Part 1: Introduction to SGX

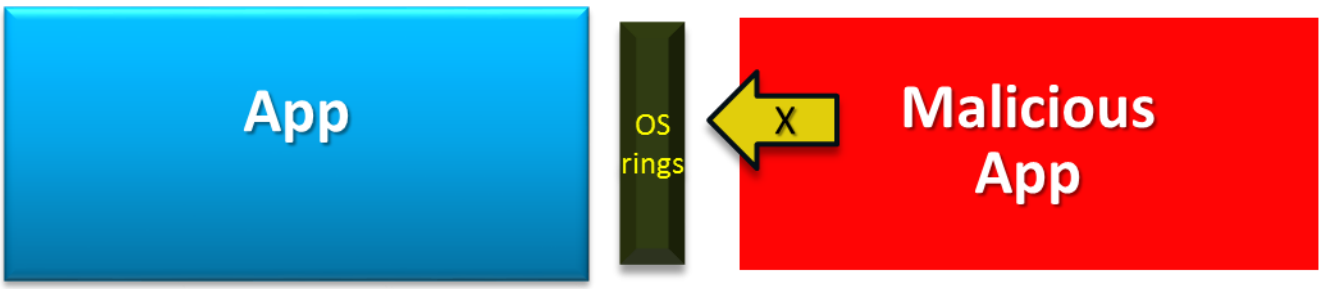
Goals:

- Explain series – what, why, how
- Provides foundation for subsequent parts.
- Discuss how SGX would affect and interact with the bare app
    - Identifying what's the secret
    - What should and should not be in an enclave, and why
- Audience should have a 100% understanding of what's going on in part 1.

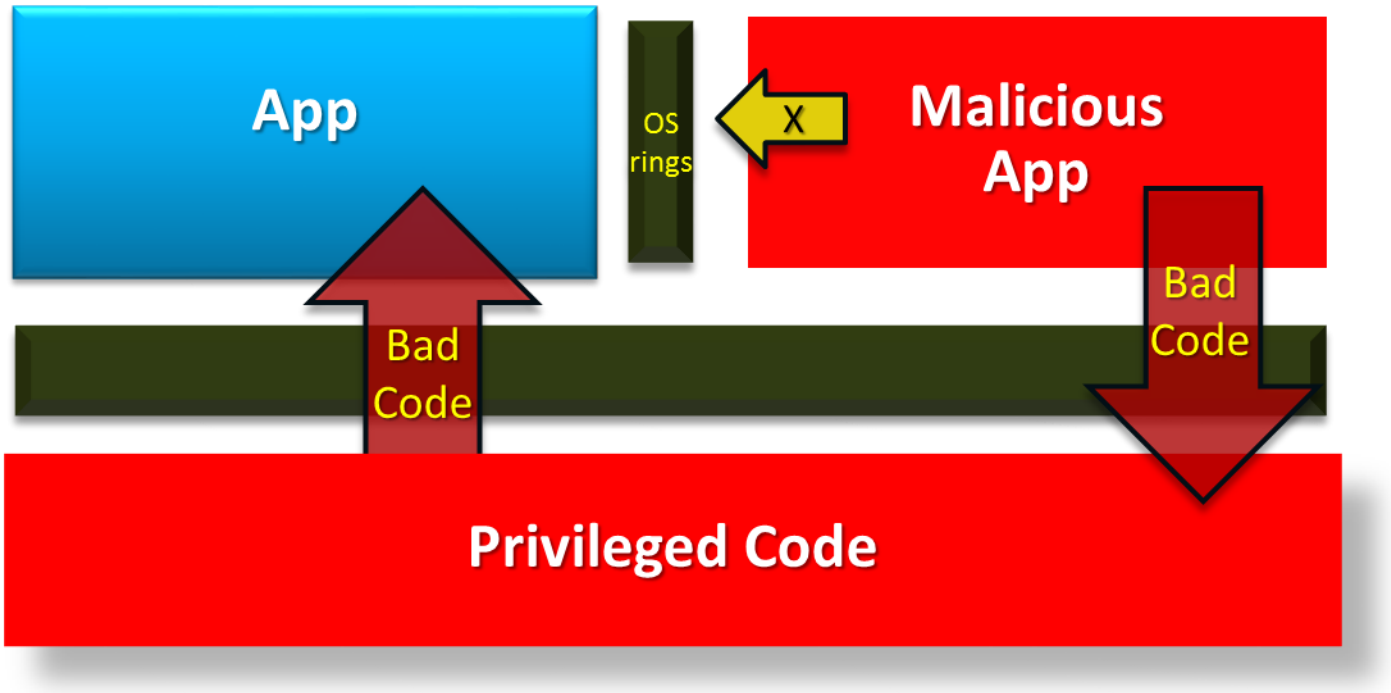
Introduction

Operating systems are responsible for storing and managing sensitive data. It is often the last line of defense against an attack. Applications must go through the OS to use any component or access any data outside of its own. This results in applications being heavily dependent on the operating system for security.

