Hunting the Haunter

Efficient Relational Symbolic Execution for Spectre with Haunted RelSE

Lesly-Ann Daniel  
CEA, LIST, Université Paris-Saclay  
France

Sébastien Bardin  
CEA, LIST, Université Paris-Saclay  
France

Tamara Rezk  
Inria  
France
Spectre attacks (2018)

• Exploit speculative execution in processors
• Affect almost all processors
• Attackers can force mispeculations: transient executions
• Transient executions are reverted at architectural level
• But not the microarchitectural state (e.g. cache)

Idea. Force victim to encode secret data in cache during transient execution & recover them with cache attacks
Spectre-PHT

Exploits conditional branch predictor

```java
if idx < size {
    v = tab[idx]
    leak(v)
}
```

- `idx` is attacker controlled
- `content of tab is public`
- `leak(v)` encodes `v` to cache

Regular execution

- Conditional bound check ensures `idx` is in bounds
- `v` contains public data
Spectre-PHT

Exploits conditional branch predictor

```java
if idx < size {
    v = tab[idx]
    leak(v)
}
```

- `idx` is attacker controlled
- `content of tab` is public
- `leak(v)` encodes `v` to cache

---

Regular execution

- Conditional bound check ensures `idx` is in bounds
- `v` contains public data

Transient Execution

- Conditional is misspeculated
- Out-of-bound array access \(\to\) load secret data in `v`
- `v` is leaked to the cache
Spectre-STL: Loads can speculatively bypass prior stores

Regular execution

\[
\begin{align*}
\text{store } a & \ s \\
\text{store } a & \ p \\
\text{store } b & \ q \\
v & = \text{load } a \\
\text{leak}(v)
\end{align*}
\]

- where \( s \) is secret, \( p \) and \( q \) are public
- where \( a \neq b \)
- \( \text{leak}(v) \) encodes \( v \) to cache
Spectre-STL: Loads can speculatively bypass prior stores

Regular execution  +  Transient Executions

\[
\begin{align*}
\text{store } & a \ s \\
\text{store } & a \ p \\
\text{store } & b \ q \\
v = & \text{load } a \\
\text{leak}(v) \\
\text{leak}(p)
\end{align*}
\]

\[
\begin{align*}
\text{store } & a \ s \\
\text{store } & a \ p \\
v = & \text{load } a \\
\text{store } & b \ q \\
\text{leak}(v) \\
\text{leak}(p)
\end{align*}
\]

- where \( s \) is secret, \( p \) and \( q \) are public
- where \( a \neq b \)
- \( \text{leak}(v) \) encodes \( v \) to cache
**Spectre-STL**: Loads can speculatively bypass prior stores

**Regular execution** + **Transient Executions**

\[
\begin{align*}
\text{store } a & \quad s \\
\text{store } a & \quad p \\
\text{store } b & \quad q \\
v & = \text{load } a \\
\text{leak}(v)
\end{align*}
\]

\[
\begin{align*}
\text{store } a & \quad s \\
\text{store } a & \quad p \\
v & = \text{load } a \\
\text{store } b & \quad q \\
\text{leak}(v)
\end{align*}
\]

\[
\begin{align*}
\text{store } a & \quad s \\
v & = \text{load } a \\
\text{store } a & \quad p \\
\text{store } b & \quad q \\
\text{leak}(v)
\end{align*}
\]

- where \( s \) is secret, \( p \) and \( q \) are public
- where \( a \neq b \)
- \(\text{leak}(v)\) encodes \( v \) to cache
Spectre-STL: Loads can speculatively bypass prior stores

Regular execution + Transient Executions

\[
\begin{align*}
\text{store } a & \quad s \\
\text{store } a & \quad p \\
\text{store } b & \quad q \\
v & = \text{load } a \\
\text{leak}(v) \\
\text{leak}(p)
\end{align*}
\]

\[
\begin{align*}
\text{store } a & \quad s \\
\text{store } a & \quad p \\
v & = \text{load } a \\
\text{store } b & \quad q \\
\text{leak}(v) \\
\text{leak}(p)
\end{align*}
\]

\[
\begin{align*}
\text{store } a & \quad s \\
v & = \text{load } a \\
\text{store } a & \quad p \\
\text{store } b & \quad q \\
\text{leak}(v) \\
\text{leak}(s)
\end{align*}
\]

\[
\begin{align*}
v & = \text{load } a \\
\text{store } a & \quad s \\
\text{store } a & \quad p \\
\text{store } b & \quad q \\
\text{leak}(v) \\
\text{leak}(\text{init_mem[a]})
\end{align*}
\]

- where $s$ is secret, $p$ and $q$ are public
- where $a \neq b$
- $\text{leak}(v)$ encodes $v$ to cache
Detect Spectre attacks?

