Spectector: Principled detection of speculative information flows

Jan Reineke @ Universität des Saarlandes

Joint work with
Marco Guarnieri, Jose Morales, Andres Sanchez @ IMDEA Software, Madrid
Boris Köpf @ Microsoft Research, Cambridge, UK

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

Exploits **speculative execution** to leak sensitive information
Exploits **speculative execution** to leak sensitive information

Almost all modern processors are affected

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

*In particular*: Compiler-level countermeasures

✓ *Example*: insert “fences” to selectively terminate 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

*In particular*: Compiler-level countermeasures

✓ **Example**: insert “fences” to selectively terminate speculative execution

✓ Implemented in major compilers (Microsoft Visual C++, Intel ICC, Clang)

**Problem Solved?**
Compiler-level countermeasures

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

Paul Kocher
February 13, 2018
Compiler-level countermeasures

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

Paul Kocher
February 13, 2018

“The countermeasure […] is conceptually straightforward but **challenging in practice**”
“The countermeasure [...] is conceptually straightforward but challenging in practice”

“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

“The countermeasure […] is conceptually straightforward but challenging in practice”

“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

Spectre Mitigations in Microsoft's C/C++ Compiler
Paul Kocher
February 13, 2018

“The countermeasure […] is conceptually straightforward but **challenging in practice**”

“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!
Goals
Goals

1. Introduce semantic notion of security against speculative execution attacks
Goals

1. Introduce **semantic notion of security** against **speculative execution attacks**

2. Static analysis to **detect vulnerability** or to **prove security**
Outline

1. Speculative execution attacks
2. Speculative non-interference
3. Spectector: Detecting speculative leaks
4. Challenges
1. Speculative execution attacks
Background: Speculative execution
Background: Speculative execution

- Predict instructions’ outcomes and speculatively continue execution
Background: Speculative execution

• Predict instructions’ outcomes and speculatively continue execution

• Rollback changes if speculation was wrong
Background: Speculative execution

- Predict instructions’ outcomes and speculatively continue execution
- Rollback changes if speculation was wrong

Only architectural (ISA, “logical”) state, **not** microarchitectural state
Background: Branch prediction

\[
\text{if } (x < A\_size) \\
\quad y = B[A[x]]
\]
Background: Branch prediction

if \( x < A\_size \)

\[
y = B[A[x]]
\]
Background: Branch prediction

\[
\text{if } (x < A_{\text{size}}) \text{ then } \\
y = B[A[x]]
\]
Background: Branch prediction

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

Size of array A
Background: Branch prediction

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

Predictions based on branch history & program structure
Background: Branch prediction

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

\[ y = B[A[x]] \]

Predictions based on \textit{branch history} \& \textit{program structure}
void f(int x)
    if (x < A_size)
        y = B[A[x]]
void \textbf{f}(int x)
\begin{verbatim}
    if (x < A\_size)
        y = B[A[x]]
\end{verbatim}
void $f(int \ x)$
  
  if ($x < A\_size$)
    $y = B[A[x]]$
void f(int x)
    if (x < A_size)
        y = B[A[x]]
Spectre V1

```c
void f(int x)
    if (x < A_size)
        y = B[A[x]]
```

What is in $A[128]$?
void \( f(\text{int } x) \)
if \( (x < \text{A}_\text{size}) \)
\( y = B[A[x]] \)

What is in \( A[128] \)?

1a) Training
void f(int x)
    if (x < A_size)
        y = B[A[x]]

What is in A[128]?

1a) Training

Cache state

A_size=16
B[B[0],B[1],...]

1a) Training
Spectre V1

```c
void f(int x)
if (x < A_size)
y = B[A[x]]
```

What is in $A[128]$?

```
1a) Training $f(0)$;
```

$A_{size}=16$

<table>
<thead>
<tr>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B[0]$ $B[1]$ ...</td>
</tr>
</tbody>
</table>
void f(int x)
if (x < A_size)
y = B[A[x]]

A_size = 16
B = [B[0], B[1], ...]

What is in A[128]?

1a) Training $f(0); f(1)$;
void \texttt{f}(int \ x) \\
\textbf{if} \ (x < A\_size) \\
y = B[A[x]]

\textbf{A\_size=16}

B
\begin{tabular}{ccc}
B[0] & B[1] & \ldots
\end{tabular}

What is in A[128]?

1a) Training \[ f(0); f(1); f(2); \ldots \]
void \texttt{f}(\texttt{int} \ x) \\
\quad \texttt{if} (x < \texttt{A\_size}) \\
\quad \quad y = \texttt{B}[\texttt{A}[x]]
void f(int x)
if (x < A_size)
y = B[A[x]]

What is in A[128]?

1a) Training f(0); f(1); f(2); ...
1b) Prepare cache
void \( f(int \ x) \)

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

**What is in** \( A[128] \)?

1. **Training**
   - \( f(0); f(1); f(2); ... \)

2. **Prepare cache**

3. **Run** \( f(128) \)
void f(int x)
if (x < A_size)
y = B[A[x]]

What is in A[128]?

1a) Training
f(0); f(1); f(2); ...

