KU LEUVEN

Safe Interacting Enclaves for Heterogeneous Protected Module Architectures

Sten Verbois

29 June 2018

Content

Motivation

- When deploying software, you have to trust the platform
- Isolation on OS level (e.g. LXC, Docker) is not enough

Motivation

- Protected Module Architectures provide isolation even from privileged software
- Guarantees integrity and confidentiality
- Two relevant implementations:
  - Intel Software Guard Extensions
  - Sancus
- Guarantees only hold when source code does not contain memory corruption bugs
  - e.g. buffer overflow, use-after-free, ...
- Guidelines and best practices are available

But...

Solutions

- Formal verification [1]
- Static analysis [2, 3]
- Program transformations / Runtime checks [4, 5, 6]
- Safe programming language
  - Good candidate: *Rust*

KU LEUVEN

Motivation (cont.)

- Modules require interaction with
  - untrusted context
  - other protected modules
  - other I/O (input devices, actuators)
- Interaction with secure components should be secure as well
- Case study: securing vehicular communication with VulCAN

Research Goals

- Specify requirements for setting up secure communication channels between protected modules
- Advance the VulCAN design by proposing an attestation server enclave implementation
- Use Rust to develop such enclaves with a small TCB while efficiently eliminating memory corruption vulnerabilities

Why Rust?

- Systems programming language
- Zero-cost abstractions
- Guarantees memory and thread safety
- Optional standard library
- Helpful error reporting

https://www.rust-lang.org

Ownership and Borrowing

```rust
fn main() {
    let mut v = Vec::new();
    v.push(1);
    borrow(&v); // OK, v borrowed
    v.push(2);
    take(v);
//         - value moved here
    v.push(3); // ERROR
//  ^ value used here after move
}
```

```rust
fn borrow(v: &Vec<i32>) {
    // ...
} // Borrow ends here

fn take(v: Vec<i32>) {
    // ...
}
```

Listing 1: Code snippet demonstrating the compiler-enforced ownership mechanism of Rust.

Rust enclaves

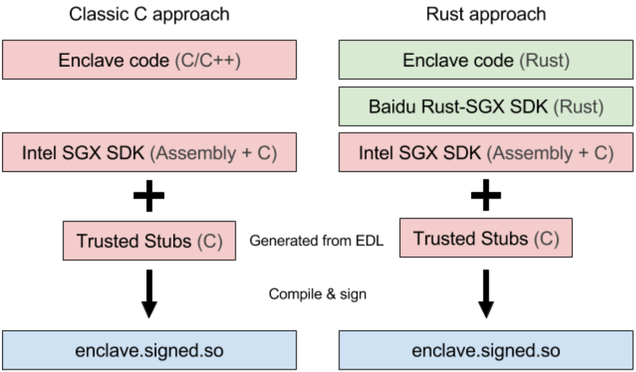

Figure: Developing SGX enclaves in C (left) and Rust (right).

Rust enclaves

- No Rust Standard Library available
- Baidu Rust-SGX SDK implements most of std in Intel SGX SDK
  - e.g. collections (String, Vector, HashMap), Rust-like file handling for SGX protected fs and Random number generation
- Other solutions: Graphene and Haven (Library OS), SCONE (Docker in Intel SGX)

Rust enclaves

- Near C performance for SPONGENT/SPONGEWRAP cryptographic functions

	C++		Rust	
Input length	256-bit	1024-bit	256-bit	1024-bit
Avg. cycles	54,368,526	201,213,242	50,396,303	184,812,155
Std. dev.	336,584	9,694,415	504,136	2,175,737

Table: Runtime performance comparison between C++ and Rust.

KU LEUVEN

Rust enclaves

- Minimal programmer effort
  - Alternatives: formal verification or exhaustive testing
  - Porting code from C/C++) to Rust is relatively easy

```c
char* input = ...;
int inputLeft = inputLength;
while (inputLeft) {
    // Use RATE bytes of input
    ...

    input += RATE;
    inputLeft -= RATE;
}
```

```rust
let input: &[u8] = ...;

for block in input.chunks(RATE) {
        // Use ‘block‘
        ...
}
```

Listing 2: Comparison between C (left) and Rust(right) of a common code pattern.

VulCAN Case Study: CAN

- CAN: Controller Area Network
  - Broadcast message bus used in vehicular control systems
  - 1991: First production vehicle from Mercedes-Benz [7]
- LeiA [8] & VatiCAN [9]
  - Authentication between Electronic Control Units (ECUs)
  - Prevents unauthorised ECUs from performing actions
- VulCAN [10]:
  - Protects against arbitrary code execution on ECUs
  - Put authentication protocol implementation in PMA

VulCAN Case Study

Figure: Picture of VulCAN hardware demo from [?].