Numerous solutions exist to help the operating system guard data from unauthorized users, while also increasing the complexity of an operating system. These solutions are typically process and memory isolation. A basic example is the use of OS protection rings, a hierarchy of access-right levels. Access control is achieved through gated access between the rings, and is a basic method of guarding an application from malicious behavior.



A process can access data at a lower, less privileged, ring. Operating systems determine which processes execute within each rings, with only the OS executing in the most privileged ring. As the most privileged process, the OS kernel is the only process that can access all data at any ring. This model creates a crucial disadvantage when relying on the OS for application security: if this process is successfully compromised, the entire system - including all applications and data – is also compromised.



In this scenario, an attacker does not need to directly attack the application to compromise it. A compromise to the OS could also result in less secure applications, regardless of developer safeguards within the application. Thus, the attack surface for an application also includes any OS vulnerability.

SGX as a solution

The Intel® Software Guard Extension™ (SGX) framework is an instruction set that allows developers to create secure regions of memory. These secure regions isolate data from an untrusted operating system.

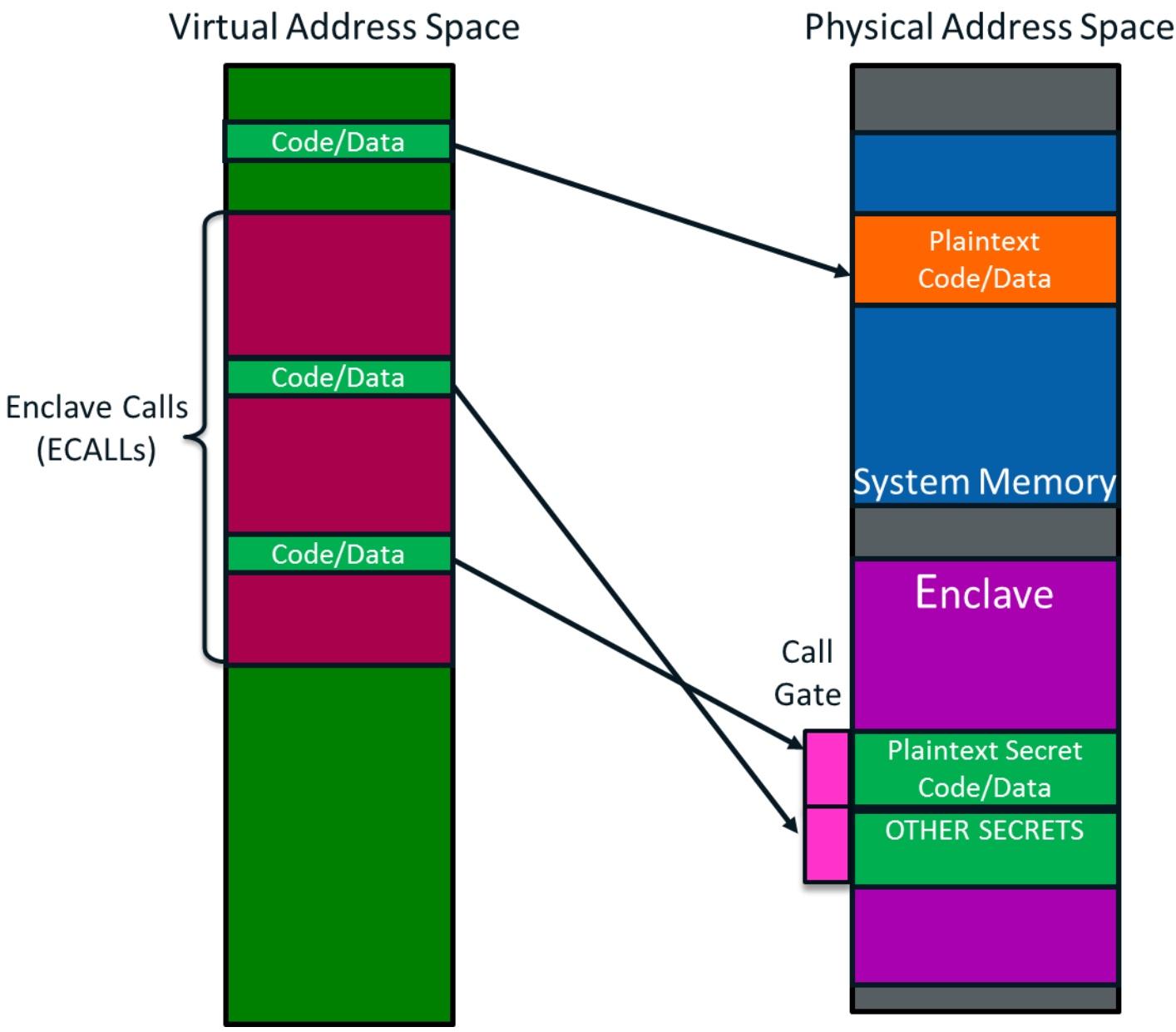
SGX enhances application security by shrinking the attack surface. Normally, unauthorized attempts to access data in another application are usually prevented through OS security. Since the OS manages all hardware on a system, any OS compromise results in a complete system compromise. This leaves the OS as a single point of failure in terms of system security.