Challenging!

- Counter-intuitive semantics
- Path explosion:
  - Spectre-STL: all possible load/store interleavings!
- Needs to hold at binary-level

Path explosion for Spectre-STL on Litmus tests (328 instr.)

<table>
<thead>
<tr>
<th>Semantics</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular semantics</td>
<td>14</td>
</tr>
<tr>
<td>Speculative semantics (Spectre-STL)</td>
<td>37M</td>
</tr>
</tbody>
</table>
Goal: New verification tools for Spectre

Goal. We need new verification tools to detect Spectre attacks!

Proposal. → Verify Speculative Constant Time (SCT) property → Use Relational Symbolic Execution (RelSE)

Challenge. Model new transient behaviors avoiding path explosion
No efficient verification tools for Spectre 😞

<table>
<thead>
<tr>
<th>Target</th>
<th>Spectre-PHT</th>
<th>Spectre-STL</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLEESpectre [1]</td>
<td>LLVM</td>
<td>🥳</td>
</tr>
<tr>
<td>SpecuSym [2]</td>
<td>LLVM</td>
<td>🤦</td>
</tr>
<tr>
<td>FASS [3]</td>
<td>Binary</td>
<td>🤦</td>
</tr>
<tr>
<td>Spectector [4]</td>
<td>Binary</td>
<td>🤦</td>
</tr>
<tr>
<td>Pitchfork [5]</td>
<td>Binary</td>
<td>🤦</td>
</tr>
</tbody>
</table>

Legend

😊 Good perfs. on crypto
😊 Good on small programs
😊 Limited perfs. On crypto
😊 Limited to small programs

LLVM analysis might miss SCT violations 😞

### No efficient verification tools for Spectre?

<table>
<thead>
<tr>
<th></th>
<th>Target</th>
<th>Spectre-PHT</th>
<th>Spectre-STL</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLEESpectre [1]</td>
<td>LLVM</td>
<td>😊</td>
<td>-</td>
</tr>
<tr>
<td>SpecuSym [2]</td>
<td>LLVM</td>
<td>😞</td>
<td>-</td>
</tr>
<tr>
<td>FASS [3]</td>
<td>Binary</td>
<td>😞</td>
<td>-</td>
</tr>
<tr>
<td>Spectector [4]</td>
<td>Binary</td>
<td>😞</td>
<td>😞</td>
</tr>
<tr>
<td>Pitchfork [5]</td>
<td>Binary</td>
<td>😊</td>
<td>😞</td>
</tr>
<tr>
<td>Binsec/Haunted</td>
<td>Binary</td>
<td>😊</td>
<td>😞</td>
</tr>
</tbody>
</table>

**Legend**

- 😊: Good perfs. on crypto
- 😞: Limited perfs. on crypto
- 😞: Limited to small programs
- 😞: LLVM analysis might miss SCT violations

---


Contributions

Haunted RelSE optimization
• Model transient and regular behaviors at the same time
  • Spectre-PHT: pruning redundant paths
  • Spectre-STL: pruning + encoding to merge paths
• Formal proof: equivalence with explicit exploration [in the paper]

Binsec/Haunted, binary-level verification tool
• Experimental evaluation on real world crypto (donna, libsodium, OpenSSL)
• Efficient on real-world crypto for Spectre-PHT 😞 → 😊
• Efficient on small programs for Spectre-STL 😞 → 😊
• Comparison with SoA: faster & more vulnerabilities found

New Spectre-STL violations
• Index-masking (countermeasure against Spectre-PHT) + proven mitigations
• Code introduced for Position-Independent-Code [in the paper]
Haunted RelSE for Spectre-PHT
Symbolic execution. An illustration.

if \( c \) then foo else bar

\[ \pi \land c \]
\[ \pi \land \neg c \]

2 regular paths
Explicit RelSE for Spectre PHT

**Spectre-PHT.** Conditional branches can be executed speculatively

```
if c
  then foo
else bar
```

- 2 regular paths
- + 2 extra transient paths

On regular and transient branches:
- Verify no secret can leak.