1b) Prepare cache

2) Run f(128)
void \( f(\text{int } x) \)

if \( (x < A\_size) \)

\[ y = B[A[x]] \]

What is in \( A[128] \)?

1a) Training

\( f(0); f(1); f(2); \ldots \)

1b) Prepare cache

2) Run \( f(128) \)
Spectre V1

void f(int x)
if (x < A_size)
y = B[A[x]]

A_size = 16

What is in A[128]?

1a) Training
   f(0); f(1); f(2); ...

1b) Prepare cache

2) Run f(128)
void $f$ (int $x$) 
if ($x < A_{\text{size}}$) 
$y = B[A[x]]$

1a) Training
$f(0); f(1); f(2); ...$

1b) Prepare cache

2) Run $f(128)$
Spectre V1

Address depends on A[128]

void f(int x)
if (x < A_size)
y = B[A[x]]

What is in A[128]?

1a) Training
   f(0); f(1); f(2); ...

1b) Prepare cache

2) Run f(128)
Spectre V1

Address depends on $A[128]$.

$A_{size}=16$

What is in $A[128]$?

$void f(int x)$

if $(x < A_{size})$

$y = B[A[x]]$

1a) Training

$f(0); f(1); f(2); ...$

1b) Prepare cache

2) Run $f(128)$

Persistent beyond rollback

Cache state

$B[0]; B[1]; B[A[128]]$
Spectre V1

Address depends on $A[128]$.

$A_{size}=16$

void $f(int \ x)$
if ($x < A_{size}$)
  $y = B[A[x]]$

What is in $A[128]$?

1a) Training $f(0); f(1); f(2); \ldots$

1b) Prepare cache

2) Run $f(128)$

3) Extract from cache

Persistent beyond rollback

Cache state
2. Speculative non-interference
Generalizing the Spectre V1 example

1a) Training $f(0); f(1); f(2); ...$

1b) Prepare cache

2) Run $f(128)$

3) Extract from cache
Generalizing the Spectre V1 example

1a) Training \( f(0); f(1); f(2); \ldots \)

1b) Prepare cache

2) Run \( f^{128} \)

3) Extract from cache

Attacker
Generalizing the Spectre V1 example

1a) Training

1b) Prepare cache

2) Run $f(128)$

3) Extract from cache

Victim

Attacker
Generalizing the Spectre V1 example

1a) Training  \( f(0); f(1); f(2); \ldots \)

1b) Prepare cache

2) Run \( f(128) \)

3) Extract from cache
Generalizing the Spectre V1 example

1) Prepares microarchitectural state

2) Leaks information into microarchitectural state

3) Extracts information from microarchitecture

Victim

{ }

Attacker

{ }
Speculative non-interference
Speculative non-interference

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

Program $P$ is **speculatively non-interferent** if

Informally:

- Leakage of $P$ in **non-speculative** execution
- Leakage of $P$ in **speculative** execution

15
Speculative non-interference

Program $P$ is **speculatively non-interferent** if

Informally:

Leakage of $P$ in non-speculative execution

More formally:

Leakage of $P$ in speculative execution
Speculative non-interference

Program $P$ is **speculatively non-interferent** if

**Informally:**
- Leakage of $P$ in **non-speculative** execution
- Leakage of $P$ in **speculative** execution

**More formally:**
For all program states $s$ and $s'$:

---

15
Speculative non-interference

Program $P$ is **speculatively non-interferent** if

**Informally:**
- Leakage of $P$ in **non-speculative** execution
- Leakage of $P$ in **speculative** execution

**More formally:**
For all program states $s$ and $s'$:

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

Program $P$ is **speculatively non-interferent** if

**Informally:**
Leakage of $P$ in non-speculative execution = Leakage of $P$ in speculative execution

**More formally:**
For all program states $s$ and $s'$:

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

Program $P$ is *speculatively non-interferent* if

**Informally:**
- Leakage of $P$ in non-speculative execution
- Leakage of $P$ in speculative execution

**More formally:**
For all program states $s$ and $s'$:

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

$$\Rightarrow \quad P_{spec}(s) = P_{spec}(s')$$
How to capture leakage into microarchitectural state?
How to capture leakage into microarchitectural state?

- Non-speculative semantics
- Speculative semantics
How to capture leakage into microarchitectural state?

Non-speculative semantics + Speculative semantics + Attacker/Observer model
\section*{μAssembly}

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

\begin{align*}
\text{rax} &\leftarrow A_{\text{size}} \\
\text{rcx} &\leftarrow x \\
\text{jmp} \ &\text{rcx} \geq \text{rax}, \ END \\
L1: &\quad \text{load} \ \text{rax}, \ A + \text{rcx} \\
&\quad \text{load} \ \text{rax}, \ B + \text{rax} \\
\text{END:} &
\end{align*}
\( \mu \text{Assembly + Non-speculative semantics} \)

