical attacks
  - Removing Hard disk from one machine and using it in another machine
  - Booting another OS/live OS to view the data (messing up with Security Accounts Manager(SAM) files in registry)
  - Using recovery option by booting live OS
  - Booting into an alternative OS ( Ophcrack run from Linux)

# Bitlocker, UEFI and Secure boot, TPM

- Default boot order is **network, USB, HDD/SSD**.
- UEFI device do not support Legacy bios mode.
- (Unified Extensible Firmware Interface (**UEFI**) is a specification for a **software program** that connects a **computer's firmware** to its **operating system** (OS)).
- **UEFI** is expected to eventually replace BIOS).
- If any one of the above setting is changed, it will trigger a complete lock out.

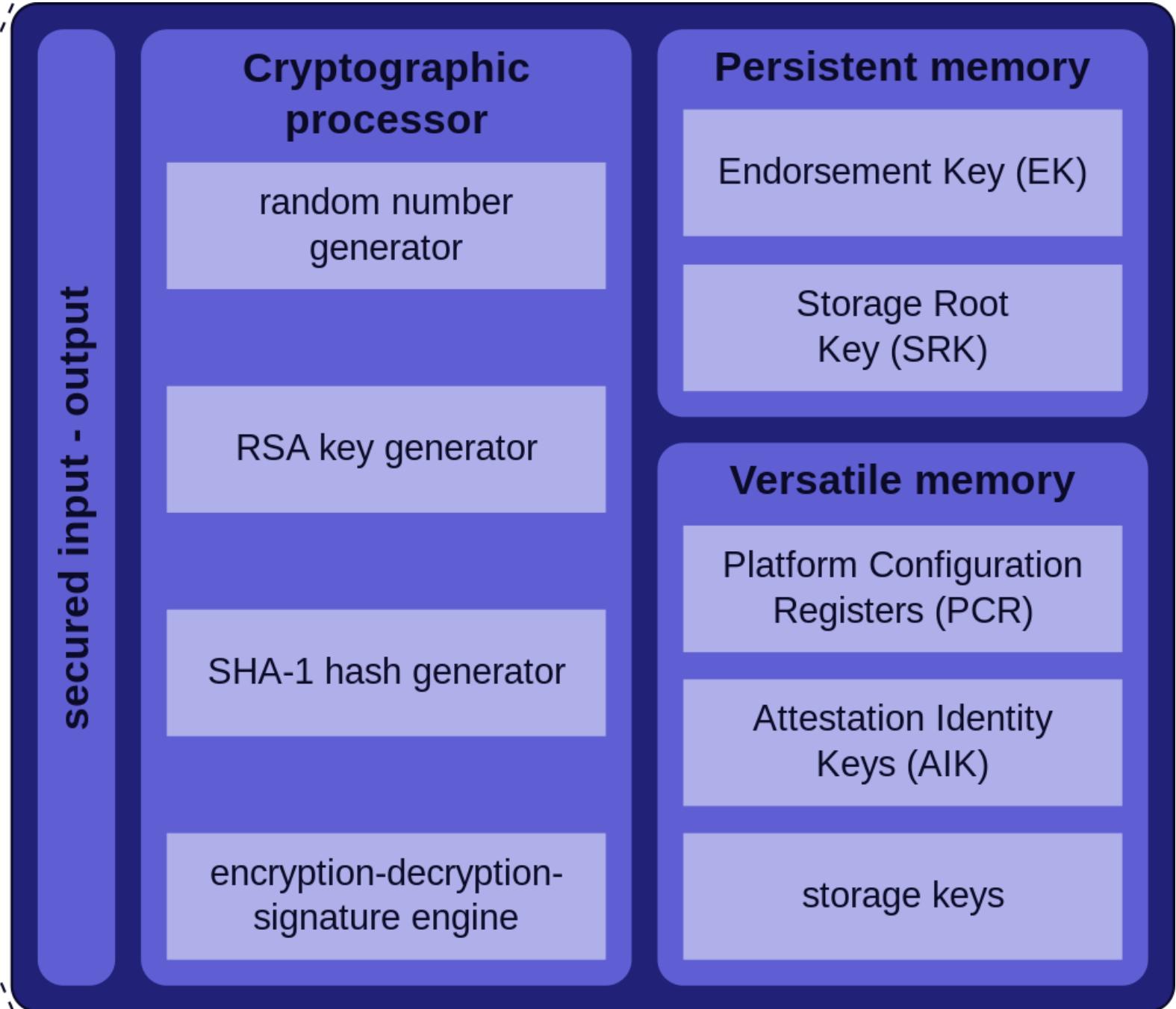
# Application: Bit locker with TPM

- Many people use a TPM to enable Windows' BitLocker Drive encryption utility.
- When you power up a system that features a TPM and BitLocker, the chip runs a series of conditional tests to see if it's safe to boot up.
- If a TPM senses the hard disk was moved to another location, as might be the case if it were stolen, it locks the system.

# Key, Certificate, Signature

- Key – used for encryption and decryption
- Certificate - A public key certificate, also known as a **digital certificate** or **identity certificate**, is an electronic document used to prove the ownership of a public key.
- Signature - A digital signature is a scheme for verifying the authenticity of digital messages or documents.

# TPM 2.0



# TPM keys

- **TPM has many type of keys**
- **Endorsement key (EK)**
  - An RSA key pair, required and created only once for the TPM's lifetime
  - Private key is inside TPM, never revealed and accessible outside TPM
- **Storage root key**
  - Created when system ownership is created
  - Based on EK and user provided password
  - Used as master wrapping key which is stored inside TPM

# Platform Configuration Registers - PCR

- Platform Configuration Registers (PCRs) is used to **cryptographically record** (measure) **software state**: both the software running on a platform and configuration data used by that software.
- The **PCR update calculation** which is a **one-way hash** so that measurements can't be removed.
- These **PCRs** can then be **read to report their state**.
- Finally, they can also be signed to return a more secure report, called an (or **quote**).

# TPM keys – cont'd

- **Attestation Identity Key (AIK)**
  - **Another pair of RSA keys**, for attestation (remote)
    - eg: when using TeamViewer or other VNC
    - eg: when using hardware and software configuration
- **How remote attestation works? Or How unforgeable key gets generated?**
  - **Public key** will be signed by **EK** and then sent to **CA**
  - **CA** then **validates** the **EK** and issue a certificate for AIK, TPM authenticate itself w.r.t to this certificate

# Storage Key

- Asymmetric key or public key
- Used to encrypt data or other keys
- Eg. Secure strings in windows

```
Windows PowerShell (x86)
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\suryakoushik> $c = Get-Credential -credential suryakoushik
PS C:\Users\suryakoushik> $c.suryakoushik
PS C:\Users\suryakoushik> $c.username
suryakoushik
PS C:\Users\suryakoushik> $c.password
System.Security.SecureString
PS C:\Users\suryakoushik> -
```

# Signing key

- Asymmetric key or public key
- Used to sign data and messages

# Bind key

- Used to **encrypt small amount data or key on one platform and decrypt it on another.**

# Authentication key

- Symmetric key or private key
- Used to protect transport sessions
- Eg. SSL/TLS pinning

# Legacy key

- Can be **exported to another TPM, for signing and encryption**

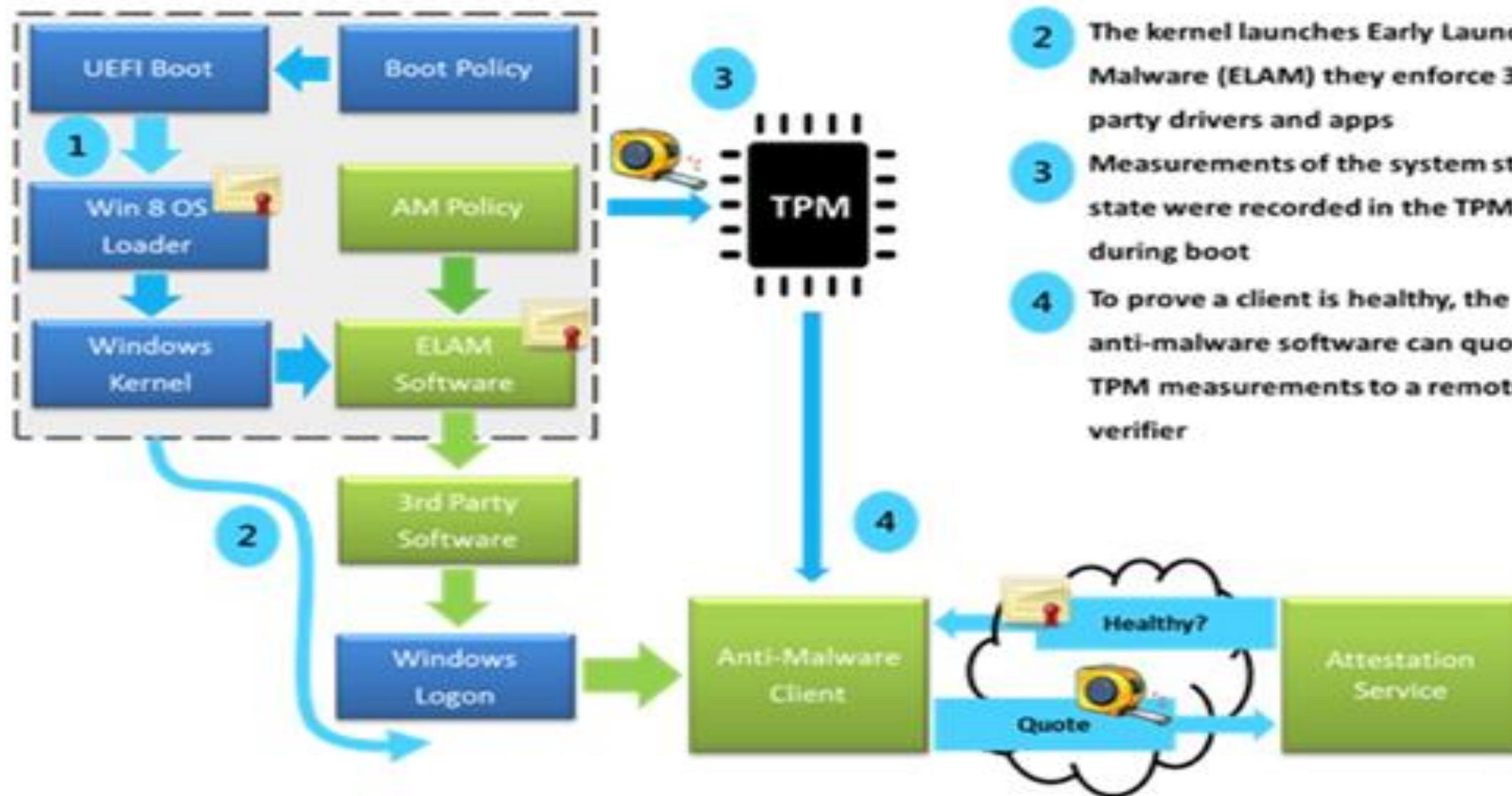
# Other keys

- Non - migratable keys
  - Bounded to a **single TPM** and present **inside the TPM**
- Migratable keys
  - Generated inside or outside TPM .
  - Integrated inside a TPM or move to another.
  - Trusted by its creator.
- Certified Migratable keys
  - Generated inside TPM
  - Can move to another TPM
  - Coordinated by **Migration authority** or **migration selection authority**.

# Real time use-cases

- Platform integrity
- Disk encryption
- Password protection

## Windows 8 Platform Integrity Architecture

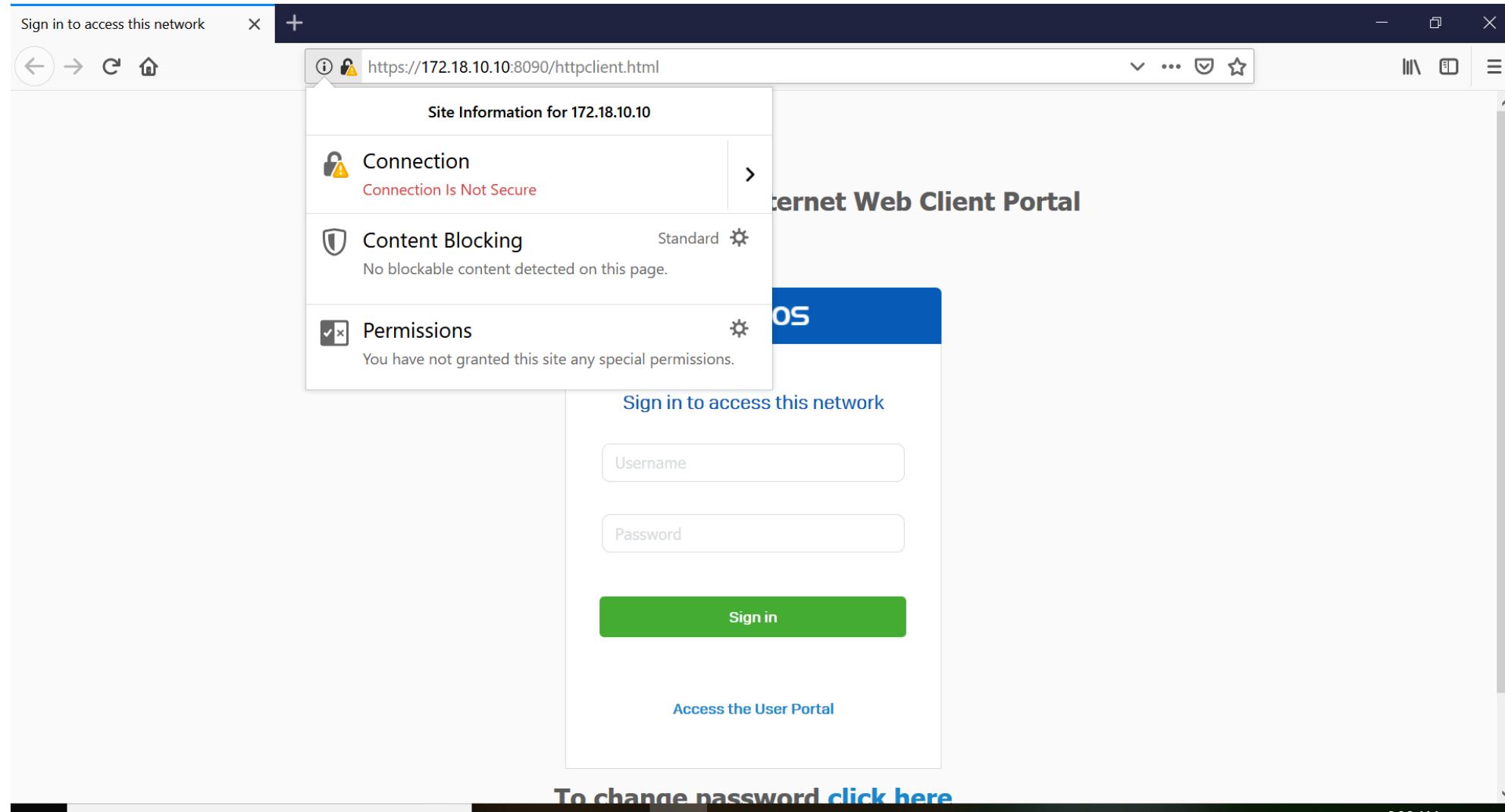


- 1 Secure boot (UEFI) prevents running a unknown OS loader
- 2 The kernel launches Early Launch Anti-Malware (ELAM) they enforce 3rd party drivers and apps
- 3 Measurements of the system start state were recorded in the TPM during boot
- 4 To prove a client is healthy, the anti-malware software can quote TPM measurements to a remote verifier

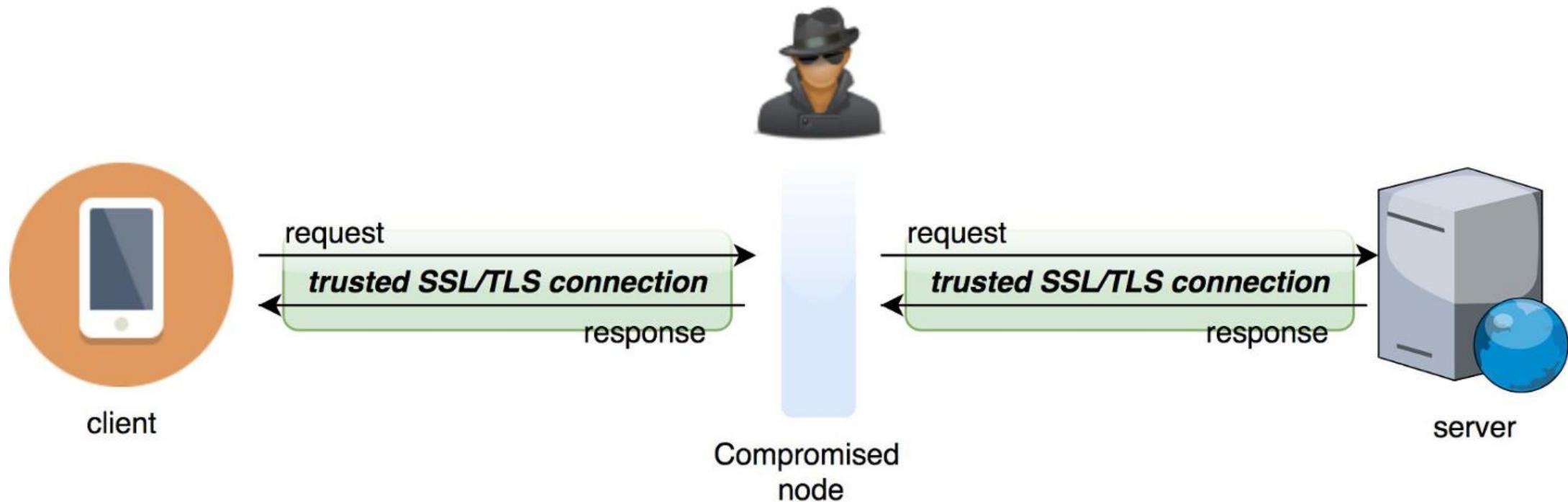
# Certificate Pinning

- Key – used for encryption and decryption
- Certificate - A public key certificate, also known as a **digital certificate** or **identity certificate**, is an electronic document used to prove the ownership of a public key.
- Signature - A digital signature is a scheme for verifying the authenticity of digital messages or documents.

# What about our captive portal?



# Problem with existing connection



# Pin or hard code the server certificates

- Typical example is **Sophos network agent**.
- **Server certificate verification on the client side.**
- **Verification requires the server certificate or its fingerprint.**
- When establishing a connection with the server, the **app** should **compare the fingerprint with a certificate from the remote server**.
- If the **fingerprints are identical**, then the connection is valid and the data transfer can proceed.
- If the **fingerprints are not identical**, then the app should reject the connection immediately

# Certificate Pinning

- Trust Agent
- Google Smart lock
- Setting → Security and Location → Trust Agents
- Setting → Security and Location → Encryption & Credentials
- Setting → Security and Location → Encryption & Credentials → Trusted credentials
  - System
  - User

# Certificate Pinning

- In a public communication, it is mandatory to ensure the security between sender and receiver.
- **SSL** and **TLS** were two main protocols responsible in providing secure communication in the internet.
- Both the protocols use **Digital Certificates** in order to **authenticate** and **encrypt the traffic**.
- **Public key** in the **digital certificate** is used to encrypt the traffic and to ensure the trust it is **digitally signed** by a **Certificate Authority**.
- **Hackers** and **advanced malware** can bogus these **digital certificates** in the name of **legitimate players** and **intercept** the **legitimate traffic** between sender and receiver.

# Certificate Pinning

- Certificate Pinning is a method of linking **X509** certificate along with its public key to a **root (CA)**.
- Certificate pinning helps reducing **trustworthiness over third party identity**.
- **X.509** is a standard defining the format of public key certificates.
- **Certificate Trust** is a feature used to pin rules for any protected sites (SSL/TLS websites).
- Certificate Pinning ensures the integrity of the site by verifying the valid certificates and any malicious target which violates the pinned rules will be discarded.

# Certificate Pinning – with/without CA

- A certificate's fingerprint is "pinned" in the used application (e.g. web browser or mobile application).
- Therefore it is not possible to initiate an encrypted communication to any other host than the one presenting the correct certificate and its fingerprint.
- This security system would even work without any **Certificate Authority** involved.

# MODULE 5

Secure Coding

The most common evasive techniques used by the modern malwares includes Obfuscation, Packers, Cryptors

- The Packers are used to compress the actual code size and to avoid reverse engineering the executable.
- The Protector is also an obfuscation technique used to perform multiple encryption and decryption to pack the same code using polymorphic encryption scheme.
- Cryptor is self-encryption to defend itself

Packer Complexity	Packer Behavior
Type I	Single layer with unpacker
Type II	Multiple layer with unpacker
Type III	Multiple layer with complex loops
Type IV	Single or multi-layer with trigger to other packer functionality
Type V	Multiple layer with Tail jump
Type VI	Complex layer with single fragment unpacker

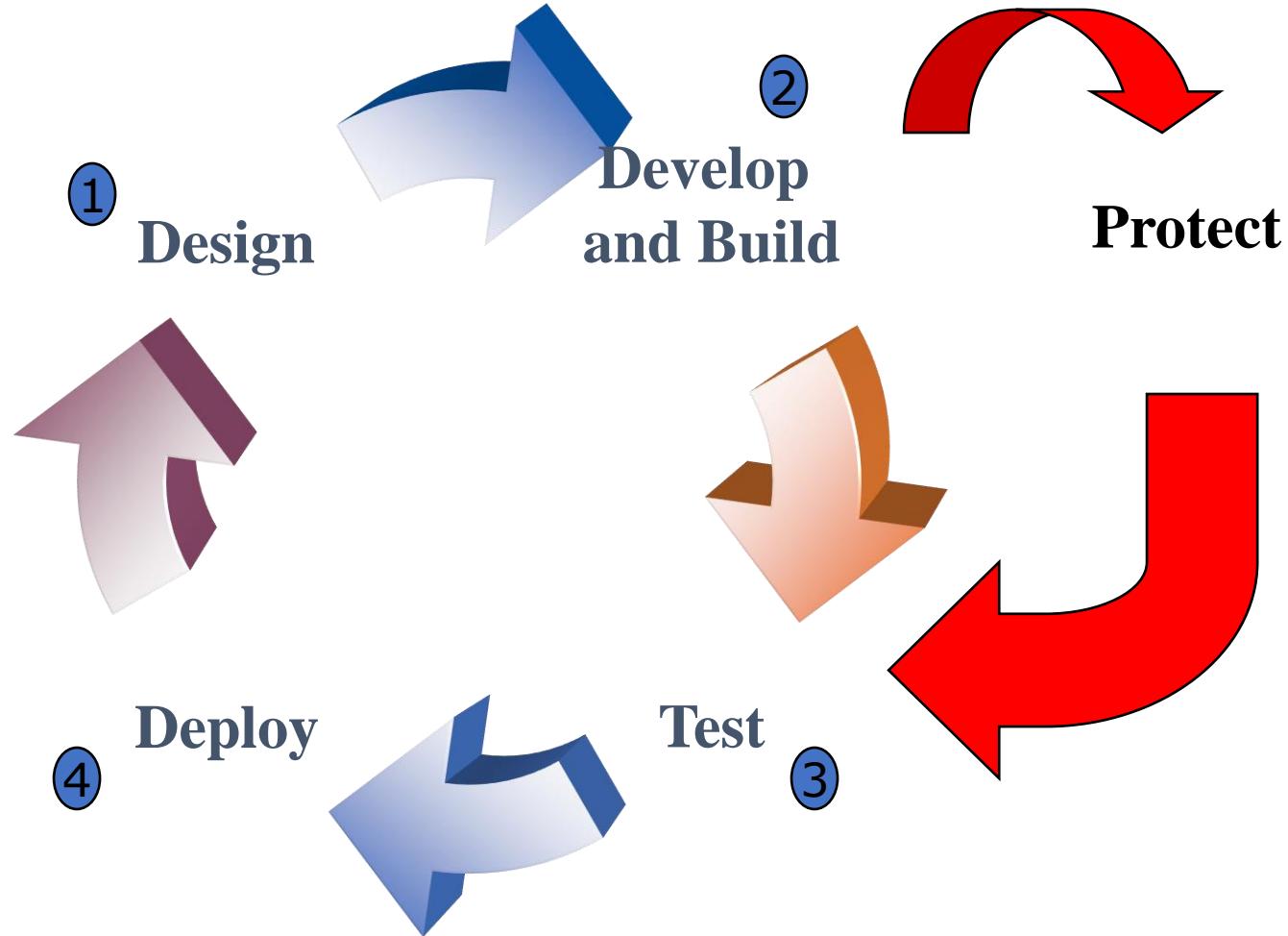
# Obfuscation

- To make something less clear and harder to understand
- **Obfuscated code** is source or machine code that has been made difficult to understand.
- Programmers may deliberately obfuscate code to conceal its purpose or its logic to prevent tampering, deter reverse engineering.

# Obfuscator

- An obfuscator: An algorithm  $O$ , such that for any program  $P$ ,  $O(P)$  is a program such that
  - ✓  $O(P)$  has the same functionality as  $P$
  - ✓  $O(P)$  is infeasible to analyze/"reverse-engineer"

# Obfuscation in SDLC



# Why Obfuscation?

- Programs are easy to reverse engineer using decompilation.
- Attackers can use any decompiler to easily reverse engineer code.
- Once reverse engineered, Anyone can peruse the details of the software and create pirated copy of the same software.
- Obfuscation reduces the size of an executable.
- Obfuscation improves the application performance at runtime.
- Properly applied obfuscation increases protection against decompilation.

