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

The LASER Workshop

Learning from Authoritative Security Experiment Results
www.laser-workshop.org

NDSS
SYMPOSIUM/2021

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

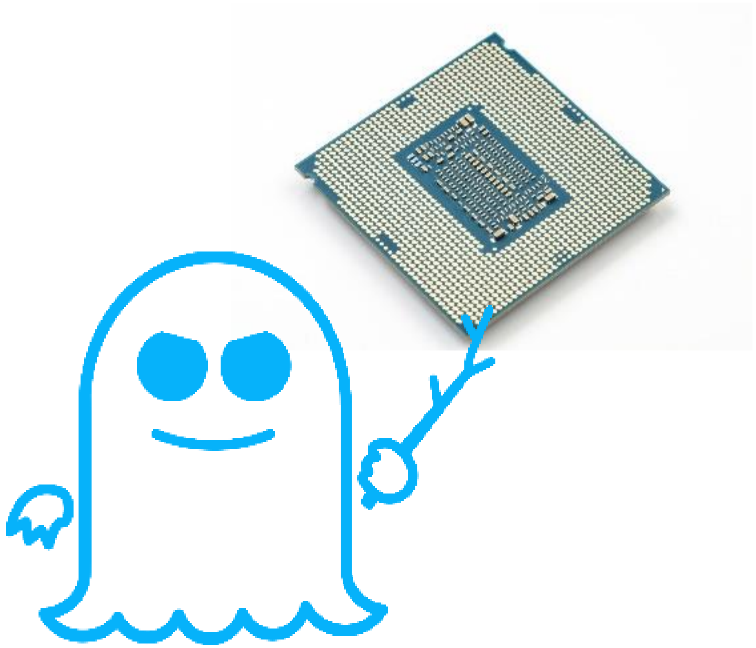
Sébastien Bardin
CEA LIST / Université Paris-Saclay
France

Tamara Rezk
INRIA Sophia Antipolis
France

Context: Detection of Spectre attacks

Spectre attacks (2018)

- Exploit speculative execution in processors
- Affect almost all processors
- Mispeculations 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

Binsec/Haunted

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

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

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

Spectre-PHT & Spectre-STL

Spectre-PHT. Exploits conditional branch predictor

```
1: if idx < size {
2:    v = tab[idx]
3:    leak(v)}
```

1. Conditional is misspeculated (`idx > size`)
2. Out-of-bound array access
   → 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)
```

```
leak(public)
```

Spectre-PHT & Spectre-STL

Spectre-PHT. Exploits conditional branch predictor

```
1: if idx < size {
2:     v = tab[idx]
3:     leak(v) }
```

1. Conditional is misspeculated (`idx > size`)
2. Out-of-bound array access
   → 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)
```

leak(public)

+

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

leak(secret)