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

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

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

\[
\text{END:}
\]
μAssembly + Non-speculative semantics

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

rax <- \( A_{\text{size}} \)
rcx <- \( x \)
jmp rcx ≥ rax, \text{ END}

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

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

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

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

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

\[END:\]
μAssembly + Non-speculative semantics

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

```
rax <- A\_size  
rcx <- x  
jmp rcx >= rax, END  
L1: load rax, A + rcx  
load rax, B + rax  
END:
```
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:
```
\muAssembly + Non-speculative semantics

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

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

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

\textbf{END:}
Non-speculative semantics: Inference Rules

Expression evaluation

\[ [n](a) = n \quad [x](a) = a(x) \quad [\Theta e](a) = \Theta[e](a) \quad [e_1 \otimes e_2](a) = [e_1](a) \otimes [e_2](a) \]

Instruction evaluation

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Skip</strong></td>
<td>[ p(a(pc)) = \text{skip} ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m, a[pc \mapsto a(pc) + 1] \rangle ]</td>
</tr>
<tr>
<td><strong>Barrier</strong></td>
<td>[ p(a(pc)) = \text{spbarr} ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m, a[pc \mapsto a(pc) + 1] \rangle ]</td>
</tr>
<tr>
<td><strong>Assign</strong></td>
<td>[ p(a(pc)) = x \leftarrow e \quad x \neq pc ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m, a[pc \mapsto a(pc) + 1, x \mapsto <a href="a">e</a>] \rangle ]</td>
</tr>
<tr>
<td><strong>ConditionalUpdate-Sat</strong></td>
<td>[ p(a(pc)) = x \leftarrow' e \quad <a href="a">e'</a> = 0 \quad x \neq pc ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m, a[pc \mapsto a(pc) + 1, x \mapsto <a href="a">e</a>] \rangle ]</td>
</tr>
<tr>
<td><strong>ConditionalUpdate-Unsat</strong></td>
<td>[ p(a(pc)) = x \leftarrow' e \quad <a href="a">e'</a> \neq 0 \quad x \neq pc ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m, a[pc \mapsto a(pc) + 1] \rangle ]</td>
</tr>
<tr>
<td><strong>Load</strong></td>
<td>[ p(a(pc)) = \text{load} x, e \quad x \neq pc ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m, a[pc \mapsto a(pc) + 1, x \mapsto <a href="a">e</a>] \rangle ]</td>
</tr>
<tr>
<td><strong>Store</strong></td>
<td>[ p(a(pc)) = \text{store} x, e ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m[n \mapsto a(x)], a[pc \mapsto a(pc) + 1] \rangle ]</td>
</tr>
<tr>
<td><strong>BEQZ-Sat</strong></td>
<td>[ p(a(pc)) = \text{beqz} x, \ell \quad a(x) = 0 ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m, a[pc \mapsto \ell] \rangle ]</td>
</tr>
<tr>
<td><strong>BEQZ-Unsat</strong></td>
<td>[ p(a(pc)) = \text{beqz} x, \ell \quad a(x) \neq 0 ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m, a[pc \mapsto a(pc) + 1] \rangle ]</td>
</tr>
<tr>
<td><strong>JMP</strong></td>
<td>[ p(a(pc)) = \text{jmp} e ]</td>
</tr>
<tr>
<td></td>
<td>[ \langle m, a \rangle \rightarrow \langle m, a[pc \mapsto \ell] \rangle ]</td>
</tr>
</tbody>
</table>
Speculative semantics

rax <- $A_{size}$
rcx <- $x$
jmp rcx ≥ rax, END

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

END:
Speculative semantics

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

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

\texttt{END:}

Starts \texttt{speculative transactions} upon branches
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 branches

Committed upon correct speculation
Speculative semantics

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

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

\textit{END:}

Starts \textbf{speculative transactions} upon branches

Committed upon correct speculation

Rolled back upon misspeculation
Speculative semantics

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

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

END:

- Starts **speculative transactions** upon branches
- Committed upon correct speculation
- Rolled back upon misspeculation
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 branches

Committed upon correct speculation

Rolled back upon misspeculation
Speculative semantics

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

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

END:

- Starts **speculative transactions** upon branches
- Committed upon correct speculation
- Rolled back upon misspeculation
Speculative semantics

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

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

END:

- Starts **speculative transactions** upon branches
- Committed upon correct speculation
- Rolled back upon misspeculation
Speculative semantics

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

Starts \textbf{speculative transactions} upon branches

Committed upon correct speculation

Rolled back upon misspeculation
Speculative semantics

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

Starts \textit{speculative transactions} upon branches

Committed upon correct speculation

Rolled back upon misspeculation
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 branches
Committed upon correct speculation
Rolled back upon misspeculation
Speculative semantics

rax ← \texttt{A\_size}
rcx ← \texttt{x}

\texttt{jmp rcx} ≥ \texttt{rax}, \texttt{END}

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

\texttt{END:}

Starts \textbf{speculative transactions} upon branches

Committed upon correct speculation

Rolled back upon misspeculation
Speculative semantics

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

\textbf{Prediction Oracle} \ O \text{ determines branch prediction + length of speculative window}

\textbf{Starts \textit{speculative transactions} upon branches}

\textbf{Committed upon correct speculation}

\textbf{Rolled back upon misspeculation}
Observer model: Leakage into \( \mu \)architectural state

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

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

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

\text{END:}

Attacker can observe:
- locations of \textit{memory accesses}
- \textit{branch/jump} targets
- \textit{start/end} speculative execution
Observer model: Leakage into μarchitectural state

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

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

\texttt{END:}

Attacker can observe:
- locations of \textit{memory accesses}
- \textit{branch/jump} targets
- \textit{start/end} speculative execution
Observer model: Leakage into μarchitectural state

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
Observer model: Leakage into μarchitectural state

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
Observer model: Leakage into μarchitectural state

\begin{align*}
\text{rax} &\leftarrow \text{A\_size} \\
\text{rcx} &\leftarrow x \\
\text{jmp} &\text{ rcx} \geq \text{rax}, \text{ END} \\
\end{align*}

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

Attacker can observe:
- locations of \textit{memory accesses}
- \textit{branch/jump} targets
- \textit{start/end} speculative execution
Observer model: Leakage into μarchitectural state

\[ \text{rax} \leftarrow \text{A\_size} \]
\[ \text{rcx} \leftarrow x \]
\[ \text{jmp \ rcx} \geq \text{rax}, \hspace{0.5cm} \text{END} \]

\( L1: \) \text{load} \ \text{rax, \ A + rcx}  \\
\text{load} \ \text{rax, \ B + rax}  \\
\text{END:} \]

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

\( \begin{align*}
\text{start;}  \\
\text{pc \ L1}
\end{align*} \)
Observer model: Leakage into μarchitectural state

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
Observer model: Leakage into µarchitectural state

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

load B + A[x]
Observer model: Leakage into μarchitectural state

