On Kernel Safety and Speculative Execution
Work in progress

Davide Davoli\textsuperscript{1,2} Tamara Rezk\textsuperscript{2} Martin Avanzini\textsuperscript{2}

\textsuperscript{1}Université Côte d’Azur
\textsuperscript{2}INRIA

3\textsuperscript{rd} April 2024 – Annual Meeting of the WG “Formal Methods in Security”
Layout randomization is a software mechanism to enforce memory safety and control flow integrity.
Address Space Layout Randomization

Layout randomization is a software mechanism to enforce memory safety and control flow integrity.

Memory
Address Space Layout Randomization

Layout randomization is a software mechanism to enforce memory safety and control flow integrity.
Address Space Layout Randomization

Layout randomization is a software mechanism to enforce memory safety and control flow integrity.

Key idea: attackers do not know where things are stored.
Address Space Layout Randomization

Layout randomization is a software mechanism to enforce memory safety and control flow integrity.
Address Space Layout Randomization

Layout randomization is a software mechanism to enforce *memory safety* and *control flow integrity*.

**Key idea:** attackers do not know where things are stored.
Open Problems, Research Questions and Contributions

Open Problems:

▶ No formal study of Layout Randomization for kernel memory (KASLR).

Research questions and contributions:

▶ Is KASLR effective without side-channels and speculative execution? Yes.

▶ Is it effective against side-channels and speculative execution? Not really...

▶ Can we do something else? Yes.
Open Problems, Research Questions and Contributions

Open Problems:

- No formal study of Layout Randomization for kernel memory (KASLR).
- Attacks to KASLR based on:

Research questions and contributions:

- Is KASLR effective without side-channels and speculative execution? Yes.
- Is it effective against side-channels and speculative execution? Not really...
- Can we do something else? Yes.
Open Problems:

- No formal study of Layout Randomization for kernel memory (KASLR).
- Attacks to KASLR based on: side-channels
Open Problems, Research Questions and Contributions

Open Problems:

▶ No formal study of Layout Randomization for kernel memory (KASLR).
▶ Attacks to KASLR based on: side-channels, speculative execution.
Open Problems, Research Questions and Contributions

Open Problems:

- No formal study of Layout Randomization for kernel memory (KASLR).
- Attacks to KASLR based on: side-channels, speculative execution.

Research questions and contributions:

- Is KASLR effective without side-channels and speculative execution?
Open Problems, Research Questions and Contributions

Open Problems:

- No formal study of Layout Randomization for kernel memory (KASLR).
- Attacks to KASLR based on: side-channels, speculative execution.

Research questions and contributions:

- Is KASLR effective without side-channels and speculative execution? Yes.
Open Problems, Research Questions and Contributions

Open Problems:

▶ No formal study of Layout Randomization for kernel memory (KASLR).
▶ Attacks to KASLR based on: side-channels, speculative execution.

Research questions and contributions:

▶ Is KASLR effective without side-channels and speculative execution? Yes.
▶ Is it effective against side-channels and speculative execution?
Open Problems, Research Questions and Contributions

Open Problems:

▸ No formal study of Layout Randomization for kernel memory (KASLR).

▸ Attacks to KASLR based on: side-channels, speculative execution.

Research questions and contributions:

▸ Is KASLR effective without side-channels and speculative execution? Yes.

▸ Is it effective against side-channels and speculative execution? Not really...
Open Problems, Research Questions and Contributions

Open Problems:

▶ No formal study of Layout Randomization for kernel memory (KASLR).
▶ Attacks to KASLR based on: side-channels, speculative execution.

Research questions and contributions:

▶ Is KASLR effective without side-channels and speculative execution? Yes.
▶ Is it effective against side-channels and speculative execution? Not really...
▶ Can we do something else?
Open Problems, Research Questions and Contributions

**Open Problems:**

▶ No formal study of Layout Randomization for kernel memory (KASLR).
▶ Attacks to KASLR based on: side-channels, speculative execution.

