CopyCat: Controlled Instruction-Level Attacks on Enclaves

- Daniel Moghimi
- Jo Van Bulck
- Nadia Heninger
- Frank Piessens
- Berk Sunar
• Intel Software Guard eXtensions (SGX)
Trusted Execution Environment (TEE) - Intel SGX

- Intel Software Guard eXtensions (SGX)
- **Enclave**: Hardware protected user-level software module
  - Mapped by the Operating System
  - Loaded by the user program
  - Authenticated and Encrypted by CPU
Trusted Execution Environment (TEE) - Intel SGX

• Intel Software Guard eXTensions (SGX)

• **Enclave:** Hardware protected user-level software module
  - Mapped by the Operating System
  - Loaded by the user program
  - Authenticated and Encrypted by CPU

• Protects against system level adversary

**New Attacker Model:**
Attacker gets full control over OS
• **Intel’s Responsibility**
  • Microcode Patches / Hardware mitigation
  • TCB Recovery
    • Old Keys are Revoked
    • Remote attestation succeeds only with mitigation.

---

• Intel’s Responsibility
  • Microcode Patches / Hardware mitigation
  • TCB Recovery
    • Old Keys are Revoked
    • Remote attestation succeeds only with mitigation.

SGX Attacks

Intel Hardware
  Foreshadow [1]
  Plundervolt [2]

Software Dev Responsibility

Intel SGX Attack Taxonomy

• Intel’s Responsibility
  • Microcode Patches / Hardware mitigation
  • TCB Recovery
    • Old Keys are Revoked
    • Remote attestation succeeds only with mitigation.
  • Hyperthreading is out
    • Remote Attestation Warning

• µarch Side Channel
  • Constant-time Coding
  • Flushing and Isolating buffers
  • Probabilistic

Intel SGX Attack Taxonomy

- Intel’s Responsibility
  - Microcode Patches / Hardware mitigation
  - TCB Recovery
    - Old Keys are Revoked
    - Remote attestation succeeds only with mitigation.
  - Hyperthreading is out
    - Remote Attestation Warning
- μarch Side Channel
  - Constant-time Coding
  - Flushing and Isolating buffers
  - Probabilistic
- Deterministic Attacks
  - Page Fault, A/D Bit, etc. (4kB Granularity)

---

CopyCat Attack
• Malicious OS controls the interrupt handler
• Malicious OS controls the interrupt handler
• A threshold to execute 1 or 0 instructions
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
CopyCat Attack

• Malicious OS controls the interrupt handler
• A threshold to execute 1 or 0 instructions
• Filtering Zeros out: Clear the A bit before, Check the A bit after
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the A bit before, Check the A bit after
- Deterministic Instruction Counting
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the A bit before, Check the A bit after
- Deterministic Instruction Counting
- Counting from start to end is not useful.
  - A Secondary oracle
  - Page table attack as a deterministic secondary oracle
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the A bit before, Check the A bit after
- Deterministic Instruction Counting
- Counting from start to end is not useful.
  - A Secondary oracle
  - Page table attack as a deterministic secondary oracle
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the A bit before, Check the A bit after
- Deterministic Instruction Counting
- Counting from start to end is not useful.
  - A Secondary oracle
  - Page table attack as a deterministic secondary oracle
CopyCat Attack

- Previous Controlled Channel attacks leak Page Access Patterns
- CopyCat additionally leaks number of instructions per page
if(c == 0) {
    r = add(r, d);
} else {
    r = add(r, s);
}

test %eax, %eax
je label
mov %edx, %esi
label:
call add
mov %eax, -0xc(%rbp)
if(c == 0) {
    r = add(r, d);
} else {
    r = add(r, s);
}

C Code

test %eax, %eax
je label
mov %edx, %esi
label:
call add
mov %eax, -0xc(%rbp)
if (c == 0) {
    r = add(r, d);
} else {
    r = add(r, s);
}

test %eax, %eax
je label
mov %edx, %esi
label:
call add
mov %eax, -0xc(%rbp)
if (c == 0) {
    r = add(r, d);
} else {
    r = add(r, s);
}

```
c C Code

