Spectector: Principled detection of speculative information flows

Jan Reineke
Saarland University

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”
Exploits speculative execution to leak sensitive information

SPECTRE

Exploits speculative execution to leak sensitive information

Almost all modern CPUs 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

“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

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

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

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

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

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

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

Predictions based on

*branch history* &
*program structure*
Background: Branch prediction

```latex
\text{if} \ (x < A_{\text{size}}) \quad 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 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

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

What is in A[128]?
void f(int x)
  if (x < A_size)
    y = B[A[x]]

What is in A[128]?

1) Training

A_size=16

B[0] B[1] ...

What is in A[128]?
void f(int x)
if (x < A_size)
y = B[A[x]]

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

What is in A[128]?

1a) Training

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

What is in A[128]?

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

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

A_size = 16
B = [B[0], B[1], ..., B[15]]

What is in A[128]?

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

1b) Prepare cache

Cache state
void \textbf{f}(\textbf{int} \ x) \\
\textbf{if} (x < \textbf{A}_{\text{size}}) \\
y = \textbf{B}[\textbf{A}[x]]
void \( f(\text{int } x) \)  
if \( x < A_{\text{size}} \)  
y = B[A[x]]

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

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

1b) Prepare cache

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

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

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

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

1b) Prepare cache

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

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

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

1b) Prepare cache

2) Run \( f(x) \) with \( x = 128 \)
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(x) with x = 128

What is in A[128]?

Cache state
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 $f(x)$ with $x = 128$
Spectre V1

Address depends on $A[128]$.

$A_{\text{size}} = 16$

Cache state

```
```

What is in $A[128]$?

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

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

1b) Prepare cache

2) Run $f(x)$ with $x = 128$

Persistent beyond rollback
void \(f(int \ x)\)
if (\(x < A\_size\))
\(y = B[A[x]]\)

Address depends on \(A[128]\)

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

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

1b) Prepare cache

2) Run \(f(x)\) with \(x = 128\)

3) Extract from cache

Persistently beyond rollback

Cache state

\(A\_size = 16\)
2. Speculative non-interference
Generalizing the Spectre V1 example

1a) Training

1b) Prepare cache

2) Run $f(x)$ with $x = 128$

3) Extract from cache
Generalizing the Spectre V1 example

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

1b) Prepare cache

2) Run \( f(x) \) with \( x = 128 \)

3) Extract from cache
Generalizing the Spectre V1 example

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

1b) Prepare cache

2) Run \( f(x) \) with \( x = 128 \)

3) Extract from cache
Generalizing the Spectre V1 example

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

1b) Prepare cache

2) Run $f(x)$ with $x = 128$

3) Extract from cache
Generalizing the Spectre V1 example

1) Prepares microarchitectural state

2) Leaks information into microarchitectural state

3) Extracts information from microarchitecture
Speculative non-interference
Speculative non-interference

Program $P$ is **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

$=$
Speculative non-interference

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

\textit{Informally:}

Leakage of $P$ in \textbf{non-speculative execution} $\ ? $ Leakage of $P$ in \textbf{speculative execution}

\textit{More formally:}
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 \( s' \):

\[ ? \]
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 $s'$:

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

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

Non-speculative semantics

Speculative semantics

Attacker/Observer model
μAssembly

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

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

\( L1: \) 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

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

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

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

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

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

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

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

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

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

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

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

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

```plaintext
rax <- A\_size
rcx <- x
jmp rcx >= rax, \text{ END}
L1: load rax, A + rcx
load rax, B + rax
END:
```
μAssembly + Non-speculative semantics

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

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

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

\[
\text{END:}
\]
Speculative semantics

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

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

\texttt{END:}
Speculative semantics

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

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

\textit{END:}

\textit{Starts 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 ← A\_size
rcx ← x
jmp rcx ≥ rax, END

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

\textbf{END:}

Starts \textbf{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 >= 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 \leftarrow A\_size
rcx \leftarrow 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 <- \texttt{A\_size}
rcx <- \texttt{x}
jmp rcx \geq rax, \texttt{END}

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

\texttt{END:}

Starts \textit{speculative transactions} upon branches

Committed upon correct speculation
Rolled back upon misspeculation
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:}

\textbf{Starts speculative transactions} upon branches

Committed upon correct speculation

Rolled back upon misspeculation
Speculative semantics

rax ← \texttt{A}_{\text{size}}
rcx ← x
jmp rcx ≥ rax, END

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

