Experimental evaluation of a binary-level symbolic analyzer for Spectre: Binsec/Haunted

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

Sébastien Bardin
CEA LIST / Université Paris-Saclay
France

Tamara Rezk
INRIA Sophia Antipolis
France
Spectre attacks (2018)

- Exploit speculative execution in processors
- Affect almost all processors
- Misspeculations lead to incorrect or transient executions
- Transient executions are reverted at architectural level
- But not the microarchitectural state (e.g. cache)

**Problem.** Transient executions can leak secret data
A new verification tool for Spectre

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

**Challenge.** Model new transient behaviors avoiding path explosion

**Contributions.**
- Optimization Haunted RelSE: transient and regular behaviors *at the same time*
- Binsec/Haunted, binary-level verification tool for Spectre-PHT & STL
- New Spectre-STL violations [paper]

**In this talk.**
- Methodology for evaluating Haunted RelSE against Explicit RelSE
- Binsec/Haunted experimental evaluation
- Comparison with other tools KLEESpectre and Pitchfork
- Challenges: Spectre detection, binary analysis, symbolic execution, etc.
Background Spectre-PHT & Spectre-STL

Experimental Evaluation

- Methodology & results: research questions, benchmark, results
- How did we get there? Implementation of Binsec/Haunted & Experimental setup
- Challenges: binary analysis, specifying secrets, validation, usability

Discussion

- Comparison against other tools
- Intermediate/unsuccessful results
- Failures with experimental evaluation & reproduction
- Availability of Binsec/Haunted

Wrap-up
Background Spectre-PHT & Spectre-STL

Experimental Evaluation
- Methodology & results: research questions, benchmark, results
- How did we get there? Implementation of Binsec/Haunted & Experimental setup
- Challenges: binary analysis, specifying secrets, validation, usability

Discussion
- Comparison against other tools
- Intermediate/unsuccessful results
- Failures with experimental evaluation & reproduction
- Availability of Binsec/Haunted

Wrap-up
Spectre-PHT & Spectre-STL

**Spectre-PHT.** Exploits *conditional branch* predictor

1. **if** idx < size {  
2. \( v = \text{tab}[\text{idx}] \)  
3. \( \text{leak}(v) \)}

1. **Conditional** is *misspeculated* \((\text{idx} > \text{size})\)
2. **Out-of-bound** array access  
   \(\rightarrow\) load *secret* data in \(v\)
3. \(v\) is leaked to the attacker 🐜
**Spectre-PHT.** Exploits *conditional branch* predictor

1. If `idx < size` {
2. \[ v = \text{tab}[\text{idx}] \]
3. \[ \text{leak}(v) \]}

1. Conditional is *misspeculated* (`idx > size`)
2. **Out-of-bound** array access
   \[ \rightarrow \text{load secret data in } v \]
3. \[ v \] is leaked to the attacker

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

```
store a secret
store a public
v = load a
leak(v)
```

\[ \text{leak(public)} \]
**Spectre-PHT.** Exploits **conditional branch** predictor

1. If \( \text{idx} < \text{size} \) {
2. \( v = \text{tab}[\text{idx}] \)
3. \( \text{leak}(v) \)

1. **Conditional is misspeculated** (\( \text{idx} > \text{size} \))
2. **Out-of-bound** array access
   \( \rightarrow \) load secret data in \( v \)
3. \( v \) is leaked to the attacker

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

- **store a secret**
- **store a public**
- \( v = \text{load a} \)
- **leak\((v)\)**

- **store a secret**
- **store a public**
- \( \text{leak}(\text{public}) \)
- **leak\((\text{secret})\)**
**Spectre-PHT.** Exploits conditional branch predictor

1. If \( \text{idx} < \text{size} \) {
2. \( \text{v} = \text{tab[\text{idx}]} \)
3. \( \text{leak(v)} \)

1. Conditional is misspeculated (\( \text{idx} > \text{size} \))
2. Out-of-bound array access → load secret data in \( \text{v} \)
3. \( \text{v} \) is leaked to the attacker

