Spectector: Principled detection of speculative information flows

Marco Guarnieri
IMDEA Software Institute

Supported by Intel Strategic Research Alliance (ISRA)
“Information Flow Tracking across the Hardware-Software Boundary”

Joint work with
José F. Morales, Andrés Sánchez @ IMDEA Software Institute
Boris Köpf @ Microsoft Research
Jan Reineke @ Saarland University

To appear at IEEE Security & Privacy 2020
Spectre

Exploits speculative execution

Almost all modern CPUs are affected

Exploits *speculative execution*

Countermeasures
Countermeasures

Long Term: Co-design of software and hardware countermeasures
Countermeasures

**Long Term:** Co-design of software and hardware countermeasures

**Short and Mid Term:** Software countermeasures

**Compiler-level countermeasures**

- Example: insert LFENCE to selectively stop speculative execution
- Implemented in major compilers (Microsoft Visual C++, Intel ICC, Clang)
Countermeasures

**Long Term:** Co-design of software and hardware countermeasures

**Short and Mid Term:** Software countermeasures

**Compiler-level countermeasures**
- Example: insert LFENCE to selectively stop speculative execution
- Implemented in major compilers (Microsoft Visual C++, Intel ICC, Clang)
Compiler-level countermeasures
Compiler-level countermeasures

“compiler [...] produces unsafe code when the static analyzer is unable to determine whether a code pattern will be exploitable”
Compiler-level countermeasures

Spectre Mitigations in Microsoft’s C/C++ Compiler

Paul Kocher
February 13, 2018

https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html

“compiler [...] produces unsafe code when the static analyzer is unable to determine whether a code pattern will be exploitable”

“there is no guarantee that all possible instances of [Spectre] will be instrumented”
Compiler-level countermeasures

“compiler [...] produces unsafe code when the static analyzer is unable to determine whether a code pattern will be exploitable”

"there is no guarantee that all possible instances of [Spectre] will be instrumented"

Bottom line: No guarantees!
Contributions
Contributions

1. **Semantic notion of security** against **speculative execution attacks**
Contributions

1. *Semantic notion of security* against *speculative execution attacks*

2. Analysis to *detect vulnerability* or *prove security*
Outline

1. Speculative execution 101
2. Speculative non-interference
3. Detecting speculative leaks
4. Spectector + Case studies
Speculative execution + branch prediction
Speculative execution + branch prediction

if \( x < A_{\text{size}} \)
\[
y = B[A[x]]
\]
Speculative execution + branch prediction

if ($x < \text{A\_size}$)
    $y = B[A[x]]$
Speculative execution + branch prediction

\[
\text{if } (x < A\_\text{size}) \quad \text{help}
\]
\[
y = B[A[x]]
\]

Size of array $A$

Branch predictor
Speculative execution + branch prediction

\[
\text{if } (x < A\_size) \quad \text{HELP} \\
y = B[A[x]]
\]

Prediction based on \textit{branch history} & \textit{program structure}

Size of array \(A\)

Branch predictor
Speculative execution + branch prediction

if \( x < A_{\text{size}} \)

\( y = B[A[X]] \)

Prediction based on **branch history & program structure**

Size of array \( A \)

Branch predictor
Speculative execution + branch prediction

\[
\text{if } (x < A\text{\_size}) \\
y = B[A[x]]
\]

Wrong prediction? **Rollback changes**!
- Architectural (ISA) state
- Microarchitectural state

Prediction based on *branch history & program structure*

Branch predictor
Speculative non-interference
Speculative non-interference
Speculative non-interference

Program $P$ is \textit{speculatively non-interferent} if
Speculative non-interference

Program $\mathcal{P}$ is **speculatively non-interferent** if

Informally:

Leakage of $\mathcal{P}$ in **non-speculative** execution $\quad = \quad$ Leakage of $\mathcal{P}$ in **speculative** execution
How to capture leakage?

- Non-speculative semantics
- Speculative semantics
- Attacker model
How to capture leakage?

- Non-speculative semantics
- Speculative semantics
- Attacker model
- Model program’s behavior
How to capture leakage?