**Research questions and contributions:**

▶ Is KASLR effective without side-channels and speculative execution? Yes.
▶ Is it effective against side-channels and speculative execution? *Not really*...
▶ Can we do something else? Yes.
Kernel’s Execution Model

- Privilege level: user or kernel
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
- Attacker: *user-space* program.
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
- Attacker: *user-space* program.

```c
void A (void){
    x = get_uid();    //syscall
    y = f(x);         //ordinary call
    y = y + z;
    print(y);         //syscall
}
```
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
- Attacker: *user-space* program.

```c
void A (void){
    x = get_uid();  //syscall
    y = f(x);       //ordinary call
    y = y + z;
    print(y);       //syscall
}
```
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
- Attacker: *user-space* program.

```c
void A (void){
    x = get_uid();  //syscall
    y = f(x);      //ordinary call
    y = y + z
    print(y);      //syscall
}
```
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
- Attacker: *user-space* program.

```c
void A (void){
    x = get_uid();  //syscall
    y = f(x);      //ordinary call
    y = y + z
    print(y);      //syscall
}
```
Kernel’s Execution Model

- Privilege level: user or kernel
- Syscalls change privilege level
- Disjoint address spaces
- Attacker: *user-space* program.

```c
void A (void) {
    x = get_uid() //syscall
    y = f(x)     //ordinary call
    y = y + z    //ordinary call
    print(y)     //syscall
}
```
Kernel Safety (no side-channels and speculative execution)

For every collection of system calls $\gamma$:

$$\text{KASLR} \land \text{LNI}(\gamma) \Rightarrow \text{probabilistic safety}$$
Kernel Safety (no side-channels and speculative execution)

For every collection of system calls \( \gamma \):

\[
\text{KASLR} \land \text{LNI}(\gamma) \Rightarrow \text{probabilistic safety}
\]

\[
\text{LNI}(\gamma) := \text{the semantics of the syscalls in } \gamma \text{ must not depend on the layout}
\]
Kernel Safety (no side-channels and speculative execution)

For every collection of system calls $\gamma$:

$$\text{KASLR} \land \text{LNI}(\gamma) \implies \text{probabilistic safety}$$

$$\text{LNI}(\gamma) := \text{the semantics of the syscalls in } \gamma \text{ must not depend on the layout}$$

<table>
<thead>
<tr>
<th>Victim</th>
<th>LNI</th>
</tr>
</thead>
<tbody>
<tr>
<td>return $p$</td>
<td>no</td>
</tr>
</tbody>
</table>
KASLR in the presence of *Side-Channel Attackers*

Instructions leak information on the layout.
KASLR in the presence of *Side-Channel* Attackers

Instructions leak information on the layout.

<table>
<thead>
<tr>
<th>Victim</th>
<th>Leaked information</th>
</tr>
</thead>
</table>

\[
\text{Victim} \quad \text{Leaked information}
\]
KASLR in the presence of *Side-Channel Attackers*

Instructions leak information on the layout.

<table>
<thead>
<tr>
<th>Victim</th>
<th>Leaked information</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>0xf1a34 = x;</em></td>
<td>data_op 0xf1a34</td>
</tr>
</tbody>
</table>
KASLR in the presence of *Side-Channel Attackers*

Instructions leak information on the layout.

<table>
<thead>
<tr>
<th>Victim</th>
<th>Leaked information</th>
</tr>
</thead>
<tbody>
<tr>
<td>*0xf1a34 = x;</td>
<td>data_op 0xf1a34</td>
</tr>
<tr>
<td>x = *0xf1a34;</td>
<td>data_op 0xf1a34</td>
</tr>
</tbody>
</table>
KASLR in the presence of *Side-Channel Attackers*

Instructions leak information on the layout.