Spectre-PHT & Spectre-STL

Spectre-PHT. Exploits conditional branch predictor

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

1. Conditional is misspeculated (`idx > size`)
2. Out-of-bound array access
   → 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)
```

leak(public)

+

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

leak(secret)

+

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

leak(init_mem[a])

Definitions

- **Transient executions:** incorrect execution (mispeculated)
- **RelSE:** Relational Symbolic Execution (SE for information-flow)
- **Expicit 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 RelSE compare vs. Explicit RelSE?
Implemented baseline Explicit in Binsec/Haunted

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

Metrics
• #X86 instructions • #Paths • Time • Bug • Timeout • Secure/Insecure

Binsec/Haunted

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) ↗

	Paths	Time	Timeout	Bugs
Explicit	**1546**	**≈3h**	2	21
Haunted	**370**	**15s**	0	22

Libsodium & OpenSSL (3 programs) ↗

	X86 Instr.	Time	Timeout	Bugs
Explicit	**2273**	18h	**3**	43
Haunted	**8634**	≈8h	**1**	47

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)

	Paths	X86 Ins.	Time	Timeouts	Bugs	Secure	Insecure
Explicit	93M	2k	30h	15	22	3/4	13/23
Haunted	42	17k	24h	8	148	4/4	23/23

- 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)

	Target	Programs	PHT	STL
KLEESpectre	LLVM	Litmus tests Tea & donna	**Explicit** 😐 (≈240× slower) ☺ (≈equivalent)	NA
Pitchfork	Binary	Litmus tests Tea & donna	**Optims** ☺ (≈equivalent) ☹ (50× slower & TO)	**Explicit** ☹ 6/10 TO ☹ TO
Binsec/Haunted	Binary	Litmus tests Tea & donna	**Haunted** ☺ ☺	**Haunted** ☺ 😐

Challenges in discussion

How did we get there?

Implementation of Binsec/Haunted

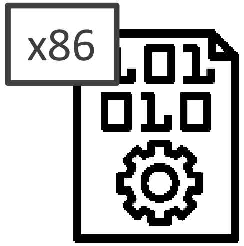
- Built on top of Binsec/Rel (RelSE for constant-time)
- Written in Ocaml (5+2 kLoCs)

+ Explicit RelSE
+ Haunted RelSE

Info on binary
- entrypoint
- initial memory

Specification
- secret input

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



Binsec/Haunted

https://github.com/binsec/haunted

Experimental Setup

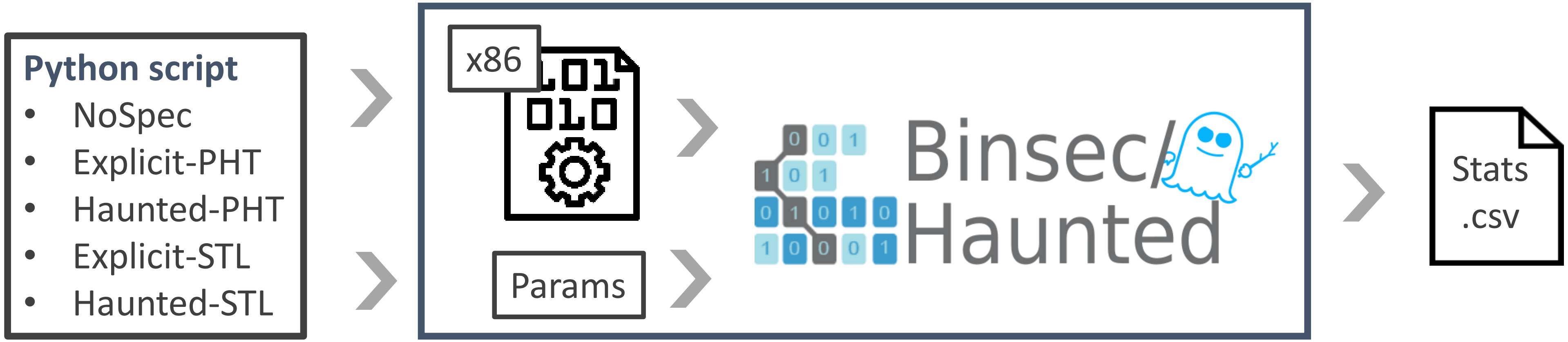
Run expes with python script

For `prog` ∈ { tea, donna, litmus-pht, … }

Just run `cd prog; pyton3 expe.py`

Params set according to file

timeout, location of secrets, entrypoint, memory



Often changing !

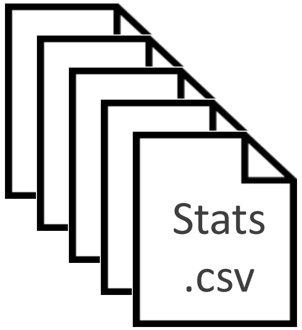
Laptop Intel(R) Xeon(R) CPU E3-1505M v6 @ 3.00GHz and 32GB of RAM

https://github.com/binsec/haunted_bench

Experimental Setup

Interpret results with python script

Just run `pyton3 stats.py` to get tables from paper



Python script
- pandas

Often changing !
(Previously R)

Programs	PHT	I_{x86}	P	T (s)	🐞	⧗	✓	✗
`litmus-pht`	NoSpec	733	48	3	-	0	16/16	-
	Explicit	761	**703**	**10331**	**21**	**2**	-	16/16
	Haunted	761	**188**	**7**	**22**	**0**	-	16/16
`litmus-pht masked`	NoSpec	911	48	5	-	0	16/16	-
	Explicit	950	**843**	**169**	-	0	16/16	-
	Haunted	950	**182**	**8**	-	0	16/16	-
`tea`	NoSpec	326	5	.56	-	0	5/5	-
	Explicit	326	172	.62	-	0	5/5	-
	Haunted	326	172	.62	-	0	5/5	-
`donna`	NoSpec	22k	5	2948	-	0	5/5	-
	Explicit	21k	1.0M	6153	-	1	4/5	-
	Haunted	21k	1.0M	6162	-	1	4/5	-
`secretbox`	NoSpec	2721	1	5	-	0	1/1	-
	Explicit	**769**	15k	**21600**	**13**	**1**	-	1/1
	Haunted	**3583**	2.2M	**2421**	**17**	**0**	-	1/1
`ssl3-digest`	NoSpec	1809	1	4	-	0	1/1	-
	Explicit	**808**	9k	**21600**	13	**1**	-	1/1
	Haunted	**2502**	428k	**4694**	13	**0**	-	1/1
`mee-cbc`	NoSpec	6383	1	448	-	0	1/1	-
	Explicit	**696**	74k	21600	17	1	-	1/1
	Haunted	**2549**	22M	21600	17	1	-	1/1
Total	NoPHT	35k	109	3415	0	0	45/45	-
	Explicit	**25k**	1.1M	**81453**	**64**	**6**	25/25	19/19
	Haunted	**32k**	25.7M	**34892**	**69**	**2**	25/25	19/19

Latex table

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

csv with 84 columns

- Value of parameters
- Number of paths
- Size of formulas
- Status, …

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 RelSE)

- 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 unititialized 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 ☺