- Non-speculative semantics
- Speculative semantics

+ Attack model
- Capture attacker’s observational power

Model program’s behavior
μAssembly + non-speculative semantics

\[
\begin{align*}
&\text{if } (x < A\_size) \\
&\quad y = B[A[x]] \\
\end{align*}
\]

\[
\begin{align*}
\text{rax} &\gets A\_size \\
\text{rcx} &\gets x \\
\text{jmp } &\text{rcx} \geq \text{rax},\; END \\
L1: &\; \text{load } \text{rax}, \; A + \text{rcx} \\
&\; \text{load } \text{rax}, \; B + \text{rax} \\
END: &
\end{align*}
\]
\[ \text{if } (\mathbf{x} < \text{A\_size}) \quad y = \text{B}[\text{A}[\mathbf{x}]] \]

\[
\begin{align*}
\text{rax} & \leftarrow \text{A\_size} \\
\text{rcx} & \leftarrow \mathbf{x} \\
& \text{jmp } \text{rcx} \geq \text{rax}, \quad \text{END} \\
\text{L1: load } \text{rax}, \text{ A + rcx} \\
& \text{load } \text{rax}, \text{ B + rax} \\
\text{END:}
\end{align*}
\]
μAssembly + non-speculative semantics

\[\text{if } (x < A\_size)\]
\[y = B[A[x]]\]

\[
\begin{align*}
\text{rax} & \leftarrow A\_size \\
\text{rcx} & \leftarrow x \\
\text{jmp } & \text{ rcx} \geq \text{rax}, \text{ END} \\
L1: & \text{ load rax, A + rcx} \\
& \text{ load rax, B + rax} \\
END:
\end{align*}
\]
μAssembly + non-speculative semantics

\[
\begin{align*}
\text{if } (x < A\_size) \\
y &= B[A[x]]
\end{align*}
\]

\[
\begin{align*}
x &= A\_size \\
x &= x \\
jmp\ \text{rcx} \geq \text{rax},\ \text{END} \\
L1: &\ \text{load} \ \text{rax},\ A + \text{rcx} \\
\ &\text{load} \ \text{rax},\ B + \text{rax} \\
\text{END:}
\end{align*}
\]
μAssembly + non-speculative semantics

if \( x < A_{\text{size}} \)
\( y = B[A[x]] \)

\[
\begin{align*}
\text{rax} & \leftarrow A_{\text{size}} \\
\text{rcx} & \leftarrow x \\
\text{jmp} & \quad \text{rcx} \geq \text{rax}, \quad \text{END}
\end{align*}
\]

\text{L1: load} \quad \text{rax, } A + \text{rcx}

\text{load} \quad \text{rax, } B + \text{rax}

\text{END:}
μAssembly + non-speculative semantics

```
if (x < A_size)
  y = B[A[x]]
```

```
rax <- A_size
rcx <- x
jmp rcx≥rax, END
L1: load rax, A + rcx
load rax, B + rax
END:
```
Speculative semantics

rax <- A_size
rcx <- x
jmp rcx≥rax, END

L1: load rax, A + rcx
load rax, B + rax

END:
Speculative semantics

rax <- $A_{\text{size}}$
rcx <- $x$
jmp rcx $\geq$ rax, $END$

$L1$: load rax, $A + rcx$
load rax, $B + rax$

$END$:

Prediction Oracle $O$ : branch prediction + length of speculative window
Speculative semantics

rax <- A_size
rcx <- x
jmp rcx ≥ rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Starts speculative transactions upon branch instructions

Prediction Oracle O: branch prediction + length of speculative window
Speculative semantics

rax <- \texttt{A\_size}
rcx <- \texttt{x}
jmp rcx≥rax, \texttt{END}

\texttt{L1: load} rax, \texttt{A} + rcx
\texttt{load} rax, \texttt{B} + rax

\texttt{END:}

Starts \textit{speculative transactions} upon branch instructions

Committed upon correct speculation

Prediction Oracle \textbf{O}: branch prediction + length of speculative window
Speculative semantics

\[
\begin{align*}
\text{rax} & \leftarrow A_{\text{size}} \\
\text{rcx} & \leftarrow x \\
\text{jmp} \text{ rcx} \geq \text{rax}, \ END \\
L1: \text{ load} & \text{ rax, A + rcx} \\
\text{load} & \text{ rax, B + rax} \\
\text{END:} &
\end{align*}
\]

Starts *speculative transactions* upon branch instructions

Committed upon correct speculation

Rolled back upon misspeculation

**Prediction Oracle** \(O\): branch prediction + length of speculative window
Speculative semantics

rax \leftarrow \texttt{A\_size} \\
rcx \leftarrow x \\
jmp rcx \geq rax, \textit{END} \\
\textit{L1: load} \ rax, \ A + rcx \\
\text{load} \ rax, \ B + rax \\
\textit{END:} \\

\textbf{Prediction Oracle} \ O \ : \ branch \ prediction + length \ of \ speculative \ window
Speculative semantics

```
rax ← A_size
rcx ← x
jmp rcx ≥ rax, END

L1: load rax, A + rcx
load rax, B + rax

END:
```

Starts *speculative transactions* upon branch instructions

Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle $O$: branch prediction + length of speculative window
Speculative semantics

```plaintext
rax <- A_size
rcx <- x
jmp rcx ≥ rax, END

L1: load rax, A + rcx
load rax, B + rax

END:
```