**Explicit RelSE.**

Fork execution into 4 at conditionals:
- 2 regular branches
- 2 transient branches (until max speculation depth)

(e.g. KLEESpectre)
Haunted RelSE for Spectre PHT

### Spectre-PHT

Conditional branches can be executed speculatively

- **if** \(c\) **then** foo **else** bar

2 speculative paths:

- \(\pi \land (c \lor \neg c)\)
- \(\pi \land (c \lor \neg c)\)

\[\begin{align*}
\pi \land c \\
\pi \land c
\end{align*}\]

- \(\pi \land \neg c\)
- \(\pi \land \neg c\)

### Haunted RelSE

Fork execution into 2 speculative paths:
- **speculative** = regular \(\lor\) transient
- After max spec. depth, add constraint to invalidate transient path

\(\rightarrow\) can spare two paths at conditionals
Haunted RelSE for Spectre-STL
Explicit RelSE for Spectre-STL

\[
\begin{align*}
\text{store } a & \ \text{s} \\
\text{store } a & \ \text{p} \\
\text{store } b & \ \text{q} \\
v & = \text{load } a
\end{align*}
\]

where \( a \neq b \)
Explicit RelSE for Spectre-STL

**Spectre-STL.** Loads can speculatively bypass prior stores

\[
\begin{align*}
\text{store } a & \rightarrow s \\
\text{store } a & \rightarrow p \\
\text{store } b & \rightarrow q \\
v & = \text{load } a \\
\end{align*}
\]

where \( a \neq b \)

\[
\begin{align*}
\text{store } a & \rightarrow s \\
\text{store } a & \rightarrow p \\
v & = \text{load } a \\
\text{store } b & \rightarrow q \\
\end{align*}
\]

\[
\begin{align*}
\text{store } a & \rightarrow s \\
\text{store } a & \rightarrow p \\
\text{store } b & \rightarrow q \\
v & = \text{load } a \\
\end{align*}
\]

\[ v \mapsto \beta \]

\[ v \mapsto \gamma \]

**1 regular path**

\[ + 3 \text{ extra transient paths} \]

**Explicit RelSE.**

At load instructions: fork execution for each load/store interleaving.

→ Path explosion

(e.g. Pitchfork)
Explicit RelSE for Spectre-STL

**Spectre-STL.** Loads can speculatively bypass prior stores

\[
\begin{align*}
\text{store } a & \rightarrow s \\
\text{store } a & \rightarrow p \\
\text{store } b & \rightarrow q \\
v & \rightarrow \text{load } a \\
\end{align*}
\]

where \( a \neq b \)

\[
\begin{align*}
\text{store } a & \rightarrow s \\
\text{store } a & \rightarrow p \\
v & \rightarrow \text{load } a \\
\text{store } b & \rightarrow q \\
\text{store } a & \rightarrow s \\
\text{store } a & \rightarrow p \\
\text{store } b & \rightarrow q \\
\end{align*}
\]

1 regular path
+ 3 extra transient paths

**Redundant case**
Can be eliminated with \textit{read-over-write}
Explicit RelSE for Spectre-STL

**Spectre-STL.** Loads can speculatively bypass prior stores

\[
\begin{align*}
\text{store a s} \\
\text{store a p} \\
\text{store b q} \\
\text{v = load a}
\end{align*}
\]

\[
\begin{align*}
\text{store a s} \\
\text{store a p} \\
\text{v = load a} \\
\text{store b q}
\end{align*}
\]

where \(a \neq b\)

---

\[
\begin{align*}
\text{store a s} \\
\text{store a p} \\
\text{store b q} \\
\text{v = load a}
\end{align*}
\]

---

**Haunted RelSE.**
- Cut redundant cases
- Encode remaining ones in 1 path
  - symbolic \(\text{ite}\)
  - free booleans \(\beta_0, \beta_1\)

\[
v \mapsto \text{ite } \beta_0 \text{ then } \alpha \text{ else } (\text{ite } \beta_1 \text{ then } s \text{ else } p)
\]

\[
\begin{align*}
\beta_0 &= \text{false} \\
\beta_1 &= \text{false}
\end{align*}
\]
Experimental evaluation
Experimental evaluation

Binsec/Haunted. Implementation of Haunted RelSE

More details on Feb, 25th at LASER workshop!