# Obfuscation features

- ✓ Renaming
- ✓ Control Flow Obfuscation
- ✓ String Encryption
- ✓ Linking
- ✓ Watermarking

# Obfuscation features - Renaming

- Renames as many methods as possible to a same name/different/default.
- For example
  - **Namespace hiding**
  - **Namespace hierarchy**
  - **Removing the namespace**

# Renaming

Original Name	New Name
Preemptive.Application.Main	Preemptive.Application.a
Preemptive.Application.LoadData	Preemptive.Application.b
Preemptive.Tools.BinaryTree	Preemptive.Tools.a
Preemptive.Tools.LinkedList	Preemptive.Tools.b

- Namespace hiding

Original Name	New Name
Preemptive.Application.Main	a.a.a
Preemptive.Application.LoadData	a.a.b
Preemptive.Tools.BinaryTree	a.b.a
Preemptive.Tools.LinkedList	a.b.b

## Namespace hierarchy

Original Name	New Name
Preemptive.Application.Main	a
Preemptive.Application.LoadData	b
Preemptive.Tools.BinaryTree	c
Preemptive.Tools.LinkedList	d

Default renaming

# Obfuscation features – Control Flow Obfuscation

Traditional control flow obfuscation **Introduces false conditions and other misleading constructs** in order to confuse and break decompilers.

- It destroys the code patterns.
- The end result is semantically equivalent to original.

# Obfuscation features – String Encryption

- No strings are encrypted unless you specifically include a method.
- The intention is that you will only want to encrypt strings in the sensitive parts of your application.

# Obfuscation features – Linking

- Also called **merging**, **Links multiple assemblies** into one or more **output assemblies**.

# Obfuscation features – Watermarking

Used to Embed data (copyright info/unique nos.) into applications, making them unique. This is one method that can be used to track unauthorized copies of your software back to the source.

## To watermark an application

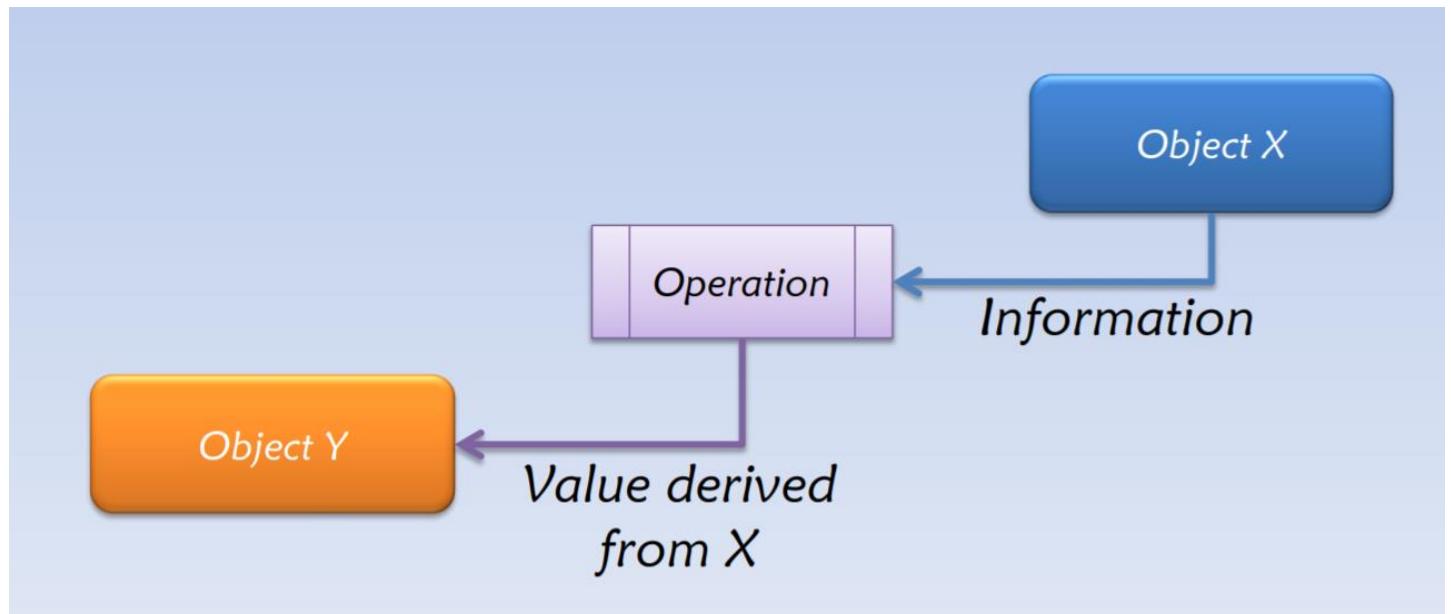
- Select the assemblies to watermark.
- Select whether the watermark string is to be encrypted and provide a passphrase if so.
- Provide a string and an encoding that will be the watermark.

# Taint analysis

- The **taint analysis** is a popular method which consists to **check which variables can be modified by the user input**.
- **All user input can be dangerous** if they aren't properly checked.
- **Taint analysis** is designed to increase security by preventing malicious users from executing commands on a host computer.

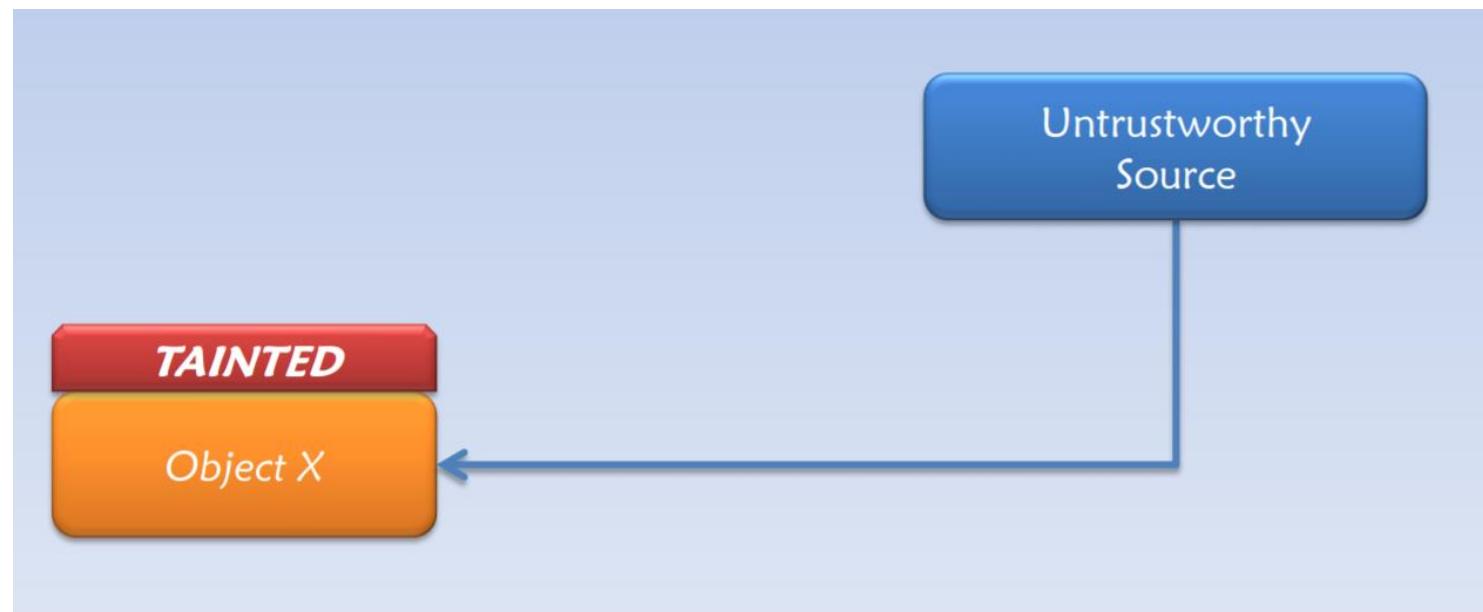
# Taint analysis – data flow

- Flow of data within:
  - An operation, or series of operations, that uses the value of some object, say x, to derive a value for another, say y, causes a flow from x to y



# Tainted objects

- If the source value of the object X is untrustworthy, we say that X is tainted.



# How Taint analysis works

- To “taint” user data is to insert some kind of **tag or label** for each **object** of the **user data**.
- The **tag allow us to track** the influence of the **tainted object** along the **execution of the program**.

## Taint analysis in exploit detection

- If we can track user data, we can detect if non-trusted data reaches a privileged location (any)
- Attacks which take advantages of user inputs can be easily prevented

# How Taint analysis works

## Steps:

1. Program execution **normally derived from trusted sources, not attacker input.**
2. Mark all input data to the computer as “tainted” (e.g., network, stdin, etc.)
3. Monitor program execution and track how tainted data propagates (follow bytes, arithmetic operations, jump addresses, etc.)
4. Detect when tainted data is used in dangerous ways

# MODULE 6

Hardware Security

# What we learnt so far

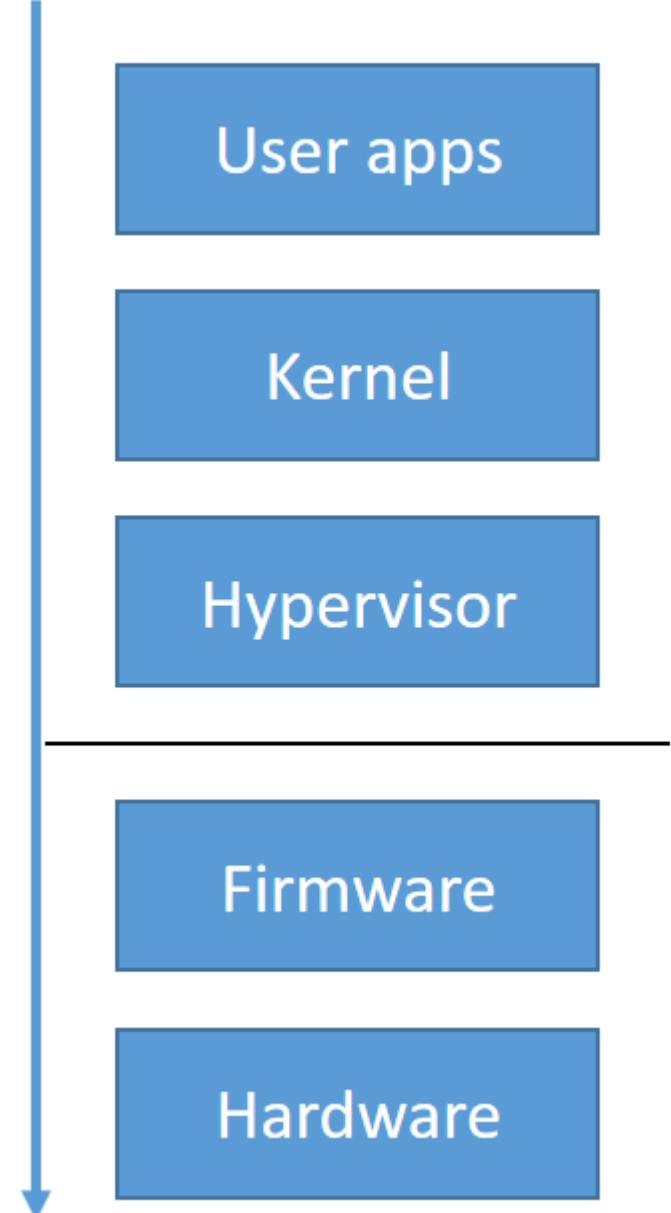
- Security Principles
- Vulnerabilities
- Security Validation
- Secure coding
- Application Hardening techniques

# Secure Coding - Hypothesis 1

- **Security of a computer system** has been traditionally related to the **security of the software** or the **information being processed**.

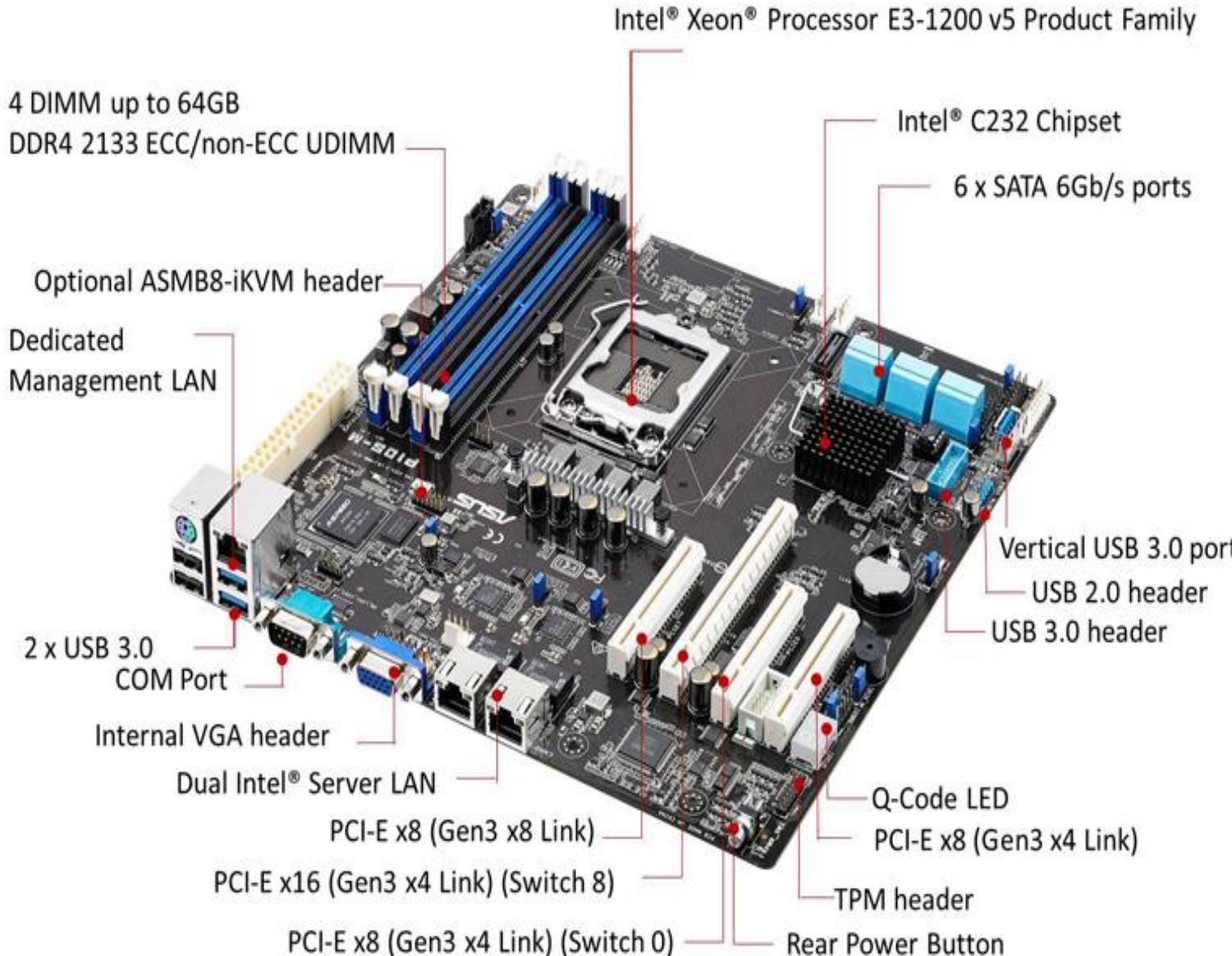
# Secure Coding - Hypothesis 2

- The **underlying hardware (Trusted Computing Base - TPM,DEP)** used for information processing has been considered trusted.

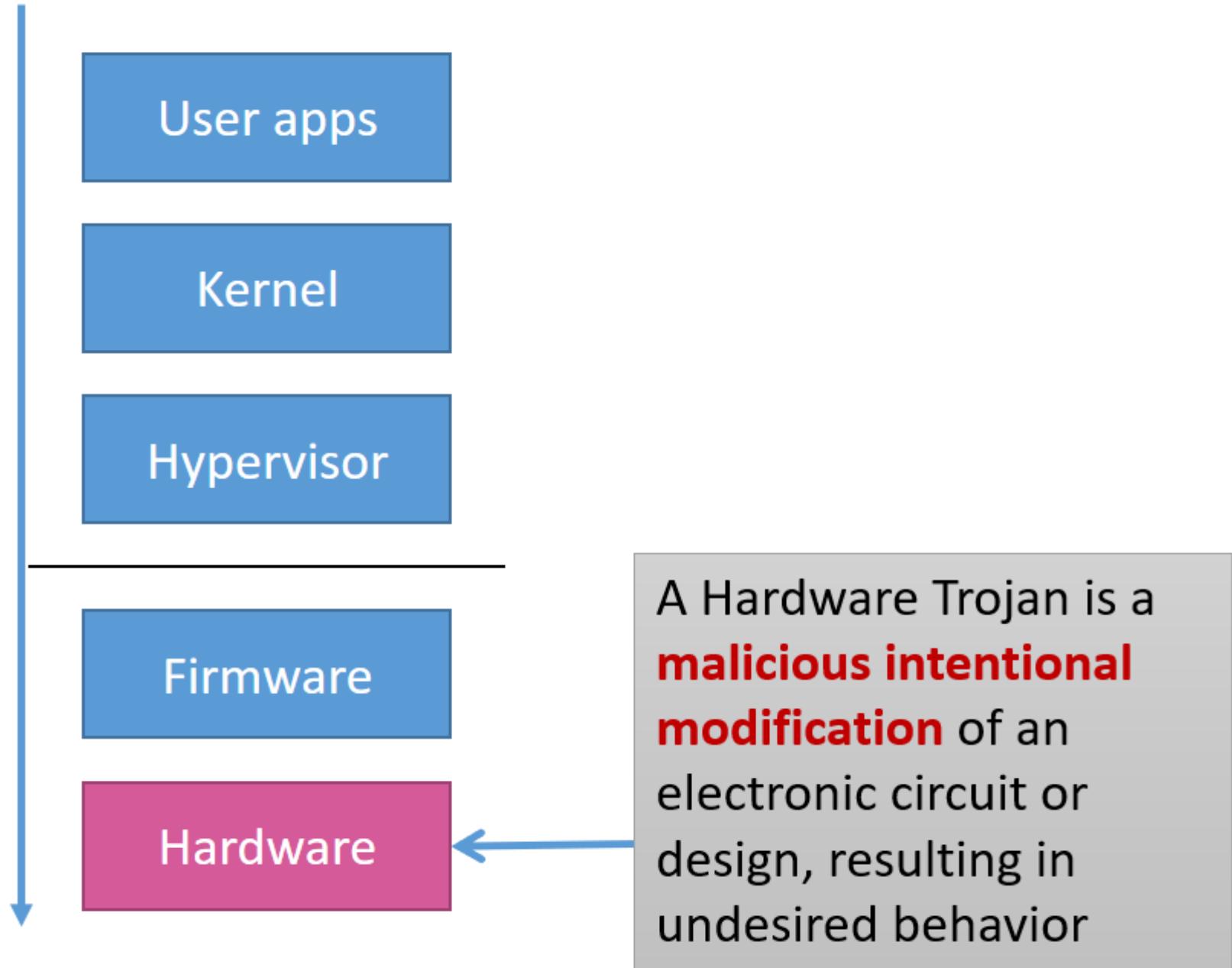


# Motherboard

- Integrated circuits
- Non integrated circuits



# Present day issue



# Malware Types



# Trojan Horse

- A Trojan horse, or Trojan, is a type of **malicious code or software** that looks **legitimate** but can take control of your computer.
- A Trojan is designed to damage, disrupt, steal, or in general to perform some other harmful action on your data or network.



# Common types of Trojan malware

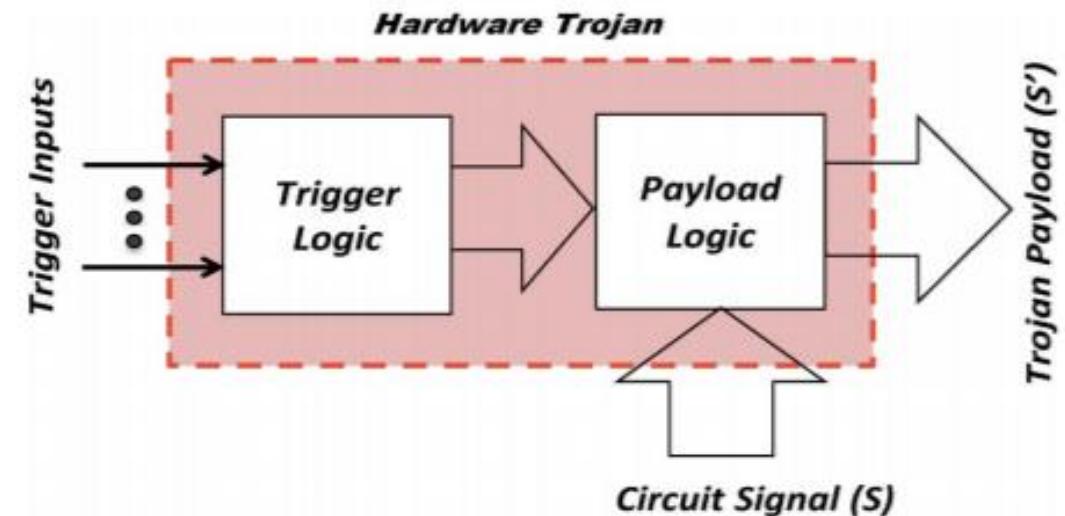
- Backdoor Trojan
- Distributed Denial of Service (DDoS) attack Trojan
- Fake AV Trojan
- Game-thief Trojan
- Info-stealer Trojan
- Ransom Trojan
- Remote Access Trojan
- Trojan banker

# Problem with existing system

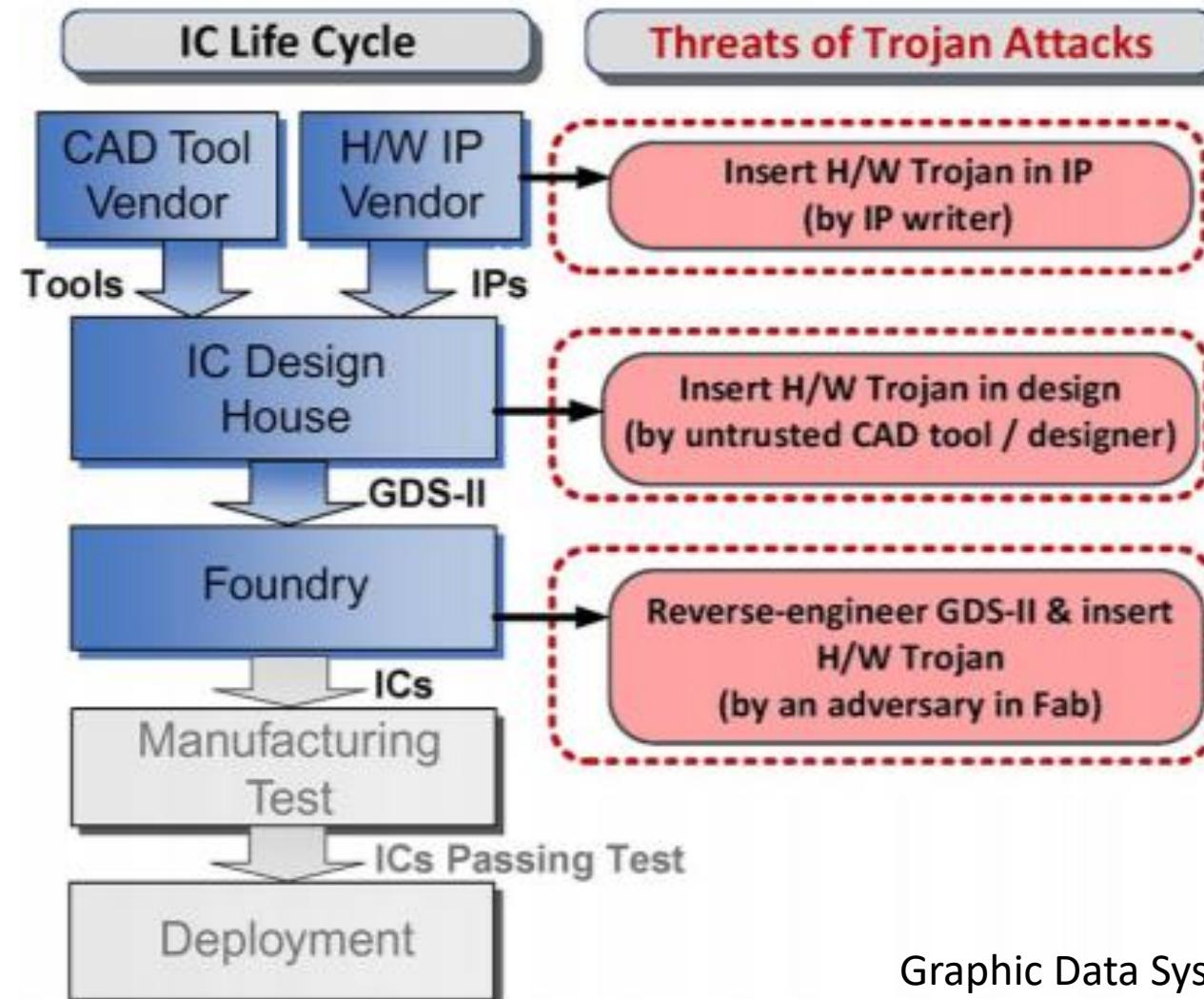
- Today's modern operating systems have a very large and complex trusted computing base (like TPM, DEP etc.). Explain why this is not a good idea! What could be the consequences?
- The underlying hardware used for information processing has been considered trusted.