Starts *speculative transactions* upon branch instructions

Committed upon correct speculation

Rolled back upon misspeculation

**Prediction Oracle** $O$: branch prediction + length of speculative window
Speculative semantics

rax <- \texttt{A\_size}
rcx <- \texttt{x}
\texttt{jmp rcx} \geq \texttt{rax}, \texttt{END}

L1:\ load \texttt{rax}, \texttt{A} + \texttt{rcx}
load \texttt{rax}, \texttt{B} + \texttt{rax}

END:

Starts \textit{speculative transactions} upon branch instructions
Committed upon correct speculation
Rolled back upon misspeculation

Prediction Oracle $O$: branch prediction + length of speculative window
Speculative semantics

rax <- A\_size
rcx <- x
jmp rcx\geq rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Starts \textit{speculative transactions} upon branch instructions

Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle $O$ : branch prediction + length of speculative window
Speculative semantics

rax <- \texttt{A\_size}
rcx <- \texttt{x}
\texttt{jmp rcx\geq rax, END}

\texttt{L1: load rax, A + rcx}
\texttt{load rax, B + rax}

\texttt{END:}

\textbf{Starts \textit{speculative transactions} upon branch instructions}

\textbf{Committed upon correct speculation}

\textbf{Rolled back upon misspeculation}

\textbf{Prediction Oracle} \textbf{O}: branch prediction + length of speculative window
Speculative semantics

rax <- A_size
rcx <- x
jmp rcx >= rax, END

L1: load rax, A + rcx
load rax, B + rax
END:

Starts *speculative transactions* upon branch instructions

Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle $O$: branch prediction + length of speculative window
Speculative semantics

rax <- A_size
rcx <- x
jmp rcx ≥ rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Starts *speculative transactions*
upon branch instructions

Committed upon correct speculation

Rolled back upon misspeculation

Prediction Oracle \( O \): branch prediction + length of speculative window
Leakage into μarchitecture

rax <- \textit{A\_size}
rce <- x
jmp rcx\geq rax, \textit{END}

\textit{L1: load} rax, A + rcx
load rax, B + rax

\textit{END:}
Leakage into μarchitecture

rax <- _A_size_
rcx <- x
jmp rcx ≥ rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Attacker can observe:
- locations of memory accesses
- branch/jump targets
- start/end speculative execution
Leakage into μarchitecture

\[ \text{rax} \leftarrow \text{A\_size} \]
\[ \text{rcx} \leftarrow x \]
\[ \text{jmp rcx} \geq \text{rax}, \text{ END} \]

**L1:** load rax, A + rcx
load rax, B + rax

**END:**

Attacker can observe:
- locations of memory accesses
- branch/jump targets
- start/end speculative execution

Inspired by “constant-time” rqmts
Leakage into μarchitecture

rax <- A_size
rcx <- x
jmp rcx >= rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Attacker can observe:
- locations of memory accesses
- branch/jump targets
- start/end speculative execution

Inspired by “constant-time” rqmts
Leakage into μarchitecture

rax ← \textit{A\_size}
rcx ← \textit{x}
jmp rcx\geq rax, \textit{END}
\textit{L1}: load rax, \textit{A} + rcx
load rax, \textit{B} + rax
\textit{END}:

Attacker can observe:
- locations of \textit{memory accesses}
- \textit{branch/jump} targets
- \textit{start/end} speculative execution

Inspired by “constant-time” rqmts
Leakage into μarchitecture

rax ← A\textunderscore size
rcx ← x
jmp rcx ≥ rax, END
L1: load rax, A + rcx
load rax, B + rax
END:

Attacker can observe:
- locations of memory accesses
- branch/jump targets
- start/end speculative execution

Inspired by “constant-time” rqmts
Leakage into μarchitecture

rax <- A_size
rcx <- x
jmp rcx≥rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Attacker can observe:
- locations of memory accesses
- branch/jump targets
- start/end speculative execution

Inspired by “constant-time” reqmts
Leakage into μarchitecture

rax <- \texttt{A\_size}
rcx <- \texttt{x}
jmp rcx\geq\texttt{rax}, \texttt{END}

\textbf{L1:} load rax, \texttt{A + rcx}
load rax, \texttt{B + rax}

\textbf{END:}

\begin{itemize}
\item Attacker can observe:
\begin{itemize}
\item locations of \texttt{memory accesses}
\item \texttt{branch/jump} targets
\item \texttt{start/end} speculative execution
\end{itemize}
\end{itemize}

Inspired by “constant-time” reqmts

\begin{itemize}
\end{itemize}
Leakage into μarchitecture

```plaintext
rax <- A_size
rcx <- x
jmp rcx >= rax, END

L1: load rax, A + rcx
load rax, B + rax

END:
```