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

- Store a secret
- Store a public
- \( \text{v} = \text{load a} \)
- \( \text{leak(v)} \)
- \( \text{leak(public)} \)

- Store a secret
- Store a public
- \( \text{v} = \text{load a} \)
- \( \text{store a public} \)
- \( \text{leak(v)} \)
- \( \text{leak(secret)} \)

- Store a secret
- Store a public
- \( \text{v} = \text{load a} \)
- \( \text{store a public} \)
- \( \text{leak(v)} \)
- \( \text{leak(init_mem[a])} \)
Definitions

• **Transient executions**: incorrect execution (misspeculated)

• **RelSE**: Relational Symbolic Execution (SE for information-flow)

• **Explicit RelSE**: *baseline* technique to model speculative execution

• **Haunted RelSE**: our optimization, models transient and regular behaviors *at the same time*

• **Binsec/Haunted**: binary-analysis tool that implements Haunted RelSE
Background Spectre-PHT & Spectre-STL

Experimental Evaluation

- **Methodology & results**: research questions, benchmark, results
- **How did we get there?** Implementation of Binsec/Haunted & Experimental setup
- **Challenges**: binary analysis, specifying secrets, validation, usability

Discussion

- Comparison against other tools
- Intermediate/unsuccessful results
- Failures with experimental evaluation & reproduction
- Availability of Binsec/Haunted

Wrap-up
Experimental methodology & results
Clear Research Questions

**RQ1. Effectiveness**
Is Binsec/Haunted able to scale on real-world cryptographic code?
*Perfs on donna, OpenSSL, Libsodium*

**RQ2. Haunted vs. Explicit**
How does Haunted ReISE compare vs. Explicit ReISE?
*Implemented baseline Explicit in Binsec/Haunted*

**RQ3. Binsec/Haunted vs. SoA tools**
Comparison against Pitchfork and KLEESpectre
*(Details in Discussion)*

- #X86 instructions
- #Paths
- Time
- Bug
- Timeout
- Secure/Insecure

Links:
https://github.com/binsec/haunted
Benchmark

• Small test cases.
  • Paul Kocher’s litmus tests for Spectre-PHT*
  • + a version that we patched with index-masking
  • A set of litmus tests for Spectre-STL (that we designed)

• Cryptographic primitives, compiled with -O0, -O1, -O2, -O3, -Ofast.
  • Tea & donna *

• More complex cryptographic primitives with stack protectors.
  • Libsodium secretbox *
  • OpenSSL ssl3-digest-record *
  • OpenSSL mee-cbc-decrypt *

* From Pitchfork

https://github.com/binsec/haunted_bench
## Haunted vs. Explicit for Spectre-PHT (RQ1-RQ2)

**Litmus tests** (32 programs)

<table>
<thead>
<tr>
<th></th>
<th>Paths</th>
<th>Time</th>
<th>Timeout</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit</td>
<td>1546</td>
<td>≈3h</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td>Haunted</td>
<td>370</td>
<td>15s</td>
<td>0</td>
<td>22</td>
</tr>
</tbody>
</table>

**Libsodium & OpenSSL** (3 programs)

<table>
<thead>
<tr>
<th></th>
<th>X86 Instr.</th>
<th>Time</th>
<th>Timeout</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit</td>
<td>2273</td>
<td>18h</td>
<td>3</td>
<td>43</td>
</tr>
<tr>
<td>Haunted</td>
<td>8634</td>
<td>≈8h</td>
<td>1</td>
<td>47</td>
</tr>
</tbody>
</table>

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

Take away, Haunted RelSE vs Explicit RelSE.
- At worse: no overhead compared to Explicit ≈
- At best: faster, more coverage, less timeouts ↑

Take away from methodology: sometimes difficult (not desirable) to aggregate results
Haunted vs. Explicit for Spectre-STL (RQ1-RQ2)

<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!*
Comparison Binsec/Haunted against Pitchfork & KLEESpectre (RQ3)

