DR.SGX
Automated and Adjustable Side-Channel Protection for SGX using Data Location Randomization

Tommaso Frassetto
Ferdinand Brasser
Ahmad-Reza Sadeghi
TU Darmstadt, Germany

Srdjan Capkun
Kari Kostiainen
ETH Zurich, Switzerland

Alexandra Dmitrienko
University of Würzburg, Germany
The Big Picture

SGX

Side channel leakage

RAM

DR.SGX
Background: Intel Software Guard Extensions
Background: Intel Software Guard Extensions

EPC: Enclave Page Cache
Background: Intel Software Guard Extensions

EPC: Enclave Page Cache
Background: Intel Software Guard Extensions

EPC: Enclave Page Cache
Background: Intel Software Guard Extensions

EPC: Enclave Page Cache
Background: Intel Software Guard Extensions
Background: Intel Software Guard Extensions
Background: Side-channel Attacks

Paging

• Trivially achievable by a malicious OS
• Just mark all pages as unavailable, then monitor page faults

Caching

• Requires more involved attacks, e.g., Prime+Probe:
  1. Access cache lines
  2. Let victim execute
  3. Check if own lines still in cache

• Demonstrated against SGX

[Xu et al., Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. IEEE S&P ’15.]
[Brasser et al., Software Grand Exposure: SGX Cache Attacks Are Practical. WOOT ’17]
Side-channel Mitigations: Other Approaches

- **Side-channel resilient code**: Requires:
  - High expertise
  - Vast effort

- **Annotation-based protections**: Requires:
  - High expertise
  - Significant effort

- **Oblivious Execution**: Extremely high overhead (up to 220×)

[Ahmad et al., Obfusuro, NDSS 2019]
DR.SGX: Goals

• Semantic-agnostic side-channel protection for data accesses
• No program-specific knowledge required
• More practical overhead than Oblivious Execution
Adversary Model

The attacker can:

• Launch software-based side-channel attacks on enclave data
  • Caches + Paging
  • 64 byte granularity
• Fully control OS and host application

The attacker cannot:

• Use physical side channels
  • Power
• Target code accesses
  • DR.SGX only protect data accesses
• Use transient execution bugs
  • E.g., no Foreshadow
DR.SGX: Data Location Randomization

```
static const unsigned char FSb[256] = {
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
    0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    ...);

int mbedtls_aes_setkey_enc(...)
{
    ... (uint32_t) FSb[(RK[3] >> 8) & 0xFF]) ... 
} 
```
DR.SGX: Data Location Randomization

```
static const unsigned char FSb[256] = {
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5,
    0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
    ...
};

int mbedtls_aes_setkey_enc( ... ) {
    ... (uint32_t) FSb[(RK[3] >> 8) & 0xFF] ... 
}
```

Run-time address resolver
DR.SGX: Components

Compile time
- Data access instrumentation

Run time
- Efficient data permutation
- Run-time address resolver
- Oblivious (re-)randomization
DR.SGX: Compiler-based Instrumentation

\[
\begin{align*}
\text{var}[256] &= 42; \\
*(\text{var} + 256) &= 42; \\
*(\text{addr\_resolve}(\text{var} + 256)) &= 42;
\end{align*}
\]
Efficient Data Permutation

• How to efficiently store a permutation?
• A permutation table requires space
  • Very little storage space in the CPU
  • Storing the table in memory requires oblivious access to it...
• Solution: Small-domain Format-preserving Encryption
Efficient Data Permutation

Application cache line identifier → FFX (Format-Preserving Encryption) → Randomized cache line identifier

FFX (Format-Preserving Encryption) → Cache line offset (6 bits)

Cache line offset (6 bits) → FFX (Format-Preserving Encryption) → Randomized cache line identifier

Cache line offset (6 bits)
Improving Performance: Permutation Buffer

- FFX requires 10 AES encryptions
  - 100 AES rounds per address resolution
- Our solution: the Permutation Buffer
- Buffer of 256 recently used addresses
  - Substantially improves performance
- We access every cache line in the buffer at every access
  - No leakage due to the permutation buffer
(Re-)Randomization under attacker’s observation

- DR.SGX uses non-temporal instructions that evade caches
  - Attacker still gets page-granularity data
- Initial randomization: we access all pages for every write
  - Attacker gets no information during initial permutation
Security

• Security depends on the enclave’s access patterns:
  • No predictable accesses: Initial randomization is sufficient
  • Predictable accesses: Information may leak during 1-2 randomization cycles after the predictable access
• More details in the paper
Conclusion

• **DR.SGX** introduces semantic-agnostic side-channel protection for SGX
• Reasonable performance overhead due to optimized implementation
• Configurable side-channel protection
DR.SGX: Automated and Adjustable Side-Channel Protection for SGX using Data Location Randomization