Attacker can observe:
- locations of memory accesses
- branch/jump targets
- start/end speculative execution

Inspired by “constant-time” rqmts
Leakage into μarchitecture

rax <- A_size
rcx <- x
jmp rcx >= rax, END

L1: load rax, A + rcx
load rax, B + rax
END:

Attacker can observe:
- locations of memory accesses
- branch/jump targets
- start/end speculative execution

Inspired by “constant-time” rqmts

load B + A[x]
Leakage into μarchitecture

rax <- \texttt{A\_size}
rcx <- \texttt{x}
\texttt{jmp rcx} \geq \texttt{rax}, \texttt{END}

L1: \texttt{load} rax, \texttt{A} + rcx
load rax, \texttt{B} + rax

END:

Attacker can observe:
- locations of \texttt{memory accesses}
- \texttt{branch/jump} targets
- \texttt{start/end} speculative execution

Inspired by “constant-time” rqmts
Leakage into µarchitecture

rax <- $A_{\text{size}}$
rcx <- $x$
jmp $rcx \geq rax$, END

$L1$: load rax, $A + rcx$
load rax, $B + rax$

END:

Attacker can observe:
- locations of memory accesses
- branch/jump targets
- start/end speculative execution

Inspired by “constant-time” rqmts
Speculative non-interference
Speculative non-interference

Program $P$ is **speculatively non-interferent** for prediction oracle $O$ if
Speculative non-interference

Program $\mathcal{P}$ is **speculatively non-interferent** for prediction oracle $\mathcal{O}$ if

For all program states $s$ and $s'$:
Speculative non-interference

Program $P$ is **speculatively non-interferent** for prediction oracle $O$ if

Formally!

For all program states $s$ and $s'$:

$$P_{\text{non-spec}}(s) = P_{\text{non-spec}}(s')$$
Program $P$ is **speculatively non-interferent** for prediction oracle $O$ if

For all program states $s$ and $s'$:

\[ P_{\text{non-spec}}(s) = P_{\text{non-spec}}(s') \]

\[ \Rightarrow P_{\text{spec}}(s, O) = P_{\text{spec}}(s', O) \]
Reasoning about arbitrary oracles
Reasoning about arbitrary oracles

Always-mispredict speculative semantics

- Mispredict all branch instructions
- Fixed speculative window
- Rollback of every transaction
Reasoning about arbitrary oracles

Always-mispredict speculative semantics

- Mispredict all branch instructions
- Fixed speculative window
- Rollback of every transaction

Always-mispredict is worst-case

\[ P_{am}(s) = P_{am}(s') \iff \forall O. P_{spec}(s,O) = P_{spec}(s',O) \]
Reasoning about arbitrary oracles

Always-mispredict speculative semantics

- Mispredict all branch instructions
- Fixed speculative window
- Rollback of every transaction

Always-mispredict is worst-case:

\[ P_{am}(s) = P_{am}(s') \iff \forall O. P_{spec}(s,O) = P_{spec}(s',O) \]

If program \( P \) satisfies

\[ \forall s,s'. P_{non-spec}(s) = P_{non-spec}(s') \Rightarrow P_{am}(s) = P_{am}(s') \]

then \( P \) satisfies \( SNI \) w.r.t. all \( O \)
Detecting speculative leaks
Detecting speculative leaks

rax <- A\_size
rcx <- x
jmp rcx\geq rax, END

L1: load rax, A + rcx
load rax, B + rax

END:
Detecting speculative leaks

rax <- A\_size
rcx <- x
jmp rcx\geq rax, END
L1: load rax, A + rcx
load rax, B + rax
END:

Detect leaks

Symbolic trace: path condition + observations along the symbolic path
Symbolic execution

rax <- A_size
rcx <- x
jmp rcx ≥ rax, END
L1: load rax, A + rcx
load rax, B + rax
END:
Symbolic execution

rax <- \texttt{A\_size}
rcx <- \texttt{x}
jmp \texttt{rcx} \geq \texttt{rax}, \texttt{END}

\texttt{L1: load rax, A + rcx}
\texttt{load rax, B + rax}

\texttt{END:}

\texttt{Always mispredict branch instructions}
Symbolic execution

rax <- A\_size
rcx <- x
jmp rcx >= rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

true

*Always mispredict*

branch instructions
Symbolic execution

```plaintext
rax <- A_size
rcx <- x
jmp rcx ≥ rax, END
L1: load rax, A + rcx
    load rax, B + rax
END:
```