```
data            = dword ptr -28h
out             = dword ptr -20h
key             = dword ptr -18h

lea     eax, [ebp+key]
sub     esp, 24h
push    eax             ; k
lea     eax, [ebp+out]
push    eax             ; w
lea     eax, [ebp+data]
push    eax             ; v
call    encipher
```

Use C stubs

- Use stubs to specify secrets
- Automatic ☺
- Not so much realistic ☹
- Adds stores: ☹ Spectre-STL

```
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:

```
   Value  Size Type    Bind   Vis      Ndx Name
080e5c84     8 OBJECT  GLOBAL DEFAULT   24 out
080e5c8c     8 OBJECT  GLOBAL DEFAULT   24 data
080e5c94    16 OBJECT  GLOBAL DEFAULT   24 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

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

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

[1] https://github.com/cdisselkoen/pitchfork/blob/master/new-testcases/spectrev1.c
[2] https://github.com/IAIK/transientfail/tree/master/pocs/spectre/STL
[3] https://github.com/binsec/haunted_bench/blob/master/src/litmus-stl/programs/spectrev4.c

Interpreting results: case Spectre-PHT

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

Binsec/Haunted

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

Interpreting results: case Spectre-PHT

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

Binsec/Haunted

With a bit of reverse

- Insecure memory access 0x000011d3

```
mov  cl, (publicarray2 - 4000h)[eax+edx]
```

load publicarray[idx]

- Counterexample:

```
0xffffcc1d: 0x00020024
secretarray[4] = is_secret
[…]
```

initial esp + RE → idx = 0x20024

publicarray[0x20024] = secretarray[4]

Interpreting results: case Spectre-PHT

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

Binsec/Haunted

With a bit of reverse

- Insecure memory access 0x000011d3