```test %eax, %eax
je label
mov %edx, %esi
label:
call add
mov %eax, -0xc(%rbp)
```
Crypto means Cryptoattacks
Binary Extended Euclidean Algorithm (BEEA)

- Previous attacks only leak some of the branches w/ some noise

```plaintext
1: procedure MODINV(u, modulus v)
2: \[ b_0 \leftarrow 0, d_0 \leftarrow 1, u_0 \leftarrow u, v_0 \leftarrow v, \]
3: \begin{align*}
4: & \text{while isEven}(u_i) \text{ do} \\
5: & \quad u_i \leftarrow u_i / 2 \\
6: & \text{if isOdd}(b_i) \text{ then} \\
7: & \quad b_i \leftarrow b_i - u \\
8: & \quad b_i \leftarrow b_i / 2 \\
9: & \text{while isEven}(v_i) \text{ do} \\
10: & \quad v_i \leftarrow v_i / 2 \\
11: & \text{if isOdd}(d_i) \text{ then} \\
12: & \quad d_i \leftarrow d_i - u \\
13: & \quad d_i \leftarrow d_i / 2 \\
14: & \text{if } u_i > v_i \text{ then} \\
15: & \quad u_i \leftarrow u_i - v_i, b_i \leftarrow b_i - d_i \\
16: & \text{else} \\
17: & \quad v_i \leftarrow v_i - u_i, d_i \leftarrow d_i - b_i \\
18: \end{align*}
19: \text{return } d_i
```
• Previous attacks only leak some of the branches w/ some noise
• CopyCat synchronously leaks all the branches wo/ any noise

```plaintext
1: procedure MODINV(u, modulus v)
2:   b_i ← 0 d_i ← 1, u_i ← u, v_i = v,
3:   while isEven(u_i) do
4:     u_i ← u_i / 2
5:     if isOdd(b_i) then
6:       b_i ← b_i − u
7:     b_i ← b_i / 2
8:   while isEven(v_i) do
9:     v_i ← v_i / 2
10:    if isOdd(d_i) then
11:       d_i ← d_i − u
12:     d_i ← d_i / 2
13:     if u_i > v_i then
14:       u_i ← u_i − v_i, b_i ← b_i − d_i
15:     else
16:       v_i ← v_i − u_i, d_i ← d_i − b_i
17: return d_i
```
• Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \mod n$
  • Iterative over the entire recovered trace with $n$ as input $\Rightarrow k_{inv}$
  • Plug $k_{inv}$ in $s_1 = k_1^{-1}(h - r_1.x) \mod n \Rightarrow$ get private key $x$
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \mod n$
  - Iterative over the entire recovered trace with $n$ as input $\to k_{inv}$
  - Plug $k_{inv}$ in $s_1 = k_1^{-1}(h - r_1.x) \mod n \to$ get private key $x$

- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \mod p$
  - We know that $p.q = N$
CopyCat on WolfSSL - Cryptanalysis

• Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \mod n$
  • Iterative over the entire recovered trace with $n$ as input $\rightarrow k_{inv}$
  • Plug $k_{inv}$ in $s_1 = k_1^{-1}(h - r_1.x) \mod n \rightarrow$ get private key $x$

• Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \mod p$
  • We know that $p.q = N$
  • Branch and prune Algorithm with the help of the recovered trace