*Always mispredict* branch instructions
Symbolic execution

rax <- \textit{A\_size}
rcx <- \textit{x}

\textbf{jmp} \textit{rcx} \geq \textit{rax, END}

\textbf{L1:} load \textit{rax, A + rcx}
load \textit{rax, B + rax}

\textit{END:}

\textit{Always mispredict}
branch instructions
Symbolic execution

\[
\begin{align*}
&\text{rax} \leftarrow \text{A\_size} \\
&\text{rcx} \leftarrow \text{x} \\
&\text{jmp rcx} \geq \text{rax}, \text{ END} \\
&L1: \text{load rax, A } + \text{ rcx} \\
&\text{load rax, B } + \text{ rax} \\
&\text{END:}
\end{align*}
\]

*Always mispredict branch instructions*
Symbolic execution

rax <- \textit{A}_size
rcx <- x
\texttt{jmp rcx} \geq rax, \textit{END}

\textit{L1}: load rax, A + rcx
load rax, B + rax

\textit{END}:

Always mispredict branch instructions
Symbolic execution

rax <- A\_size
rcx <- x
jmp rcx \geq rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Always mispredict branch instructions
Symbolic execution

rax <- A_size
rcx <- x
jmp rcx ≥ rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Always mispredict branch instructions
Symbolic execution

rax <- A_size
rcx <- x
jmp rcx ≥ rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Always mispredict branch instructions
Symbolic execution

rax <- A_size
cx <- x
jmp rcx>=rax, END

L1: load rax, A + rcx
load rax, B + rax

END:

Always mispredict branch instructions

start pc L1 load A+x load B+A[x] rollback pc END
Symbolic execution

\[ \text{rax} \leftarrow \text{A}_\text{size} \]
\[ \text{rcx} \leftarrow \text{x} \]
\[ \text{jmp rcx} \geq \text{rax}, \quad \text{END} \]

\[ \text{L1: load rax, A + rcx} \]
\[ \text{load rax, B + rax} \]

\[ \text{END:} \]

Always mispredict branch instructions

![Diagram showing symbolic execution flow](image)
Symbolic execution

rax <- \( A\_size \)
rcx <- \( x \)
jmp rcx\( \geq \)rax, END

\( L1: \) load rax, \( A + rcx \)
load rax, \( B + rax \)

Always mispredict branch instructions

\( x \geq A\_size \)
\( x < A\_size \)

start pc \( L1 \) load \( A+x \) load \( B+A[x] \) rollback pc END
Detecting speculative leaks

Symbolic trace: path condition + observations along the symbolic path

rax <- \texttt{A\_size}
rcx <- \texttt{x}
jmp rcx\geq rax, \texttt{END}

\texttt{L1: load } \texttt{rax, A + rcx}
\texttt{load } \texttt{rax, B + rax}

\texttt{END:}
Detecting speculative leaks

For each symbolic trace $\tau \in traces(prg)$

- if $MemLeak(\tau)$ then
  - return $INSECURE$
- if $CtrlLeak(\tau)$ then
  - return $INSECURE$
- return $SECURE$
Detecting speculative leaks

For each symbolic trace $\tau \in \text{traces}(\text{prg})$

- if $\text{MemLeak}(\tau)$ then
  - return $\text{INSECURE}$
- if $\text{CtrlLeak}(\tau)$ then
  - return $\text{INSECURE}$
- return $\text{SECURE}$
Memory leaks

Speculative memory accesses must depend only on

- Non-sensitive information
- Non-speculative observations
Memory leaks

Speculative memory accesses must depend only on

- Non-sensitive information
- Non-speculative observations
Memory leaks

Speculative memory accesses must depend only on

- Non-sensitive information
- Non-speculative observations

\[ pathCnd(\tau) \land obsEqv(\tau|_{\text{non-spec}}) \land \neg obsEqv(\tau|_{\text{spec}}) \]
Memory leaks

Speculative memory accesses must depend only on

- Non-sensitive information
- Non-speculative observations

\[ \text{pathCnd}(\tau) \land \text{obsEqv}(\tau|_{\text{non-spec}}) \land \neg \text{obsEqv}(\tau|_{\text{spec}}) \]

Check with self-composition
Memory leaks

Speculative memory accesses must depend only on

• Non-sensitive information

• Non-speculative observations

\[ \text{pathCnd}(\tau) \land \text{obsEqv}(\tau|_{\text{non-spec}}) \land \neg \text{obsEqv}(\tau|_{\text{spec}}) \]

Check with self-composition
Memory leaks

Speculative memory accesses \textit{must} depend only on

- Non-sensitive information
- Non-speculative observations

\[
pathCnd(\tau) \land \text{obsEqv}(\tau|_{\text{non-spec}}) \land \neg\text{obsEqv}(\tau|_{\text{spec}})
\]