<table>
<thead>
<tr>
<th>Target</th>
<th>Programs</th>
<th>PHT</th>
<th>STL</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLEESpectre</td>
<td>LLVM</td>
<td>Explicit</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>😞 (≈240× slower)</td>
<td>😊 (≈equivalent)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tea &amp; donna</td>
<td></td>
</tr>
<tr>
<td>Pitchfork</td>
<td>Binary</td>
<td>Optims</td>
<td>Explicit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>😊 (≈equivalent)</td>
<td>😞 6/10 TO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>😞 (50× slower &amp; TO)</td>
<td>😞 TO</td>
</tr>
<tr>
<td>Binsec/Haunted</td>
<td>Binary</td>
<td>Haunted</td>
<td>Haunted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>😊</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tea &amp; donna</td>
<td>😞</td>
</tr>
</tbody>
</table>
How did we get there?
Implementation of Binsec/Haunted

• Built on top of Binsec/Rel (RelSE for constant-time)
• Written in Ocaml (5+2 kLoCs)

+ Explicit RelSE
+ Haunted RelSE

Info on binary
• entrypoint
• initial memory

Specification
• secret input

Microarchitectural state
• max spec. depth (200)
• store buffer (20)

https://github.com/binsec/haunted
Experimental Setup

Run expes with python script
For \( \text{prog} \in \{ \text{tea, donna, litmus-pht, … } \} \)
Just run `cd prog; python3 expe.py`

Params set according to file timeout, location of secrets, entrypoint, memory

Python script
- NoSpec
- Explicit-PHT
- Haunted-PHT
- Explicit-STL
- Haunted-STL

Often changing!

https://github.com/binsec/haunted_bench

Laptop Intel(R) Xeon(R) CPU E3-1505M v6 @ 3.00GHz and 32GB of RAM
Experimental Setup

Interpret results with python script
Just run `python3 stats.py` to get
tables from paper

```latex
csv with 84 columns
- Value of parameters
- Number of paths
- Size of formulas
- Status, ...
```

```python
Stats .csv
```

```latex
Latex table
- X86 instructions
- Paths
- Time
- Bug
- Timeout
- Secure
- Insecure
```

https://github.com/binsec/haunted_bench
Take away on methodology

- Clear research questions
  - Clear objectives
  - Associated metrics & protocol
  - Clear conclusions

- We compare with other tools + in a controlled setup
  (re-implementing the baseline for Explicit ReISE)

- Better too much stats than not enough!
  - Rerun all expes to get static instructions count for coverage
Challenges
Standard challenges of binary analysis

• **Entrypoint**: start from main or other function symbol
  - stripped binaries are more challenging
• Only for **statically** compiled binaries (or you have to provide stubs)
• Configuration of **initial memory**
  - Sections to load from file: .data, .rodata, .got, .got.plt
  - .bss for both uninitialized variables (symbolic) & variables set to 0 (concrete)
• Choose an implementation for **memset_ifunc** (indirect functions)
  - __memset_ia32, __memset_sse2?

*All these steps might require reverse engineering*
Specifying secrets: a challenge at binary-level

Reverse Engineering
- Open IDA & find offset of secrets from initial esp
- Manual 😞
- Close to reality 😊

Use C stubs
- Use stubs to specify secrets
- Automatic 😊
- Not so much realistic 😞
- Adds stores: 😞 Spectre-STL

Use global variables
- Put secret in global variables
- Automatic 😊
- Not so much realistic 😞

Global variables have symbols:

```
int main() {
    unsigned long key[4];
    unsigned long data[2];
    unsigned long out[2];
    high_input_16(key);
    high_input_8(data);
    high_input_8(out);
    decipher(data, out, key);
}
```

Just give high symbols to binsec:
```
binsec relse -relse-high-sym key,data,out
```
## Validation of Binsec/Haunted

### Problem.

- Spectre attacks are difficult to find manually
- No ground truth (esp. for Spectre-STL)

### Spectre-PHT

Paul Kocher’s Litmus tests for Spectre-PHT [1]
- Set of 16 insecure simple test cases 😊
- Still not easy to precisely identify vulnerabilities 😞
  - Number of vulnerabilities, locations, etc.
  - We added patched versions with index-masking