# Hardware Trojan

- Any Addition or Modification of the circuit or a system with a malicious intention.
- Characteristics of Hardware Trojan
  - **Change or control functionality**
  - **Leak sensitive information**
  - **Reduce circuit reliability**



# Origination of Hardware Trojan in production environment



# Example of Hardware Trojan

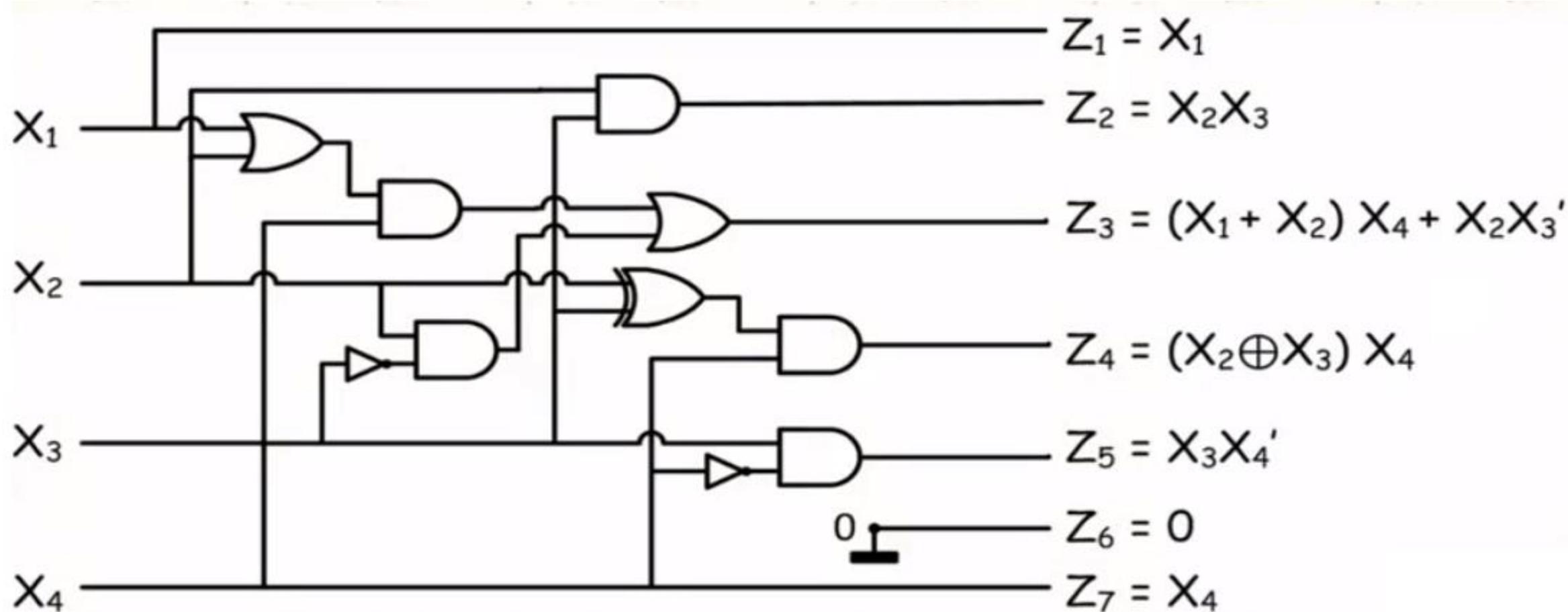
- Alice ask bob to design a circuit that computes  $F(x)$  so she can authenticate the  $(x, f(x))$  pairs as **(username, password)**.
- Assume that  $f(x) = x^2$  for 10 different users  $x = 0, \dots, 9$
- Username (number) and password is square of the number
- Bob's design
- Input :  $\{x_1, x_2, x_3, x_4\}$  // binary bit for 10 users
- Output:  $\{z_1, z_2, z_3, z_4, z_5, z_6, z_7\}$  // binary bit for 10 user's password
- Functionality:  $z = f(x) = x^2$



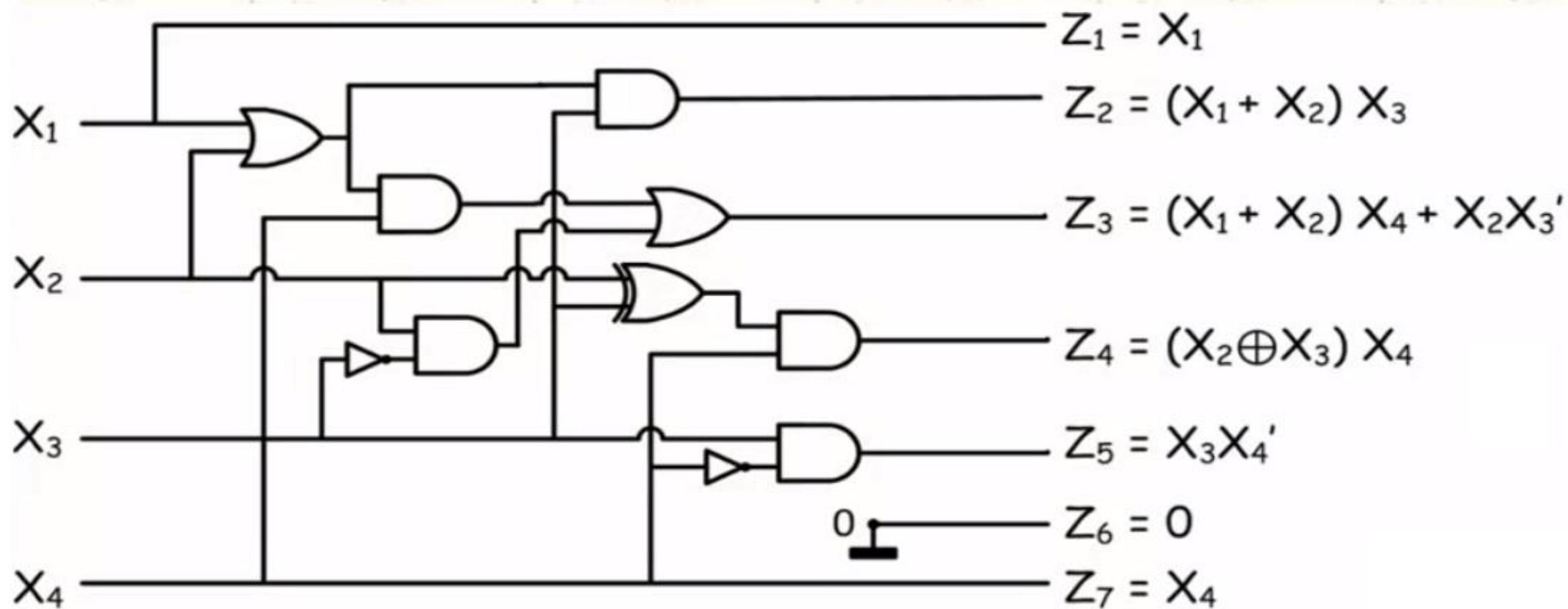
# Exact design requirement (z)

$X_1$	$X_2$	$X_3$	$X_4$	$X$	$X^2$	$Z_1$	$Z_2$	$Z_3$	$Z_4$	$Z_5$	$Z_6$	$Z_7$
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	1
0	0	1	0	2	4	0	0	0	0	1	0	0
0	0	1	1	3	9	0	0	0	1	0	0	1
0	1	0	0	4	16	0	0	1	0	0	0	0
0	1	0	1	5	25	0	0	1	1	0	0	1
0	1	1	0	6	36	0	1	0	0	1	0	0
0	1	1	1	7	49	0	1	1	0	0	0	1
1	0	0	0	8	64	1	0	0	0	0	0	0
1	0	0	1	9	81	1	0	1	0	0	0	1

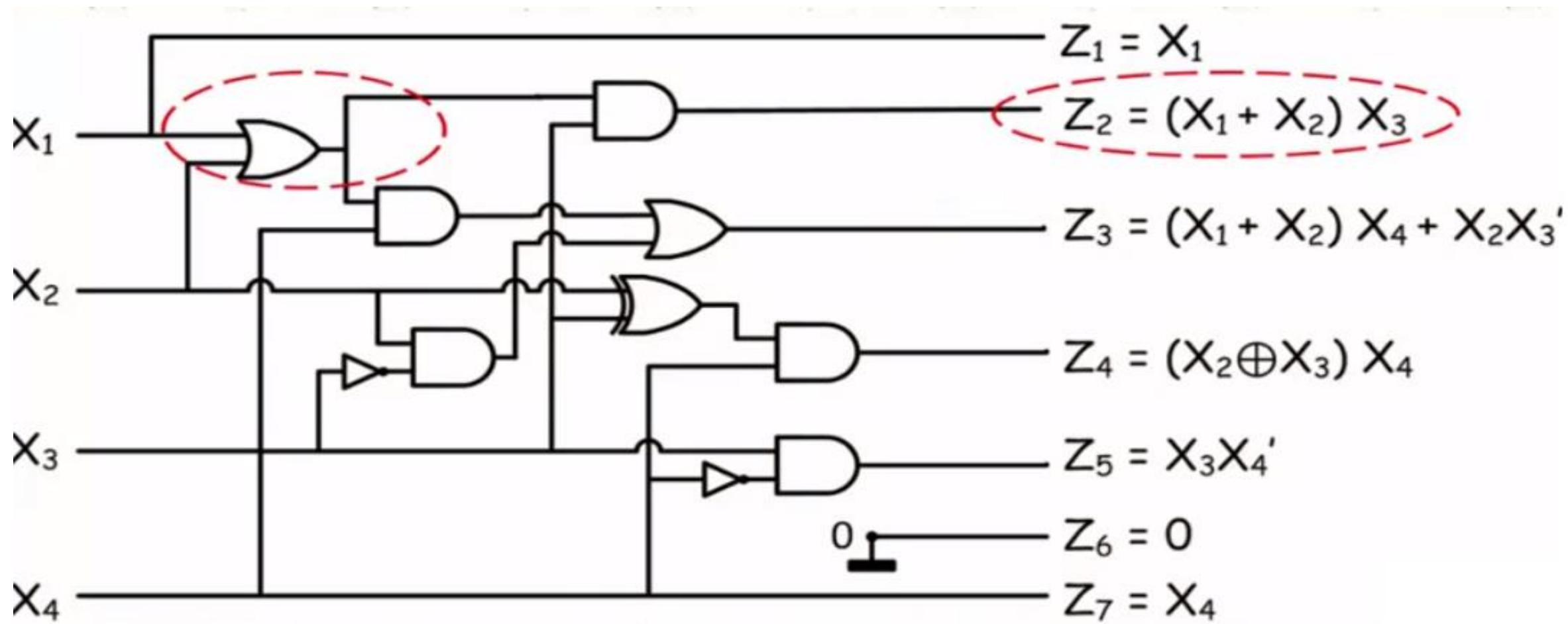
# Circuit design 1



# Circuit design 2



# Circuit design – Modified with malicious intention



What happens if the circuit has a backdoor in its design and accepts 10<sup>th</sup> user and so on.

- If the design is perfect, it should not be in the position to accept
- Circuit design 1
- User input : 10 & 11

$$\begin{array}{ll} \# Z_1 = X_1 & \# Z_5 = X_3 X_4' \\ \# Z_2 = X_2 X_3 & \# Z_6 = 0 \\ \# Z_3 = (X_1 + X_2) X_4 + X_2 X_3' & \# Z_7 = X_4 \\ \# Z_4 = (X_2 \oplus X_3) X_4 & \end{array}$$

$X_1$	$X_2$	$X_3$	$X_4$	$\times$	$F(X)$	$Z_1$	$Z_2$	$Z_3$	$Z_4$	$Z_5$	$Z_6$	$Z_7$
1	0	1	0	10	68	1	0	0	0	1	0	0
1	0	1	1	11	89	1	0	1	1	0	0	1

## Circuit design 2, User input : 10 & 11

# $Z_1 = X_1$	# $Z_5 = X_3 X_4'$
# $Z_2 = (X_1 + X_2) X_3$	# $Z_6 = 0$
# $Z_3 = (X_1 + X_2) X_4 + X_2 X_3'$	# $Z_7 = X_4$
# $Z_4 = (X_2 \oplus X_3) X_4$	

$X_1$	$X_2$	$X_3$	$X_4$	$\times$	$F(X)$	$Z_1$	$Z_2$	$Z_3$	$Z_4$	$Z_5$	$Z_6$	$Z_7$
1	0	1	0	10	100	1	1	0	0	1	0	0
1	0	1	1	11	121	1	1	1	1	0	0	1

# Physical attacks

- Understanding the vulnerabilities and threats to a system from hardware.
- **Requirement**
  - **Access to the chip**
  - **Connection to signal wires (making mal-connection, cut wire etc.)**
  - Equipment, tools, skills and knowledge (hardware specific, cryptographic specific)

# Physical attacks

- **Two phases**
  - **Interaction:** Attacker exploits some **physical characteristics of the device.**
  - **Exploitation:** Analyzing the gathered information to recover the secret

# Physical attacks vs network or software attacks

- **Compared to network or software attacks**
  - Physical attacks have **higher requirement**
    - **Physical access** to the system
  - **Specialized equipment, tools and knowledge**
    - Physical attacks are harder to launch

# Physical attacks : Attackers

- **Class I : Clever outsider**
  - Insufficient knowledge of the system
  - Limited access to the system
- **Class II: Knowledgeable Insider**
  - Knowledge of the system
  - Access to the tools and organization
- **Class III : Funded organization**
  - Access to all resources

# What we are going to learn today

- Attack motivation
- Attack categories
- Attack model
- Attack classification

# Physical attacks: Motivation

- **Ultimate motivation is money**
- **Direct theft of service or money**
  - Smart card, TV set up box, game console
- **Sell/ re-sell of products**
  - IP piracy, Cloning (device)
- **Interrupting or DoS**
  - Competitor's device

# Physical attack categories

- Invasive attacks
- Non-invasive attacks
- Semi-invasive attacks

# Physical attack categories – Invasive attacks

- **Direct access** to inside of the chip/device
- Reversible
- Device damaged or tamper evidence left
- Challenges
- **Cost to launch such attacks are high and requires high knowledge and skills**

# Physical attack categories – Non-Invasive attacks

- **Interacts with the device/ chip through its interface**  
(voltage, current, IO, clock, fan speed, hdd sound)
- **Passive vs active**
- **No device damage**
- No tamper evidence
- Low cost and repeatable
- Challenges
- **Too slow, success rate is very low**

# Physical attack categories – Semi-Invasive attacks

- **Access to the surface of the chip**
- **Normally does not damage the system**
- May or may not leave the **tamper evidence**
- Moderate cost (depends on the intact of the system: high or low)
- **Challenges**
- Requires high knowledge and skills
- Repeatable

# Physical attack model : Invasive attacks

- **Decapsulation and de-processing**
  - Remove the package to expose the silicon die
- **Reverse engineering**
  - Reveal the inner structure and functionality
- **Micro-probing**
- **Chip modification**

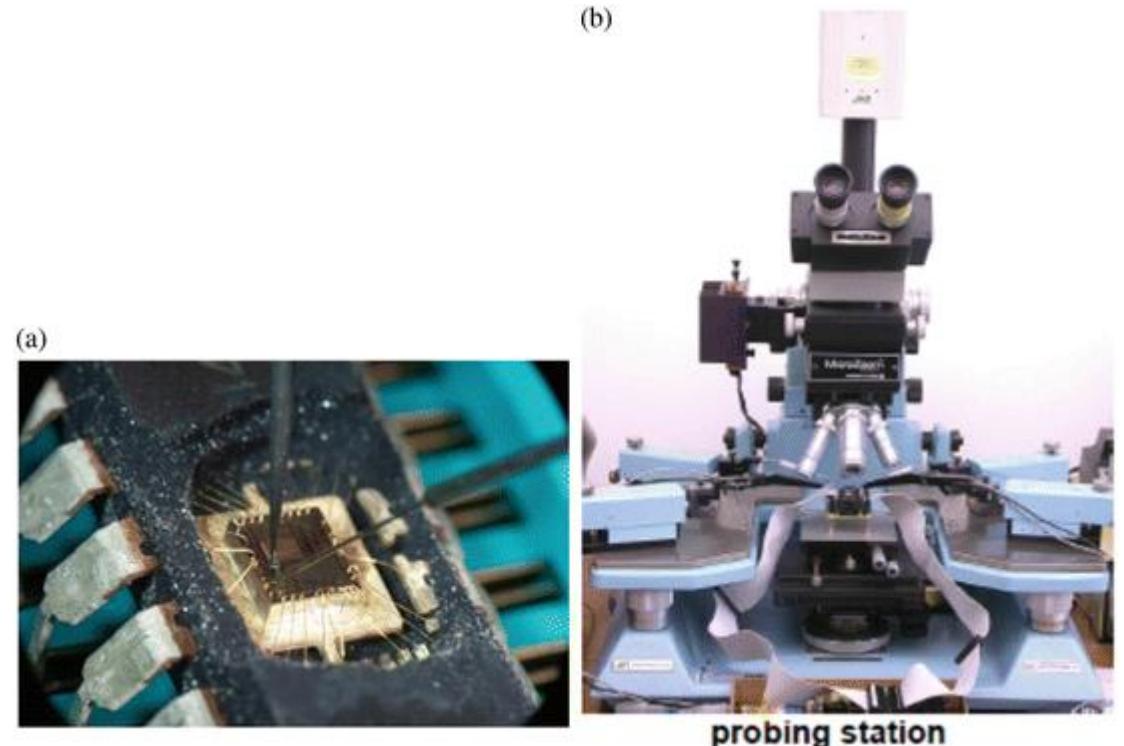
# Physical attacks: classification

- **Reverse engineering** (invasive)
  - Study chip's **inner structure** and **functionality**
  - High cost, similar capability of the **designer**.



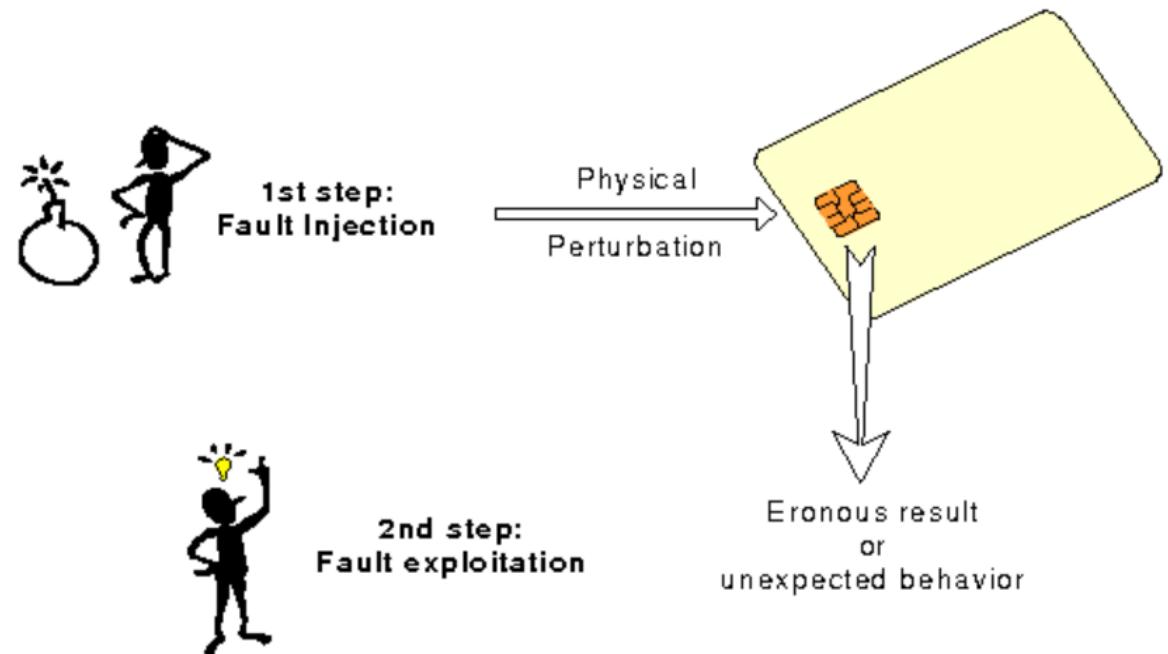
# Physical attacks: classification

- **Micro-probing** (invasive)
- Directly access the chip interface
- Observe, manipulate, interfere with the chip/chip data
  - **Eavesdropping** on signals inside a chip
  - **Injection of test signals** and analyzing the Behaviour.
  - Used for extraction of **secret key** and **memory contents**.



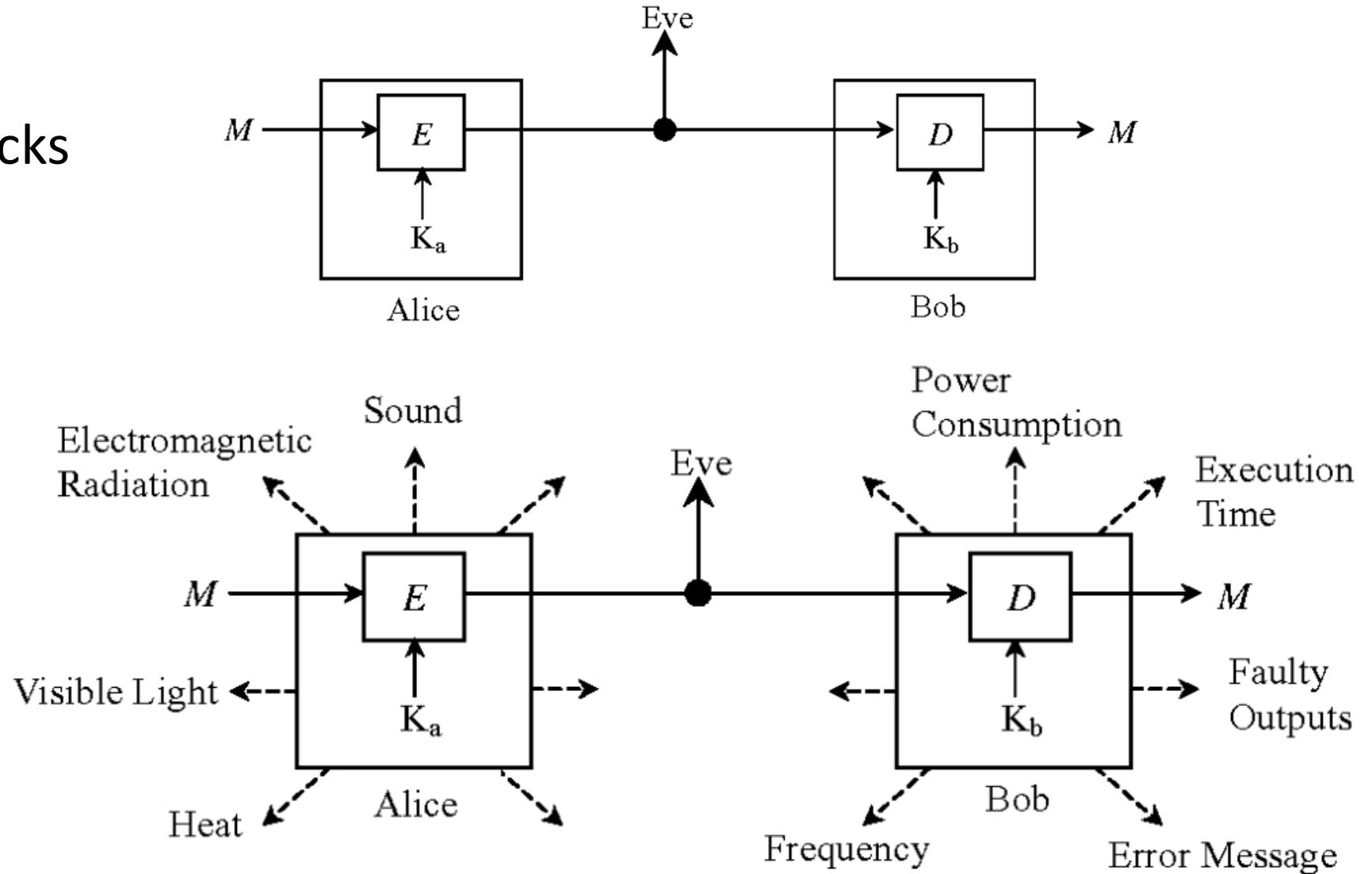
# Physical attacks: classification

- **Fault generation** (semi-invasive or non-invasive)
  - **Faulty data** are given to the chip
    - **Run in abnormal condition**
  - **Chip malfunctioning** and information leakage
  - **Repeatable**



# Physical attacks: classification

- Side channel attacks



Cryptography is not only solution to protect your data (based on CIA)

Breaking a cryptographic algorithm?  
**Impossible**

Suppose if we want to break a cryptosystem

- **Cryptanalysis**
  - Mathematic properties of a cryptographic system

# Side Channel attacks

- A malware “Fansmitter”
- Hijacks the computer's fans to transmit data.
- **Limitation**
  - For a **computer** that is **air gapped**, infecting it with a **malware isn't so easy**.
  - Infecting it usually requires **some sort of physical access** and using **infected USBs** have been a common form of getting this required access.
  - **Fansmitter malware** is extremely slow and can extract data at a rate of only 900 bits per hour

