ADDRESS SPACE LAYOUT RANDOMIZATION (ASLR) AND STACK PROTECTION AND MITIGATING TECHNIQUES FOR SIDE CHANNEL ATTACKS

OPERATING SYSTEM

SUBMITTED BY: SUSHAN SHRESTHA

INTRODUCTION

- - Operating system security ensures that memory and system resources are protected from unauthorized access.
- - Memory protection helps prevent various attacks such as buffer overflows.
- - This presentation will explain ASLR and Stack Protection as two key security techniques.

ADDRESS SPACE LAYOUT RANDOMIZATION (ASLR)

- - ASLR is a security technique that randomizes the memory addresses used by system components.
- - It makes it harder for attackers to predict the location of critical data.
- - Memory segments like stack, heap, and libraries are randomized to prevent exploitation.

WHY ASLR IS IMPORTANT?

- - ASLR protects against buffer overflow and code injection attacks.
- - It makes memory addresses unpredictable, forcing attackers to guess locations.
- - Without ASLR, attackers can easily exploit vulnerabilities in programs.

LIMITATIONS OF ASLR

- - ASLR can be bypassed if the randomization algorithm is weak.
- - Brute force attacks can guess randomized addresses.
- - ASLR is ineffective if combined with vulnerabilities like information leakage.

WHAT IS STACK PROTECTION?

- - Stack Protection prevents attackers from overwriting return addresses on the stack.
- - It is designed to prevent stack-based buffer overflow attacks.
- - Commonly used techniques include placing canary values before return addresses.

TECHNIQUES OF STACK PROTECTION

- Stack Canaries: Random values placed between local variables and return addresses.
- Non-Executable Stack (NX Bit): Marks stack memory as non-executable to prevent code execution.
- Safe SEH: Ensures only valid exception handlers are executed.
- Stack Smashing Protector (SSP): Compiler feature that adds security checks.

CONCLUSION

- - Both ASLR and Stack Protection are essential for modern system security.
- - They work together to prevent buffer overflows and code execution attacks.
- - A combination of multiple security mechanisms provides stronger protection.

COMPARISON BETWEEN ASLR AND STACK PROTECTION

Feature	ASLR	Stack Protection
Focus	Memory Address Randomization	Stack Overflow Prevention
Type	System-level protection	Application-level protection
Effectiveness	Reduces predictability	Stops direct overflow attacks

WHAT IS A SIDE-CHANNEL ATTACK?

- A **security breach** where information is gathered from the **system's physical behavior** (not the data itself).
- **Example**: Observing power consumption or timing of cryptographic operations.

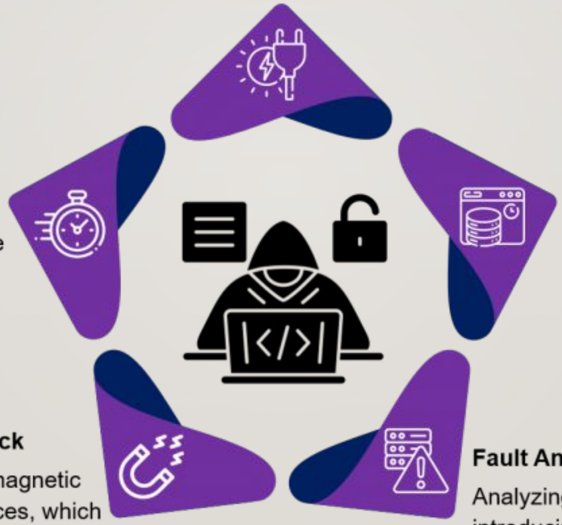
SIDE-CHANNEL ATTACK

Power Analysis Attacks

Rely on the variations of power traces consumed by a computing device

Timing Attack

Rely on the variations in execution times of cryptographic operations, i.e. by observing and analyzing the time taken by a specific task



Cache-based Attack

Rely on the behavior of cache memory, which can reveal information about data access patterns in a shared physical system or a type of cloud service

Electromagnetic (EM) Attack

Rely on the emitted electromagnetic radiation by computing devices, which can directly provide plaintexts and other information

Fault Analysis Attack

Analyzing the faulty output after introducing faults in a cyber physical system

HARDWARE-LEVEL MITIGATION TECHNIQUES

Implement Secure Hardware Design

- **Goal**: Reduce physical leakage.
- **Method**: Design logic gates, circuits, and layouts to make them tamper-resistant.
- **Example**: Shielding sensitive signals in silicon layouts using **special metal routing**.

HARDWARE-LEVEL MITIGATION TECHNIQUES

Power and Electromagnetic (EMF) Analysis Countermeasures

- **Dynamic Voltage Variation**: Randomly change voltage to make it hard to detect patterns in power consumption.
- **EMF Shielding**: Reduce electromagnetic emissions from the hardware.
- **Power Analysis Counters**: Power gating and balancing power consumption to prevent correlation between operations and power usage.

HARDWARE-LEVEL MITIGATION TECHNIQUES

Real-World Testing and Regular Auditing

- **Testing under Real Conditions**: Expose cryptographic devices to environmental factors (e.g., power fluctuations) to test resistance.
- **Security Audits**: Regular reviews and checks to find vulnerabilities.

SOFTWARE-LEVEL MITIGATION TECHNIQUES

Randomizing Operations

- **Goal**: Obfuscate data access patterns.
- **Method**: Introduce random delays, algorithmic noise, or dummy instructions in code.
- **Why?**: Makes it harder for attackers to correlate side-channel signals to the actual data or keys.

SOFTWARE-LEVEL MITIGATION TECHNIQUES

Constant-Time Algorithm Usage

- **Goal**: Ensure uniform processing times.
- **Method**: Algorithms should take the same amount of time, no matter the input.
- **Why?**: Prevents timing attacks that exploit variations in execution time.

SOFTWARE-LEVEL MITIGATION TECHNIQUES

Cryptographic Algorithms Integration

- **Masking or Hiding Techniques**: Modify data to make it harder to extract secrets.
- **DPA-Resistant Algorithms**: Use algorithms that resist Differential Power Analysis (DPA).
- **White-Box Cryptography**: Secure cryptographic algorithms by obfuscating the keys and operations.

THANK YOU