Check with self-composition
Memory leaks

Speculative memory accesses must depend only on

- Non-sensitive information
- Non-speculative observations

$$\tau$$

Check with self-composition

Equivalent wrt policy

$$s_1 \models \varphi$$

$$s_2 \models \varphi$$
Memory leaks

Speculative memory accesses **must** depend only on

- Non-sensitive information
- Non-speculative observations

\[ \tau \]

\[ \text{pathCnd}(\tau) \land \text{obsEqv}(\tau|_{\text{non-spec}}) \land \neg \text{obsEqv}(\tau|_{\text{spec}}) \]

\[ s_1 \models \varphi \]

\[ s_2 \models \varphi \]

Check with self-composition

Equivalent wrt policy
Memory leaks

Speculative memory accesses must depend only on

- Non-sensitive information
- Non-speculative observations

Check with self-composition

Equivalent wrt policy

\[
\tau \equiv \text{pathCnd}(\tau) \land \text{obsEqv}(\tau|_{\text{non-spec}}) \land \neg \text{obsEqv}(\tau|_{\text{spec}})
\]

\[S_1 \models \varphi \quad \text{||} \quad \models \quad \not\models \quad S_2 \models \varphi\]
Spectector + Case studies
Spectector

```
mov    rax, A_size
mov    rcx, x
cmp    rcx, rax
jae    END
mov    rax, A[rcx]
mov    rax, B[rax]
```

x64 to μASM

```
rax <- A_size
rcx <- x
jmp  rcx≥rax, END

L1:
load rax, A + rcx
load rax, B + rax

END:
```

Symbolic execution

Check for speculative leaks
Spectector

```assembly
mov rax, A_size
mov rcx, x
cmp rcx, rax
jae END
mov rax, A[rcx]
mov rax, B[rax]
L1: mov rax, A[rax]
    mov rax, B[rax]
END:
```

x64 to μASM

```assembly
rax <- A_size
rcx <- x
jmp rcx >= rax, END
load rax, A + rcx
load rax, B + rax
```

More details

- Built in Prolog
- **Z3** for symbolic execution and leak detection

Symbolic execution

Check for speculative leaks
Case study: compiler mitigations

Target:

• 15 variants of Spectre V1 by Paul Kocher*

• Compiled with Microsoft Visual C++, Intel ICC, and Clang with different mitigations and optimization levels

• 240 assembly programs of up to 200 instructions each

How:

• Use Spectector to prove security or detect leaks

* Paul Kocher - Spectre Mitigations in Microsoft C/C++ Compiler — https://www.paulkocher.com/doc/MicrosoftCompilerSpectreMitigation.html
### Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>VCC</th>
<th>ICC</th>
<th>CLANG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN 19.15</td>
<td>FEN 19.20</td>
</tr>
<tr>
<td></td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>01</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>02</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>03</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>04</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>05</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>06</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>07</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>08</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>09</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>10</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>11</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>12</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>13</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>15</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
</tbody>
</table>

- • denotes that SPECTECTOR detects a speculative leak, whereas
- o indicates that SPECTECTOR proves the program secure.
## Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>VCC</th>
<th>ICC</th>
<th>CLANG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN 19.15</td>
<td>FEN 19.20</td>
</tr>
<tr>
<td></td>
<td>UNP</td>
<td>FEN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UNP</td>
<td>FEN</td>
<td>SLH</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ex.</th>
<th>-00</th>
<th>-02</th>
<th>-00</th>
<th>-02</th>
<th>-00</th>
<th>-02</th>
<th>-00</th>
<th>-02</th>
<th>-00</th>
<th>-02</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>02</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>03</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>04</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>05</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>06</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>07</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>08</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>09</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>10</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>11</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>12</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>13</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>14</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>15</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

- ○ denotes that SPECPECTOR detects a speculative leak, whereas ● indicates that SPECPECTOR proves the program secure.
Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>VCC Unp</th>
<th>VCC FEN 19.15</th>
<th>VCC FEN 19.20</th>
<th>ICC Unp</th>
<th>ICC FEN</th>
<th>CLANG Unp</th>
<th>CLANG FEN</th>
<th>CLANG SLH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-00 -02</td>
<td>-00 -02</td>
<td>-00 -02</td>
<td>-00 -02</td>
<td>-00 -02</td>
<td>-00 -02</td>
<td>-00 -02</td>
<td>-00 -02</td>
</tr>
<tr>
<td>01</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>02</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>03</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>04</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>05</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>06</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>07</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>08</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>09</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>10</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>11</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>12</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>13</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>15</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
</tbody>
</table>