CLEVER ATTACK USES THE SOUND OF A COMPUTER'S FAN TO STEAL DATA

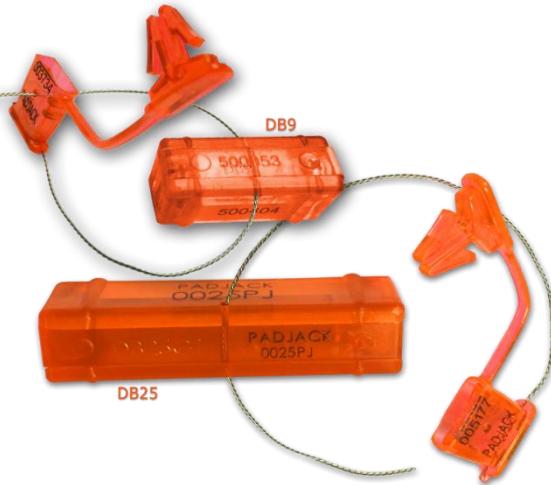


# Top four attack methods to steal data from air-gapped network

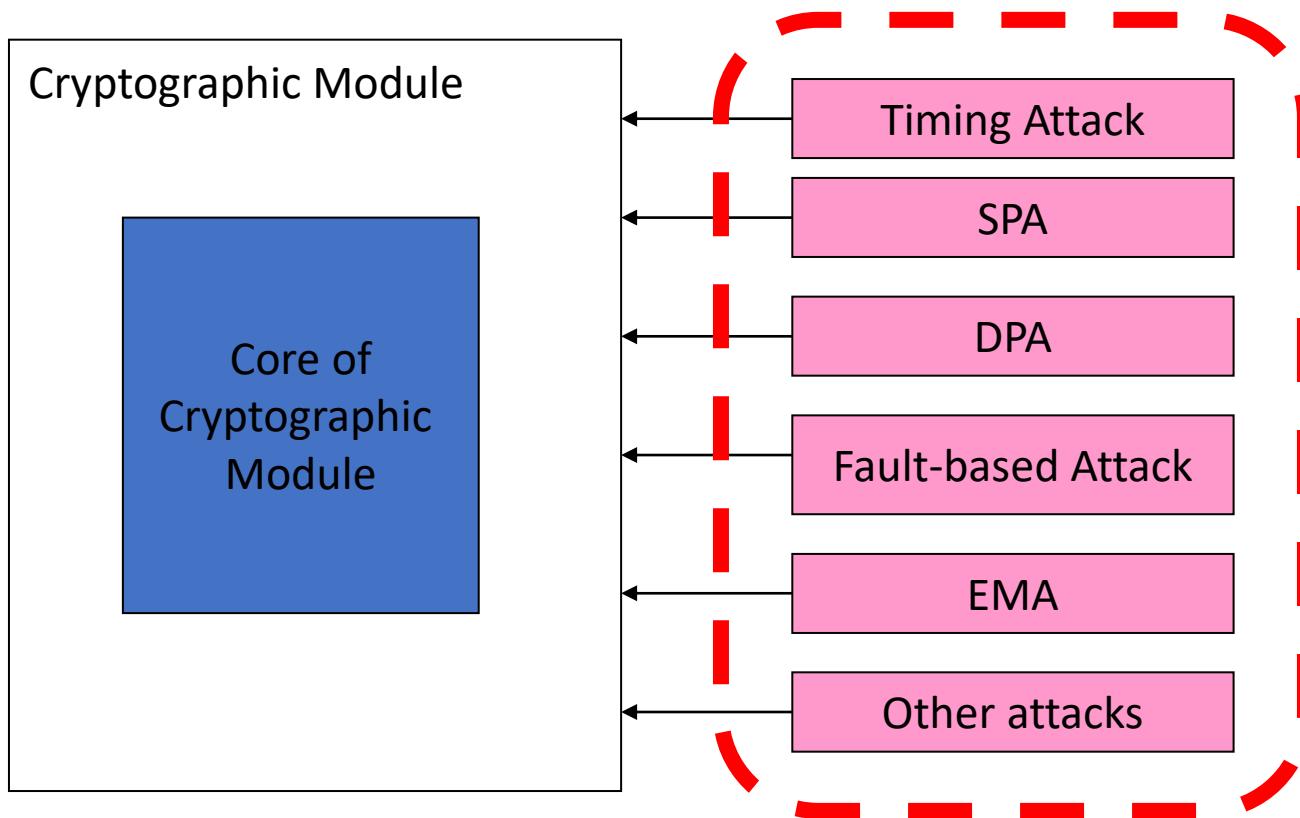
- Electromagnetic
  - **EM radiation from the memory bus**
  - leakage from **USB ports and cables**
- Acoustic
  - Speakers
  - Hard disk noise
- Thermal (very low speeds possible)
  - Power analysis attacks
- Optical

# Tamper Resistance

- High Physical Security
  - Locks
  - Biometrics etc.
- Cryptographic module
  - Disk protection
  - RAM protection

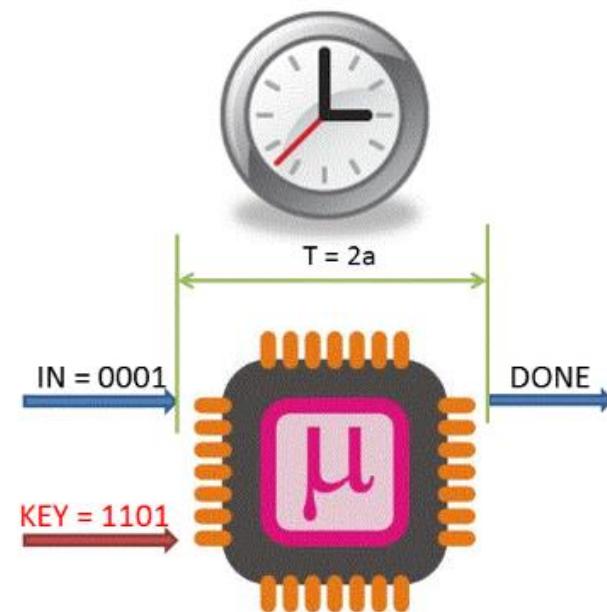


# Possible physical attacks



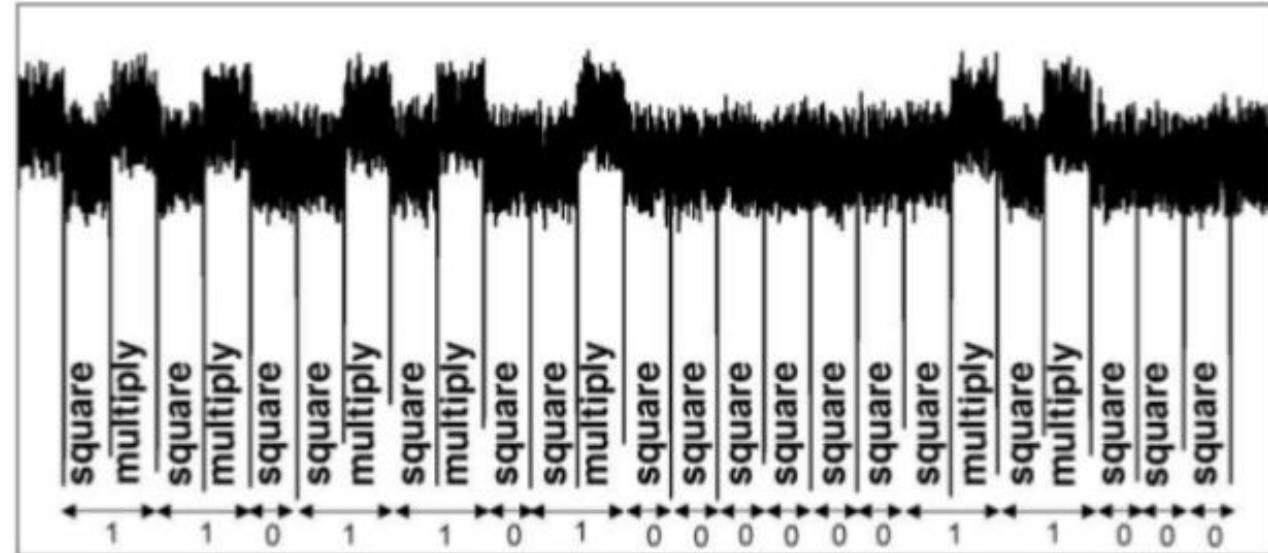
# Timing attack

- A **timing attack** is a side channel **attack** in which the attacker attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms.



# SPA attack

- Simple power analysis (**SPA**) is a side-channel **attack** which involves visual examination of graphs of the current used by a device over time.
  - Simple power analysis (**SPA**) is a side-channel **attack** which involves visual examination of graphs of the current used by a device over time.

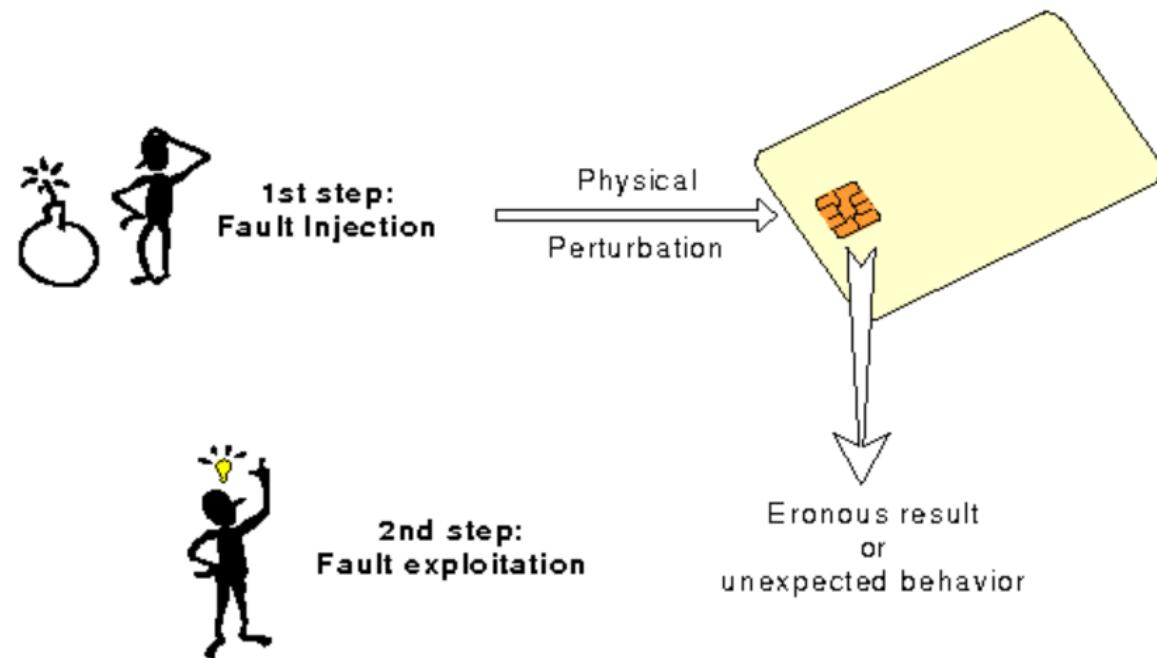


# DPA attacks

- Differential power analysis (**DPA**) is a **side-channel attack** which involves **statistically analyzing power consumption** measurements from a cryptosystem.
- The **attack** exploits varying power consumption of microprocessors or other hardware while performing operations using secret keys.

# Fault-based attack

- **Faulty data** are given to the chip
  - Run in **abnormal condition**
- **Chip malfunctioning** and information leakage



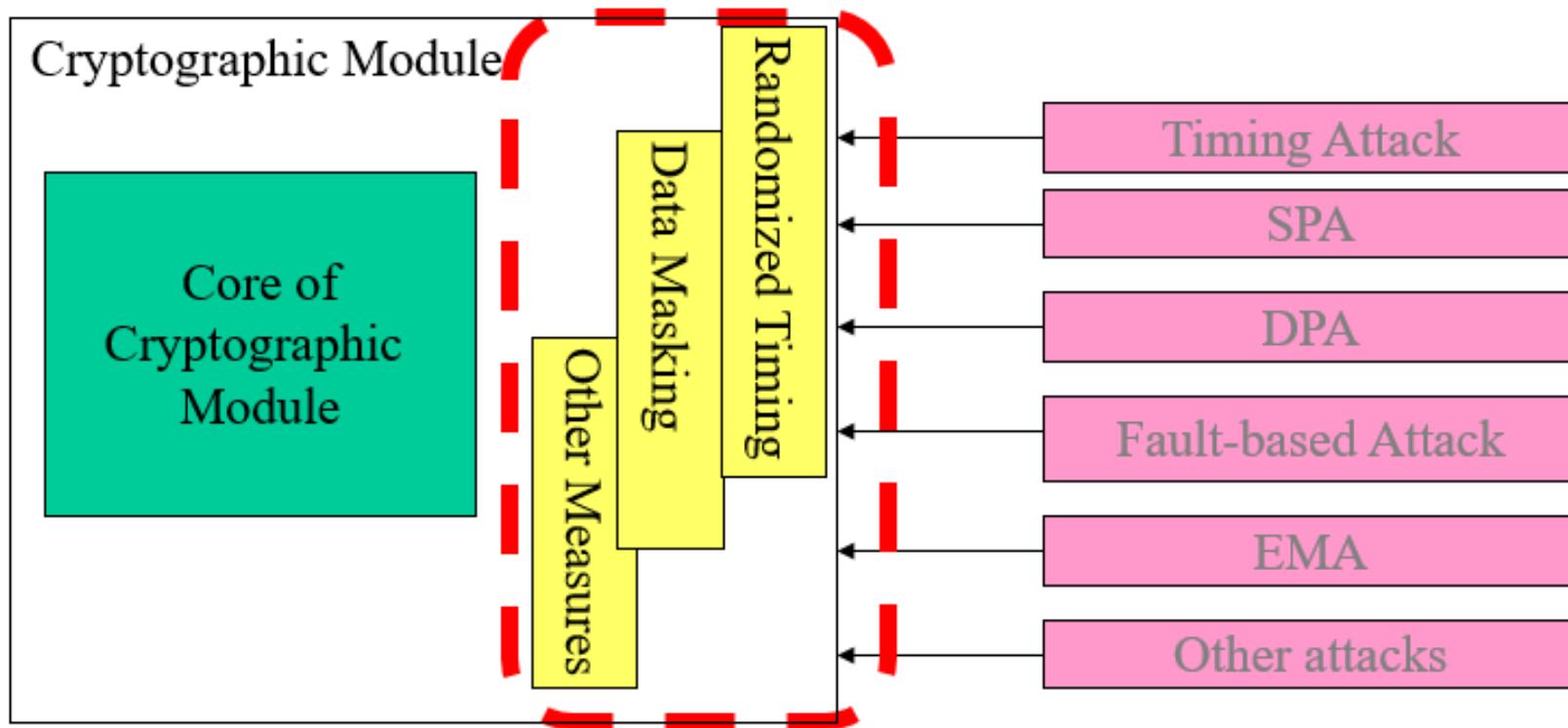
# Electro-magnetic attack

- In cryptography, **electromagnetic attacks** are side-channel attacks performed by measuring the electromagnetic radiation emitted from a device and performing signal analysis on it.

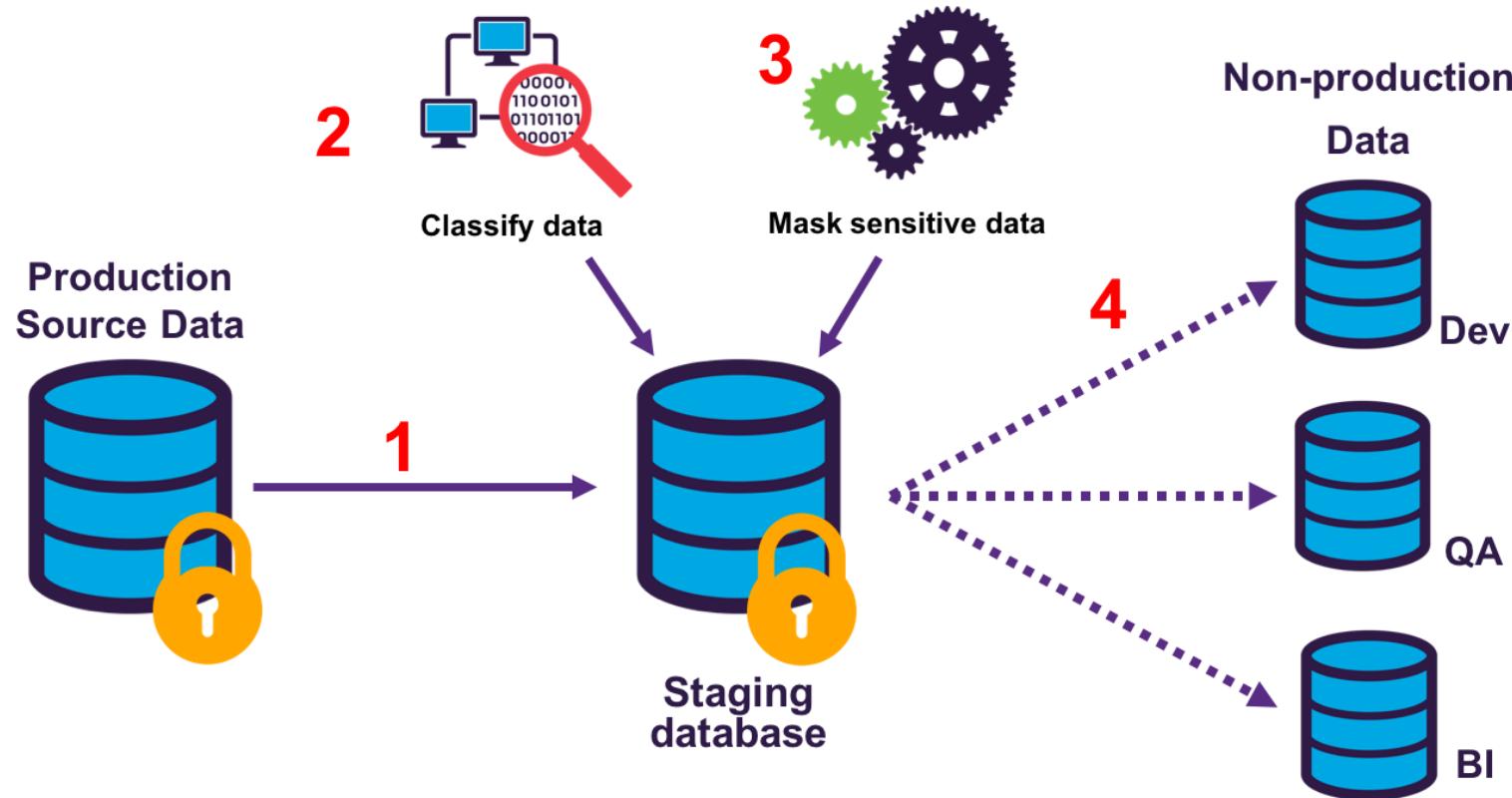
# Other attacks

- **Cold boot attacks**
  - A **cold boot attack** (or a **platform reset attack**) is a type of **side channel attack** in which an attacker with **physical access** to a computer performs a **memory dump** of a computer's **random access memory** by performing a **hard reset of the target machine**.
  - The attack relies on the **data remanence** property of **DRAM** and **SRAM** to retrieve memory contents that **remain readable** in the seconds to minutes after power has been removed.

# Possible Countermeasures



# Data masking (data security)

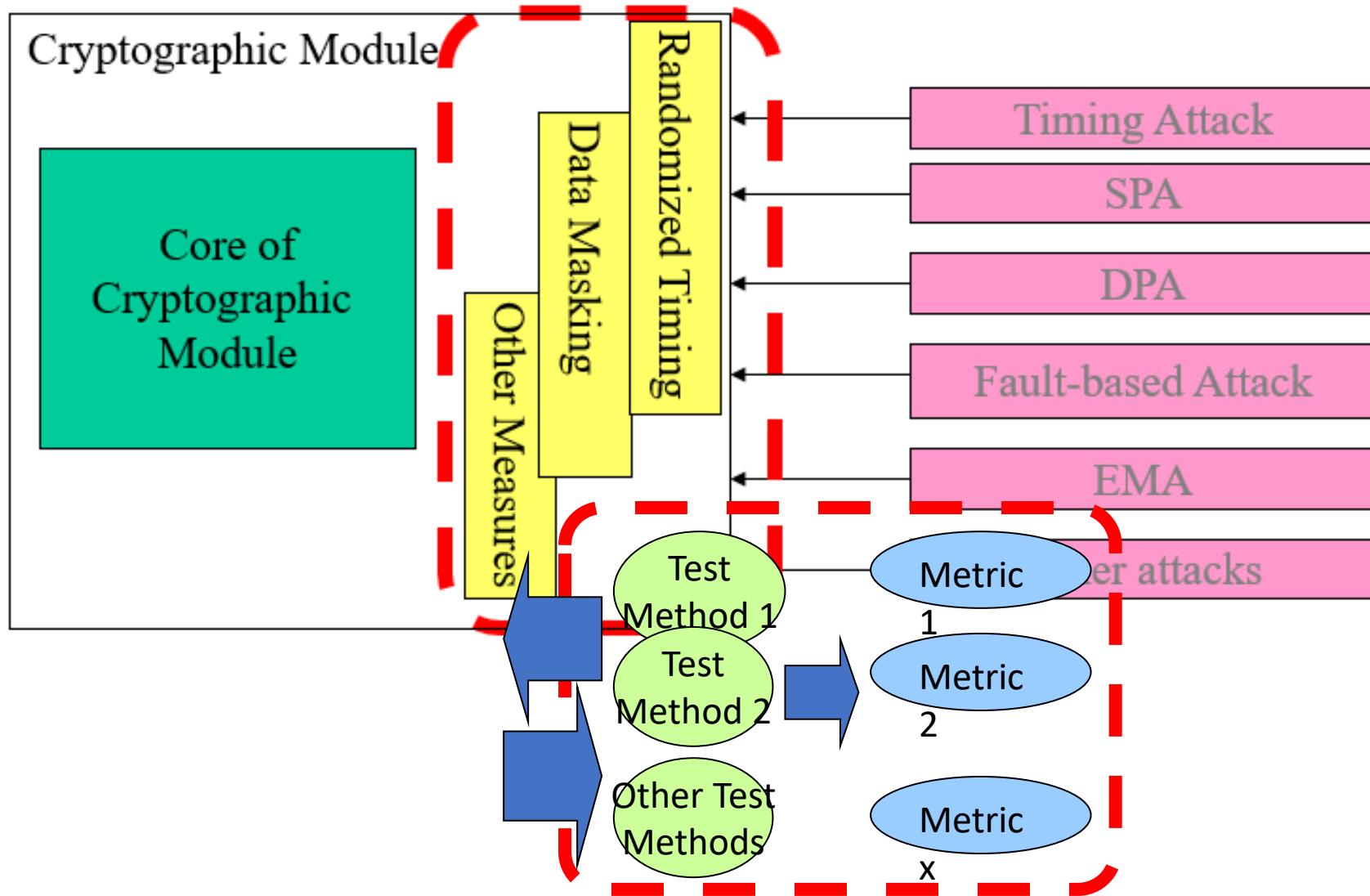


- Can you give any real time example for data masking?

# Real time example – Data masking

- Forgot password
- OTP msg

# Possible Countermeasures – cont'd...



# Immutable Technology

- Immutable storage is storage whose content cannot be changed once it has been written.
- Works based on Write-Once-Read-Many times (WORM) principle.
- **Categories**
  - Physical WORM technology or P-WORM
  - Embedded WORM technology or E-WORM
  - Software WORM technology or S-WORM

# Physical Immutable Technology

- The physical storage technologies include media that is by nature immutable and unchangeable.
  - Optical WORM
    - CD/ DVD - R
  - Magneto Optical WORM

# Embedded Immutable Technology

- Device driver and firmware
- Widely used in magnetic tapes and magnetic disks

# Software WORM technology

- Operating system provides immutable protection using mechanisms like capability-based schemes (**attrib** in windows/ **chattr** in linux).
- Works based on the privileges.
- System utilities like chattr and lsattr can be used to set the immutability flag and render a file impossible to modify.
- **Software WORM fails** when an attacker has elevated privileges.

# Drawback of existing countermeasures

- Ideal approach -- **if appropriate metrics and test method** are defined
- Searching for **appropriate metrics** is a big issue --- **Intensive research** is required

# Example Scenario



# Present day security implications

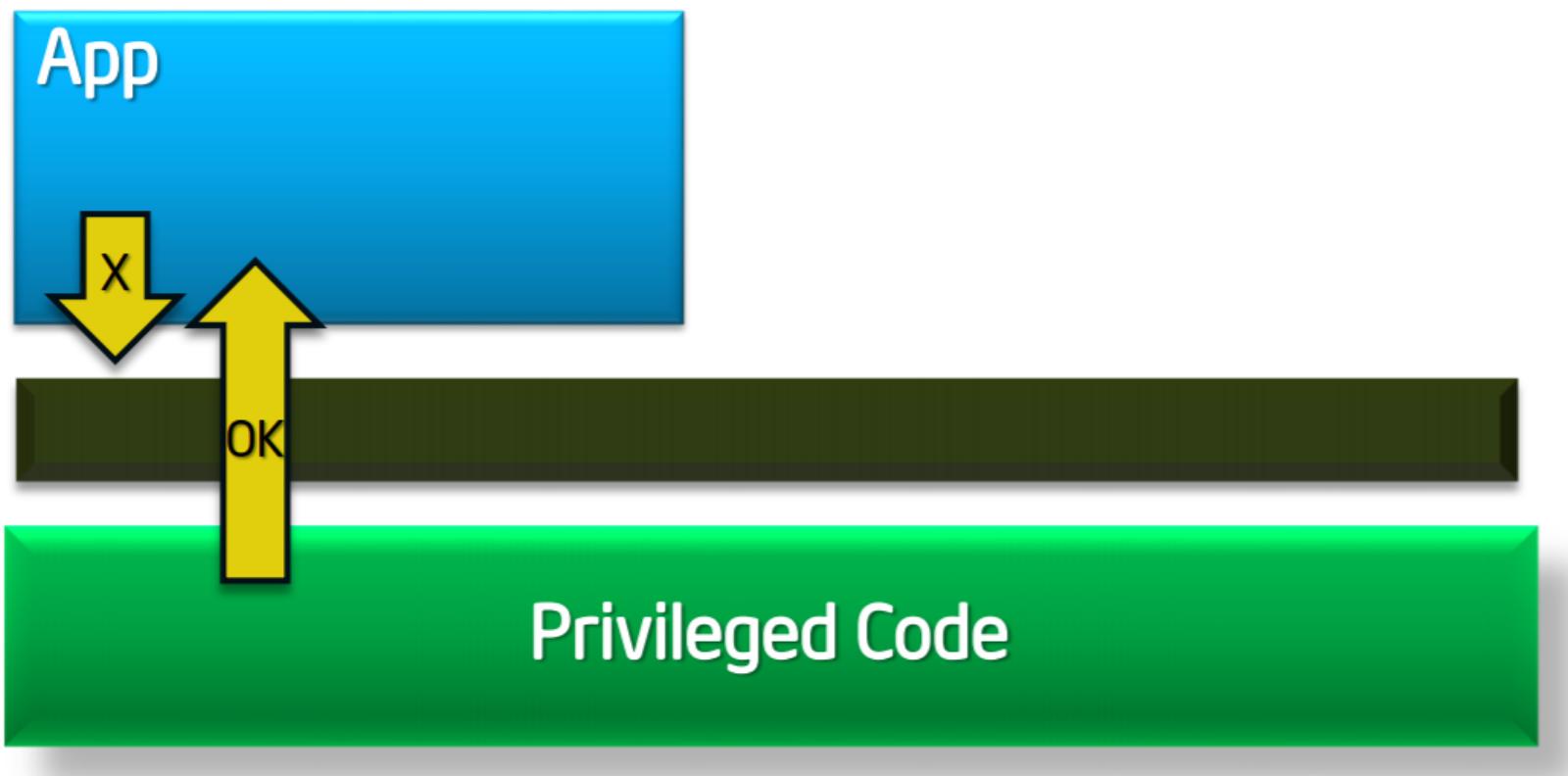
- **Hardware protected (encrypted) storage:**
  - Only “authorized” software can decrypt data
    - e.g.: **protecting key for decrypting file system**
- **Secure boot:** method to “authorize” software (OS)
- **Attestation:** Prove to remote server what software is running on my machine.