```plaintext
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

**Prediction Oracle O** determines branch prediction + length of speculative window
Observer model: Leakage into µarchitectural state

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

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

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

END:

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

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

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

END:

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

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

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

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

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, \text{END}
```

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

\text{END:}

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

\text{start; pc L1}
Observer model: Leakage into μarchitectural state

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

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

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

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

\textit{END:}

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

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

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

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

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

\textbf{L1:} load rax, A + rcx
load rax, 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” requirements
Observer model: Leakage into μarchitectural state

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

L1: load rax, A + rcx
load rax, 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” 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

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

\[
\begin{align*}
L1: \quad \text{load} & \quad r ax, \quad A + rcx \\
\text{load} & \quad r ax, \quad B + r ax
\end{align*}
\]

\[
\begin{align*}
\text{END:}
\end{align*}
\]
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

rax <- $A_{size}$  
rcx <- $x$

jmp rcx$\geq$rax, END

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

END:

Always mispredict branch instructions’ outcomes

Fixed speculative window
Always-mispredict speculative semantics

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

Always mispredict branch instructions’ outcomes

Fixed speculative window

Rollback of every transaction
Always-mispredict leaks maximally

Speculative semantics
+ Prediction oracle

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

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

\[ \Leftrightarrow \forall O: P_{\text{spec},0}(s) = P_{\text{spec},0}(s') \]
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

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

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

END:
Speculative non-interference: Example

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

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

END:
Speculative non-interference: Example

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

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

END:
Speculative non-interference: Example

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

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

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

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

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

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

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

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

\begin{align*}
\text{L1: load } & \texttt{rax, A + rcx} \\
\text{load } & \texttt{rax, B + rax} \\
\texttt{END:}
\end{align*}
Speculative non-interference: Example

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

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

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

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

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

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

\text{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 the speculative execution process with load operations for B+0 and B+1 and memory accesses to A at index 128 with values 0 and 1.]
Speculative non-interference: Example

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

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

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

END:

\begin{align*}
\text{END:} \\
\text{load } B + 0 \\
\text{load } B + 1 \\
x = 128 \\
A_{\text{size}} = 16 \\
A[128] = 0 \\
x = 128 \\
A_{\text{size}} = 16 \\
A[128] = 1
\end{align*}
3. Spectector: Detecting speculative leaks
Spectector: Detecting speculative leaks
Spectector: Detecting speculative leaks