```
mov   cl, (publicarray2 - 4000h)[eax+edx]
```

load publicarray[idx]

- Counterexample:
  0xffffcc1d: 0x00020024
  secretarray[4] = is_secret
  [...]

initial esp + RE → idx = 0x20024

publicarray[0x20024] = secretarray[4]

Interpreting results requires manual effort

Interpreting results: case Spectre-STL

Binsec/Haunted

- 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

KLEESpectre (KLEE, SE)	Pitchfork (Angr, SE + tainting secrets)
• Could not compare programs with syscalls (restrict to litmus, tea & donna) • Outputs only vulnerabilities found & exec time	
• LLVM tool • Spectre-PHT only • Not exactly the same property (loads only) • False positive (one nested spec. cond?)	• Adapted to match Binsec/Haunted: Pitchfork-cont • Have to deal with TO & OOM • Spurious vulnerabilities (in .data section)?

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: 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 !

  $pc \wedge c_l = T \wedge c_r = T$ when $c_l = c_r$ ❯ $pc \wedge c_l = T$

  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 !

  $pc \wedge c_l = T \wedge c_r = T$ when $c_l = c_r$ ❯ $pc \wedge c_l = T$ 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: $\mathrm{v} \mapsto \{a, b, c, d\}$

Formula: φ ❯  Solver

Retire value a (v1)

$\mathrm{v} \mapsto \{a, b, c, d\}$

$\mathrm{v} \neq a \wedge \varphi$ ❯ 

Retire value a (v2)

$\mathrm{v} \mapsto \{\cancel{a,} b, c, d\}$

$\mathrm{v} \neq a \wedge \varphi$ ❯ 

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

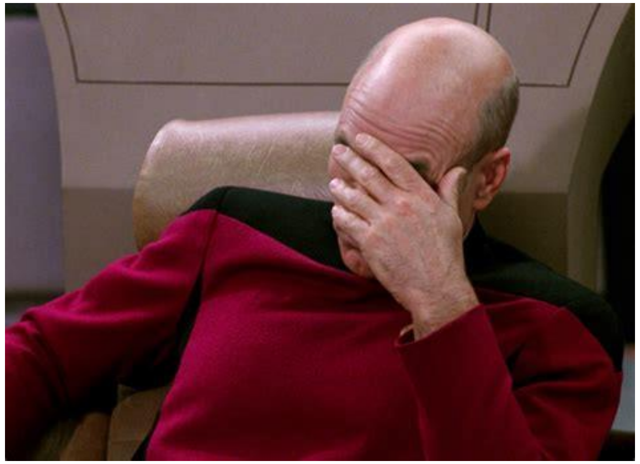
- **Linux kernel** (inspired from [2])
  - Get compare & execute gadgets
  - Had to search & identify myself
  - Not easy ☹ (macros + inl. asm)
  - → Analysis of syscall handler

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

Gadget	Example (Spectre-PHT)	#Occurrences
Prefetch	`if(i<LEN_A){a[i];}`	172
Compare	`if(i<LEN_A){if(a[i]==k){};}`	127
Index	`if(i<LEN_A){y = b[a[i]*x];}`	0
Execute	`if(i<LEN_A){a[i](void);}`	16

[1] Wu, Meng, and Chao Wang. "Abstract interpretation under speculative execution." PLDI '19.
[2] Canella, Claudio, et al. "A systematic evaluation of transient execution attacks and defenses." USENIX Security '19

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× 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

Binsec/Haunted

Sources & Bench on Github:

https://github.com/binsec/haunted

https://github.com/binsec/haunted_bench

Docker image on zenodo:

DOI 10.5281/zenodo.4442337

- 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

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

Binsec/Haunted

https://github.com/binsec/haunted

13

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

Binsec/Haunted
https://github.com/binsec/haunted

Specifying secrets: a challenge at binary-level

Reverse Engineering

- Open IDA & find offset of secrets from initial esp
- Manual ☹
- Close to reality ☺