```
\begin{equation}
\begin{array}{c}
p = \ldots X \\
q = \ldots X \\
\end{array}
\end{equation}
\begin{align*}
p &= \ldots 0 \\
q &= \ldots 0 \\
p &= \ldots 0 \\
q &= \ldots 1 \\
p &= \ldots 1 \\
q &= \ldots 0 \\
p &= \ldots 1 \\
q &= \ldots 1
\end{align*}
```
CopyCat on WolfSSL - Cryptanalysis

• Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \mod n$
  • Iterative over the entire recovered trace with $n$ as input → $k_{inv}$
  • Plug $k_{inv}$ in $s_1 = k_1^{-1}(h - r_1.x) \mod n$ → get private key $x$

• Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \mod p$
  • We know that $p \cdot q = N$, and $N$ is public
  • Branch and prune Algorithm with the help of the recovered trace

\[ \begin{array}{c}
\text{p} = \ldots X \\
\text{q} = \ldots X
\end{array} \]

\[ \begin{array}{c}
\text{p} = \ldots X 0 \\
\text{q} = \ldots X 0
\end{array} \quad \begin{array}{c}
\text{p} = \ldots 0 \\
\text{q} = \ldots 1
\end{array} \quad \begin{array}{c}
\text{p} = \ldots 1 \\
\text{q} = \ldots 0
\end{array} \quad \begin{array}{c}
\text{p} = \ldots X 1 \\
\text{q} = \ldots X 1
\end{array} \]

\[ N = 1110 \]
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \mod n$
  - Iterative over the entire recovered trace with $n$ as input $\Rightarrow k_{inv}$
  - Plug $k_{inv}$ in $s_1 = k_1^{-1}(h - r_1.x) \mod n \Rightarrow$ get private key $x$

- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \mod p$
  - We know that $p.q = N$, and $N$ is public
  - Branch and prune Algorithm with the help of the recovered trace

\[
\begin{array}{c}
p = \ldots X \\
q = \ldots X \\
\hline
p = \ldots X 0 \\
q = \ldots X 0 \\
\hline
p = \ldots 0 \\
q = \ldots 1 \\
\hline
p = \ldots 1 \\
q = \ldots 0 \\
\hline
p = \ldots X 1 \\
q = \ldots X 1 \\
\hline
\end{array}
\]

\[
\begin{array}{c}
p = \ldots 0 0 \\
q = \ldots 1 0 \\
\hline
p = \ldots 1 0 \\
q = \ldots 0 0 \\
\hline
p = \ldots 0 0 \\
q = \ldots 1 0 \\
\hline
p = \ldots 1 1 \\
q = \ldots 0 1 \\
\end{array}
\]
CopyCat on WolfSSL - Cryptanalysis

• Single-trace Attack during DSA signing: \( k_{inv} = k^{-1} \mod n \)
  • Iterative over the entire recovered trace with \( n \) as input \( \Rightarrow k_{inv} \)
  • Plug \( k_{inv} \) in \( s_1 = k_1^{-1}(h - r_1.x) \mod n \) \( \Rightarrow \) get private key \( x \)

• Single-trace Attack during RSA Key Generation: \( q_{inv} = q^{-1} \mod p \)
  • We know that \( p \cdot q = N \), and \( N \) is public
  • Branch and prune Algorithm with the help of the recovered trace
• Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \mod n$
  - Iterative over the entire recovered trace with $n$ as input $\rightarrow k_{inv}$
  - Plug $k_{inv}$ in $s_1 = k_1^{-1}(h - r_1.x) \mod n \rightarrow$ get private key $x$

• Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \mod p$
  - We know that $p.q = N$, and $N$ is public
  - Branch and prune Algorithm with the help of the recovered trace
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: \( k_{inv} = k^{-1} \mod n \)
  - Iterative over the entire recovered trace with \( n \) as input \( \rightarrow k_{inv} \)
  - Plug \( k_{inv} \) in \( s_1 = k_1^{-1}(h - r_1.x) \mod n \) \( \rightarrow \) get private key \( x \)

- Single-trace Attack during RSA Key Generation: \( q_{inv} = q^{-1} \mod p \)
  - We know that \( p.q = N \), and \( N \) is public
  - Branch and prune Algorithm with the help of the recovered trace

- Single-trace Attack during RSA Key Generation: \( d = e^{-1} \mod \lambda(N) \)
CopyCat on WolfSSL - Cryptanalysis Results

• Executed each attack 100 times.