<table>
<thead>
<tr>
<th>Victim</th>
<th>Leaked information</th>
</tr>
</thead>
<tbody>
<tr>
<td>*0xf1a34 = x;</td>
<td>data_op 0xf1a34</td>
</tr>
<tr>
<td>x = *0xf1a34;</td>
<td>data_op 0xf1a34</td>
</tr>
<tr>
<td>(*0xf1a34)();</td>
<td>jump 0xf1a34</td>
</tr>
</tbody>
</table>
KASLR in the presence of *Side-Channel Attackers*

Instructions leak information on the layout.

<table>
<thead>
<tr>
<th>Victim</th>
<th>Leaked information</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ast 0xf1a34 = x; )</td>
<td>data_op 0xf1a34</td>
</tr>
<tr>
<td>( x = \ast 0xf1a34; )</td>
<td>data_op 0xf1a34</td>
</tr>
<tr>
<td>(( \ast 0xf1a34)();</td>
<td>jump 0xf1a34</td>
</tr>
</tbody>
</table>

To restore the protection offered by KASLR, LNI needs to be strengthened:
KASLR in the presence of *Side-Channel* Attackers

Instructions leak information on the layout.

<table>
<thead>
<tr>
<th>Victim</th>
<th>Leaked information</th>
</tr>
</thead>
<tbody>
<tr>
<td>(*0xf1a34 = x;)</td>
<td>data_op 0xf1a34</td>
</tr>
<tr>
<td>(x = *0xf1a34;)</td>
<td>data_op 0xf1a34</td>
</tr>
<tr>
<td>((*0xf1a34)();)</td>
<td>jump 0xf1a34</td>
</tr>
</tbody>
</table>

To restore the protection offered by KASLR, LNI needs to be strengthened:

\[
\text{LNI}(\gamma) := \text{the semantics of the syscalls in } \gamma \text{ must not depend on the layout}
\]
KASLR in the presence of *Side-Channel* Attackers

Instructions leak information on the layout.

\[
\begin{array}{ll}
\text{Victim} & \text{Leaked information} \\
*0xf1a34 = x; & \text{data\_op } 0xf1a34 \\
x = *0xf1a34; & \text{data\_op } 0xf1a34 \\
(*0xf1a34)(); & \text{jump } 0xf1a34 \\
\end{array}
\]

To restore the protection offered by KASLR, LNI needs to be strengthened:

\[\text{LNI}(\gamma) := \text{the semantics and the side-channel leaks of the syscalls in } \gamma \text{ must not depend on the layout}\]
KASLR in presence of Speculative Attackers

The system call $s$ is a threat in presence of speculative attackers (BlindSide), and we can model it.

```c
void s(x, y){
    if(x)
        (*y)(x);
}
```
KASLR in presence of Speculative Attackers

The system call s is a threat in presence of speculative attackers (BlindSide), and we can model it.

\[
\text{for}(i = 0; i < 8; i++)\
\quad \text{force(true);}\
\quad s(false, i);\
\]

\[
\text{void s}(x, y)\
\quad \text{if}(x)\
\quad \quad (*y)(x);\
\]

Randomized Kernel Memory

0: 
1: 
2: 
3: 
4: lambda 
5: 
6: 
7: 
KASLR in presence of Speculative Attackers

The system call \( s \) is a threat in presence of speculative attackers (BlindSide), and we can model it.

```
for(i = 0; i < 8; i++){
    force(true);
    s(false, i);
}
void s(x, y){
    if(x)
        (*y)(x);
}
```
KASLR in presence of *Speculative Attackers*

The system call \( s \) is a threat in presence of speculative attackers (BlindSide), and we can model it.

\[
\rightarrow \quad \text{for}(i = 0; i < 8; i + +)\{
\quad \text{force}(\text{true});
\quad s(\text{false}, i);
\}
\]

\[
\text{void \( s \) (\( x, y \))}\{
\quad \text{if}(x)
\quad \quad \text{(*y)}(x);
\}
\]

Randomized Kernel Memory

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>( \lambda x \ldots )</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
KASLR in presence of *Speculative Attackers*

The system call \( s \) is a threat in presence of speculative attackers (BlindSide), and we can model it.