```
data              = dword ptr -28h
out               = dword ptr -20h
key               = dword ptr -18h

lea     eax, [ebp+key]
sub     esp, 24h
push    eax                 ; k
lea     eax, [ebp+out]
push    eax                 ; w
lea     eax, [ebp+data]
push    eax                 ; v
call    encipher
```

Use C stubs

- Use stubs to specify secrets
- Automatic ☺
- Not so much realistic ☹
- Adds stores: ☹ Spectre-STL

```
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:

```
   Value  Size Type    Bind   Vis      Ndx Name
080e5c84     8 OBJECT  GLOBAL DEFAULT   24 out
080e5c8c     8 OBJECT  GLOBAL DEFAULT   24 data
080e5c94    16 OBJECT  GLOBAL DEFAULT   24 key
```

Just give high symbols to binsec

```
binsec relse -relse-high-sym 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 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

Binsec/Haunted
https://github.com/binsec/haunted

Specifying secrets: a challenge at binary-level

Reverse Engineering

- Open IDA & find offset of secrets from initial esp
- Manual ☹
- Close to reality ☺

```
data              = dword ptr -28h
out               = dword ptr -20h
key               = dword ptr -18h

lea     eax, [ebp+key]
sub     esp, 24h
push    eax                  ; k
lea     eax, [ebp+out]
push    eax                  ; w
lea     eax, [ebp+data]
push    eax                  ; v
call    encipher
```

Use C stubs

- Use stubs to specify secrets
- Automatic ☺
- Not so much realistic ☹
- Adds stores: ☹ Spectre-STL

```
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:

```
   Value  Size Type    Bind   Vis      Ndx Name
080e5c84     8 OBJECT  GLOBAL DEFAULT   24 out
080e5c8c     8 OBJECT  GLOBAL DEFAULT   24 data
080e5c94    16 OBJECT  GLOBAL DEFAULT   24 key
```

Just give high symbols to binsec

```
binsec relse -relse-high-sym key,data,out
```


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

KLEESpectre (KLEE, SE)	Pitchfork (Angr, SE + tainting secrets)
• Could not compare programs with syscalls (restrict to litmus, tea & donna) • Outputs only vulnerabilities found & exec time	
• LLVM tool • Spectre-PHT only • Not exactly the same property (loads only) • False positive (one nested spec. cond?)	• Adapted to match Binsec/Haunted: Pitchfork-cont • Have to deal with TO & OOM • Spurious vulnerabilities (in .data section)?

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 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

Binsec/Haunted
https://github.com/binsec/haunted

Specifying secrets: a challenge at binary-level

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

```
data            = dword ptr -28h
out             = dword ptr -20h
key             = dword ptr -18h

lea     eax, [ebp+key]
sub     esp, 24h
push    eax                 ; k
lea     eax, [ebp+out]
push    eax                 ; w
lea     eax, [ebp+data]
push    eax                 ; v
call    encipher
```

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

```
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:

```
   Value  Size Type    Bind   Vis      Ndx Name
080e5c84     8 OBJECT  GLOBAL DEFAULT   24 out
080e5c8c     8 OBJECT  GLOBAL DEFAULT   24 data
080e5c94    16 OBJECT  GLOBAL DEFAULT   24 key
```

Just give high symbols to binsec

```
binsec relse -relse-high-sym key,data,out
```


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

KLEESpectre (KLEE, SE)	Pitchfork (Angr, SE + tainting secrets)
• Could not compare programs with syscalls (restrict to litmus, tea & donna) • Outputs only vulnerabilities found & exec time	
• LLVM tool • Spectre-PHT only • Not exactly the same property (loads only) • False positive (one nested spec. cond?)	• Adapted to match Binsec/Haunted: Pitchfork-cont • Have to deal with TO & OOM • Spurious vulnerabilities (in .data section)?

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 ☺!

Availability of Binsec/Haunted

Binsec/Haunted

Sources & Bench on Github: https://github.com/binsec/haunted
https://github.com/binsec/haunted_bench

Docker image on zenodo: DOI 10.5281/zenodo.4442337

- 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