```
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
Observer model: Leakage into μarchitectural state

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
Observer model: Leakage into μarchitectural state

```
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” programming requirements
Observer model: Leakage into μarchitectural state

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

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

Inspired by “constant-time” programming requirements

No need for detailed model of memory hierarchy:
- possibly pessimistic
- more robust
Reasoning about arbitrary prediction oracles

Speculative semantics + Prediction oracle \rightarrow Always-mispredict speculative semantics
Always-mispredict speculative semantics

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

\begin{align*}
\text{L1: load } & \text{ rax, A + rcx} \\
\text{load } & \text{ rax, B + rax}
\end{align*}

END:
Always-mispredict speculative semantics

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

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

END:

Always mispredict branch instructions’ outcomes
Always-mispredict speculative semantics

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

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

**END:**

Always mispredict branch instructions’ outcomes

Fixed speculative window
Always-mispredict speculative semantics

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

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

\textit{END:}

- Always mispredict branch instructions’ outcomes
- Fixed speculative window
- Rollback of every transaction
Always-mispredict speculative semantics: Inference Rules

**Se-NoBranch**

\[ p(\sigma(\text{pc})) \neq \text{beqz } x, \ell \quad \sigma \xrightarrow{\tau_s} \sigma' \quad \text{enabled}'(s) \]

\[ s' = \begin{cases} 
\text{decr}'(s) & \text{if } p(\sigma(\text{pc})) \neq \text{spbarr} \\
\text{zeroes}'(s) & \text{otherwise}
\end{cases} \]

\[ \langle \text{ctr}, \sigma, s \rangle \xrightarrow{\tau_s} \langle \text{ctr}, \sigma', s' \rangle \]

**Se-Branch-Symb**

\[ p(\sigma(\text{pc})) = \text{beqz } x, \ell'' \quad \text{enabled}'(s) \]

\[ \sigma \xrightarrow{\text{symPc}(\text{se}, \text{pc})} \ell' \quad \sigma' = \begin{cases} 
\sigma(\text{pc}) + 1 & \text{if } \ell' \neq \sigma(\text{pc}) + 1 \\
\ell'' & \text{if } \ell' = \sigma(\text{pc}) + 1
\end{cases} \]

\[ s' = s \cdot \langle \sigma, \text{ctr}, \min(w, \text{wndw}(s) - 1), \ell \rangle \quad \text{id} = \text{ctr} \]

\[ \langle \text{ctr}, \sigma, s \rangle \xrightarrow{\text{symPc}((\text{se}, \text{pc}) \cdot \text{start}) \cdot \text{id} \cdot \text{pc}} \ell' \xrightarrow{s} \langle \text{ctr} + 1, \sigma[\text{pc} \mapsto \ell], s' \rangle \]

**Se-Rollback**

\[ \sigma' \xrightarrow{\tau_s} \sigma'' \]

\[ \langle \text{ctr}, \sigma, s \cdot \langle \sigma', \text{id}, 0, \ell \rangle \rangle \xrightarrow{\text{rollback} \cdot \text{id} \cdot \text{pc} \cdot \sigma''(\text{pc})} \xrightarrow{s} \langle \text{ctr}, \sigma'', s \rangle \]
Always-mispredict leaks maximally

Speculative semantics +
Prediction oracle

Always-mispredict speculative semantics

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

$$P_{spec}(s) = P_{spec}(s')$$

$$\Leftrightarrow \forall O: P_{spec,O}(s) = P_{spec,O}(s')$$
Recap: Speculative non-interference

Program $P$ is **speculatively non-interferent** if

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

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

$\Rightarrow P_{\text{spec}}(s) = P_{\text{spec}}(s')$
Speculative non-interference: Example

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

\[
\begin{align*}
\text{rax} & \leftarrow A\_\text{size} \\
\text{rcx} & \leftarrow x \\
\text{jmp} & \text{ rcx} \geq \text{rax}, \quad \text{END} \\
\text{L1: load} & \quad \text{rax, A } + \text{ rcx} \\
\text{load} & \quad \text{rax, B } + \text{ rax} \\
\text{END:}
\end{align*}
\]
Speculative non-interference: Example

rax ← \( A_{\text{size}} \)
rcx ← \( x \)
jmp rcx ≥ rax, \( END \)

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

\( END: \)
Speculative non-interference: Example

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

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

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

END:
Speculative non-interference: Example

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

\texttt{L1: load} rax, A + rcx
load rax, B + rax
\texttt{END}:
Speculative non-interference: Example

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

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

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

\( END: \)
Speculative non-interference: Example

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

![Diagram showing speculative non-interference example.]
Speculative non-interference: Example

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