VulCAN Case Study: Simplified

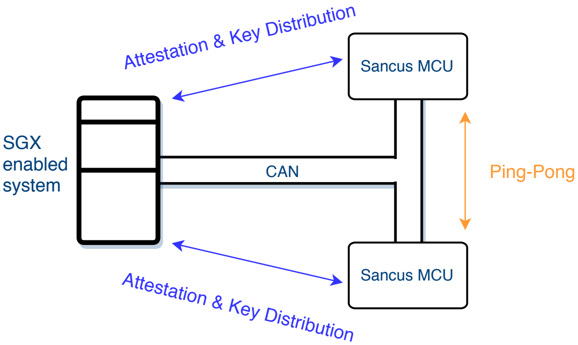

Figure: Schematic representation of hardware setup.

VulCAN Case Study: Logging Server

- Passive logging module
- If the log indicates an authenticated message, then it must have been produced by a trusted component in the network
- Non-repudiation but no availability
- Limited usability

VulCAN Case Study: Attestation Server

Attestation

- Goals
  - Expected module content ...
  - ... running on expected platform ...
  - ... with expected memory layout
- Challenge-response
- Utilizing module-specific key K_{PM}

KU LEUVEN

VulCAN Case Study: Attestation Server

Attestation

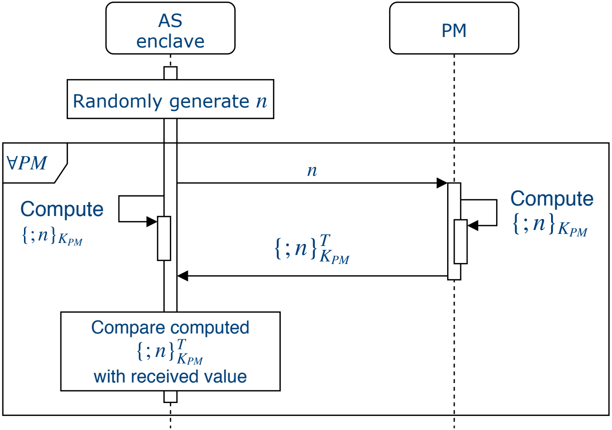

Figure: Attestation protocol between *AS* and each *PM*.

VulCAN Case Study: Attestation Server

Key distribution

- Goals
  - Securely deliver connection keys K_C to PMs
  - Proof received $\Rightarrow$ key successfully installed
- Again, encrypt payload with module-specific key K_{PM}

VulCAN Case Study: Attestation Server

Key distribution

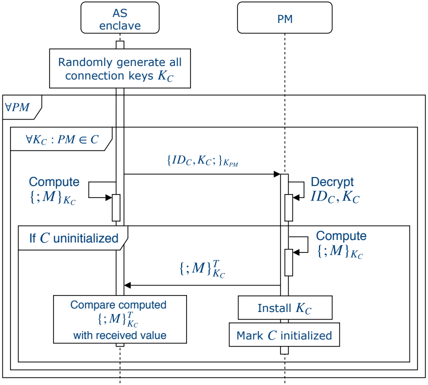

Figure: Key distribution protocol* between *AS* and each *PM*.

KU LEUVEN

Questions

Thank you for your attention!
Questions?

References I

- B. Jacobs, J. Smans, and F. Piessens, "The Verifast program verifier: A tutorial," *Tech. Rep.*, 2014.
- M. Das, S. Lerner, and M. Seigle, "ESP: Path-sensitive program verification in polynomial time," in *ACM Sigplan Notices*, vol. 37, pp. 57–68, ACM, 2002.
- J. Berdine, B. Cook, and S. Ishtiaq, "SLAyer: Memory safety for systems-level code," in *International Conference on Computer Aided Verification*, pp. 178–183, Springer, 2011.

KU LEUVEN

References II

- S. Nagarakatte, J. Zhao, M. M. Martin, and S. Zdancewic, "Softbound: Highly compatible and complete spatial memory safety for C," *ACM Sigplan Notices*, vol. 44, no. 6, pp. 245–258, 2009.
- P. Akritidis, M. Costa, M. Castro, and S. Hand, "Baggy Bounds Checking: An Efficient and Backwards-Compatible Defense against Out-of-Bounds Errors.," in *USENIX Security Symposium*, pp. 51–66, 2009.
- K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov, "Addresssanitizer: A Fast Address Sanity Checker.," in *USENIX Annual Technical Conference*, pp. 309–318, 2012.

References III

- Wikipedia, "Can bus — Wikipedia, the free encyclopedia," 2017.
  [Online; accessed 09-December-2017].
- A.-I. Radu and F. D. Garcia, "Leia: a lightweight authentication protocol for can," in *European Symposium on Research in Computer Security*, pp. 283–300, Springer, 2016.
- S. Nürnberger and C. Rossow, "–vatiCAN–Vetted, Authenticated CAN Bus," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 106–124, Springer, 2016.

KU LEUVEN

References IV

J. Van Bulck, J. T. Mühlberg, and F. Piessens, "VulCAN: Efficient component authentication and software isolation for automotive control networks," in *Proceedings of the 33th Annual Computer Security Applications Conference (ACSAC'17)*, ACM, 2017.

KU LEUVEN