• DSA $k^{-1} \mod n$
  • Average 22,000 IRQs
  • 75 ms to iterate over an average of 6,320 steps
• RSA $q^{-1} \mod p$
  • Average 106490 IRQs
  • 365 ms to iterate over an average of 39,400 steps
• RSA $e^{-1} \mod \lambda(N)$
  • $e^{-1} \mod \lambda(N)$
  • Average 230,050 IRQs
  • 800ms to iterate over an average of 81,090 steps
• Experimental traces always match the leakage model in all experiments → Successful single-trace key recovery
How about other Crypto libraries?

- Libgcrypt uses a variant of BEEA
  - Single trace attack on DSA, Elgamal, ECDSA, RSA Key generation
- OpenSSL uses BEEA for computing GCD
  - Single trace attack on RSA Key generation when computing $\gcd(q - 1, p - 1)$

<table>
<thead>
<tr>
<th>Operation (Subroutine)</th>
<th>Implementation</th>
<th>Secret Branch</th>
<th>Exploitable</th>
<th>Computation → Vulnerable Callers</th>
<th>Single-Trace Attack</th>
</tr>
</thead>
<tbody>
<tr>
<td>WallSSL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalar Multiply (wc_ecm_mulmod_ex)</td>
<td>Montgomery Ladder w/ Branches</td>
<td>✓</td>
<td>✓</td>
<td>$(k \times G) \rightarrow \text{wc_eco_sign_hash} $</td>
<td>X</td>
</tr>
<tr>
<td>Greatest Common Divisor (fp_gcd)</td>
<td>Euclidean (Divisions)</td>
<td>✓</td>
<td>✗</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Modular Inverse (fp_invmod)</td>
<td>BEEA</td>
<td>✓</td>
<td>✓</td>
<td>$\gcd(q - 1, p - 1) \rightarrow \text{RSA_X931_derive_ex}$</td>
<td>✓</td>
</tr>
<tr>
<td>Libgcrypt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greatest Common Divisor (npi_gcd)</td>
<td>Euclidean (Divisions)</td>
<td>✓</td>
<td>✗</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Modular Inverse (npi_invmod)</td>
<td>Modified BEEA [43, Vol II, §4.5.2]</td>
<td>✓</td>
<td>✓</td>
<td>$\gcd(q - 1, p - 1) \rightarrow \text{RSA_X931_derive_ex}$</td>
<td>✓</td>
</tr>
<tr>
<td>OpenSSL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greatest Common Divisor (BN_gcd)</td>
<td>BEEA</td>
<td>✓</td>
<td>✓</td>
<td>$gcd(q - 1, p - 1) \rightarrow \text{RSA_X931_derive_ex}$</td>
<td>✓</td>
</tr>
<tr>
<td>Modular Inverse (BN_mod_inverse_no_branch)</td>
<td>BEEA w/ Branches</td>
<td>❌</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>IPP Crypto</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Greatest Common Divisor (ippsGcd_BN)</td>
<td>Modified Lehmer's GCD</td>
<td>✓</td>
<td>✗</td>
<td>$\gcd(q - 1, e) \rightarrow \text{cpIsCoPrime}$</td>
<td>N/A</td>
</tr>
<tr>
<td>Modular Inverse (ippsModInv_BN)</td>
<td>Euclidean (Divisions)</td>
<td>✓</td>
<td>✗</td>
<td>$\gcd(p - 1, q - 1) \rightarrow \text{isValidPrivl_rsa}$</td>
<td>N/A</td>
</tr>
</tbody>
</table>
WolfSSL fixed the issues in 4.3.0 and 4.4.0
  - Blinding for $k^{-1} \mod n$ and $e^{-1} \mod \lambda(N)$
  - Alternate formulation for $q^{-1} \mod p$: $q^{p-2} \mod p$
  - Using a constant-time (branchless) modular inverse [11]

Libgcrypt fixed the issues in 1.8.6
  - Using a constant-time (branchless) modular inverse [11]

OpenSSL fixed the issue in 1.1.1e
  - Using a constant-time (branchless) GCD algorithm [11]

Conclusion

• Instruction Level Granularity
  • Imbalance number of instructions
  • Leak the outcome of branches

• Fully Deterministic and reliable
  • Millions of instructions tested
  • Attacks match the exact leakage model of branches

• Easy to scale and replicate
  • No reverse engineering of branches and microarchitectural components
  • Tracking all the branches synchronously

• Branchless programming is hard!
Questions?!