+ validation against Pitchfork and KLEESpectre on these litmus test (when possible)
  & manually check in case of deviation
  + used for regression testing

### Spectre-STL

No ground truth except for Spectre-STL PoC [2]
- Even more difficult to identify vulnerabilities
  - We crafted 14 STL-litmus tests [3]
    - Still needs more doc (coming soon!) to be usable

Interpreting results: case Spectre-PHT

```c
void case_1(uint64_t idx) {
    if (idx < publicarray_size) {
        temp &= publicarray2[publicarray[idx] * 512];
    }
}
```

- Insecure memory access 0x000011d3
- Counterexample:
  0xffffcc1d: 0x00020024
  secretarray[4] = is_secret [...]

Interpreting results: case Spectre-PHT

```c
void case_1(uint64_t idx) {
    if (idx < publicarray_size) {
        temp &= publicarray2[publicarray[idx] * 512];
    }
}
```

- Insecure memory access 0x000011d3
- Counterexample:
  
  ```c
  mov cl, (publicarray2 - 4000h)[eax+edx]  
  initial esp + RE → idx = 0x20024  
  secretarray[4] = is_secret [...]
  publicarray[0x20024] = secretarray[4]
  ```
Interpreting results: case Spectre-PHT

```c
void case_1(uint64_t idx) {
    if (idx < publicarray_size) {
        temp &= publicarray2[publicarray[idx] * 512];
    }
}
```

- Insecure memory access 0x000011d3
- Counterexample: 0xfffffccc1d: 0x00020024
  - initial esp + RE → idx = 0x20024
  - secretarray[4] = is_secret
  - publicarray[0x20024] = secretarray[4]

Interpreting results requires manual effort
Interpreting results: case Spectre-STL

- Location of violation
- Initial memory configuration
- List of loads that bypass a store

Encode in smt-formula.
- Address of out-of-order loads
- Address of forwarding store

Solver will return its choice in counterexample.
load_08049d27_from_main-mem: True
load_08049d1c_from_08049cf5: True
Summary of challenges

• **Standard to binary analysis**
  – Difficult to use, might require reverse engineering
    ✓ We can automate many things if we have symbols

• **Specifying secrets**
  • Tradeoff between realism & usability

• **Spectre attacks**
  – Validation is not easy, still a manual process
    ✓ Existing litmus tests for Spectre-PHT + new litmus for Spectre-STL
    ✓ Cross-validated against Pitchfork and KLEESpectre
  – Difficult to understand vulnerabilities
    ✓ Encoding in smt-formula for Spectre-STL

*Usability* crucial for running more experiments & validation & sharing
Background Spectre-PHT & Spectre-STL

Experimental Evaluation

- Methodology & results: research questions, benchmark, results
- How did we get there? Implementation of Binsec/Haunted & Experimental setup
- Challenges: binary analysis, specifying secrets, validation, usability

Discussion

- Comparison against other tools
- Intermediate/unsuccessful results
- Failures with experimental evaluation & reproduction
- Availability of Binsec/Haunted

Wrap-up
Comparison against other tools: not so easy

Use cases from Pitchfork
Recompiled for 32-bit architecture
No execution time reported in paper
Rerun Pitchfork for comparison

<table>
<thead>
<tr>
<th>KLEE Spectre (KLEE, SE)</th>
<th>Pitchfork (Angr, SE + tainting secrets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• LLVM tool</td>
<td>• Adapted to match Binsec/Haunted:</td>
</tr>
<tr>
<td>• Spectre-PHT only</td>
<td>Pitchfork-cont</td>
</tr>
<tr>
<td>• Not exactly the same property (loads only)</td>
<td>• Have to deal with TO &amp; OOM</td>
</tr>
<tr>
<td>• False positive (one nested spec. cond?)</td>
<td>• Spurious vulnerabilities (in .data section)?</td>
</tr>
</tbody>
</table>

Results to take with pinch of salt, not always related to what we want to measure
→ Need to compare Explicit vs Haunted in Binsec/Haunted
Tools easy adapt & run on my test cases 😊!
Intermediate results