\[
\text{for}(i = 0; i < 8; i++)\
\rightarrow \quad \text{force(true);}\
\quad s(\text{false}, i);\
\]

\[
\text{void } s(x, y)\
\quad \text{if}(x)\
\quad \quad (*y)(x);\
\]

Randomized Kernel Memory

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>( \lambda x \ldots )</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
KASLR in presence of *Speculative Attackers*

The system call \texttt{s} is a threat in presence of speculative attackers (BlindSide), and we can model it.

\[
\text{for}(i = 0; i < 8; i++)\{
    \text{force}(\text{true});
    \text{s}(\text{false}, i);
}\]

\[
\text{void } \text{s}(x, y)\{
    \text{if}(x)\{
        (*y)(x);
    \}
\}
\]

Randomized Kernel Memory

\[
\begin{array}{c}
0: \_\_\_\_\_\_\_\_\_\_ \\
1: \_\_\_\_\_\_\_\_\_ \\
2: \_\_\_\_\_\_\_\_\_ \\
3: \_\_\_\_\_\_\_\_\_ \\
4: \_\_\_\_\_\_\_\_\_ \\
5: \_\_\_\_\_\_\_\_\_ \\
6: \_\_\_\_\_\_\_\_\_ \\
7: \_\_\_\_\_\_\_\_\_ \\
\end{array}
\]
KASLR in presence of *Speculative Attackers*

The system call \( s \) is a threat in presence of speculative attackers (BlindSide), and we can model it.

\[
\text{for}(i = 0; i < 8; i++)\{
    \text{force}(\text{true});
    s(\text{false}, i);
}\]

\[
\text{void } s(x, y)\{
    \text{if}(x)
    \quad \rightarrow \quad (*y)(x);
}\]

Randomized Kernel Memory

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>lambdax...</td>
<td></td>
</tr>
<tr>
<td>5:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
KASLR in presence of Speculative Attackers

The system call $s$ is a threat in presence of speculative attackers (BlindSide), and we can model it.

```c
for(i = 0; i < 8; i++){
    force(true);
    s(false, i);
}
void s(x, y){
    if(x)
        (*y)(x);
}
```

Randomized Kernel Memory

```
0: [ ]
1: [ ]
2: [ ]
3: [ ]
4: [ ]
5: [ ]
6: [ ]
7: [ ]
```

\[\lambda x.\ldots\]
KASLR in presence of Speculative Attackers

The system call $s$ is a threat in presence of speculative attackers (BlindSide), and we can model it.

for($i = 0; i < 8; i++$)
    force(true);
    s(false, i);
}

void s(x, y){
    if(x)
        (*y)(x);
    }

Randomized Kernel Memory

\[
\begin{array}{c}
0: \\
1: \\
2: \\
3: \\
4: \lambda x.. \\
5: \\
6: \\
7: \\
\end{array}
\]
KASLR in presence of Speculative Attackers

The system call $s$ is a threat in presence of speculative attackers (BlindSide), and we can model it.

$\rightarrow$  \hspace{1cm} \textbf{for}(i = 0; i < 8; i++){
     \hspace{1cm} \textbf{force}(\text{true});
     \hspace{1cm} s(\text{false}, i);
  }

\textbf{void} s(x, y){
  \hspace{1cm} \textbf{if}(x)
      \hspace{1cm} (*y)(x);
  }

Randomized Kernel Memory

\begin{tabular}{c|c|c|c|c|c|c}
  0: & & & & & & \\
  1: & & & & & & \\
  2: & & & & & & \\
  3: & & & & & & \\
  4: & & & & & & $\lambda x$ \\
  5: & & & & & & \\
  6: & & & & & & \\
  7: & & & & & & \\
\end{tabular}
KASLR in presence of *Speculative Attackers*

The system call \( s \) is a threat in presence of speculative attackers (BlindSide), and we can model it.