Fig. 7. Analysis of Kocher's examples [16] compiled with compilers and options. For each of the 15 examples, we analyzed the unpatched version (denoted by U), the version patched with speculation barriers (denoted by F), and the version patched using speculative load hardening (denoted by S). Programs have been compiled without optimizations (-O0) or with compiler optimizations (-O2) using the compilers VISUAL C++ (two versions), ICC, and CLANG. "•" denotes that SPECTECTOR detects a speculative leak, whereas "o" indicates that SPECTECTOR proves the program secure.
## Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Vcc</th>
<th>ICC</th>
<th>Clang</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN 19.15</td>
<td>FEN 19.20</td>
</tr>
<tr>
<td>00</td>
<td>-02</td>
<td>-02</td>
<td>-02</td>
</tr>
<tr>
<td>01</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>02</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>03</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>04</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>05</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>06</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>07</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>08</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>09</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>10</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>11</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>12</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>13</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>15</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
</tbody>
</table>

**Fig. 7.** Analysis of Kocher’s examples compiled with compilers and options. For each of the 15 examples, we analyzed the unpatched version (denoted by U), the version patched with speculation barriers (denoted by F), and the version patched using speculative load hardening (denoted by S). Programs have been compiled without optimizations (-00) or with compiler optimizations (-02) using the compilers VIALC++ (two versions), ICC, and CLANG. O indicates that SPECTECTOR detects a speculative leak, whereas 24 indicates that SPECTECTOR proves the program secure.
# Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>VCC</th>
<th>ICC</th>
<th>CLANG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unp -00 -02</td>
<td>Fen 19.15 -00 -02</td>
<td>Unp -00 -02</td>
</tr>
<tr>
<td>01</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>02</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>03</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>04</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>05</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>06</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>07</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>08</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>09</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>10</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>11</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>12</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>13</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>14</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>15</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
</tbody>
</table>

*Fig. 7. Analysis of Kocher's examples [16] compiled with compilers and options. For each of the 15 examples, we analyzed the unpatched version (denoted by Unp), the version patched with speculation barriers (denoted by Fen), and the version patched using speculative load hardening (denoted by Slh). Programs have been compiled without optimizations (-O0) or with compiler optimizations (-O2) using the compilers VCC++ (two versions), ICC, and CLANG.*

- ○ denotes that SPECTECTOR detects a speculative leak,
- ● indicates that SPECTECTOR proves the program secure.
## Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Vcc</th>
<th>ICC</th>
<th>CLANG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN</td>
<td>UNP</td>
</tr>
<tr>
<td></td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>01</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>02</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>03</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>04</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>05</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>06</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>07</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>08</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>09</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>10</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>11</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>12</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>13</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>15</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
</tbody>
</table>

Automated insertion of fences
## Results

The table below shows the analysis of Kocher's examples compiled with various compilers and options. For each of the 15 examples, we analyzed the unpatched version (denoted by U), the version patched with speculation barriers (denoted by F), and the version patched using speculative load hardening (denoted by S). Programs have been compiled without optimizations (−O0) or with compiler optimizations (−O2) using the compilers Visual C++ (two versions), ICC, and CLANG.

<table>
<thead>
<tr>
<th>Ex.</th>
<th>VCC</th>
<th>ICC</th>
<th>CLANG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN 19.15</td>
<td>FEN 19.20</td>
</tr>
<tr>
<td>01</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>02</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>03</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>04</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>05</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>06</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>07</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>08</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>09</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>10</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>11</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>12</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>13</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>14</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>15</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

*•* denotes that SPECTECTOR detects a speculative leak, whereas • indicates that SPECTECTOR proves the program secure.
# Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>VCC</th>
<th>ICC</th>
<th>CLANG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN 19.15</td>
<td>FEN 19.20</td>
</tr>
<tr>
<td></td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>01</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>02</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>03</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>04</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>05</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>06</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>07</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>08</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>09</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>10</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>11</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>12</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>13</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>14</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>15</td>
<td>○</td>
<td>●</td>
<td>○</td>
</tr>
</tbody>
</table>

Figure 7. Analysis of Kocher’s examples [16] compiled with compilers and options. For each of the 15 examples, we analyzed the unpatched version (denoted by U), the version patched with speculation barriers (denoted by F), and the version patched using speculative load hardening (denoted by S). Programs have been compiled without optimizations (-O0) or with compiler optimizations (-O2) using the compilers VCC++ (two versions), ICC, and CLANG.