• **Which solver to use:** boollector, z3, yices, cvc4?
  boollector is better but sometime it is stuck while z3 solves the query (overflow on memory indexes)

• **Path constraint** as a big conjunction at the end of the formula or just assert constraints when they come ?
  → Does not matter

• Simpler is not always better !
  \[ pc \land c_l = T \land c_r = T \text{ when } c_l = c_r \]  \Rightarrow  \[ pc \land c_l = T \]  Simpler but slower to solve
Intermediate results

• **Which solver to use:** boolector, z3, yices, cvc4?
  boolector is better but sometime it is stuck while z3 solves the query (overflow on memory indexes)

• **Path constraint** as a big conjunction at the end of the formula or just assert constraints when they come?
  → Does not matter

• Simpler is not always better!
  \[ pc \land c_l = T \land c_r = T \text{ when } c_l = c_r \implies pc \land c_l = T \]

  Simpler but slower to solve

*Things I tested quickly, results not really recorded 😊*

Lesson learned: It is a good practice to document the intermediate results
Things I tried that did not succeed

Trying to help the solver.
- Reduce size of query by removing redundant insecurity formulas
  → up to 50% size reduction, usually around 30% but no impact on time

Propagate info in symbolic store to simplify expressions.

Symbolic store: $v \mapsto \{a, b, c, d\}$

Formula: $\varphi \quad \xrightarrow{\text{Solver}}

Retire value a (v1)
$v \mapsto \{a, b, c, d\}$
$v \neq a \land \varphi \quad \xrightarrow{\text{Solver}}$

Retire value a (v2)
$v \mapsto \{a, b, c, d\}$
$v \neq a \land \varphi \quad \xrightarrow{\text{Solver}}$

Things I tested quickly, results not really recorded 😞

Lesson learned:
- SMT-Solver can be hard to satisfy
- Investigate bottlenecks & focus on them
Other things I tried but couldn’t put in the paper

- **Explore different strategies for computing speculation depth ['1]**
  - **Static**: Speculate for 200 instructions
  - **Hybrid**: Speculate only when conditional depends on memory
  - **Dynamic**: Retire conditional instructions when older memory access is retired

- **Linux kernel (inspired from [2])**
  - Get compare & execute gadgets
  - Had to search & identify myself
  - Not easy 😞 (macros + inl. asm)
  → Analysis of syscall handler

Table 7: Spectre-PHT gadget classification and the number of occurrences per gadget type in Linux kernel v5.0.

<table>
<thead>
<tr>
<th>Gadget</th>
<th>Example (Spectre-PHT)</th>
<th>#Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefetch</td>
<td>if(i&lt;LEN_A){a[i];}</td>
<td>172</td>
</tr>
<tr>
<td>Compare</td>
<td>if(i&lt;LEN_A){if(a[i]==k);}</td>
<td>127</td>
</tr>
<tr>
<td>Index</td>
<td>if(i&lt;LEN_A){y = b[a[i]*x];}</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>if(i&lt;LEN_A){a<a href="void">i</a>;}</td>
<td>16</td>
</tr>
</tbody>
</table>

Fails with experiments

When trying to run my expes.

• oomkiller stories (50% swap is too late)
• Beware other programs running
• Don't forget caffeine (disables auto-suspend)
• Don't forget to plug your laptop (-50% perfs on battery)

When trying to reproduce.

• Why are my experimental results $4 \times$ slower than usual ?
  → Because CPU freq is blocked at 800MHz instead of 4GHz
• Why can’t I reproduce last month results ?
  → Because new boolector version 3.2.0 → 3.2.1 = ↑ memory consumption + oom

Lots of possible causes, often time-consuming to debug
Record commit hash can help
Availability of Binsec/Haunted

Sources & Bench on Github:

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

Docker image on zenodo:

- Binsec/Haunted
- Expes: csv results + binaries + scripts
- Ocaml 4.05
- Boolector 3.2.0

- Compiler I used for expes
- Exact version of python packages
- Exact version of all opam dependencies
- KLEESpectre & Pitchfork setup

DOI 10.5281/zenodo.4442337
Takeaways

• Difficult to compare to other tools
  ➢ Implementing our own baseline gives control on what is measured

• Solvers are sometimes difficult to satisfy

• Document unsuccessful/intermediate experimental results
  ➢ Otherwise they are forgotten 😞

• Sometime it is difficult to reproduce old results
  ➢ Log commit hash during expes & beware changing versions of dependencies!

