Developing Secure SGX Enclaves

New Challenges on the Horizon

Raoul Strackx Frank Piessens

imec - Distrinet, KU Leuven

December 12, 2016

A Short Trip Trough Memory Lane

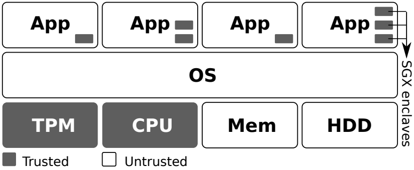
In the bad old days (a.k.a. today):

- Buffer overflows were rampant
- Computer systems were exploited frequently
- Update your system, run an AV and keep your fingers crossed

But in the near future. . .

Intel SGX to the rescue

- Extract sensitive parts of an application
- Isolate it from all privilege level



Unfortunately there are some assumptions. . .

SGX implementation:

- Provide perfect isolation
- Avoid side-channels[1]
- Presents problems to even low-end PMA (e.g., Sancus)
- **Out of scope for today**

Enclave implementation:

- Do not leak secrets!
- Integrate smoothly with new/legacy applications
- **How do we do this!?**

[1] Xu, Cui, and Peinado. "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems". 2015. *36th IEEE Symposium on Security and Privacy*

What do we need?

Fully abstract compilation: If we can't differentiate two enclaves at source code level, we also shouldn't be able to differentiate them at machine code level (and vice versa)

`Disclaimer: Some details are omitted for clarity!`

What do we need?

```
object o {
  int secret;

  int m() {
    if (secret == 0)
      return 0;
    else
      return 0;
  }
}
```

```
object o {
  int secret;

  int m() {
    return 0;
  }
```

What do we need?

```
object o {
  int secret;

  int m() {
    if (secret == 0)
      return 0;
    else
      return 0;
  }
}
```

```
object o {
  int secret;

  int m() {
    return 0;
  }
```

Only fully-abstract when:

- Regular registers are cleared
- Status registers are cleared
- Secure stack is used
- ...

What do we need?

```
object o {
  int m(bool x) {
    if ( x )
      return x;
    else
      return false;
  }
}
```

```
object o {
  int m(bool x) {
    if ( x )
      return true;
    else
      return false;
  }
}
```

What do we need?

```
object o {
  int m(bool x) {
    if ( x )
      return x;
    else
      return false;
  }
}
```

```
object o {
  int m(bool x) {
    if ( x )
      return true;
    else
      return false;
  }
}
```

Only fully-abstract if x is enforced to be

- `0000 0000` for `false`
- `0000 0001` for `true`

What do we need?

```
object o {
  b enclaved &m() {
    return new b();
  }
}
```

```
object o {
  b enclaved &m() {
    new b();
    new b();
    new b();
    return new b();
  }
}
```

What do we need?

```
object o {
  b enclaved &m() {
    return new b();
  }
}
```

```
object o {
  b enclaved &m() {
    new b();
    new b();
    new b();
    return new b();
  }
}
```

Only fully-abstract if object ids are returned, not pointers.

How do we get there!?

`note that low-level vulnerabilities also break fully abstract compilation.`

Option 1: Memory-safe Languages

- Extremely rich languages
  - When to pass references/copy objects
  - How do to garbage collection
  - ...
- Provides a lot of compile-time data
- Candidates:
  - Java, C#, ... rely on a huge runtime system
  - Rust, ... are not widely known

→ The most secure, but a challenging option

Option 2: Automatic Hardening of Enclaves

- Wellknown languages: C/C++!
- Available toolchain!
- Existing applications[2]/libraries!
- Hardening legacy code has been toroughly studied

→ Fasted approach!

The same secure measures to ensure fully abstract compilation needs to be provided!

[2] Baumann, Peinado, and Hunt. “Shielding applications from an untrusted cloud with Haven”. 2014. *USENIX Symposium on Operating Systems Design and Implementation (OSDI’14)*

Option 2: Automatic Hardening of Enclaves

Problem 1: hardening approach assume that the *entire* is protected or at least non-malicious!

- Metadata needs to be strictly separated!
- Example: Garbage collection / Reference counting
- Disclosing information is hard:
  - Do not trust information from outside the enclave
  - Do not expose enclave internal state (does the enclave still have a reference?)
  - Multiple enclaves may need to co-operate

Not all existing approach are suitable

Option 2: Automatic Hardening of Enclaves

Problem 2: The attack model affects how we discuss things

- Example: PointGuard
- Idea: encrypt pointers to prevent buffer overflow exploits
- "[provides] integrity of pointers so that pointers cannot be modified in ways the programmer did not intend"
- This cannot replace translation of enclaved pointers to references
  - `xor` is not a real encryption mechanism
  - Easily forgeable
  - No integrity check
  - Leaks data on the internal state of the enclave
  - Does not deal with replays of stale references

Using "easy" solutions may actually hurt security

Option 2: Automatic Hardening of Enclaves

Problem 3: Most approaches assume (quasi) single-threaded applications

- `free(ptr); ptr = NULL;`
- Assumption that an attacker cannot influence scheduling
- Incorrect for enclaves!
- Weichbrodt et al. "AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves". 2016. *Proceedings of the 21st European Symposium on Research in Computing Security*

Option 2: Automatic Hardening of Enclaves

Problem 4: Not all information can be hidden from an attacker

- For example layout in memory (e.g., ASLR)
- Xu's[3] attack still isn't solved
- Ensure performance: enclave is limited in size and should not be in EPC memory completely
- Seo et al.[4] propose a finer-grained approach
- Increases security, but entropy may be limited

---

[3] Xu, Cui, and Peinado. "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems". 2015. *36th IEEE Symposium on Security and Privacy*

[4] "SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs". 2017. *Network and Distributed System Security Symposium (NDSS 2017)*

Conclusion

No silver bullet!

- Writing secure, fully abstract enclaves will not be easy!
- Catch-22:
  - Memory-safe languages: Hard + business world won't be eager to adopt
  - Automatic hardening unsafe languages: Hard + need to re-evaluate security

Questions?

raoul.strackx@cs.kuleuven.be