\[
\text{for} (i = 0; i < 8; i++) \\
\quad \text{force} (\text{true}); \\
\quad s (\text{false}, i); \\
\}
\]

\[
\text{void } s (x, y) \\
\quad \text{if} (x) \\
\quad \quad (*y)(x); \\
\}
\]

Randomized Kernel Memory

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>λx...</th>
<th>←</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
KASLR in presence of *Speculative Attackers*

The system call `s` is a threat in presence of speculative attackers (BlindSide), and we can model it.

```c
for(i = 0; i < 8; i ++)
{
    force(true);
    s(false, i);
}
void s(x, y){
    if(x)
        (*y)(x);
}
```

Randomized Kernel Memory

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 7 |   |   |   |   |   |   | λx... | ←
KASLR in presence of *Speculative Attackers*

The system call \( s \) is a threat in presence of speculative attackers (BlindSide), and we can model it.

\[
\text{for}(i = 0; i < 8; i + +)\{
\text{force}(\text{true});
\text{s(false, i);}
\}
\]

\[
\text{void } s(x, y)\{
\rightarrow \text{if}(x)
\text{--→ } (*y)(x);
\}
\]
KASLR in presence of *Speculative Attackers*

The system call \( s \) is a threat in presence of speculative attackers (BlindSide), and we can model it.

\[
\text{for}(i = 0; i < 8; i++)\{
    \text{force}(\text{true});
    s(\text{false}, i);
}\]

\[
\text{void } s(x, y)\{
    \text{if}(x)
        (*y)(x);
}\]

\[
\text{LNI}(\gamma) := \text{the semantics the side channel leaks of the syscalls in } \gamma \text{ must not depend on the layout}
\]
KASLR in presence of *Speculative Attackers*

The system call $s$ is a threat in presence of speculative attackers (BlindSide), and we can model it.

```c
for(i = 0; i < 8; i++){
    force(true);
    s(false, i);
}

void s(x, y){
    if(x)
        (*y)(x);
}
```

the speculative semantics and the side channel

$SLNI(\gamma) := $ leaks of the syscalls in $\gamma$ must not depend on the layout
KASLR with Speculative Attackers and Side-Channels

SLNI(γ) ⇒ safety
KASLR with Speculative Attackers and Side-Channels

\[ \text{SLNI}(\gamma) \Rightarrow \text{safety} \]
\[ \Downarrow \]
\[ \text{KASLR} \land \text{SLNI}(\gamma) \Rightarrow \text{probabilistic safety} \]
KASLR with Speculative Attackers and Side-Channels

\[ \text{SLNI}(\gamma) \Rightarrow \text{safety} \]
\[ \Downarrow \]
\[ \text{KASLR} \land \text{SLNI}(\gamma) \Rightarrow \text{probabilistic safety} \]

Victim

SLNI
KASLR with Speculative Attackers and Side-Channels

\[
\text{SLNI}(\gamma) \Rightarrow \text{safety} \\
\downarrow \\
\text{KASLR} \land \text{SLNI}(\gamma) \Rightarrow \text{probabilistic safety}
\]

Victim  SLNI  
p[0] = 42  (p array pointer)  no
KASLR with Speculative Attackers and Side-Channels

\[ \text{SLNI}(\gamma) \Rightarrow \text{safety} \]

\[ \Downarrow \]

\[ \text{KASLR} \land \text{SLNI}(\gamma) \Rightarrow \text{probabilistic safety} \]

<table>
<thead>
<tr>
<th>Victim</th>
<th>SLNI</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p[0] = 42 ) (( p ) array pointer)</td>
<td>no</td>
</tr>
<tr>
<td>( f() )       (( f ) procedure)</td>
<td>no</td>
</tr>
</tbody>
</table>
Kernel Safety $\implies$ Speculative Kernel Safety (1/3)

$\gamma$ is safe against speculative attackers

$\gamma$ is safe against ordinary attackers
Kernel Safety $\Rightarrow$ Speculative Kernel Safety (1/3)

$\gamma$ is safe against speculative attackers

$\uparrow$?