# Problem with present TCB

- TPM relies on BIOS.
- Third party.
- Resetting TPM is possible.
- Hard fact is **BIOS runs before any defenses.**
- **If not secure boot, BIOS is no longer protected.**
- **Legacy boot** leads the attacker to have a complete control over the BIOS.
- Attacker can disable TPM (possibly) at any point when the hardware has multi boot.

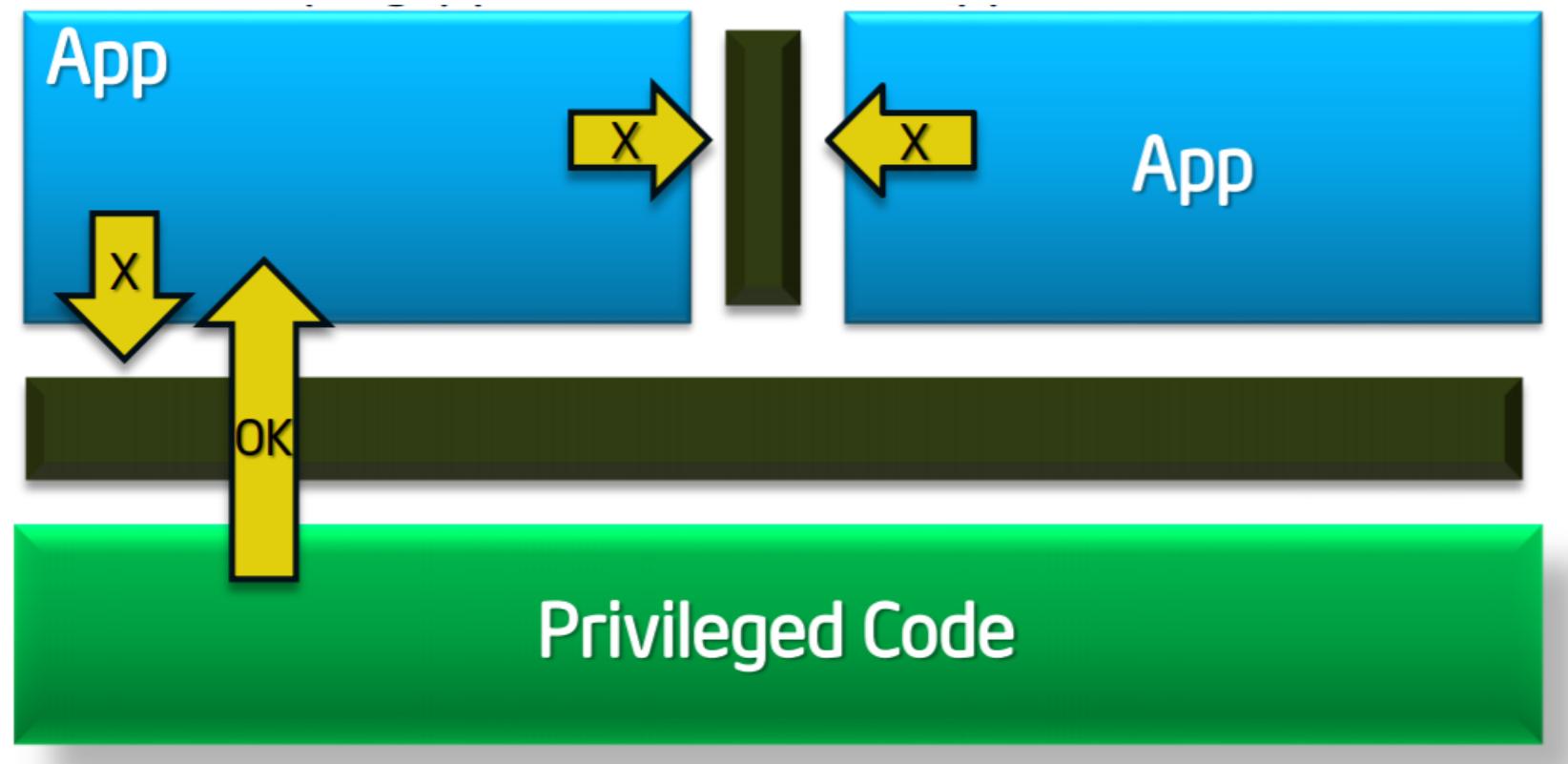
# Problem with present TCB.

- Protected Mode (rings) protects OS from apps



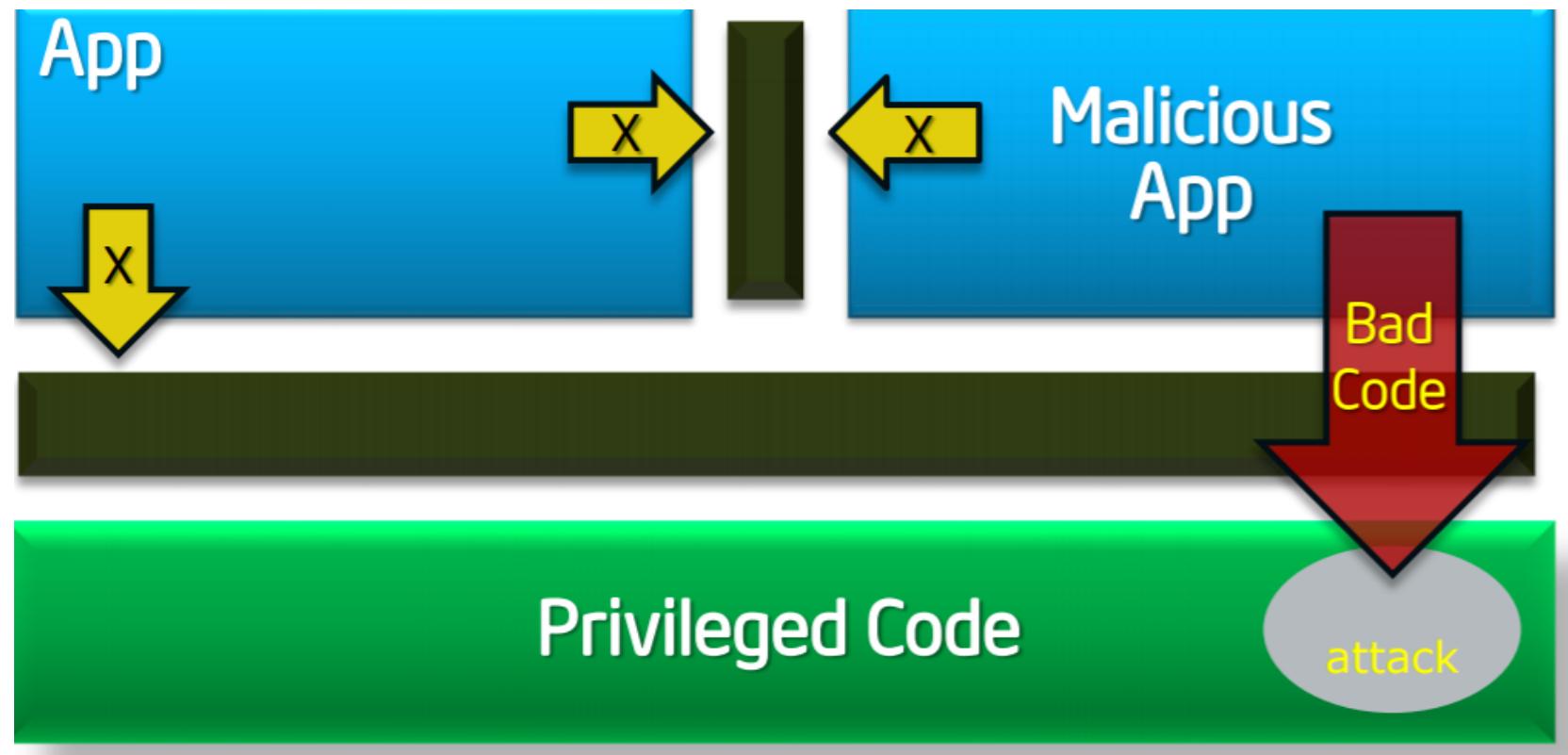
# Problem with present TCB - cont'd ...

- Protected Mode (rings) protects OS from apps
- Apps communicate with each other with respect to the privileges



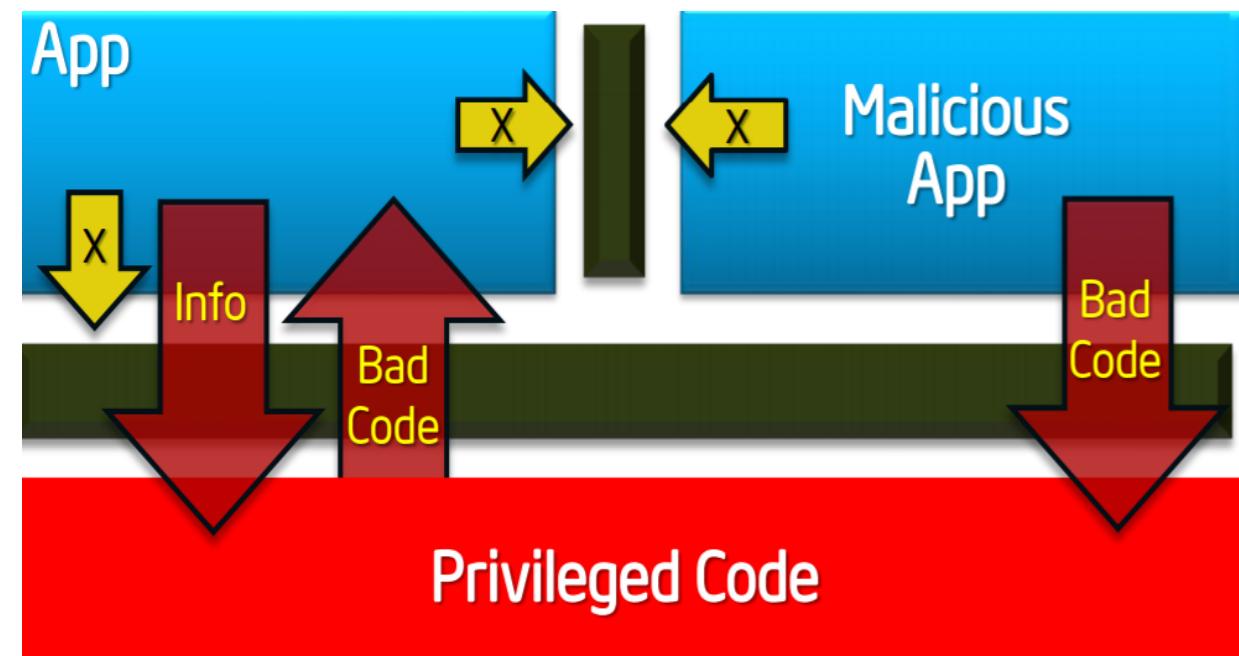
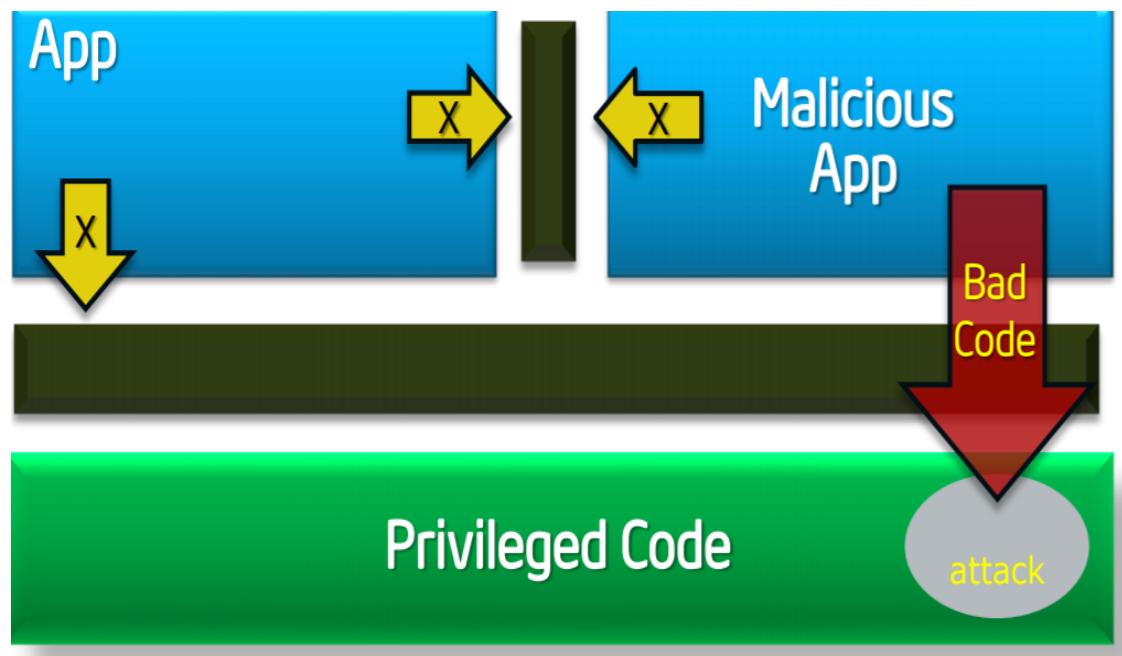
# Problem with present TCB - cont'd ...

- UNTIL a **malicious app exploits a flaw to gain full privileges and then tampers with the OS or other apps.**



# Problem with present TCB - cont'd ...

- UNTIL a **malicious app exploits a flaw to gain full privileges and then tampers with the OS or other apps.**



# What happens if your TCB is completely compromised?

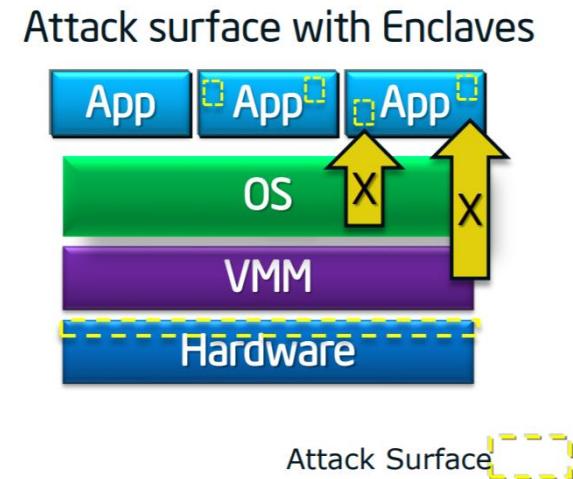
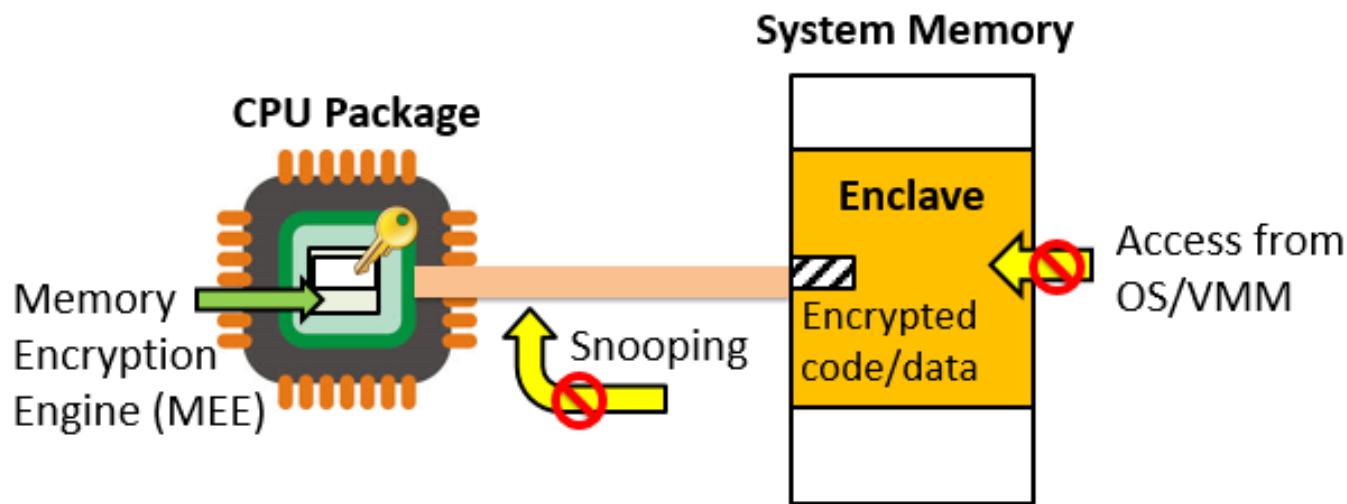
- What happens if a malware that subverts OS/VMM, BIOS, Drivers etc.?
- What happens if your TPM was compromised?

# Software Guard Extension

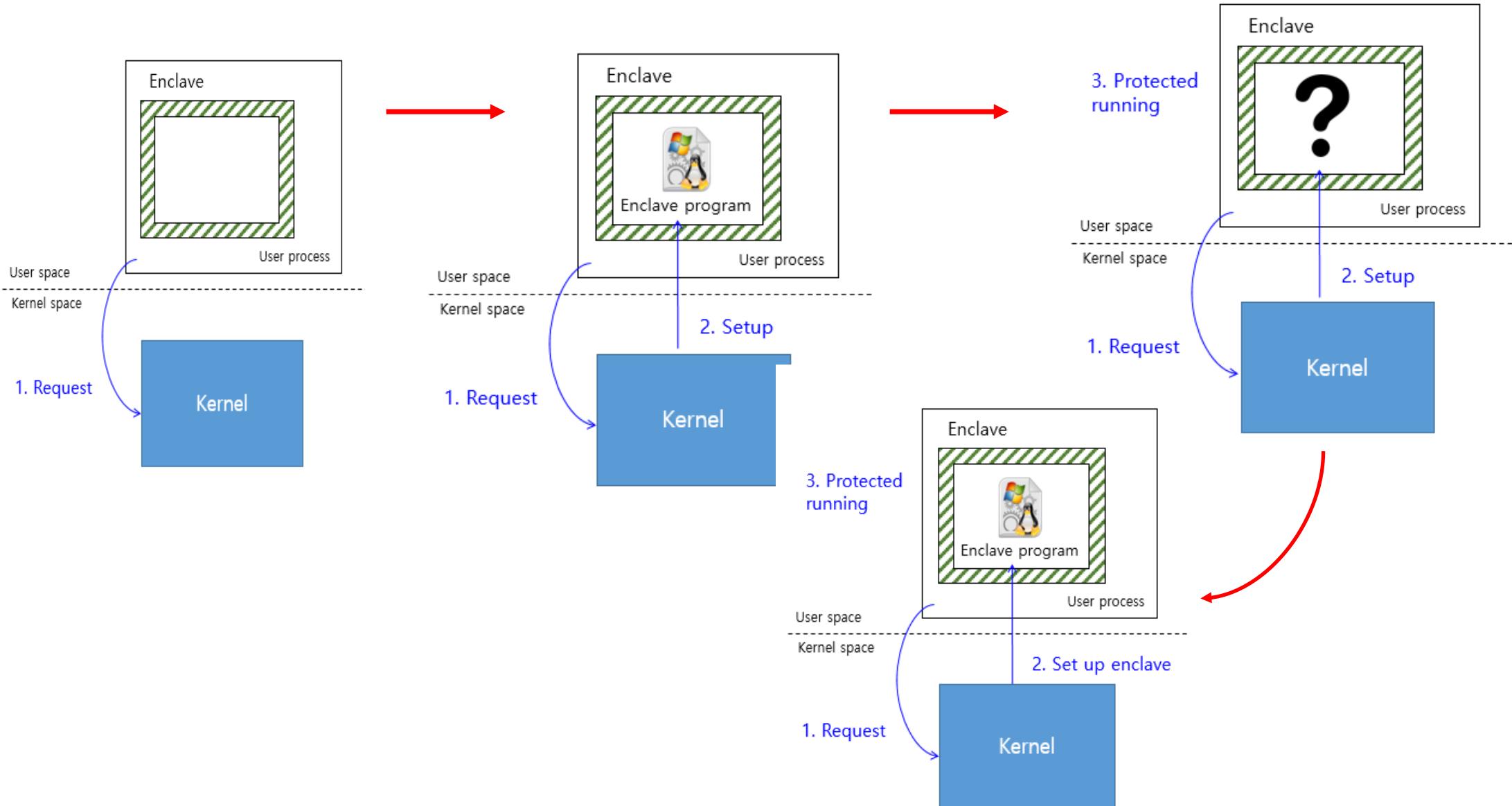
- SGX allows part of application code to run in isolation inside an enclave.
- The enclave region of the main memory is encrypted.
- The content is only decrypted inside the CPU package using processor specific keys.
- Thus, even if a malicious adversary has full control over the hardware, the adversary cannot access/modify the enclave.
- Also, the enclave is protected from other software running in the host, including the OS and hypervisor.

# SGX : Isolated Execution

- Application keeps its data/code inside the “**enclave**”
  - Protected from **all software stacks** including the kernel.
  - **CPU encrypts the data written in physical memory.**
  - **Smallest attack surface** by reducing TCB (App + processor)
  - Protect app’s secret from **untrusted privileged software** (e.g., OS, VMM)



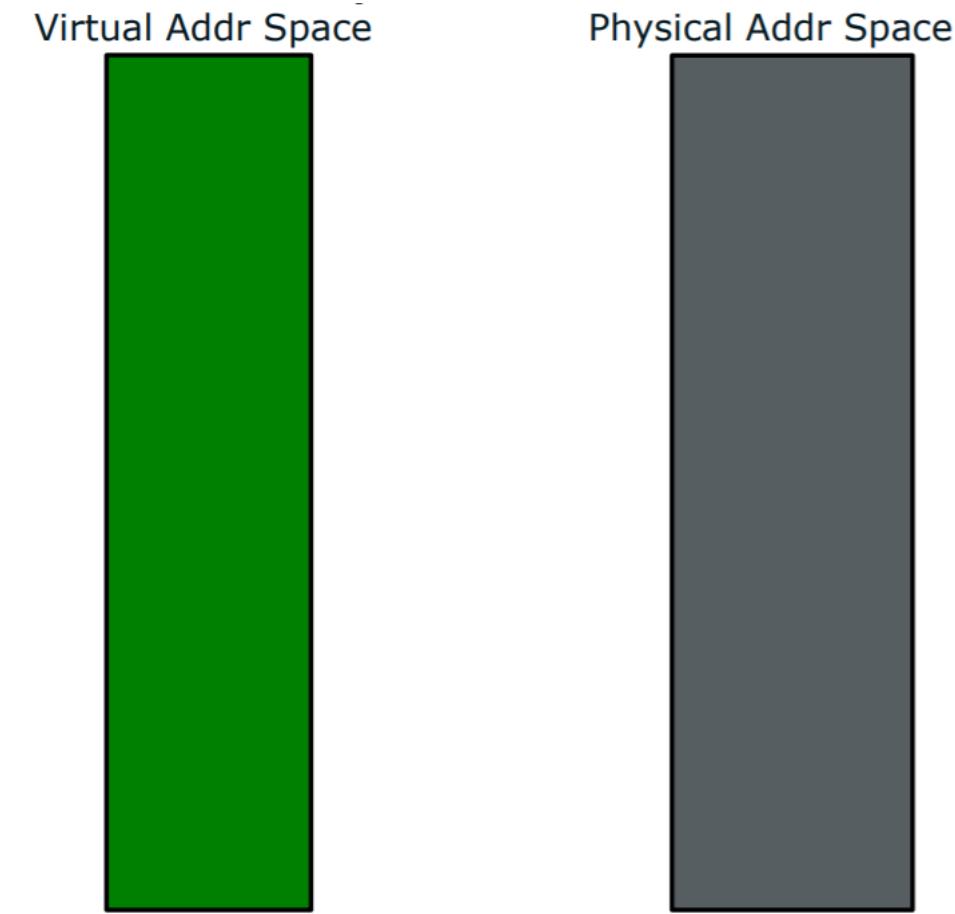
# SGX : Setting up enclave – User process



# SGX : Setting up enclave – User process

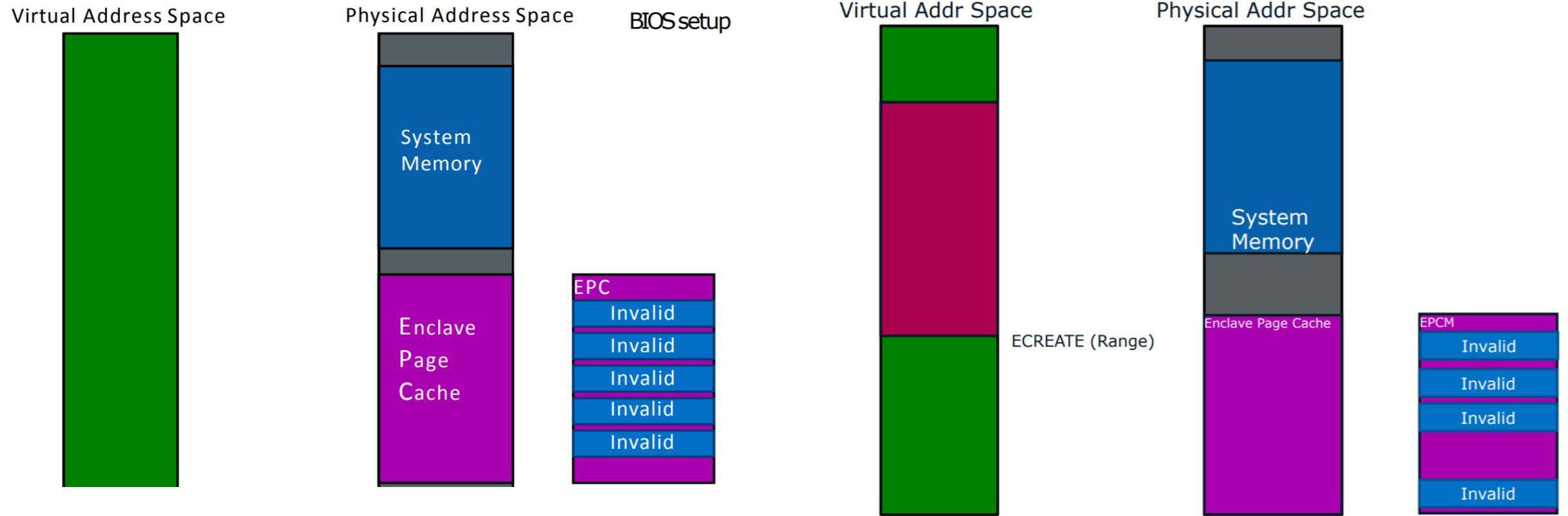
- Step 1: When setting up an enclave, ring-0 instructions are needed, so a user process must request it to the kernel.
- Step 2: The kernel initializes the enclave.
  - In this **setup process**, the kernel also **determine the initial state of enclave memory** and the **enclave program** is also **loaded by the kernel**.
- Step 3: After finishing the initialization, the **confidentiality** and **integrity** are guaranteed by CPU.
- Note: The memory layout of enclave program is completely visible to the kernel (User process)

# Life cycle of Enclave



# Life cycle of Enclave – cont'd..

- EPC – Enclave Page Cache
- EPCM - **EPCM** is the **security meta-data attached** to each **EPC** page.
- There is a 1:1 mapping between an **EPC** page and an **EPCM** entry.