• Community is great 😊
  ➢ Nice use cases + easy to use tools
Background Spectre-PHT & Spectre-STL

Experimental Evaluation

- **Methodology & results**: research questions, benchmark, results
- **How did we get there?** Implementation of Binsec/Haunted & Experimental setup
- **Challenges**: binary analysis, specifying secrets, validation, usability

Discussion

- Comparison against other tools
- Intermediate/unsuccessful results
- Failures with experimental evaluation & reproduction
- Availability of Binsec/Haunted

Wrap-up
Next steps

• Improving **usability** is still work in progress

• Better **documentation** for Spectre-STL litmus tests

• Try to build a more **reproducible setup**
  • Pinning versions of dependencies
  • ...?

• Thinking of **systematic ways to avoid failed** experiments?
Clear Research Questions

**RQ1. Effectiveness**
Is Binsec/Haunted able to scale on real-world cryptographic code?
*Perfs on donna, OpenSSL, Libsodium*

**RQ2. Haunted vs. Explicit**
How does Haunted RelSE compare vs. Explicit RelSE?
*Implemented baseline Explicit in Binsec/Haunted*

**RQ3. Binsec/Haunted vs. SoA tools**
Comparison against Pitchfork and KLEESpectre

---

**Metrics**
- #X86 instructions
- #Paths
- Time
- Bug
- Timeout
- Secure/Insecure

---

[GitHub link](https://github.com/binsec/haunted)
Clear Research Questions

**RQ1. Effectiveness**
Is Binsec/Haunted able to scale on real-world cryptographic code?
*Perfs on donna, OpenSSL, Libsodium*

**RQ2. Haunted vs. Explicit**
How does Haunted ReISE compare vs. Explicit ReISE?
*Implemented baseline Explicit in Binsec/Haunted*

**RQ3. Binsec/Haunted vs. SoA tools**
Comparison against Pitchfork and KLEESpectre

---

**Metrics**
- #X86 instructions
- #Paths
- Time
- Bug
- Timeout
- Secure/Insecure

---

**Reverse Engineering**
- Open IDA & find offset of secrets from initial esp
- Manual 😞
- Close to reality 😔

```plaintext
data = dword ptr -28h
out = dword ptr -28h
key = dword ptr -18h
lea eax, [ebp-key]
sub esp, 24h
push eax
lea eax, [ebp+out]
push eax
lea eax, [ebp+data]
push eax
call encipher
```

---

**Use C stubs**
- Use stubs to specify secrets
- Automatic 😞
- Not so much realistic 😐
- Adds stores: Spectre-STL

```plaintext
int main() {
    unsigned long key[4];
    unsigned long data[2];
    unsigned long out[2];
    high_input_16(key);
    high_input_8(data);
    high_input_8(out);
    decipher(data, out, key);
}
```

---

**Use global variables**
- Put secret in global variables
- Automatic 😞
- Not so much realistic 😐

Global variables have symbols:

```
BBH56B4 0 OBJECT GLOBAL DEFAULT 24 out
BBH56B8 0 OBJECT GLOBAL DEFAULT 24 data
BBH56C4 16 OBJECT GLOBAL DEFAULT 24 key
```

Just give high symbols to binsec

```
binsec-reise -W high -s key data out
```
Clear Research Questions

**RQ1. Effectiveness**
Is Binsec/Haunted able to scale on real-world cryptographic code?
Perfs on donna, OpenSSL, Libsodium

**RQ2. Haunted vs. Explicit**
How does Haunted ReISE compare vs. Explicit ReISE?
Implemented baseline Explicit in Binsec/Haunted

