Trusted Reference Monitors for Linux using Intel SGX Enclaves

Alexander Harri Bell-Thomas
Jesus College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Engineering in Computer Science

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: Alexander.Bell-Thomas@cl.cam.ac.uk

June 3, 2020

Declaration

I, Alexander Harri Bell-Thomas of Jesus College, being a candidate the Part III of the Computer Science Tripos, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 0

Signed:

Date:

Abstract

Write a summary of the whole thing. Make sure it fits in one page.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The task of defending computer systems against malicious programs and affording isolation to protected system components has always been exceedingly challenging to achieve. A system's *Trusted Computing Base*, or *TCB*, defines the minimal set of software, fimrware, and hardware components critical to establish and maintain system security and integrity. This traditionally includes, amogst others; the OS kernel; device drivers; device firmware; and the hardware itself. Compromise of a trusted component inside a system's *TCB* is a direct threat to any secure application running on it. A common approach to hardening a system's security is to minimise its *TCB*, diminishing its potential *attack surface*.

A modern trend is to outsource the physical layer of a system to a foreign party, for example a *cloud provider* — this is beneficial both in terms of cost and flexibility, but confuses many security considerations which assume that the physical layer itself can be trusted. In this context there is no guarantee of this, as the physical layer is usually provided as a *virtual machine*, inflating the system's *TCB* with an external and transparent software layer, the underlying *hypervisor*.

The concept of *Trusted Execution Environments*, *TEEs*, has been explored by the security community for a very long time as potential protection against this, providing isolated processing contexts in which an operation can be securely executed irrespective of the rest of the system — one such example is software *enclaves*. *Enclaves* are general-purpose *TEEs* provided by a CPU itself, protecting the logic found inside at the architectural level. Intel's Software Guard Extensions (SGX) are the most prolific example, affording a *black-box* environment and runtime for arbitrary apps to execute under. Introduced by Intel's *Skylake* architecture, a partial view of the platform's working can be found in whitepapers and previous publications.

SGX provides a *TEE* by enforcing the following:

1. Isolating a protected application, coined an enclave, from all other processes on the system at any priviledge level using hardware enforcement.
2. Protecting reserved memory against attacks using a dedicated hardware component, the *Memory Encryption Engine* (*MEE*).

... bit more here, talk about attestation and measurement briefly

A common usage pattern for enclaves in modern production systems is to build on top of a *libOS*.[1] This approach sees a trusted application built to depend on a modified operating system which is loaded alongside it in the enclave. Examples include *SGX-LKL*, *Graphene*, and *Occlum*. These projects allow SGX-unaware applications to be inexpensively ported into enclaves, but drastically inflate the *TCB* of the resulting program.

The aim of this work is to explore methods of hardening Linux with an SGX-driven *reference monitor* to track and protect host OS system resources using *information flow control* methods. Further, it aims to investigate whether bundling an entire operating environment into an enclave is a necessity, instead asking if simply using the host operating system, once hardened, would suffice in some situations.

Our Contribution

This work provides:

[1] Library Operating System

- A prototype implementation of an enclave-based, modular *reference monitor*, empowering *information flow control* techniques to operate with autonomy and protection from the host operating system. Enforcement is achieved using a modified Linux kernel, with an overall *TCB* including only a minimal footprint of the core kernel alongside the enclave application.

- A userspace interposition library to near-transparently integrate unmodified applications to fully function under the new restrictions.

- A full port of the *libtomcrypt* cryptography library for use inside an SGX enclave.

- A rigorous investigation of the performance implications of this approach, featuring a lightly-modified version of an *Nginx* production webserver. Worst-case performance shows a 35% decrease in request throughput, with the common case reporting 7-11%. Additionally we report a median overhead of 39μs (IQR 26-72μs, $n = 10^6$) per affected *system call*, matching or surpassing similar, non enclave-based, systems.

Chapter 2

Background

This chapter will cover a number of topics essential to understanding the rationale and implementation of the design as discussed in § 4. These include; Intel SGX, a brief overview of modern *libOSes*, an introduction to *Information Flow Control* (*IFC*), and a overview of key aspects of the Linux kernel relevant to the architecture of the prototype.

2.1 Intel SGX

Intel's Software Guard Extensions, SGX, was first announced and detailed in a handful of whitepaper documents published in 2013. [X,Y,Z] It described a novel approach, creating in-CPU containers with their own protected memory pools. These regions, called *enclaves*, cannot be read from or written due to fundamental protection mechanisms provided by the x86 architecture. *Enclaves* provide both integrity and secrecy to the operation running inside of it, even in the prescence of a malicious host.