```plaintext
x=128
A_size=16
A[128]=0
```

```plaintext
x=128
A_size=16
A[128]=1
```

load B+0
load B+1
Speculative non-interference: Example

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

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

\texttt{END:}
Speculative non-interference: Example

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

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

END:

\begin{align*}
\text{load } B + 0 \\
\text{load } B + 1
\end{align*}
3. Spectector: Detecting speculative leaks
Spectector: Detecting speculative leaks
Spectector: Detecting speculative leaks

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

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

END:
Spectector: Detecting speculative leaks

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

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

END:

Symbolic execution
Spectector: Detecting speculative leaks

rax <- A_size
rcx <- x
jmp rcx >= rax, END
L1: load rax, A + rcx
load rax, B + rax

END:
Spectector: Detecting speculative leaks

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

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

\texttt{END:}
Spectector: Detecting speculative leaks

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

Symbolic execution

Detect leaks

Green tick: No speculative leak detected.
Red cross: Speculative leak detected.

Spectector: Detecting speculative leaks
Spectector: Detecting speculative leaks

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

Symbolic execution

Detect leaks

END:
Symbolic execution
Symbolic execution

• Program analysis technique
Symbolic execution

- Program analysis technique

“The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols”
— James C. King
Symbolic execution

- Program analysis technique
- Execute programs over symbolic values

“The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols”
— James C. King
Symbolic execution

- Program analysis technique
- Execute programs over symbolic values
- Explore all paths, each with its own path constraint

“The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols”
— James C. King
Symbolic execution

- Program analysis technique
- Execute programs over symbolic values
  - Explore all paths, each with its own path constraint
  - Each path represents all possible executions satisfying the constraints

“The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols”
— James C. King
Symbolic execution

• Program analysis technique

• Execute programs over symbolic values
  • Explore all paths, each with its own path constraint
  • Each path represents all possible executions satisfying the constraints
  • Branch and jump instructions: fork paths and update path constraint

“The execution proceeds as in a normal execution except that values may be symbolic formulas over the input symbols”
— James C. King
Symbolic execution

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

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

END:
Symbolic execution

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

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

END:
Symbolic execution

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

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

END:
Symbolic execution

rax $\leftarrow A_{\text{size}}$
rcx $\leftarrow x$

jmp rcx $\geq$ rax, END

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

END:
Symbolic execution

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

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

END:
Symbolic execution

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

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

\[ x \geq A\_size \]
\[ x < A\_size \]
Symbolic execution

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

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

END:
Symbolic execution

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

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

END:
Symbolic execution

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

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

\[ \text{END:} \]

\[ x \geq \text{A\_size} \]
\[ x < \text{A\_size} \]
Symbolic execution

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

\[
\begin{align*}
\text{start; pc L1; load A+x; load B+A[x]; rollback; pc END}
\end{align*}
\]
Symbolic execution

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

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

END:

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

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

\[
\begin{align*}
x \geq A\_size \\
x \lt A\_size
\end{align*}
\]
Detecting speculative leaks

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

Detect leaks

Symbolic execution
Detecting speculative leaks

For each $\tau \in \text{sym-traces}(P)$

- if $\text{MemLeak}(\tau)$ then
  - return INSECURE

- if $\text{CtrlLeak}(\tau)$ then
  - return INSECURE

return SECURE
Detecting speculative leaks

For each $\tau \in \text{sym-traces}(P)$

if $\text{MemLeak}(\tau)$ then

return INSECURE

if $\text{CtrlLeak}(\tau)$ then

return INSECURE

return SECURE
Memory leaks

Speculative memory accesses must depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations
Memory leaks

Speculative memory accesses must depend only on

• Non-sensitive information (determined by policy), or
• be determined by non-speculative observations

\( \tau \)
Memory leaks

Speculative memory accesses \textbf{must} depend only on

- Non-sensitive information (determined by policy), \textit{or}
- be determined by non-speculative observations

\[ \tau \]

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

Speculative memory accesses must depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

$$\tau$$

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

$$S_1$$

$$S_2$$
Memory leaks

Speculative memory accesses must depend only on

- Non-sensitive information (determined by policy), or

- be determined by non-speculative observations

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

\[
S_1 \models \varphi
\]

\[
S_2 \models \varphi
\]
Memory leaks

Speculative memory accesses must depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

\[
\begin{align*}
\tau & \vdash \varphi \\
S_1 & \vdash \varphi \\
S_2 & \vdash \varphi \\
\end{align*}
\]

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

Speculative memory accesses must depend only on

- Non-sensitive information (determined by policy), or
- be determined by non-speculative observations

\[
\tau \models \phi
\]

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