```
rax <- A_size
rcx <- x
jmp rcx \geq 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:
Spectector: Detecting speculative leaks

```c
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}_{\texttt{size}}
rcx <- x
jmp rcx \geq \texttt{rax}, END
L1: load rax, A + rcx
load rax, B + rax
END:
Specetector: 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

Spectector: Detecting speculative leaks

`load rax, A + rcx`

`load rax, B + rax`
Spectector: Detecting speculative leaks

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

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

\texttt{END:}

Detect leaks

Symbolic execution
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 <- A_size
rcx <- x
jmp rcx≥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

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}: true
Symbolic execution

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

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

END:
Symbolic execution

rax <- $A_{\text{size}}$
rcx <- $x$

jmp rcx ≥ rax, **END**

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

END:
Symbolic execution

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

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

\texttt{END:}

x \geq \texttt{A\_size}

x < \texttt{A\_size}
Symbolic execution

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

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

END:
Symbolic execution

rax <- A_size
rcx <- x
jmp rcx\geq rax, END

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

END:

\( x \geq A_{\text{size}} \)
\( x < A_{\text{size}} \)
Symbolic execution

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

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

END:
Symbolic execution

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

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

END:

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

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
Detecting speculative leaks

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:}
Detecting speculative leaks

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

- if $\text{Mem Leak}(\tau)$ then
  - return INSECURE
- if $\text{Ctrl Leak}(\tau)$ 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
Memory leaks

Speculative memory accesses must depend only on

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

\[ \tau \]

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

Speculative memory accesses **must** depend only on

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

\[
\tau, \quad \text{where} \quad \tau \neq \text{spec}
\]

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

\[
S_1, \quad S_2
\]
Memory leaks

Speculative memory accesses must depend only on

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

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

\[
S_1 \models \varphi \\
S_2 \models \varphi
\]
Memory leaks

Speculative memory accesses must depend only on

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

\[ \tau \]

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

\[ s_1 \models \varphi \]
\[ s_2 \models \varphi \]
Memory leaks

Speculative memory accesses must depend only on

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

\[
\tau \vdash \varphi
\]

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

\[
S_1 \models \varphi
\]

\[
S_2 \models \varphi
\]
Memory leaks

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

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

\[ \text{END:} \]

\[ \tau = \text{start; pc L1; load A+x; load B+A[x]; rollback; pc END} \]
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 obsEqv(\(\tau|_{non-spec}\)) \land \neg obsEqv(\(\tau|_{spec}\))} \]
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
S2

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

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

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

\text{END:}

\text{Policy}

\text{x, A}_\text{size}, \ A, \ B \ \text{are public}

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

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

\[S_1 \land S_2 \land x_1 = x_2 \land A\text{size}_1 = A\text{size}_2 \land A_1 = A_2 \land B_1 = B_2\]
Memory leaks

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

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

\textbf{END}:

 Policy \(x, \text{A\_size}, A, B\) are public

\[\begin{align*}
\tau = & \text{start; pc } L1; \text{ load } A+x; \text{ load } B+A[x]; \text{ rollback; pc END} \\
\text{pathCnd}(\tau) & \land \text{obsEqv}(\tau|_{\text{non-spec}}) \land \neg\text{obsEqv}(\tau|_{\text{spec}}) \\
S_1 & \models x_1\geq\text{A\_size}_1 \\
S_2 & \models x_2\geq\text{A\_size}_2 \\
x_1=x_2 & \land \text{A\_size}_1=\text{A\_size}_2 \land A_1=A_2 \land B_1=B_2
\end{align*}\]
Memory leaks

rax ← A_size
rcx ← x
jmp rcx≥rax, END

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

END:

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

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

\[ S_1 \models x_1 \geq A_{-size_1} \]

\[ S_2 \models x_2 \geq A_{-size_2} \]

\[ x_1 = x_2 \land A_{-size_1} = A_{-size_2} \land A_1 = A_2 \land B_1 = B_2 \]

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})

S_1 \models x_1 ≥ A\_size_1
S_2 \models 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

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

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

\begin{align*}
S_1 & \models x_1 \geq 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
\end{align*}

Always true!
Memory leaks

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

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

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

\[
S_1 \models x_1 \geq A\_size_1
\]

\[
S_2 \models x_2 \geq A\_size_2
\]

\[
x_1 = x_2 \land A\_size_1 = A\_size_2 \land A_1 = A_2 \land B_1 = B_2
\]

Always true!

Policy

\[ x, A\_size, A, B \]
are public
Memory leaks

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

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

\[ \text{END:} \]

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

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

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

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

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

Always true!
Memory leaks

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

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

\[ \text{END:} \]

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

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

\[ S_1 \models x_1 \geq A_{\text{size}_1} \]
\[ \text{pc END} \]
\[ A_1 + x_1 \]
\[ B_1 + A_1[x_1] \]

\[ S_2 \models x_2 \geq A_{\text{size}_2} \]
\[ \text{pc END} \]
\[ A_2 + x_2 \]
\[ B_2 + A_2[x_2] \]

\[ x_1 = x_2 \land A_{\text{size}_1} = A_{\text{size}_2} \land A_1 = A_2 \land B_1 = B_2 \]

Always true!

Policy
\[ x, A_{\text{size}}, A, B \]
are public
# Experimental results

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

32
## Experimental results

<table>
<thead>
<tr>
<th></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>o</td>
<td>o</td>
</tr>
<tr>
<td>02</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>03</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>04</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>05</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>06</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>07</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>08</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>09</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>10</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>11</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>12</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>13</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>14</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>15</td>
<td>o</td>
<td>o</td>
</tr>
</tbody>
</table>

15 Spectre variants from Paul Kocher
Experimental results

<table>
<thead>
<tr>
<th></th>
<th>ICC 00</th>
<th>ICC 02</th>
<th>CLANG 00</th>
<th>CLANG 02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre variants from Paul Kocher</td>
<td>![Table with data points]</td>
<td>![Table with data points]</td>
<td>![Table with data points]</td>
<td>![Table with data points]</td>
</tr>
</tbody>
</table>

- **15** if \((x < A\_size)\)
  
  \(y = B[A[x] \times 512]\)
Experimental results

<table>
<thead>
<tr>
<th>ICC</th>
<th>Clang</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unp</td>
<td></td>
</tr>
<tr>
<td>Fen</td>
<td>Unp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></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>

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

<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>

15 Spectre variants from Paul Kocher

\[
\text{if } (x < A\text{\_size}) \\
\quad \text{if } (A[x] == k) \\
\quad y = B[0]
\]
## 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>FEN</strong></td>
<td><strong>SLH</strong></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>○</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>
</tr>
<tr>
<td>03</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>
</tr>
<tr>
<td>05</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>
</tr>
<tr>
<td>07</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>
</tr>
<tr>
<td>09</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>
</tr>
<tr>
<td>11</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>
</tr>
<tr>
<td>13</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>
</tr>
<tr>
<td>15</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</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></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>

32
## 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>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UNP</td>
<td>FEN</td>
<td></td>
<td>UNP</td>
<td>FEN</td>
<td></td>
<td>UNP</td>
<td>FEN</td>
<td></td>
<td>UNP</td>
<td>FEN</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<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>○</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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>
<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>-02</td>
<td>-00</td>
</tr>
<tr>
<td>02</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>03</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>04</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>05</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>06</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>07</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>08</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>09</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>10</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>11</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>12</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>13</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>14</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
<tr>
<td>15</td>
<td>-00</td>
<td>-02</td>
<td>-00</td>
</tr>
</tbody>
</table>
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 -00</td>
<td>-02</td>
<td>UNP</td>
<td>-00</td>
<td>UNP</td>
<td>-02</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>

Speculative load hardening
## Experimental results

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

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

<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></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>

**Performance**

- Programs \(\sim 20-200\) lines of assembly code
- Analysis terminates in less than 30 sec
- Except for example #05 (< 2 min)
4. Challenges
A sound HW/SW security contract

Instruction-set architecture: to weak for security guarantees

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

Instruction-set architecture: to weak for security guarantees

Microarchitecture: not available publicly, and too detailed for analysis
Verifying compiler-level countermeasures

How can we specify and verify such countermeasures?
Thank you for your attention!

Find out more in the paper:
https://arxiv.org/abs/1812.08639
Backup
Example #01 - SLH

\[
\begin{align*}
\text{if} \ (x &< \text{A\_size}) \\
\quad y &= B[A[x]*512]
\end{align*}
\]
Example #01 - SLH

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

\[y = B[A[x] \times 512]\]