$\gamma$ is safe against ordinary attackers
Kernel Safety $\Rightarrow$ Speculative Kernel Safety (1/3)

$\gamma$ is safe against speculative attackers

$\gamma$ is safe against ordinary attackers
Kernel Safety $\Rightarrow$ *Speculative Kernel Safety* (1/3)

\[ \gamma \text{ is safe against speculative attackers} \]
\[ \uparrow \]
\[ \gamma \text{ is safe against ordinary attackers} \]

But maybe there is an instrumentation $\zeta$ such that:

\[ \gamma \text{ is safe against ordinary attackers} \]
\[ \Downarrow \]
\[ \zeta(\gamma) \text{ is safe against *speculative* attackers} \]
Theorem

If $\zeta$: 

- preserves the semantics of the syscalls,
- prevents the transient execution of unsafe commands,
- and $\gamma$ is safe against ordinary attackers, then $\zeta(\gamma)$ is safe against speculative attackers.
Kernel Safety ⇒ Speculative Kernel Safety (2/3)

Theorem

If $\zeta$:

- preserves the semantics of the syscalls,
Kernel Safety ⇒ Speculative Kernel Safety (2/3)

Theorem

If \( \zeta \):

- preserves the semantics of the syscalls,
- prevents the transient execution of unsafe commands,
Kernel Safety ⇒ Speculative Kernel Safety (2/3)

**Theorem**

If \( \zeta \):

- preserves the semantics of the syscalls,
- prevents the transient execution of unsafe commands,

and \( \gamma \) is safe against ordinary attackers, then \( \zeta(\gamma) \) is safe against speculative attackers.
Kernel Safety ⇒ *Speculative* Kernel Safety (3/3)

Does such transformation exist?
Kernel Safety ⇒ Speculative Kernel Safety (3/3)

Does such transformation exist? Yes.
Kernel Safety ⇒ Speculative Kernel Safety (3/3)

Does such transformation exist? Yes.

\[
\begin{align*}
\zeta(\text{if}(E) \{C\} \text{ else } \{D\}) & \triangleq \text{if}(E) \{\zeta(C)\} \text{ else } \{\zeta(D)\} \\
\zeta(\text{while}(E) \{C\}) & \triangleq \text{while}(E) \{\zeta(C)\} \\
\zeta(*E = F) & \triangleq \text{lfence}; *E = F \\
\zeta(E = *F) & \triangleq \text{lfence}; E = *F \\
\zeta((*E)(F_1, \ldots, F_k)) & \triangleq \text{lfence}; (*E)(F_1, \ldots, F_k)
\end{align*}
\]
Conclusions

- Layout Randomization protects the kernel against ordinary attacks in presence of LNI,
Conclusions

- Layout Randomization protects the kernel against *ordinary* attacks in presence of LNI, but not against the *speculative* ones.
Conclusions

- Layout Randomization protects the kernel against *ordinary* attacks in presence of LNI, but not against the *speculative* ones.
- If a kernel is safe against ordinary attacks, it is possible to make it safe against speculative attacks.

Future work:
- Model indirect branch speculation.
- Evaluate the overhead of our instrumentation in practice.
Conclusions

► Layout Randomization protects the kernel against ordinary attacks in presence of LNI, but not against the speculative ones.

► If a kernel is safe against ordinary attacks, it is possible to make it safe against speculative attacks.

Future work:
Conclusions

- Layout Randomization protects the kernel against ordinary attacks in presence of LNI, but not against the speculative ones.
- If a kernel is safe against ordinary attacks, it is possible to make it safe against speculative attacks.

Future work:

- Model indirect branch speculation.
Conclusions

- Layout Randomization protects the kernel against *ordinary* attacks in presence of LNI, but not against the *speculative* ones.
- If a kernel is safe against ordinary attacks, it is possible to make it safe against speculative attacks.

Future work:

- Model indirect branch speculation.
- Evaluate the overhead of our instrumentation in practice.