- • denotes that SPECTEGER detects a speculative leak, whereas ○ indicates that SPECTEGER proves the program secure.
## Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>VCC</th>
<th>ICC</th>
<th>CLANG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unp</td>
<td>F19.15</td>
<td>F19.20</td>
</tr>
<tr>
<td>01</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>02</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>03</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>04</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>05</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>06</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>07</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>08</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>09</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>10</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>11</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>12</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>13</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>14</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>15</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
</tbody>
</table>

- ○ denotes that SPECTECTOR detects a speculative leak, whereas ● indicates that SPECTECTOR proves the program secure.
## Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>VCC</th>
<th>ICC</th>
<th>CLANG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>UNP</td>
<td>UNP</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### VCC

- **Fen 19.15**
- **Fen 19.20**

### ICC

- **Fen**

### CLANG

- **Fen**
- **SLH**

---

### Summary

- **Leaks in all unprotected programs**
  (except example #08 with optimizations)

- **Confirm all vulnerabilities in VCC pointed out by Paul Kocher**

- **Programs with fences (ICC and Clang) are secure**

- **Unnecessary fences**

- **Programs with SLH are secure except #10 and #15**
Case study: scalability

**Target:** Xen hypervisors

**Main challenges for scalability:**
- Policy definition
- ISA coverage
- Path explosion

**How:**
- Analyze scalability of checking SNI *relative to* symbolic execution
- 24’000 symbolic paths of < 10’000 instructions (from ~ 4’000 functions)
Case study: scalability

**Target:** Xen hypervisors

**Main challenges for scalability:**
- Policy definition
- ISA coverage
- Path explosion

**How:**
- Analyze scalability of checking SNI *relative to* symbolic execution
- 24’000 symbolic paths of < 10’000 instructions (from ~ 4’000 functions)

Trade-offs affect analysis soundness and completeness
Results
Results

- SNI 10x-100x faster
- 20.2% traces
Results

- SNI 10x-100x faster
- 20.2% traces
- SNI ≤10x faster
- 41.9% traces
Results

- SNI 10x-100x faster
  - 20.2% traces
- SNI ≤10x faster
  - 41.9% traces
- SNI ≤10x slower
  - 26.9% traces
Results

- SNI 10x-100x faster
  - 20.2% traces

- SNI ≤10x faster
  - 41.9% traces

- SNI ≤10x slower
  - 26.9% traces

- SNI 10x-100x slower
  - 7.9% traces
Results

- SNI 10x-100x faster
- 20.2% traces

- SNI ≤ 10x faster
- 41.9% traces

- SNI ≤ 10x slower
- 26.9% traces

- SNI 10x-100x slower
- 7.9% traces

Checking SNI scales roughly as well as discovering new paths in symbolic execution
Conclusion
Speculative non-interference

Program $P$ is **speculatively non-interferent** for prediction oracle $O$ if

For all program states $s$ and $s'$:

$$P_{\text{non-spec}}(s) = P_{\text{non-spec}}(s')$$

$$\implies P_{\text{spec}}(s, O) = P_{\text{spec}}(s', O)$$

**Results**

<table>
<thead>
<tr>
<th>Ex.</th>
<th>VCC</th>
<th>ICC</th>
<th>CLANG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN 19.15</td>
<td>FEN 19.20</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>03</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>04</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>06</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>08</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>09</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Spectator**

- SNI $10x-100x$ faster
- $20.2\%$ traces
- SNI $\leq 10x$ faster
- $41.9\%$ traces
- SNI $\leq 10x$ slower
- $26.9\%$ traces
- SNI $10x-100x$ slower
- $7.9\%$ traces

---

Just a few lines of code to illustrate the symbolic execution:

```
mov rax, A_size
mov rcx, x
cmp rcx, rax
jae END
mov rax, A[rcx]
mov rax, B[rax]
```

x64 to µASM

```
x64 to µASM
```
### Results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Vcc</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNP</td>
<td>FEN 19.15</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
</tr>
<tr>
<td>03</td>
<td>00</td>
</tr>
<tr>
<td>04</td>
<td>00</td>
</tr>
<tr>
<td>05</td>
<td>00</td>
</tr>
<tr>
<td>06</td>
<td>00</td>
</tr>
<tr>
<td>07</td>
<td>00</td>
</tr>
<tr>
<td>08</td>
<td>00</td>
</tr>
<tr>
<td>09</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>12</td>
<td>00</td>
</tr>
<tr>
<td>13</td>
<td>00</td>
</tr>
<tr>
<td>14</td>
<td>00</td>
</tr>
<tr>
<td>15</td>
<td>00</td>
</tr>
</tbody>
</table>

Program $P$ is **speculatively non-interferent** for prediction oracle $O$ if:

$$P_{non-spec}(s) = P_{non-spec}(s') \quad \text{and} \quad P_{spec}(s, O) = P_{spec}(s', O) \quad \Rightarrow \quad \text{Formally!}$$

**Spectector**

https://spectector.github.io

@ marco.guarnieri@imdea.org

@MarcoGuarnier1

- SNI ≤10x slower
- 26.9% traces
- SNI 10x-100x slower
- 7.9% traces