```assembly
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

\[
\text{if } (x < A\_size) \\
y = B[A[x]*512]
\]

```assembly
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]
```

rax is -1 whenever \( x \geq A\_size \)
We can prove security
Example #10 - SLH

```c
if (x < A_size)
    if (A[x] == k)
        y = B[0]
```
Example #10 - SLH

\[
\text{if (} x < A\_size \text{)} \\
\quad \text{if (} A[x] == k \text{)} \\
\hspace{1em} y = B[0]
\]

```assembly
mov      rax, A\_size
mov      rcx, x
mov      rdx, 0
cmp      rcx, rax
jae      END
cmovae   -1, rdx
mov      rax, A[rcx]
jne      rax, END
cmovne   -1, rdx
mov      rax, [B]
```
Example #10 - SLH

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

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

Leaks A[x] == 0 via control-flow
We detect the leak!
```

```
cmovae  -1, rdx
mov     rax, A[rcx]
jae     END
jne     rax, END

mov     rax, [B]
```
Example #08 - FEN

\[ y = B[A[x<A_{size}?(x+1):0] \times 512] \]
Example #08 - FEN

\[ y = \text{B}[A[x<A_{size}?(x+1):0]*512] \]

\[
\begin{align*}
\text{mov} & \quad \text{rax, A}_{size} \\
\text{mov} & \quad \text{rcx, x} \\
\text{lea} & \quad \text{rcx, [rcx+1]} \\
\text{xor} & \quad \text{rdx, rdx} \\
\text{cmp} & \quad \text{rcx, rax} \\
\text{cmovae} & \quad \text{rdx, rcx} \\
\text{mov} & \quad \text{rax, A}[\text{rdx}] \\
\text{shl} & \quad \text{rax, 9} \\
\text{lfence} & \\
\text{mov} & \quad \text{rax, B}[\text{rax}] 
\end{align*}
\]
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]
lfence is unnecessary