\[
S_1 \models \varphi
\]

\[
S_2 \models \varphi
\]
Memory leaks

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

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

\texttt{END:}

\( \tau = \texttt{start; pc \texttt{L1; load A+x; load B+A[x]; rollback; pc \texttt{END}} \)}
Memory leaks

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

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

\( \text{END:} \)

\[ \tau = \text{start; pc } L1; \text{ load A+x; load B+A[x]; rollback; pc END} \]

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

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

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

END:

\[
\tau = \text{start; pc L1; load A+x; load B+A[x]; rollback; pc END}
\]

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

S1
S2

Policy
x, A_size, A, B are public
Memory leaks

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

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

END:

τ = start; pc L1; load A+x; load B+A[x]; rollback; pc END

pathCnd(τ) ∧ obsEqv(τ|_{non-spec}) ∧ ¬obsEqv(τ|_{spec})

S_1

S_2

x_1 = x_2 ∧ A_size_1 = A_size_2 ∧ A_1 = A_2 ∧ B_1 = B_2

Policy

x, A_size, A, B are public
Memory leaks

rax ← A_size
rcx ← x
jmp rcx≥rax, END

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

END:

τ = start; pc L1; load A+x; load B+A[x]; rollback; pc END

pathCnd(τ) ∧ obsEqv(τ | non-spec) ∧ ¬obsEqv(τ | spec)

S_1 ⊨ x_1≥A_size_1

S_2 ⊨ x_2≥A_size_2

x_1=x_2 ∧ A_size_1=A_size_2 ∧ A_1=A_2 ∧ B_1=B_2
Memory leaks

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

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

END:

τ = start; pc L1; load A+x; load B+A[x]; rollback; pc END

pathCnd(τ) ∧ obsEqv(τ|non-spec) ∧ ¬obsEqv(τ|spec)

S₁ ⊨ x₁ ≥ A_size₁
S₂ ⊨ x₂ ≥ A_size₂

x₁=x₂ ∧ A_size₁=A_size₂ ∧ A₁=A₂ ∧ B₁=B₂

Policy
x, A_size, A, B are public

Always true!
Memory leaks

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

τ = start; pc L1; load A+x; load B+A[x]; rollback; pc END

pathCnd(τ) ∧ obsEqv(τ|_{non-spec}) ∧ ¬obsEqv(τ|_{spec})

S₁ ⊨ x₁ ≥ A_size₁
S₂ ⊨ x₂ ≥ A_size₂
x₁ = x₂ ∧ A_size₁ = A_size₂ ∧ A₁ = A₂ ∧ B₁ = B₂

Always true!

Policy
x, A_size, A, B are public
Memory leaks

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

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

\texttt{END:}

\[ \tau = \text{start; pc } \texttt{L1; load A+x;} \text{ load B+A[x]}; \text{ rollback; pc } \texttt{END} \]

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

\[ S_1 \models x_1 \geq \texttt{A\_size}_1 \]

\[ S_2 \models x_2 \geq \texttt{A\_size}_2 \]

\[ x_1 = x_2 \land \texttt{A\_size}_1 = \texttt{A\_size}_2 \land \texttt{A}_1 = \texttt{A}_2 \land \texttt{B}_1 = \texttt{B}_2 \]

Always true!
Memory leaks

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

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

END:

τ = start; pc L1; load A+x; load B+A[x]; rollback; pc END

pathCnd(τ) ∧ obsEqv(τ|_{non-spec}) ∧ ¬obsEqv(τ|_{spec})

S_1 ⊨ x_1 ≥ A_size_1

S_2 ⊨ x_2 ≥ A_size_2

x_1 = x_2 ∧ A_size_1 = A_size_2 ∧ A_1 = A_2 ∧ B_1 = B_2

Always true!

Policy
x, A_size, A, B are public
Memory leaks

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

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

\[\text{END:}\]

\[\tau = \text{start; pc L1; load A+x; load B+A[x]; rollback; pc END}\]

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

\[S_1 \models x_1 \geq A\_\text{size}_1 \\
S_2 \models x_2 \geq A\_\text{size}_2
\]

\[x_1 = x_2 \land A\_\text{size}_1 = A\_\text{size}_2 \land A_1 = A_2 \land B_1 = B_2\]
Memory leaks

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

\[
\tau = \text{start; pc L1; load A+x; load B+A[x]; rollback; pc END}
\]

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

\[
\begin{align*}
S_1 \models x_1 & \geq \text{A\_size}_1 & \text{pc END} & \text{B}_1 + \text{A}_1[\text{x}_1] \\
S_2 \models x_2 & \geq \text{A\_size}_2 & \text{pc END} & \text{A}_2 + \text{x}_2 & \text{B}_2 + \text{A}_2[\text{x}_2]
\end{align*}
\]

\[
x_1 = x_2 \land \text{A\_size}_1 = \text{A\_size}_2 \land \text{A}_1 = \text{A}_2 \land \text{B}_1 = \text{B}_2
\]

Policy
\[
\text{x, A\_size, A, B are public}
\]

Always true!
Experimental results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Visual C++</th>
<th></th>
<th>ICC</th>
<th></th>
<th>Clang</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unp</td>
<td>Fen</td>
<td>Unp</td>
<td>Fen</td>
<td>Unp</td>
<td>Fen</td>
</tr>
<tr>
<td></td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
<td>-02</td>
</tr>
<tr>
<td>01</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>03</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>04</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>06</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>08</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>09</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Experimental results

<table>
<thead>
<tr>
<th>Spectre variants from Paul Kocher</th>
<th>ICC</th>
<th>Clang</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental results

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

\[ y = B[A[x]*512] \]
### Experimental results

**Spectre variants from Paul Kocher**

<table>
<thead>
<tr>
<th></th>
<th>UNP</th>
<th>FEN</th>
<th>UNP</th>
<th>FEN</th>
<th>UNP</th>
<th>FEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</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>
</tr>
<tr>
<td>03</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>
</tr>
<tr>
<td>05</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>
</tr>
<tr>
<td>07</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>
</tr>
<tr>
<td>09</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>
</tr>
<tr>
<td>11</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>
</tr>
<tr>
<td>13</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>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ y = B[A[x<A\_size?(x+1):0]*512] \]
### Experimental results

#### 15 Spectre variants from Paul Kocher

<table>
<thead>
<tr>
<th></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>01</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>
</tr>
<tr>
<td>03</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>
</tr>
<tr>
<td>05</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>
</tr>
<tr>
<td>07</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>
</tr>
<tr>
<td>09</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>
</tr>
<tr>
<td>11</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>
</tr>
<tr>
<td>13</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>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