# Some common instructions

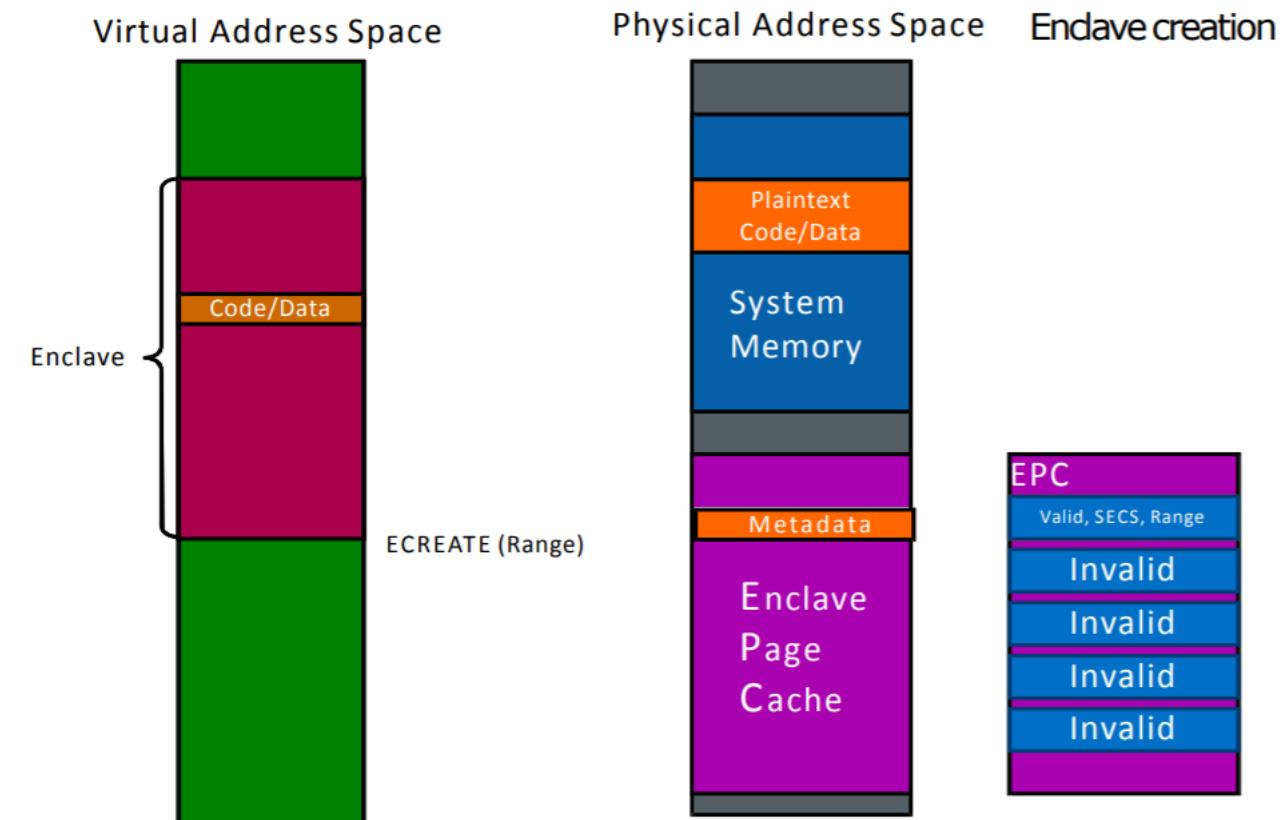
<b>Super.</b>	<b>Description</b>	<b>User</b>	<b>Description</b>
EADD	Add a page	EENTER	Enter an enclave
EBLOCK	Block an EPC page	EEXIT	Exit an enclave
ECREATE	Create an enclave	EGETKEY	Create a cryptographic key
EDBGRD	Read data by debugger	EREPORt	Create a cryptographic report
EBDGWR	Write data by debugger	ERESUME	Re-enter an enclave
EINIT	Initialize en enclave		
ELDB	Load an EPC page as blocked		
ELDU	Load an EPC page as unblocked		
EPA	Add a version array		
EREMOVE	Remove a page from EPC		
ETRACE	Activate EBLOCK checks		
EWB	Write back/invalidate an EPC page		

Supervisor

User

# Life cycle of Enclave – Enclave Creation

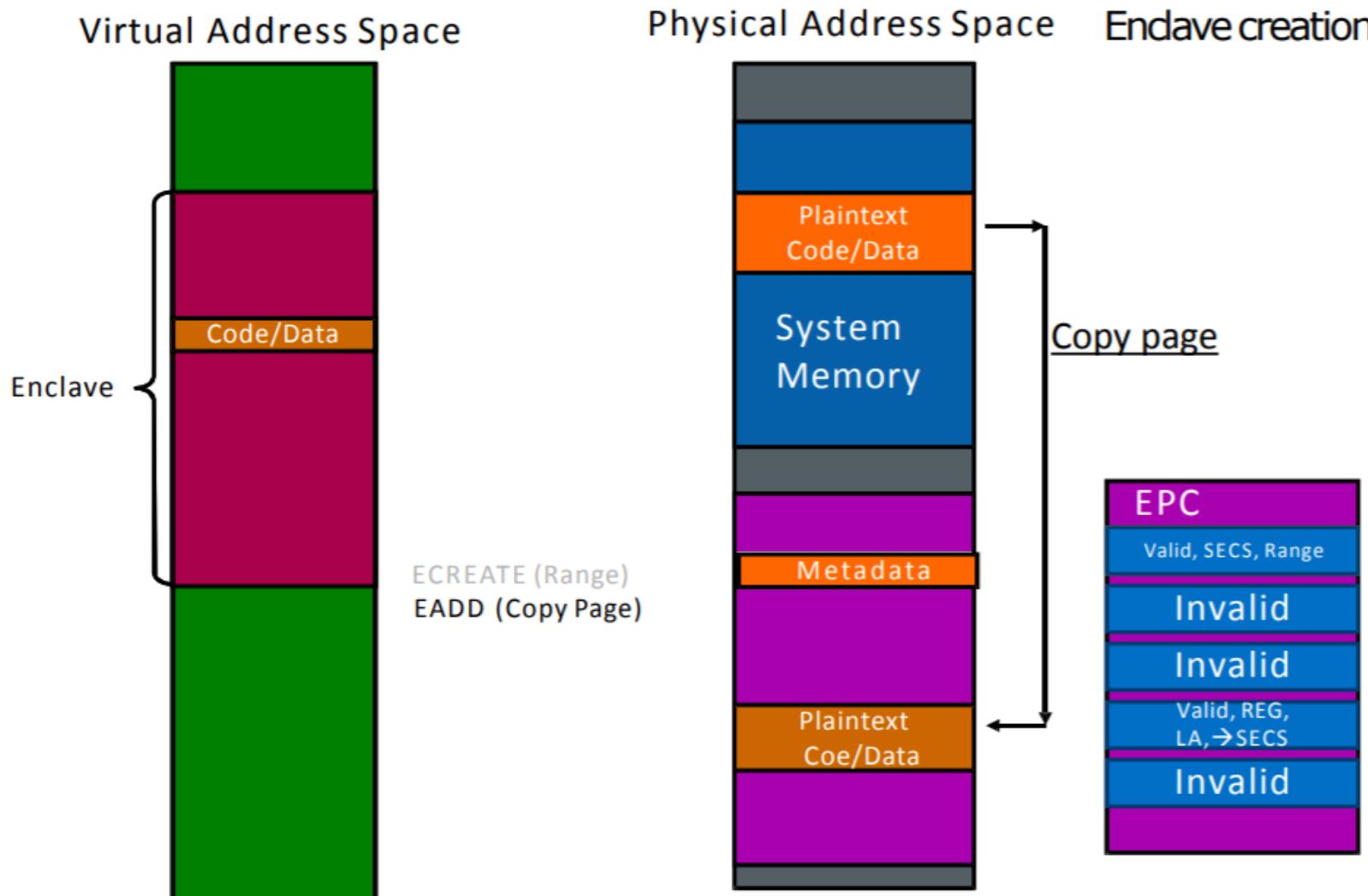
- **ECREATE** - Create an enclave (Stores enclave attributes (mode of operation, debug, etc.) in metadata.)
- Trusted virtual address within a range



\*SECS - SGX Enclave Control Structure (SECS)

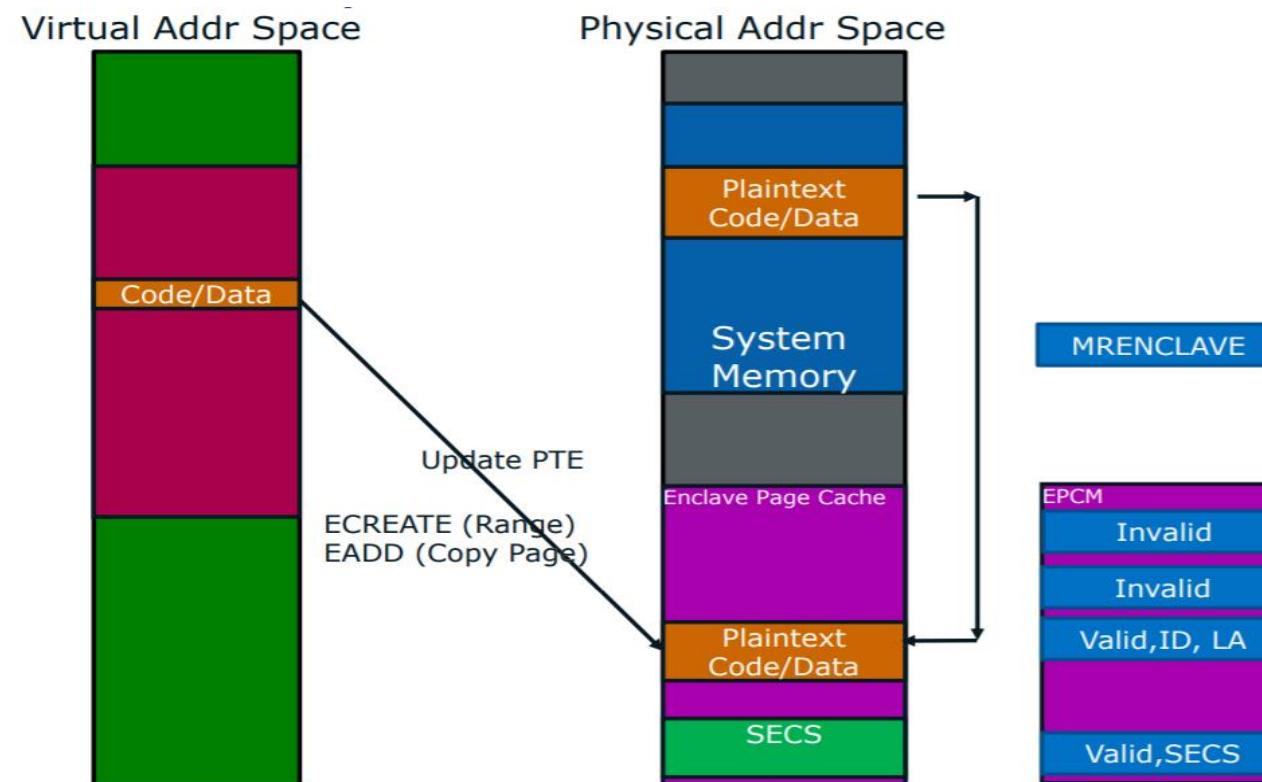
# Life cycle of Enclave – Enclave Creation

- EADD - Commits new pages to enclave & updates security metadata



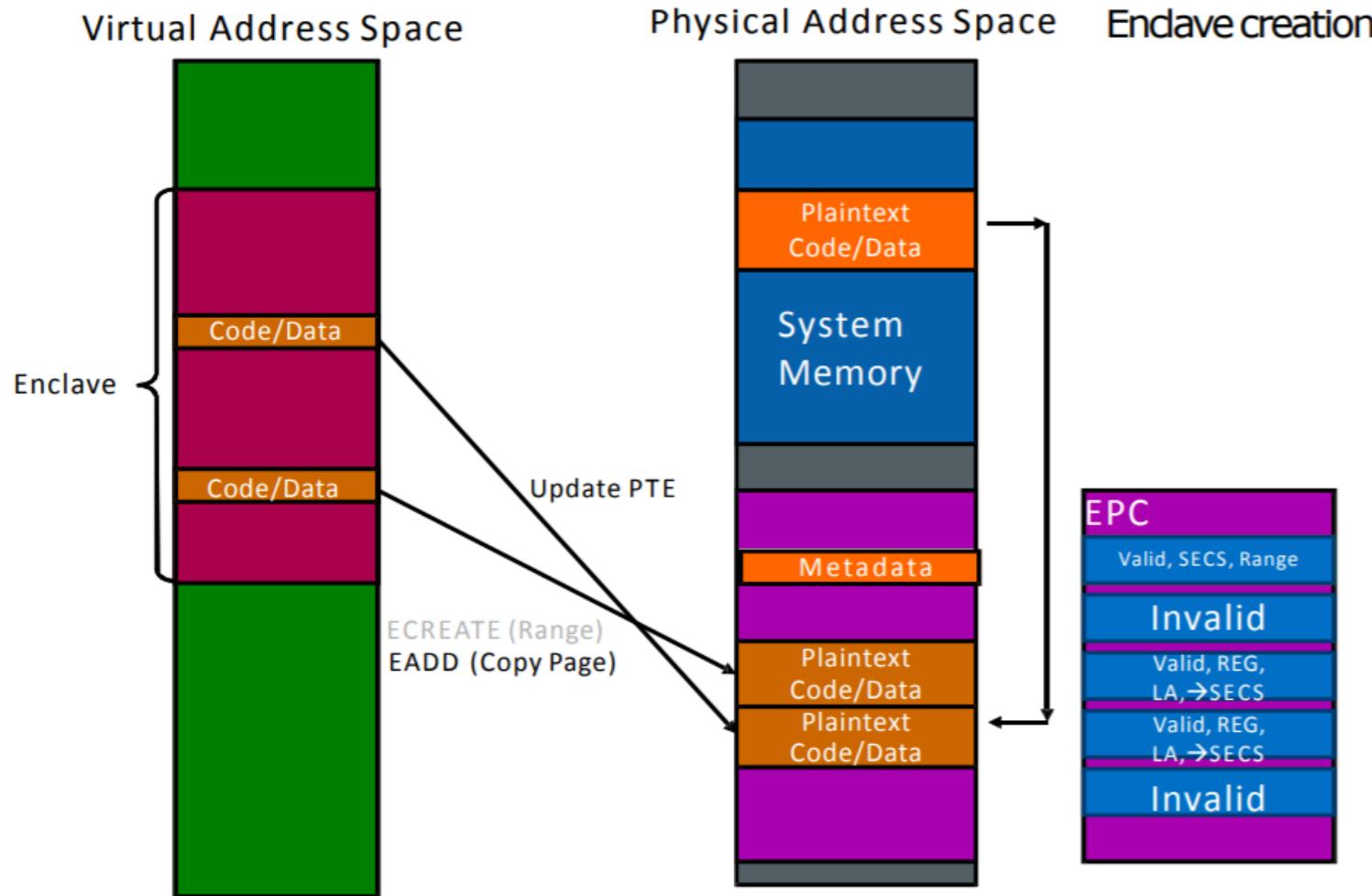
# Life cycle of Enclave – Enclave Creation

- EADD - Commits code, data or SGX Enclave Control Structure (SECS) page types.
- MREnclave (Enclave identity)- is 256 bit digest
  - Used to check the integrity (Code, Data, Stack, Heap, location of each page within enclave)



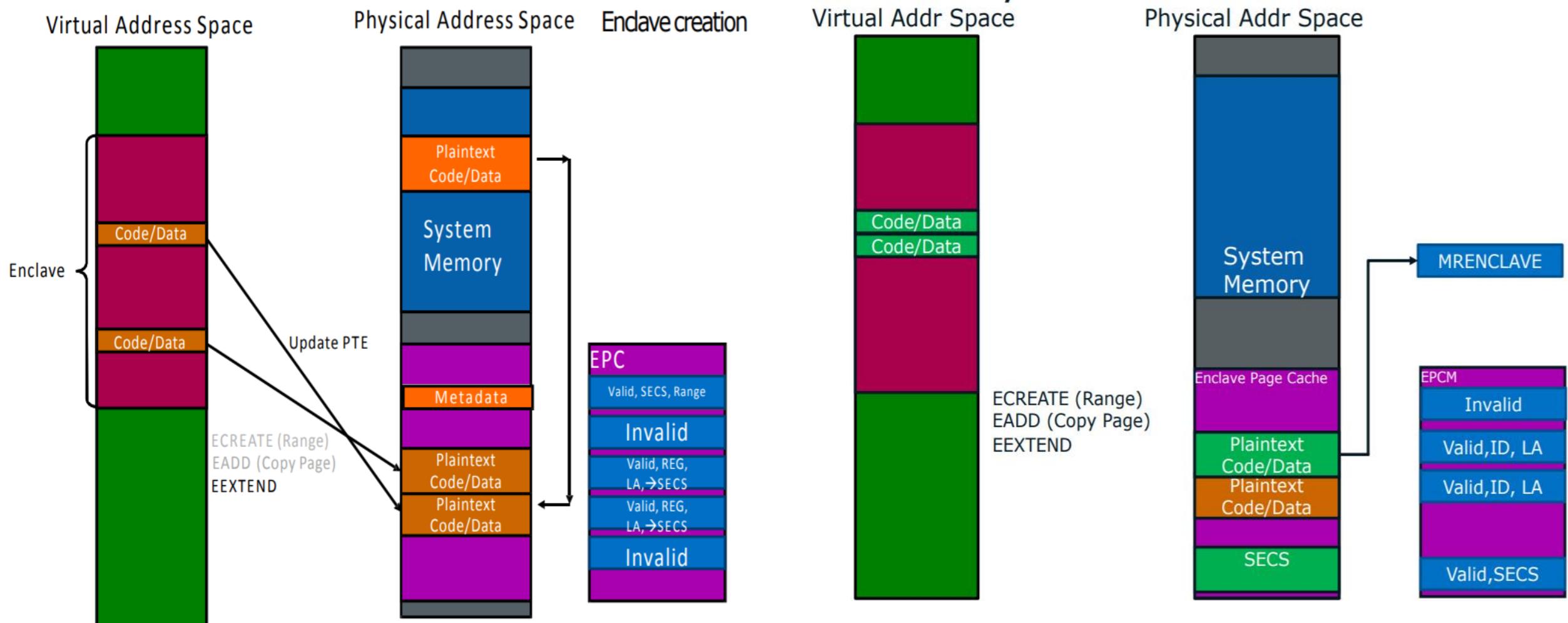
# Life cycle of Enclave – Commit code (multi)

- EADD - Commits code, data or SGX control structure page types.



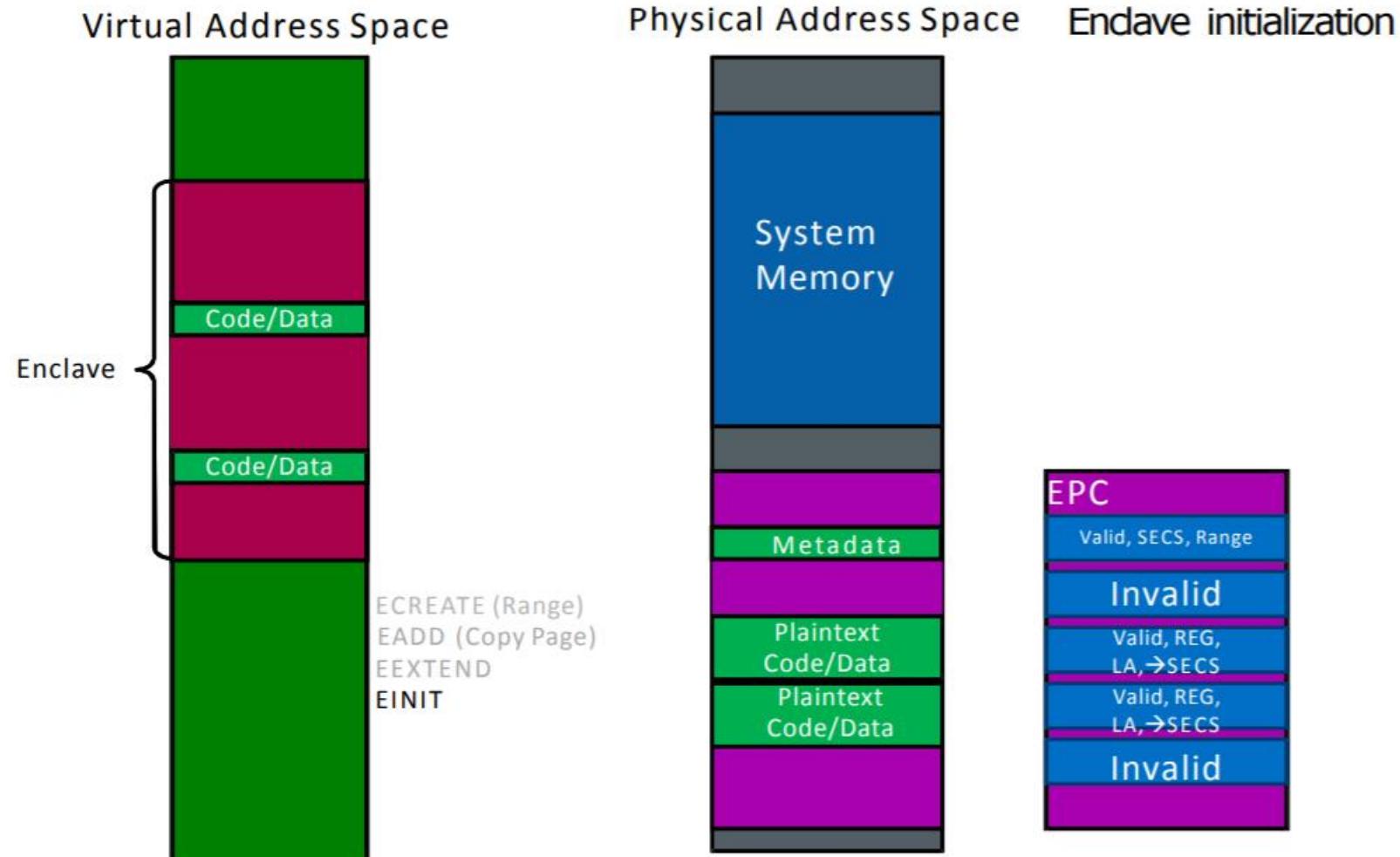
# Life cycle of Enclave – Enclave Creation

- **EEXTEND** - Measures the enclave with SHA256.