Motivation At a high level SGX aims to achieve security for sensitive application by shielding them, and the resources it uses, against tampering and to provide a guarantee to end users about the an enclaves integrity; this is achieved using measurement and attestation. A driving use case is in a cloud computing context, where users are forced to trust a foreign party with both their data and business logic. By distributing

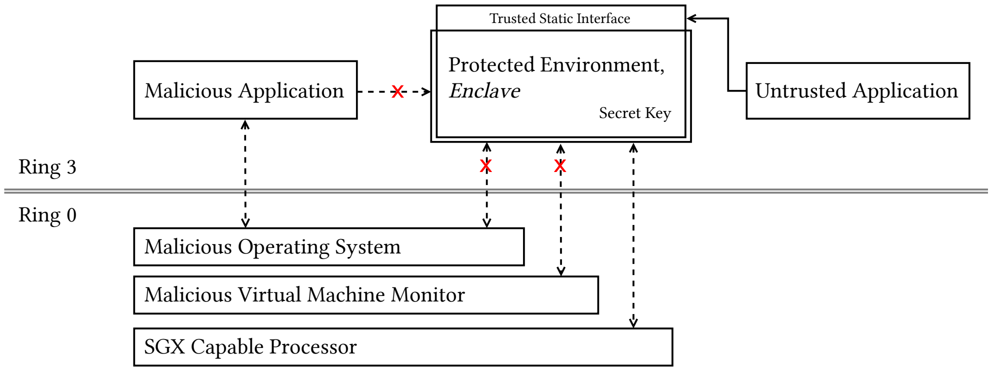

Figure 2.1: Abstract overview of an SGX enclave's protections.

encrypted, yet executable, containers targetting a single, unique SGX CPU, users can be assured that their information is safe, regardless of any virutalisation that may be taking place. Only the provisioned CPU is able to decrypt and execute the enclave, strictly in accordance with the restrictions of the SGX platform.

2.1.1 Security Characteristics

At its heart SGX is designed to be *trustworthy*; this is achieved in a number of ways, including robust enclaving provisioning, sealing and attestation. Intel enumerates SGX's protection as follows;

- Memory security against observation and modification from outside the enclave; this is achieved using an in-die *Memory Encryption Engine* (*MEE*), with a secret that rotates on every boot. This protection notably works against a host hypervisor, other enclaves, and anything running in supervisor mode.
- Attestation of an enclave to a challenger through the use of a permanent hardware security key for asymmetric encryption.
- Proxied software calls to prepare and transfer control in and out of an enclave. Arguments are securely marshalled according to a static enclave definition.
- SGX does not defend against reverse engineering or sidechannel attacks: [X,Y] this is the responsibility of the developer to mitigate.
- Debugging support is only provided via a specialised tool and only when an

enclave is compiled with debugging enabled.

2.1.2 Architecture and Implementation

Kernel module, driver. SDK.

2.1.3 Enclave Lifecycle

2.1.4 Attestation

2.1.5 Provisioning and Sealing

2.2 Modern *libOSes*

2.3 Information Flow Control

2.4 Aspects of the *Linux* Kernel

Chapter 3

Related Work

Chapter 4

Design and Implementation

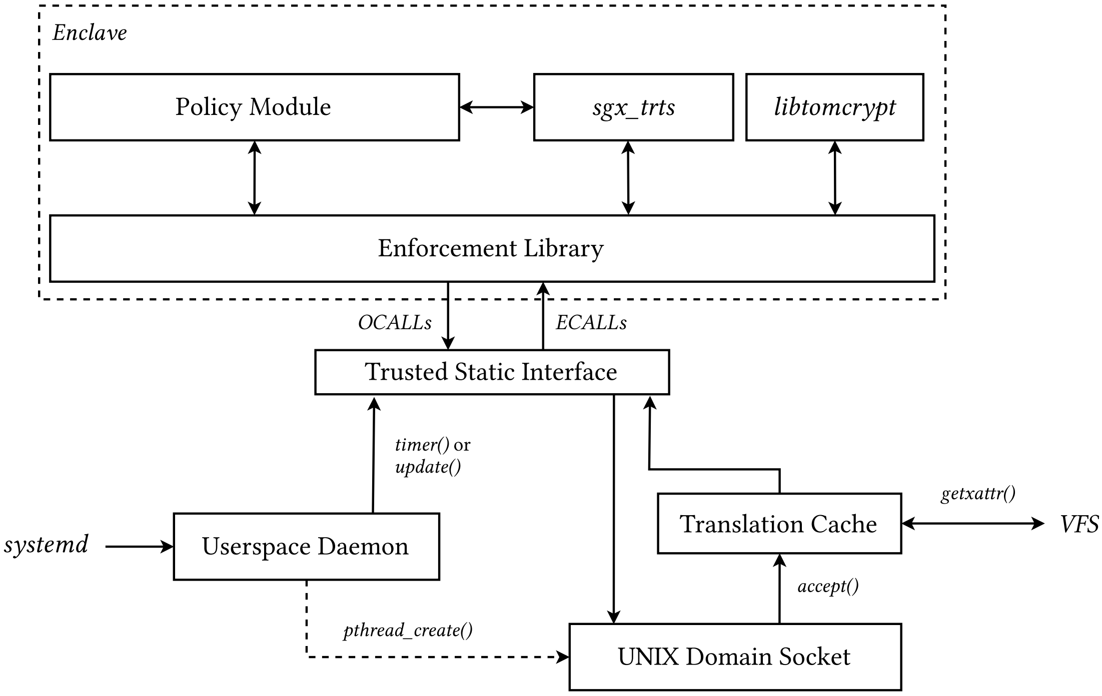

Figure 4.1: Abstract overview of an SGX enclave's protections.

Chapter 5

Evaluation

Chapter 6

Summary and Conclusions

Bibliography