**if** \( A[x] == k \)

\( y = B[0] \)
## Experimental results

<table>
<thead>
<tr>
<th>Ex.</th>
<th><strong>VISUAL C++</strong></th>
<th></th>
<th></th>
<th><strong>ICC</strong></th>
<th></th>
<th></th>
<th><strong>CLANG</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>UNP</strong></td>
<td><strong>FEN</strong></td>
<td><strong>UNP</strong></td>
<td><strong>FEN</strong></td>
<td><strong>UNP</strong></td>
<td><strong>FEN</strong></td>
<td><strong>UNP</strong></td>
<td><strong>SLH</strong></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
</tbody>
</table>
## Experimental results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Visual C++</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>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>

---

*Note: The symbols o and o represent different experimental conditions.*
<table>
<thead>
<tr>
<th>Ex.</th>
<th>Visual C++</th>
<th></th>
<th>ICC</th>
<th></th>
<th>Clang</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN</td>
<td>UNP</td>
<td>FEN</td>
<td>UNP</td>
<td>FEN</td>
</tr>
<tr>
<td>01</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>
</tr>
<tr>
<td>03</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>
</tr>
<tr>
<td>05</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>
</tr>
<tr>
<td>07</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>
</tr>
<tr>
<td>09</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>
</tr>
<tr>
<td>11</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>
</tr>
<tr>
<td>13</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>
</tr>
<tr>
<td>15</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
</tbody>
</table>

No countermeasures
## Experimental results

### Automated insertion of fences

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Visual C++</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>01</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>02</td>
<td>-02</td>
<td>-00</td>
<td>-02</td>
</tr>
<tr>
<td>03</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>04</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>05</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>06</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>07</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>08</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>09</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>10</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>11</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>12</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>13</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>14</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
<tr>
<td>15</td>
<td>-00</td>
<td>-00</td>
<td>-00</td>
</tr>
</tbody>
</table>
# Experimental results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Visual C++</th>
<th>ICC</th>
<th>Clang</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP 00</td>
<td>FEN</td>
<td>UNP 00</td>
</tr>
<tr>
<td>01</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>02</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>03</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>04</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>05</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>06</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>07</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>08</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>09</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>10</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>11</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>12</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>13</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>14</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
<tr>
<td>15</td>
<td>•</td>
<td>0</td>
<td>•</td>
</tr>
</tbody>
</table>

Speculative load hardening
## Experimental results

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Visual C++</th>
<th></th>
<th>ICC</th>
<th></th>
<th>Clang</th>
<th></th>
<th>SLH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unp</td>
<td>FEN</td>
<td>Unp</td>
<td>FEN</td>
<td>Unp</td>
<td>FEN</td>
<td>Unp</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</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>
</tr>
</tbody>
</table>
## Experimental results

<table>
<thead>
<tr>
<th>Ex.</th>
<th><strong>VISUAL C++</strong></th>
<th></th>
<th></th>
<th><strong>ICC</strong></th>
<th></th>
<th></th>
<th><strong>CLANG</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>UNP</strong></td>
<td><strong>FEN</strong></td>
<td></td>
<td><strong>UNP</strong></td>
<td><strong>FEN</strong></td>
<td></td>
<td><strong>UNP</strong></td>
<td><strong>FEN</strong></td>
<td><strong>SLH</strong></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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<td>o</td>
</tr>
</tbody>
</table>
Experimental results

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
  - But: Unnecessary fences
- Programs with SLH are secure except #10 and #15
## Experimental results

### Performance

- Programs ~20-200 lines of assembly code
- Analysis terminates in less than **30 sec**
- Except for example #05 (< 2 min)
4. Challenges
Scalable analysis

Goal:
Analysis of large, security-critical applications:
• Intel SGX SDK
• Xen hypervisor
• microkernels

Need: Scalable analysis of speculative non-interference
• Exploit “locality” of speculative execution
• Develop scalable abstractions
Verifying compiler-level countermeasures

mov rax, A_size
mov rcx, x
cmp rcx, rax
jae END
mov rax, A[rax]
shl rax, 9
mov rax, B[rax]

Inserting fences

mov rax, A_size
mov rcx, x
cmp rcx, rax
jae LFENCE
mov rax, A[rcx]
shl rax, 9
or rax, rdx
mov rax, B[rax]

Speculative load hardening

mov rax, A_size
mov rcx, x
mov rdx, 0
cmp rcx, rax
jae END
cmovae -1, rdx
mov rax, A[rcx]
shl rax, 9
or rax, rdx
mov rax, B[rax]

How can we verify such countermeasures?
A sound HW/SW security contract

Instruction-set architecture: too weak for security guarantees

Microarchitecture: not available publicly, and too detailed for analysis
A sound HW/SW security contract

Instruction-set architecture: too weak for security guarantees

Microarchitecture: not available publicly, and too detailed for analysis
Find out more in the paper:
https://arxiv.org/abs/1812.08639

Find out more in the paper: https://arxiv.org/abs/1812.08639


I am looking for PhD students and postdocs!
Find out more in the paper:
https://arxiv.org/abs/1812.08639


I am looking for PhD students and postdocs!

Thank you for your attention!
Backup
Example #01 - SLH