**RQ3. Binsec/Haunted vs. SoA tools**
Comparison against Pitchfork and KLEE Spectre

Comparison against other tools: not so easy

Use cases from Pitchfork
Recompiled for 32-bit architecture
No execution time reported in paper
Rerun Pitchfork for comparison

<table>
<thead>
<tr>
<th>KLEE Spectre (KLEE, SE)</th>
<th>Pitchfork (Angr, SE + tainting secrets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• LVMM tool</td>
<td>• Adapted to match Binsec/Haunted:</td>
</tr>
<tr>
<td>• Spectre-PHT only</td>
<td>Pitchfork-cont</td>
</tr>
<tr>
<td>• Not exactly the same</td>
<td>• Have to deal with TO &amp; OOM</td>
</tr>
<tr>
<td>property (loads only)</td>
<td>• Spurious vulnerabilities in .data</td>
</tr>
<tr>
<td>• False positive (one</td>
<td>section)?</td>
</tr>
<tr>
<td>nested spec. cond?)</td>
<td></td>
</tr>
</tbody>
</table>

Results to take with pinch of salt, not always related to what we want to measure
→ Need to compare Explicit vs Haunted in Binsec/Haunted
Tools easy adapt & run on my test cases 😊!
Clear Research Questions

**RQ1. Effectiveness**
Is Binsec/Haunted able to scale on real-world cryptographic code?
Perfs on donna, OpenSSL, Libsodium

**RQ2. Haunted vs. Explicit**
How does Haunted ReISE compare vs. Explicit ReISE?
Implemented baseline Explicit in Binsec/Haunted

**RQ3. Binsec/Haunted vs. SoA tools**
Comparison against Pitchfork and KLEESpectre

---

**Metrics**
- #x86 instructions
- #Paths
- Time
- Bug
- Timeout
- Secure/Insecure

---

**Reverse Engineering**
- Open IDA & find offset of secrets from initial esp
- Manual 🙌
- Close to reality 😐

**Use C stubs**
- Use stubs to specify secrets
- Automatic 😊
- Not so much realistic 😐
- Adds stores: 😊 Spectre-STL

**Use global variables**
- Put secret in global variables
- Automatic 😊
- Not so much realistic 😐

**Global variables have symbols:**
```
Value  Size Type          File            Name
0x0058c4 8  OBJECT GLOBAL DEFAULT 24 out
0x0058c8 8  OBJECT GLOBAL DEFAULT 24 out
0x0058c4 16 OBJECT GLOBAL DEFAULT 24 data
```
Just give high symbols to binsec

Availability of Binsec/Haunted

- **KLEESpectre (KLEE, SE)**
  - Could not compare programs with syscalls (restrict to litmus, tea & donna)
  - Outputs only vulnerabilities found & exec time

- **Pitchfork (Angr, SE + tainting secrets)**
  - Adapted to match Binsec/Haunted:
    - Pitchfork-cont
  - Have to deal with TO & OOM
  - Spurious vulnerabilities (in .data section)?

- **Do**
  - LVMM tool
  - Spectre-PHT only
  - Not exactly the same property (loads only)
  - Not false positive (one nested spec. cond?)

- **Don't**
  - Recompiled for 32-bit architecture
  - No execution time reported in paper
  - Rerun Pitchfork for comparison

---

Comparison against other tools: not so easy

```
Results to take with pinch of salt, not always related to what we want to measure
- Need to compare Explicit vs Haunted in Binsec/Haunted
  Tools easy adapt & run on my test cases 😊!
```

Sources & Bench on Github:
- Binsec/Haunted: [https://github.com/binsec/haunted](https://github.com/binsec/haunted)
- Bench: [https://github.com/binsec/haunted_bench](https://github.com/binsec/haunted_bench)

Docker image on zenodo:

- 10.5281/zenodo.4442337
  - Compiler I used for expes
  - Exact version of python packages
  - Exact version of all opam dependencies
  - KLEESpectre & Pitchfork setup

---

Binsec/Haunted

---

---