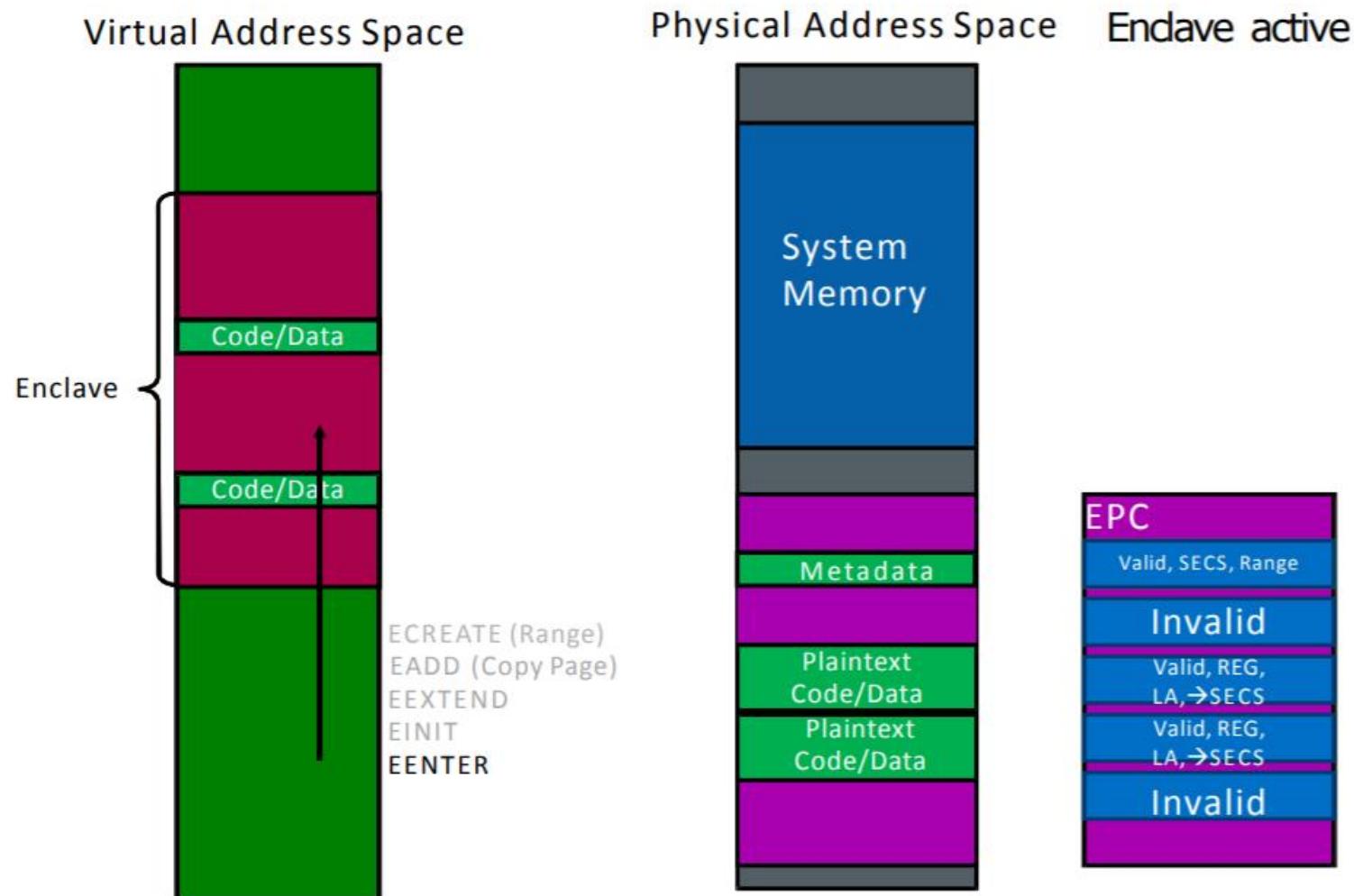
# Life cycle of Enclave – Enclave Initialization

**EINIT** - Finalizes measurements, validates them & enables enclave's entry use.



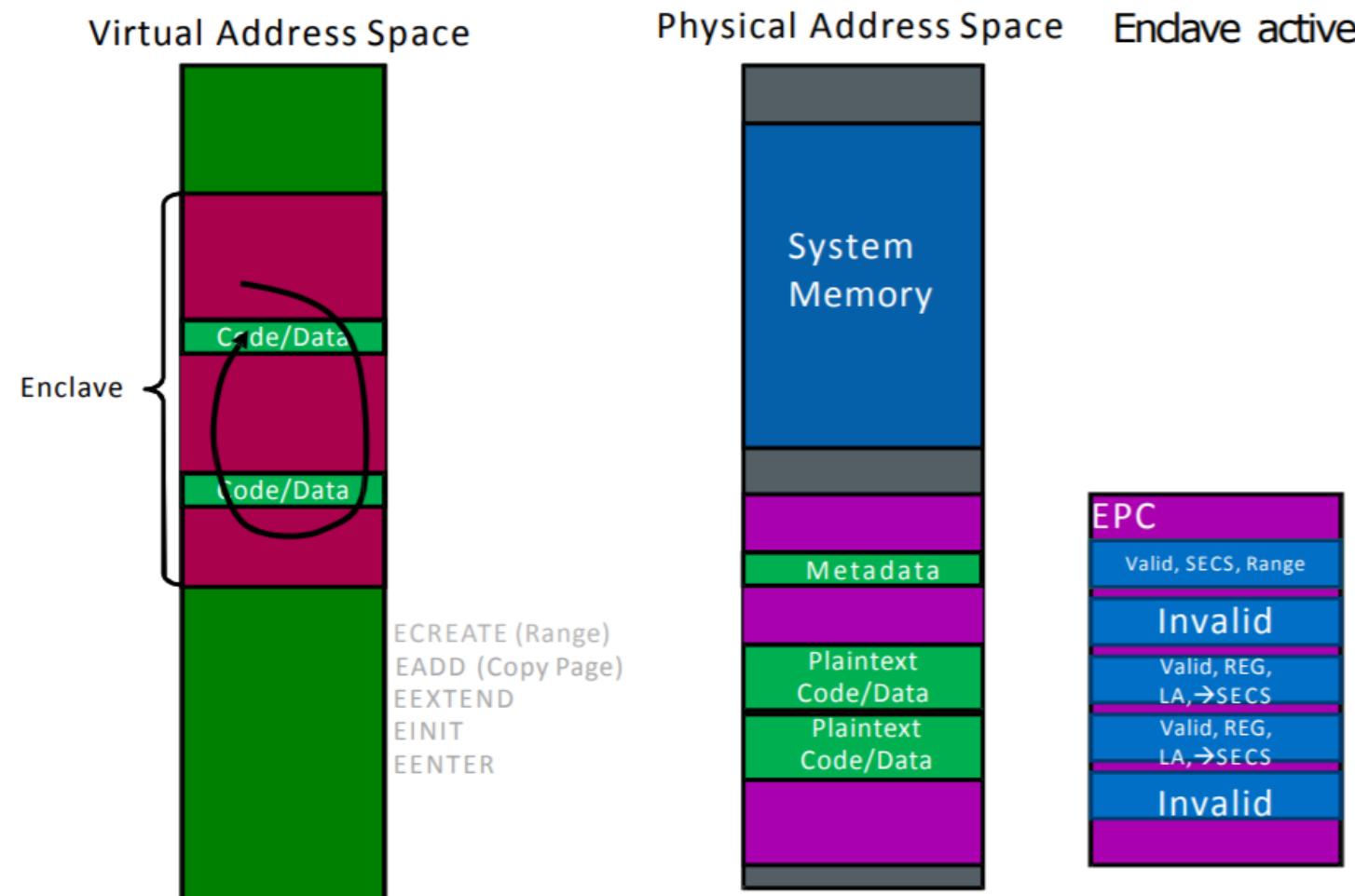
# Life cycle of Enclave – Enclave Initialization

- **EENTER** - **Verifies enclave entry** & sets **CPU operation mode** to “enclave mode”



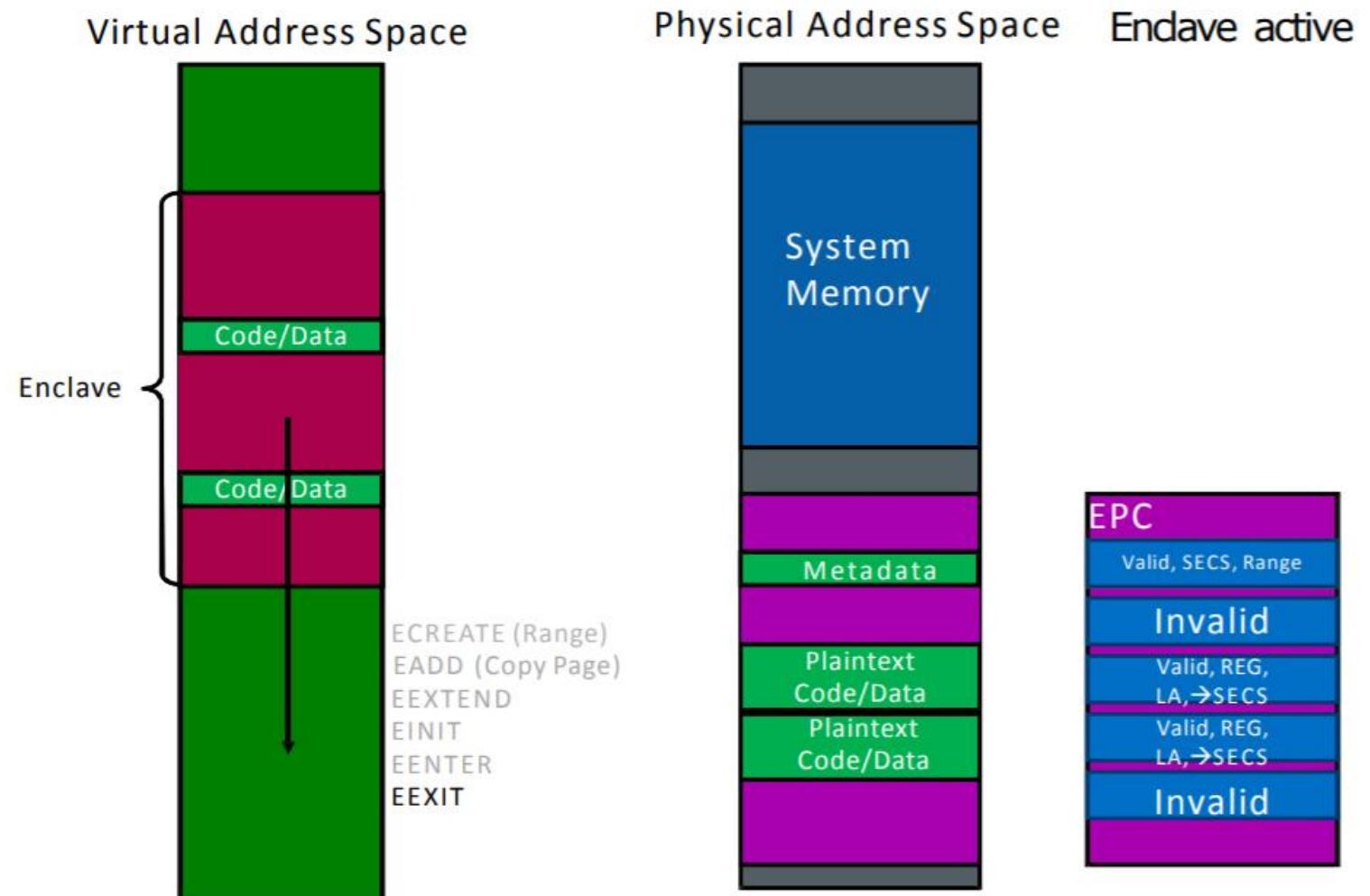
# Life cycle of Enclave – Enclave Flow

- Enclave flow is executed using the **secured address range**, obscured from all other SW.



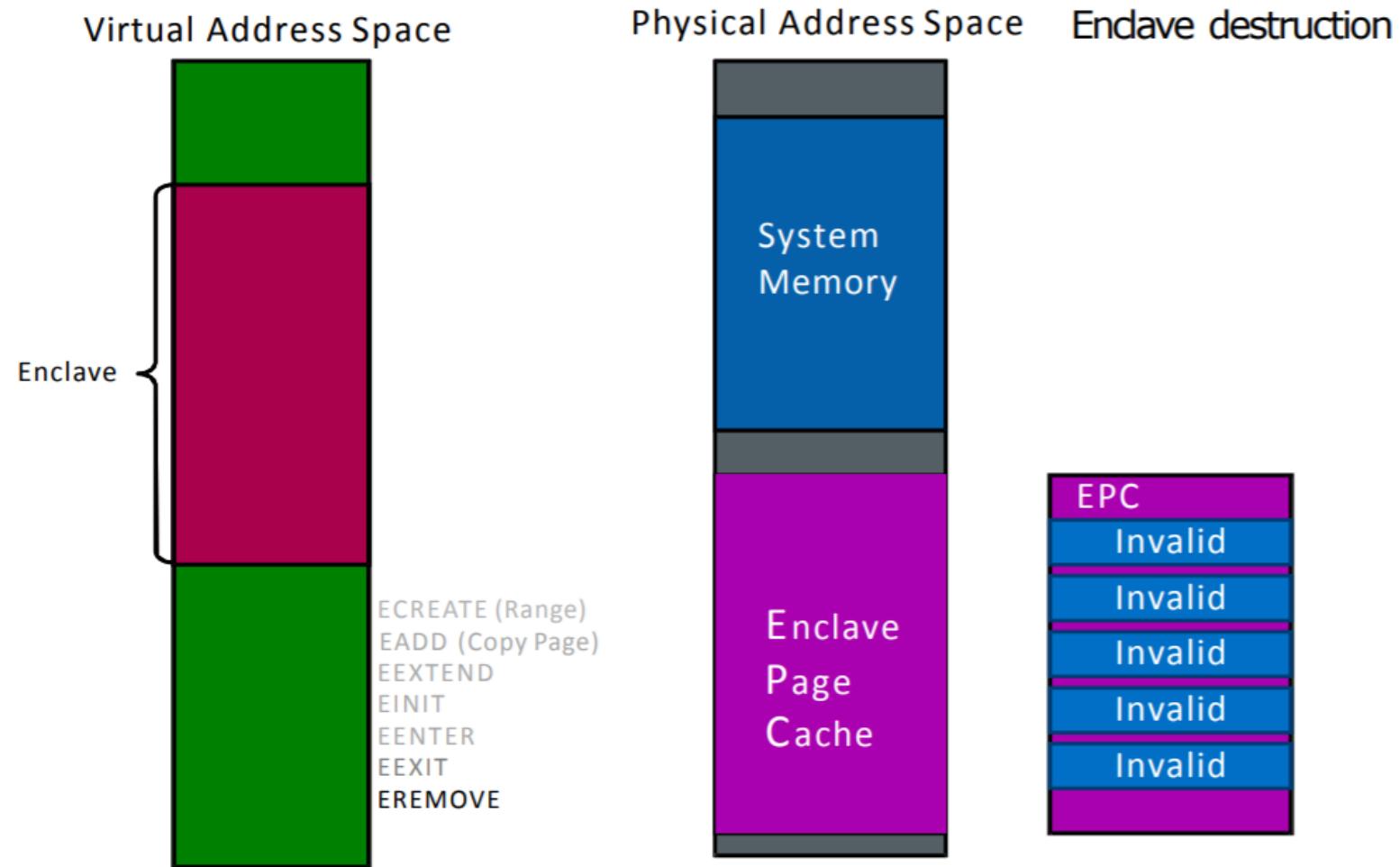
# Life cycle of Enclave – Enclave Flow

- **EEXIT** – Clears **CPU cache** & Jumps out of enclave back to **OS instruction address**.



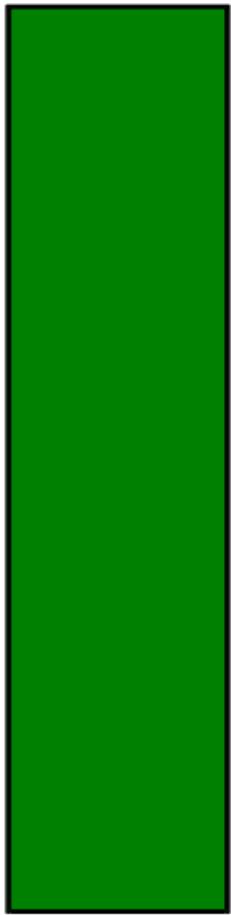
# Life cycle of Enclave – Enclave Destruction

- **EREMOVE** – Clears enclave's trusted virtual address range reserved by ECREATE.

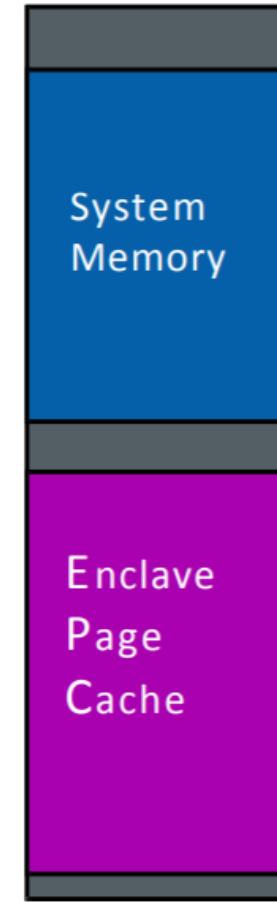


# Finally, After EREMOVE call

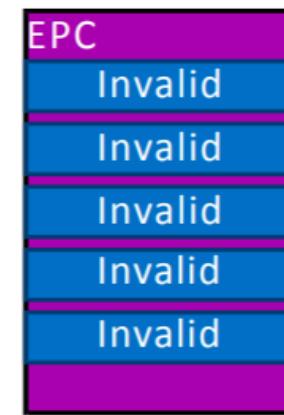
Virtual Address Space



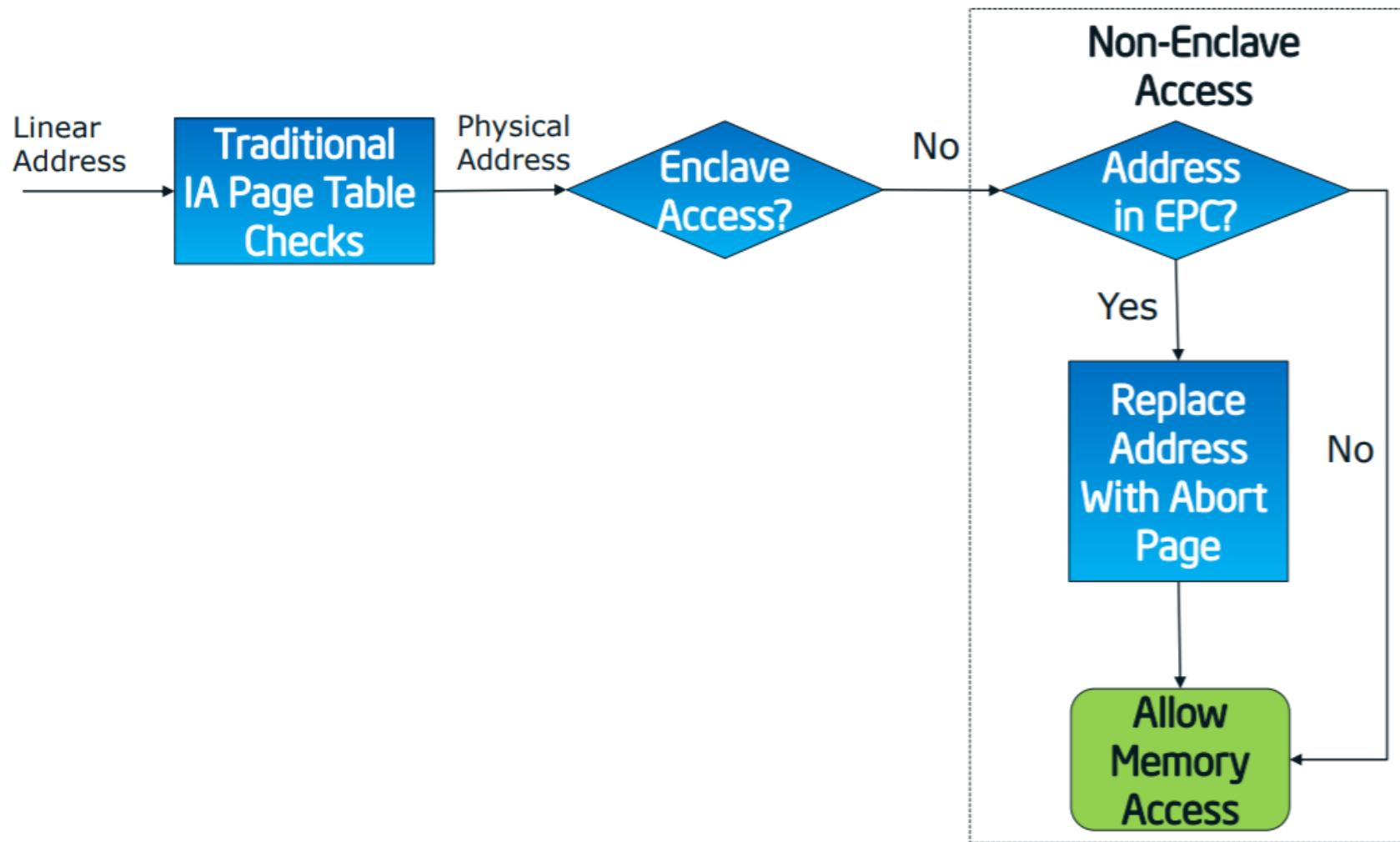
Physical Address Space



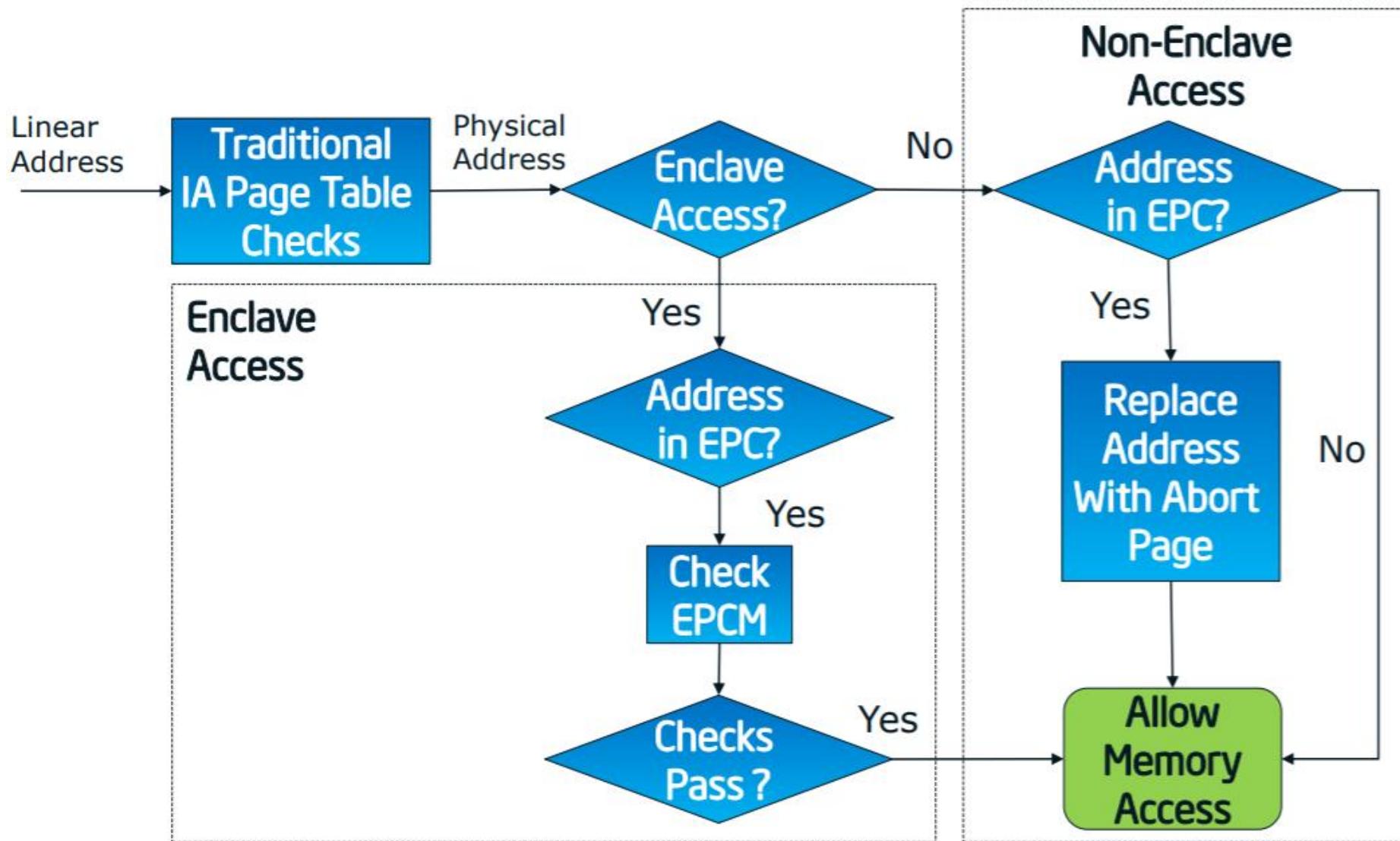
BIOS setup



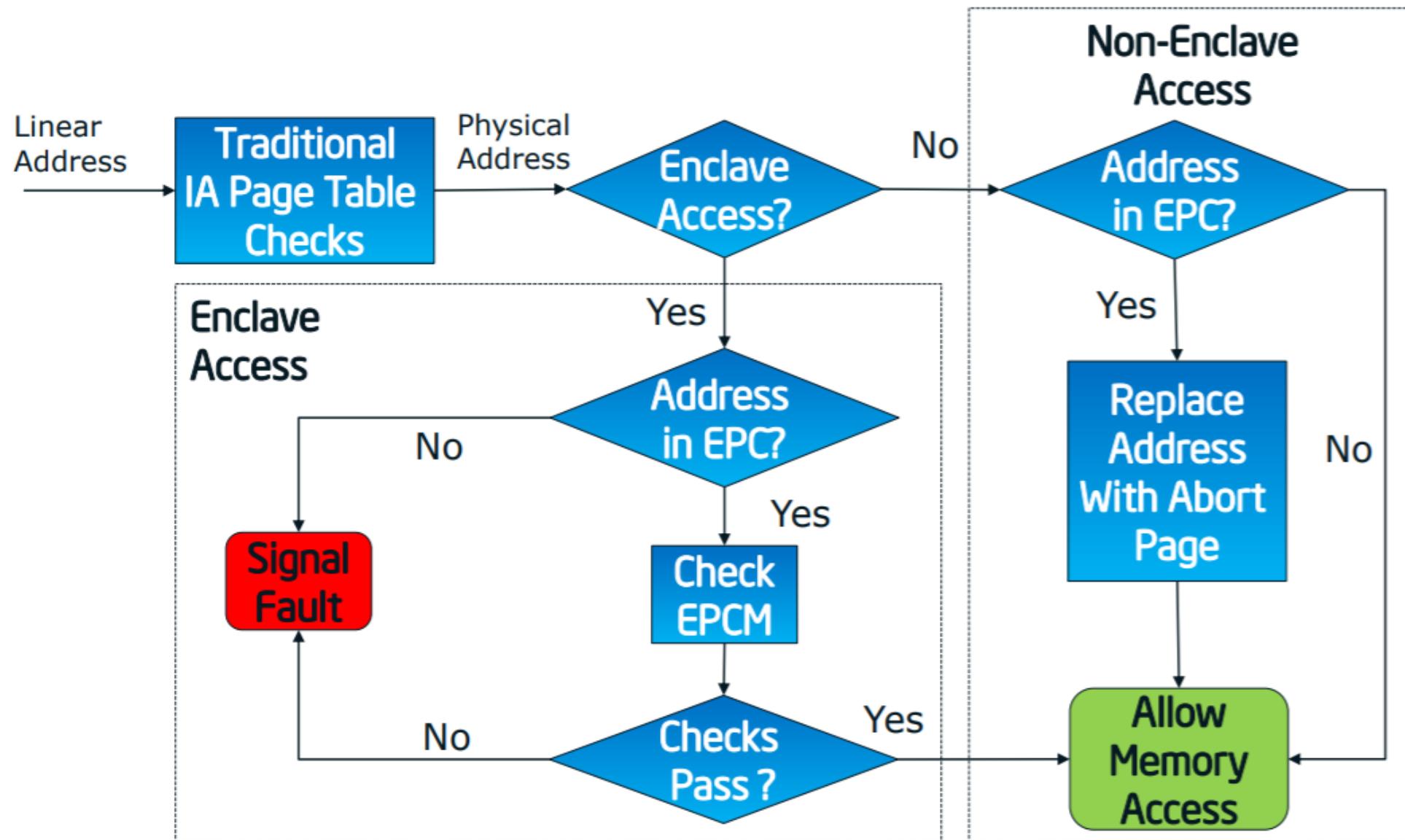
# SGX Access control – Non-Enclave access