```java
if (x < A_size)
    y = B[A[x]*512]
```
Example #01 - SLH

```c
if (x < A_size)  
y = B[A[x]*512]

mov    rax, A_size
mov    rcx, x
mov    rdx, 0
cmp    rcx, rax
jae    END
cmovae -1, rdx
mov    rax, A[rcx]
shl    rax, 9
or     rax, rdx
mov    rax, B[rax]
```
Example #01 - SLH

if \((x < A_{\text{size}})\)
\[
y = B[A[x] \times 512]
\]

mov rax, A_{\text{size}}
mov rcx, x
mov rdx, 0
cmp rcx, rax
jae END
jae cmovae -1, rdx
mov rax, A[rcx]
shl rax, 9
or rax, rdx
mov rax, B[rax]

rax is -1 whenever \(x \geq A_{\text{size}}\)
We can prove security
Example #10 - SLH

if \( x < A_{\text{size}} \)
  if \( A[x] == 0 \)
    y = B[0]
Example #10 - SLH

\[
\text{if } (x < A_{\text{size}}) \\
\quad \text{if } (A[x] == 0) \\
\quad \quad y = B[0]
\]

\[
\begin{align*}
\text{mov} & \quad rax, A_{\text{size}} \\
\text{mov} & \quad rcx, x \\
\text{mov} & \quad rdx, 0 \\
\text{cmp} & \quad rcx, rax \\
\text{jae} & \quad END \\
\text{cmovae} & \quad -1, rdx \\
\text{mov} & \quad rax, A[rcx] \\
\text{jne} & \quad rax, END \\
\text{cmovne} & \quad -1, rdx \\
\text{mov} & \quad rax, [B]
\end{align*}
\]
Example #10 - SLH

```assembly
if (x < A_size)
  if (A[x] == 0)
    y = B[0]

mov rax, A_size
mov rcx, x
mov rdx, 0
cmp rcx, rax
jae END

lea rdx, [rip+1]
cmovae -1, rdx

mov rax, A[rcx]

jne rax, END

cmovne -1, rdx

mov rax, [B]
```

Leaks $A[x] == 0$ via control-flow

We detect the leak!
Example #08 - FEN

\[ y = B[A[x<A\_size?(x+1):0]*512] \]
Example #08 - FEN

\[ y = B[A[x<A_size?(x+1):0]*512] \]

```
mov rax, A_size
mov rcx, x
lea rcx, [rcx+1]
xor rdx, rdx
cmp rcx, rax
cmovae rdx, rcx
mov rax, A[rdx]
shl rax, 9
lfence
mov rax, B[rax]
```
Example #08 - FEN

\[ y = B[A[x<A_size??(x+1):0]*512] \]

```assembly
mov rax, A_size
mov rcx, x
lea rcx, [rcx+1]
xor rdx, rdx
cmp rcx, rax
cmovae rdx, rcx
mov rax, A[rdx]
shl rax, 9
lfence
mov rax, B[rax]
```

 lfence is unnecessary