Fortunately, SGX does not rely on the OS for security: the application is protected from attacks by privileged code, even if the attacker has physical control of the system. This is accomplished through the

use of secure enclaves. An enclave is a trusted memory region within a process. This protects data within the region from unauthorized access and modification – including direct attacks on memory.

Enclave Structure

The enclave works with an application by storing the most sensitive parts of the program. Code in an enclave is written in the same way as code in a normal application, but the code resides in a separate address space.



An enclave is secure due to the method of loading and initializing an enclave, which prevents direct access to the data by an untrusted application and prevents modification without detection. Placement of enclave content in physical memory helps to protect the data in the event of an OS or BIOS compromise, or virtual machine monitoring, it limits the size of an enclave due to physical memory limitations. Secure code in the enclave is called using enclave calls, or ECALLs. ECALLs are SGX functions calls to access a secure enclave.

To use an enclave within an application, a developer must first identify the secret. A secret is the most important component of the application that must be protected from unauthorized access. A secret can be sensitive data such as passwords, or a proprietary algorithm. After identifying the secret, it is placed

into an enclave. An enclave operates at the CPU-level and uses low-level memory. Therefore, while possible in some cases, an entire application either should not or cannot be placed in an enclave due to memory availability.

SGX as a Solution

This series will demonstrate a practical use of SGX will a real-world example. Not only does it contain coding demonstrations - including how to create an enclave - it also shows considerations for designing an enclave, such as what should and should not be in the enclave and trade-offs. This series assumes:

- Reader is between beginner and intermediate-level SGX user
- Reader knows the structure for writing a basic enclave. Details can be found here
- Reader has prior C/C++ knowledge, with programming experience on windows

This tutorial series will use a basic password manager and increase its security by extend it to use SGX. The user will learn about the SGX SDK and understand its vital role in securing sensitive information on a machine.

Part 2: Application Design

Goals:

- Explain the app
- Apply principles in part 1 to the real-world app
- Design the application

Application Description

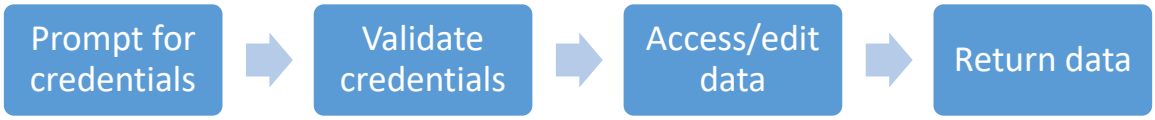
The sample application used for this series is a password manager. It will store usernames, passwords, and metadata information, and use an encryption algorithm to protect the data. The objective is to

show how SGX can be used to enhance the password manager and provide additional security measures.

A password manager authenticates user credentials by storing username-password-metadata pairs and validating against a user's attempt. This requires multiple parts: asking a user for credentials, authenticating credentials, storing credentials and data, editing credentials and data, and returning data to the user.

Application Design

Break the code into logic components.



To determine how to extend the password manager with SGX, the developer must identify which component(s) contain the 'secret'. The first step in developing an SGX application is identifying critical data that must be protected. When considering a password manager, the obvious secret would be the valid username-password combinations. After identifying the secret(s), we must isolate only the component(s) that access the secret(s).

Where is the secret? Let's examine the components:

- **Prompt for credentials**: The program asks the user for credentials. At this point, we do not know if the credentials are valid. The prompted data is not sensitive until authenticated.
- **Validate credentials**: Credentials are encrypted and/or decrypted to determine validity.
- **Access/edit data**: Raw, sensitive data is altered, possibly using proprietary algorithms, along with metadata.
- **Return data**: The data is finalized returned to the user, if necessary. The user only needs to see metadata, since there is no scenario where the user needs to see either an encrypted or decrypted password.

As mentioned in part 1, a developer could attempt to put the entire application into an enclave. With an arbitrary-sized password database, this approach could result in using too much memory – a self-inflicted denial-of-service. Due to memory and performance considerations, users should put the least possible amount of code in an SGX enclave. From the application flow chart, we see that the most sensitive components are validating and editing the data. These secrets are the parts that should be protected with SGX, while the remaining components can be placed in untrusted memory. We are ignoring communication through data transfer (the arrows in the flow chart) for now. This will be addressed in part 6.

Component Placement

The next step is identifying resources used to protect the secret, if any. This is also considered critical data. In our case, an additional resource would be the proprietary encryption and decryption algorithms. Any component that uses this resource must also be protected by an enclave. Otherwise, the secret could be compromised without needing to attack an enclave.

After determining the secret for the application, and the resources used to protect the secret, we can break down the components into the application and enclave functions based on which components either holds a secret or contains sensitive resources to protect a secret. This results in the following program structure:

-**Trusted component** (SGX ECALLs)

Encrypt data

Decrypt data

Editing data

-**Untrusted component**

Prompt for user credentials

Return data to user

The items in the untrusted component are normal functions that will be written in the main application. The trusted components are functions that will be placed in an enclave using SGX methodology.