User-Controlled Hardware Security Anchors: Evaluation And Designs

Mark Ryan, Flavio Garcia, David Oswald and Eduard Marin

2nd RISE Annual Conference
London, 21/11/2019
Attacks
Lots of attacks in recent years...

All these attacks exploit microarchitectural bugs or side channels at the hardware level.
Enclave shielding runtimes

TEE promise: enclave == “secure oasis” in a hostile environment but application writers and compilers are largely unaware of isolation boundaries!

**Trusted shielding runtime transparently acts as a secure bridge on enclave entry/exit**
A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes

Jo Van Bulck
imec-Distrinet, KU Leuven
jo.vanbulck@kuleuven.be

Abdulla Aldosseri
The University of Birmingham, UK
a.a.1170@student.bham.ac.uk

David Oswald
The University of Birmingham, UK
d.f.oswald@cs.bham.ac.uk

Flavio D. Garcia
The University of Birmingham, UK
t.f.garcia@cs.bham.ac.uk

Eduard Marin
The University of Birmingham, UK
e.marin@cs.bham.ac.uk

Frank Piessens
imec-Distrinet, KU Leuven
frank.piessens@kuleuven.be

ABSTRACT
This paper analyzes the vulnerability space arising in Trusted Execution Environments (TEEs) when interfacing a trusted enclave application with untrusted, potentially malicious code. Considerable research and industry effort has gone into developing TEE runtime libraries with the purpose of transparently shielding enclave application code from an adversarial environment. However, our analysis reveals that shielding requirements are generally not well-understood in real-world TEE runtime implementations. We expose several sanitization vulnerabilities at the level of the Application Binary Interface (ABI) and the Application Programming Interface (API) that can lead to exploitable memory safety and side-channel vulnerabilities in the compiled enclave. Mitigation of these vulnerabilities is not as simple as ensuring that pointers are outside enclave memory. In fact, we demonstrate that state-of-the-art sanitization techniques such as Intel’s strip-4, Microsoft’s “deep copy marshaling”, or even memory-safe languages like Rust fail to fully eliminate this attack surface. Our analysis reveals 35 enclave interface sanitization vulnerabilities in 8 major open-source shielding frameworks for Intel SGX, RISC-V, and Sancus TEEs. We practically exploit these vulnerabilities in several attack scenarios to leak secret keys from the enclave or enable remote code reuse.

1 INTRODUCTION
Minimization of the Trusted Computing Base (TCB) has always been one of the key principles underlying the field of computer security. With an ongoing stream of vulnerabilities in mainstream operating system and privileged hypervisor software layers, Trusted Execution Environments (TEEs) [28] have been developed as a promising new security paradigm to establish strong hardware-backed security guarantees. TEEs such as Intel SGX [8], ARM TrustZone [34], RISC-V Keystore [23], or Sancus [32] realize isolation and attestation of secure application compartments, called enclaves. Essentially, TEEs enforce a dual-world view, where even compromised or malicious system software in the normal world cannot gain access to the memory space of enclaves running in an isolated secure world on the same processor. This property allows for strong TCB reductions: only the code running in the secure world needs to be trusted for enclave computation results. Nevertheless, TEEs merely offer a relatively coarse-grained memory isolation primitive at the hardware level, leaving it up to the enclave developer to maintain useful security properties at the software level. This can become particularly complex when dealing with interactions between the untrusted host OS and the secure enclave, e.g., sending or receiving data to or from the enclave. For this rea-
What did we find?

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>SGX-SDK</th>
<th>OpenEnclave</th>
<th>Graphene</th>
<th>SGX-LKL</th>
<th>Rust-EDP</th>
<th>Asylo</th>
<th>Keystone</th>
<th>Sanctus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tier 1 (ABI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#1 Entry status flags sanitization</td>
<td>⭐️</td>
<td>⭐️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
<tr>
<td>#2 Entry stack pointer restore</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
<tr>
<td>#3 Exit register leakage</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
<tr>
<td>Tier 2 (API)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#4 Missing pointer range check</td>
<td>⚫️</td>
<td>⭐️</td>
<td>⭐️</td>
<td>⭐️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
<tr>
<td>#5 Null-terminated string handling</td>
<td>⭐️</td>
<td>⭐️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
<tr>
<td>#6 Integer overflow in range check</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
<tr>
<td>#7 Incorrect pointer range check</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
<tr>
<td>#8 Double fetch untrusted pointer</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
<tr>
<td>#9 Ocall return value not checked</td>
<td>⭐️</td>
<td>⭐️</td>
<td>⭐️</td>
<td>⭐️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
<tr>
<td>#10 Uninitialized padding leakage</td>
<td>[23]</td>
<td>⭐️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
<td>⚫️</td>
</tr>
</tbody>
</table>

Summary: > 35 enclave interface sanitization vulnerabilities across 8 projects
What did we find?
Responsible disclosure

• Fortanix-EDP => Security patch in the Rust compiler
• Intel SGX => CVE-2018-3626 and CVE-2019-14565
• Findings in open-source projects have been acknowledged in Github

All of our attack code is available in Github:
https://github.com/jovanbulck/0xbadc0de.
Paper at S&P 2020!!
Defenses
Software defined networks

Different apps written by app developers (App store)

Physical switches can now be divided into several various switches
Improve user authentication

• Use context information (e.g., geographical location)
  => Trusted input from sensors

• Bring your own device (BYOD) scenarios
  => Adversary can have root privileges in the device