Benchmark.
- Litmus tests (46 small test cases)
- Cryptographic primitives tea & donna
- More complex cryptographic primitives
  - Libsodium secretbox
  - OpenSSL ssl3-digest-record
  - OpenSSL mee-cdc-decrypt

Experiments.

RQ1. Effective on real code?
→ Spectre-PHT ☑ & Spectre-STL ☒

RQ2. Haunted vs. Explicit?
→ Spectre-PHT: ≈ or ↑ & Spectre-STL: always ↑

RQ3. Comparison against KLEESpectre & Pitchfork
→ Spectre-PHT: ≈ or ↑ & Spectre-STL: always ↑ [in paper]
Haunted vs. Explicit for Spectre-PHT

<table>
<thead>
<tr>
<th>Litmus tests (32 programs)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Paths</td>
<td>Time</td>
</tr>
<tr>
<td>Explicit</td>
<td>1546</td>
</tr>
<tr>
<td>Haunted</td>
<td>370</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Libsodium &amp; OpenSSL (3 programs)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X86 Instr.</td>
<td>Time</td>
</tr>
<tr>
<td>Explicit</td>
<td>2273</td>
</tr>
<tr>
<td>Haunted</td>
<td>8634</td>
</tr>
</tbody>
</table>

**Tea and donna (10 programs).** No difference between Explicit and Haunted ≈ 25

**Take away, Haunted RelSE vs Explicit RelSE.**
- At worse: no overhead compared to Explicit ≈
- At best: faster, more coverage, less timeouts
## Haunted vs. Explicit for Spectre-STL

<table>
<thead>
<tr>
<th></th>
<th>Paths</th>
<th>X86 Ins.</th>
<th>Time</th>
<th>Timeouts</th>
<th>Bugs</th>
<th>Secure</th>
<th>Insecure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit</td>
<td>93M</td>
<td>2k</td>
<td>30h</td>
<td>15</td>
<td>22</td>
<td>3/4</td>
<td>13/23</td>
</tr>
<tr>
<td>Haunted</td>
<td>42</td>
<td>17k</td>
<td>24h</td>
<td>8</td>
<td>148</td>
<td>4/4</td>
<td>23/23</td>
</tr>
</tbody>
</table>

- Avoids paths explosion
- More unique instruction explored
- Faster
- Less timeouts
- More bugs found
- More programs proven secure / insecure

**Take away, Haunted RelSE vs Explicit RelSE.**

*Always wins!*
Weakness of index-masking countermeasure
Index masking. Add branchless bound checks

Program vulnerable to Spectre-PHT

```c
if (idx < size) { // size = 256
    v = tab[idx]
    leak(v)
}
```
Index masking. Add branchless bound checks

```
if (idx < size) { // size = 256
    idx = idx & (0xff)
    v = tab[idx]
    leak(v)
}
```
Weakness of Spectre-PHT countermeasure

**Index masking.** Add branchless bound checks

```c
if (idx < size) {  // size = 256
    idx = idx & (0xff)
    v = tab[idx]
    leak(v)
}
```

Compiled version with gcc –O0 –m32

```c
store @idx (load @idx & 0xff)
eax = load @idx
al = [@tab + eax]
leak(al)
```

- Masked index stored in memory
- Store may be bypassed with Spectre-STL!
Weakness of Spectre-PHT countermeasure

Index masking. Add branchless bound checks

```c
if (idx < size) { // size = 256
    idx = idx & (0xff)
    v = tab[idx]
    leak(v)
}
```

**Verified mitigations:**

- Enable optimizations (depends on compiler choices)
- Explicitly put masked index in a register

Compiled version with gcc –O0 –m32

```assembly
register uint32_t ridx asm ("eax");
store @idx (load @idx & 0xff)
eax = load @idx
al = [@tab + eax]
leak (al)
```

• Masked index stored in memory
• Store may be bypassed with Spectre-STL!
Conclusion
Conclusion

• **Haunted RelSE optimization**
  • Model transient and regular behaviors at the same time
  • Significantly improves SoA methods

• **Binsec/Haunted**, binary-level verification tool
  • Spectre-PHT: efficient on real world crypto 😞 → 😄
  • Spectre-STL: efficient on small programs 😞 → 😄

• New Spectre-STL violations with index masking and PIC

https://github.com/binsec/haunted
https://github.com/binsec/haunted_bench