# SGX Access control – Enclave access



# SGX Access control – Enclave access violation



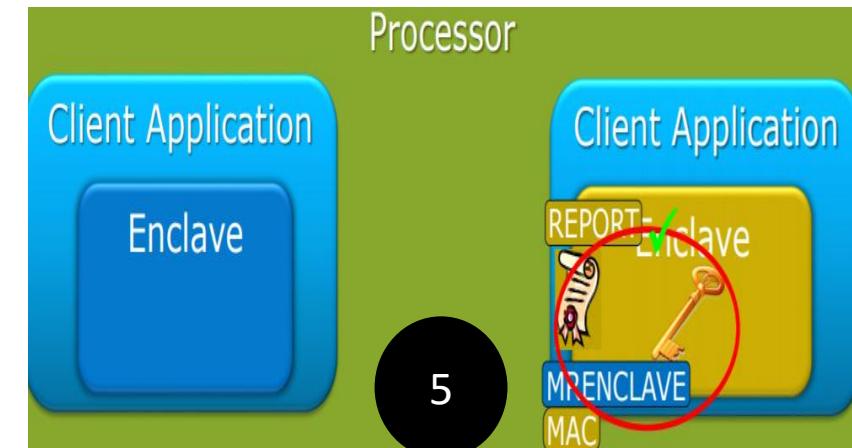
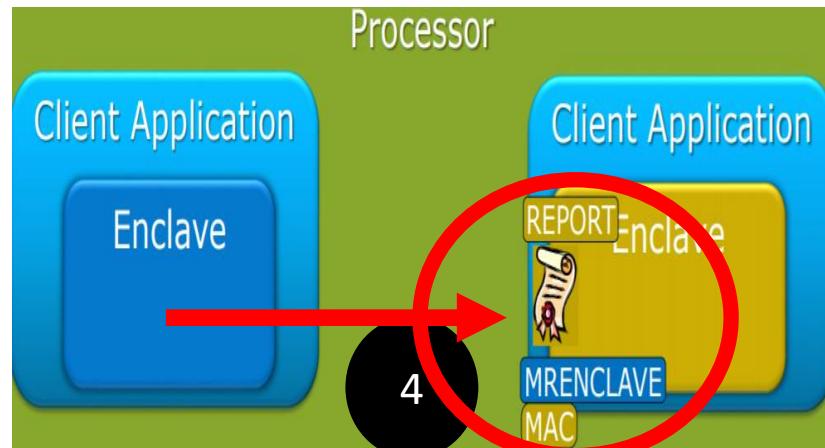
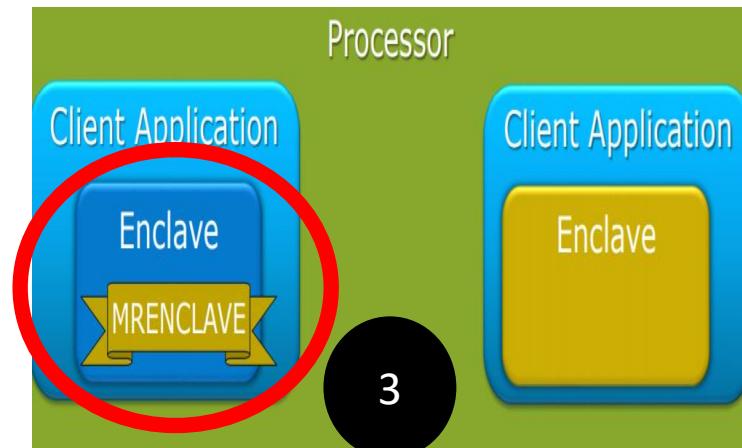
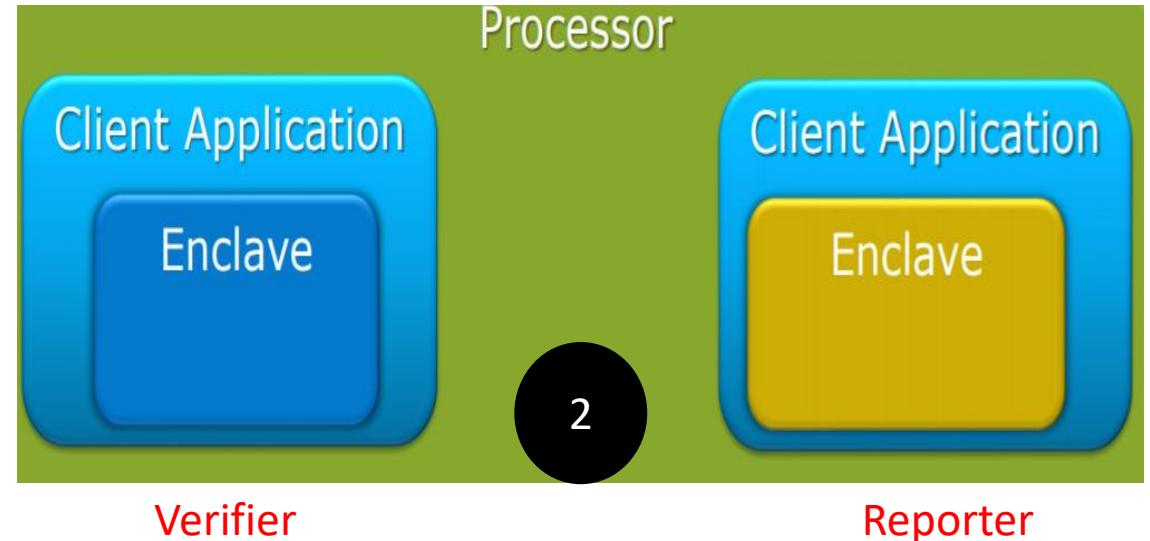
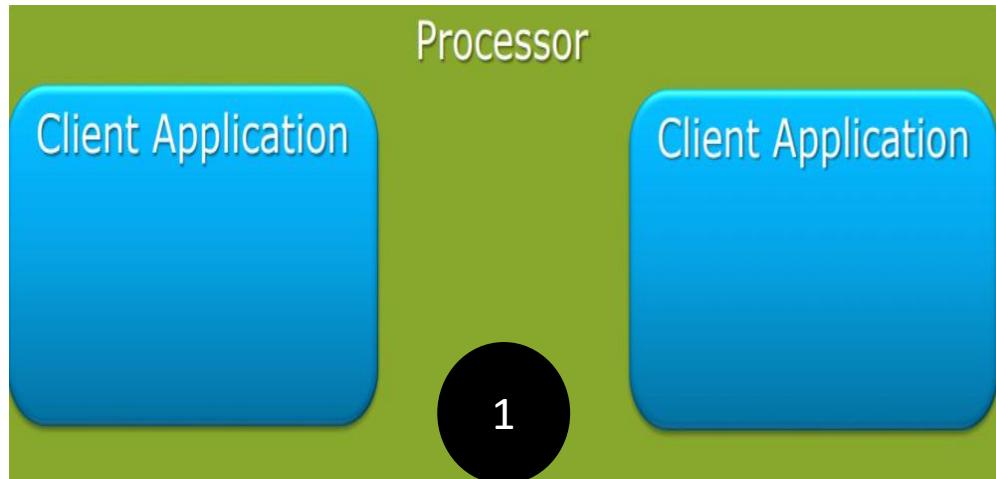
# Trusted Computing Base for SGX

- Attestation
- Sealing
  - Sealing Authority

## Attestation

- Local attestation
- Remote attestation

# Problem with Attestation



# Sealing

- “Sealing”: **Cryptographically protecting data when it leaves the enclave.**
- Enclaves use **EGETKEY** to retrieve an **enclave platform persistent key** and encrypts the data.
- **EGETKEY** uses a **combination of enclave attributes and platform unique key to generate keys for sealing**
  - Enclave Sealing Authority
  - Enclave Product ID
  - Enclave Product Security Version Number (SVN)

# Sealing Authority

- Every enclave has an **Enclave Certificate (SIGSTRUCT)** which is signed by a **Sealing Authority**.
- SIGSTRUCT includes:
  - **Enclave's Identity** (represented by MRENCLAVE)
  - **Sealing Authority's public key** (represented by MRSIGNER)
- EINIT verifies the signature over **SIGSTRUCT** prior to enclave initialization.

# SGX Summary

- Enclave Measurement
  - Mrenclave
  - Enclave runs in ring-3 privilege level deriving trust directly from hardware.
  - Application can support multiple enclaves.
- Memory Encryption Engine (MEE)
- Sealing
- Attestation

# Device Guard

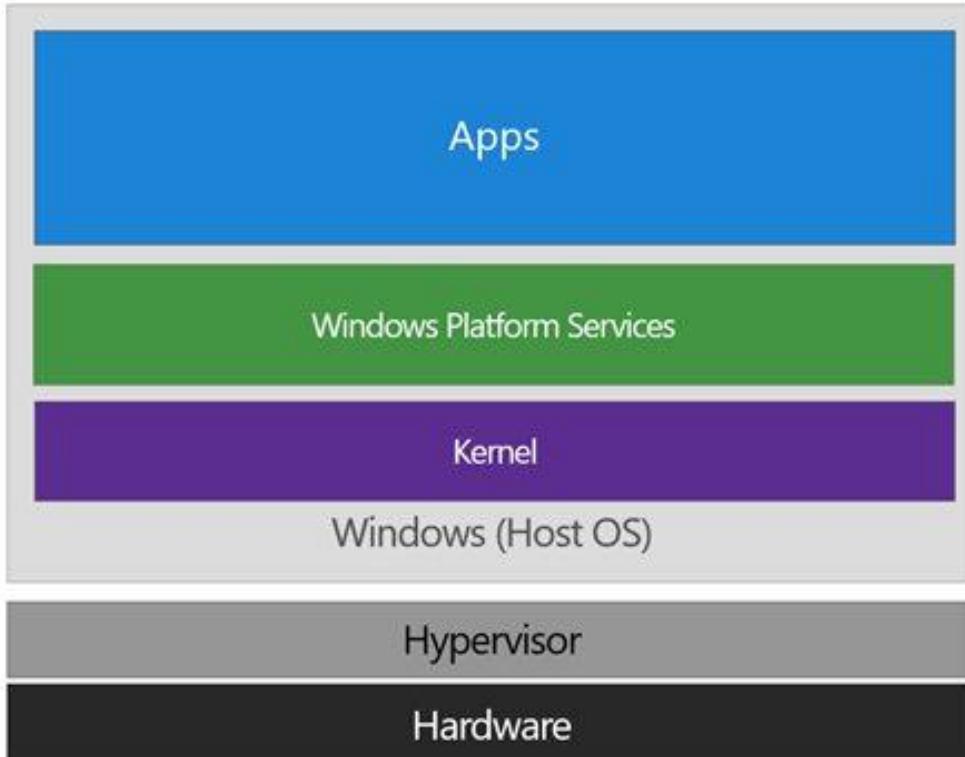
- Windows Defender Device Guard is split into two features known as **Windows Defender Exploit Guard** and **Windows Defender Application Control**.
- A combination of **hardware and software security features** that, when configured together, **will lock a device down** so **that it can only run trusted applications** that are **defined** in an **enterprise's code integrity policies**.
- If the app isn't trusted it can't run the app

# Device Guard is actually a set of three features

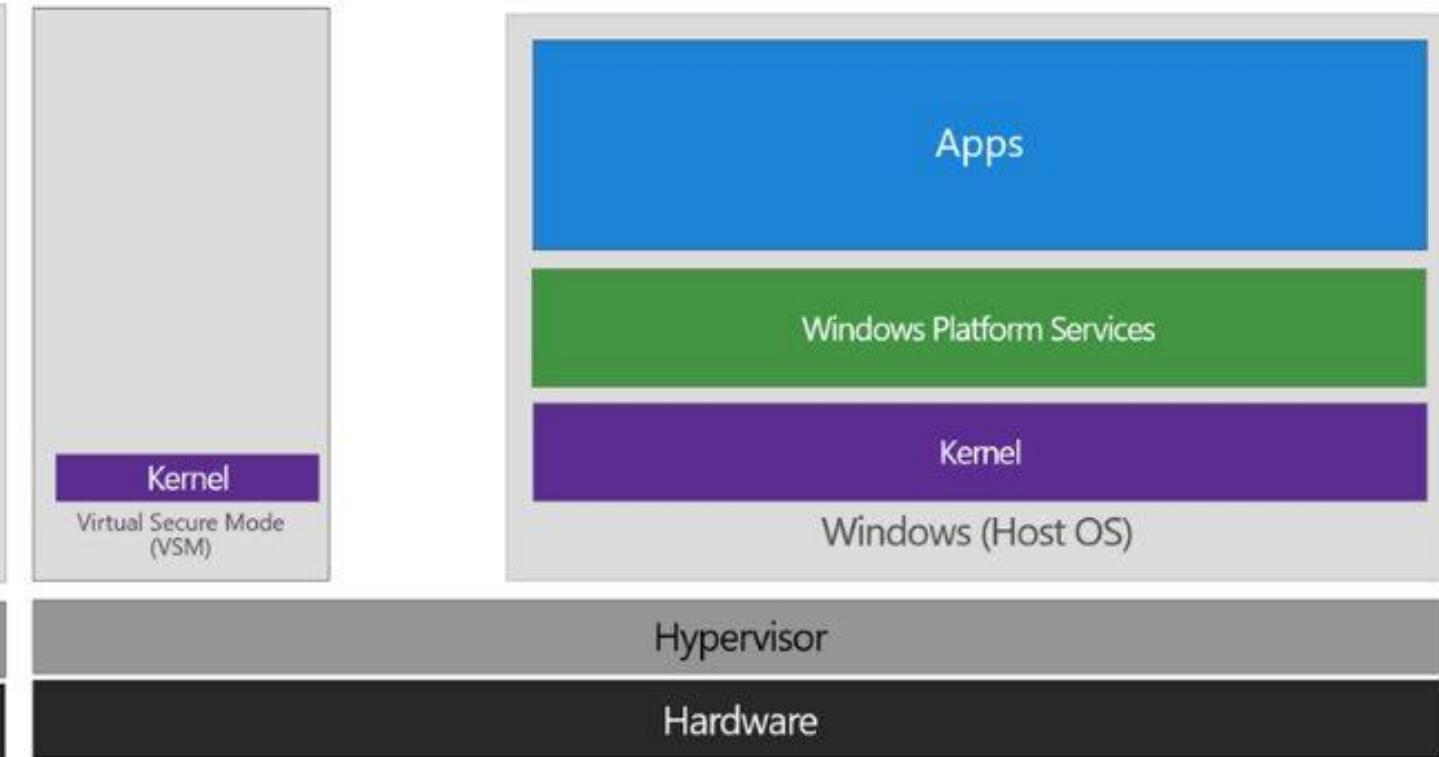
- Platform and Secure Boot locks down boot processes by requiring '**UEFI Secure Boot Mode**' which **ensures the UEFI firmware is digitally signed** and therefore all but guarantees it has not been tampered with.
- **Virtual Secure Mode (VSM)** turns on core Hyper-V components to allow Windows to isolate each application's memory.  
Note: VSM protects only domain user accounts, not local ones
- **Configurable Code Integrity (CCI)** allows only programs that an administrator approves.

# Device Guard design architecture

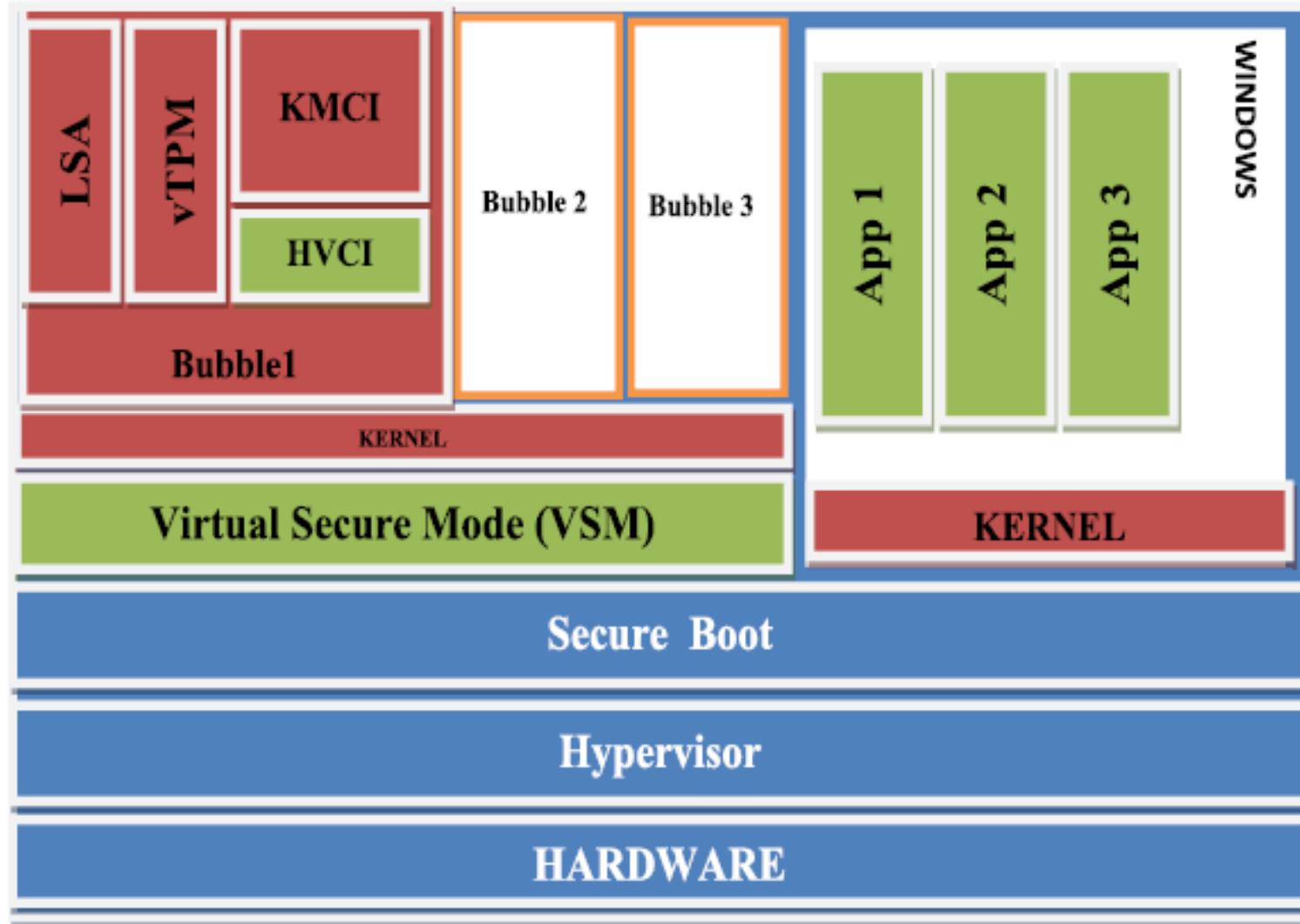
**Standard Computer Stack**



**Device Guard with VSM Computer Stack**

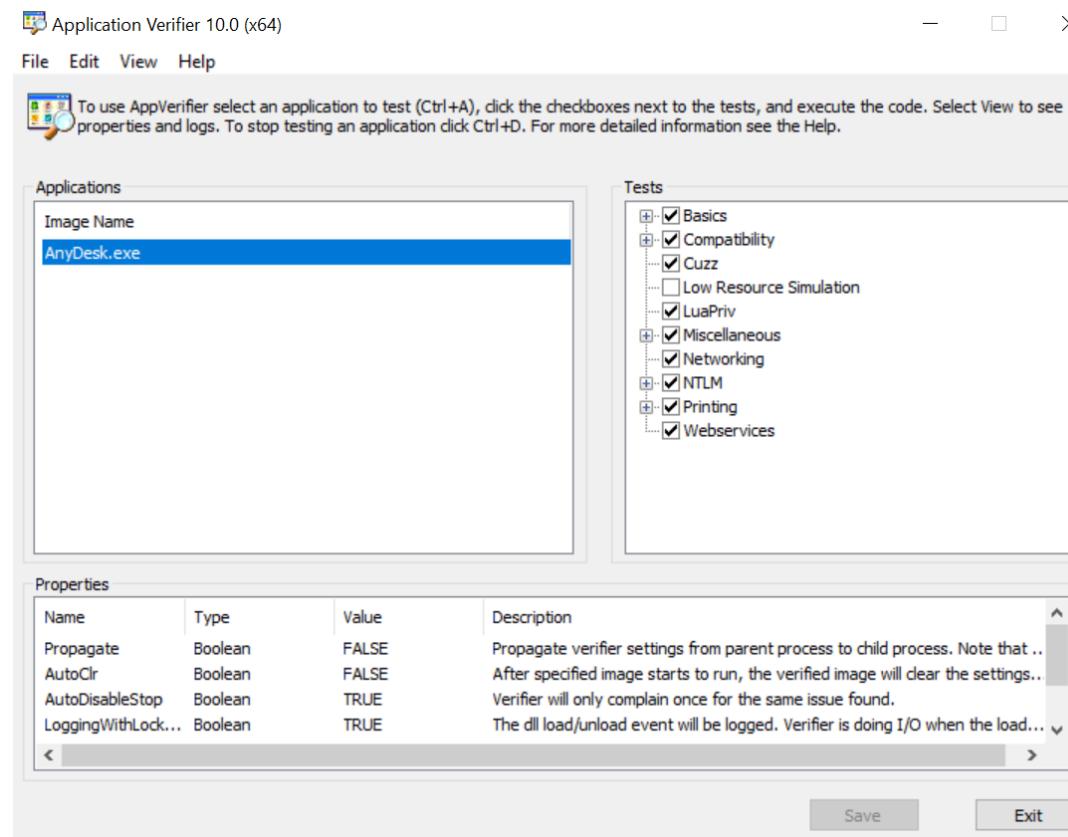


# VSM architecture



# Application code integrity verifier

- Application verifier



# Enable virtualization-based security and Windows Defender ApplicationControl

- Go to HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\DeviceGuard.
- Add a new DWORD value named **EnableVirtualizationBasedSecurity**. Set the value of this registry setting to 1 to enable virtualization-based security and set it to 0 to disable it.
- Add a new DWORD value named **RequirePlatformSecurityFeatures**. Set the value of this registry setting to 1 to use **Secure Boot** only or set it to 3 to use **Secure Boot and DMA protection**.

# Enabling virtualization based security feature

- Core Isolation
- Memory Integrity

# Trusted integrated circuit

- Policies to ensure high trustability in ICs:
- Prevent intentional or unintentional modification or tampering of the ICs
- Protect the ICs from unauthorized attempts at reverse engineering
- Minimizing the exposure of functionality or evaluation of their possible vulnerability

# key elements for Trusted ICs

- Trusted suppliers and products
- Proper design information hiding
- Incorporating anti-tamper technology
- Failure detection and forensics
- Damage mitigation
- Chip level signature authentication

THE END