Embedded Security

Chapter 3: Meltdown and Spectre

Michael Schwarz, Moritz Lipp

20.03.2018

www.iaik.tugraz.at

- Find something human readable, e.g., the Linux version

```
# sudo grep linux_banner /proc/kallsyms
ffffffff81a000e0 R linux_banner
```

```
char data = *(char*) 0xffffffff81a000e0;
printf("%c\n", data);
```

Building the Code

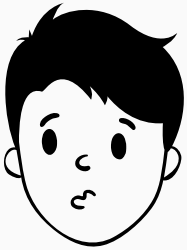
- Compile and run

```
segfault at ffffffff81a000e0 ip 0000000000400535
            sp 00007ffce4a80610 error 5 in reader
```

- Compile and run

```
segfault at ffffffff81a000e0 ip 0000000000400535
            sp 00007ffce4a80610 error 5 in reader
```

- Kernel addresses are of course not accessible
- Any invalid access throws an exception → segmentation fault



- Still no kernel memory
- Maybe it is not that straight forward
- Privilege checks seem to work
- Are privilege checks also done when executing instructions out of order?
- Problem: out-of-order instructions are not visible

- Adapted code

```
*(volatile char*) 0;
array[0] = 0;
```

- Flush+Reload over all pages of the array

- "Unreachable" code line was actually executed
- Exception was only thrown afterwards

- Out-of-order instructions leave microarchitectural traces
- We can see them for example in the cache
- Give such instructions a name: transient instructions
- We can indirectly observe the execution of transient instructions

- Combine the two things

```
char data = *(char*)0xffffffff81a000e0;
array[data * 4096] = 0;
```

- Combine the two things

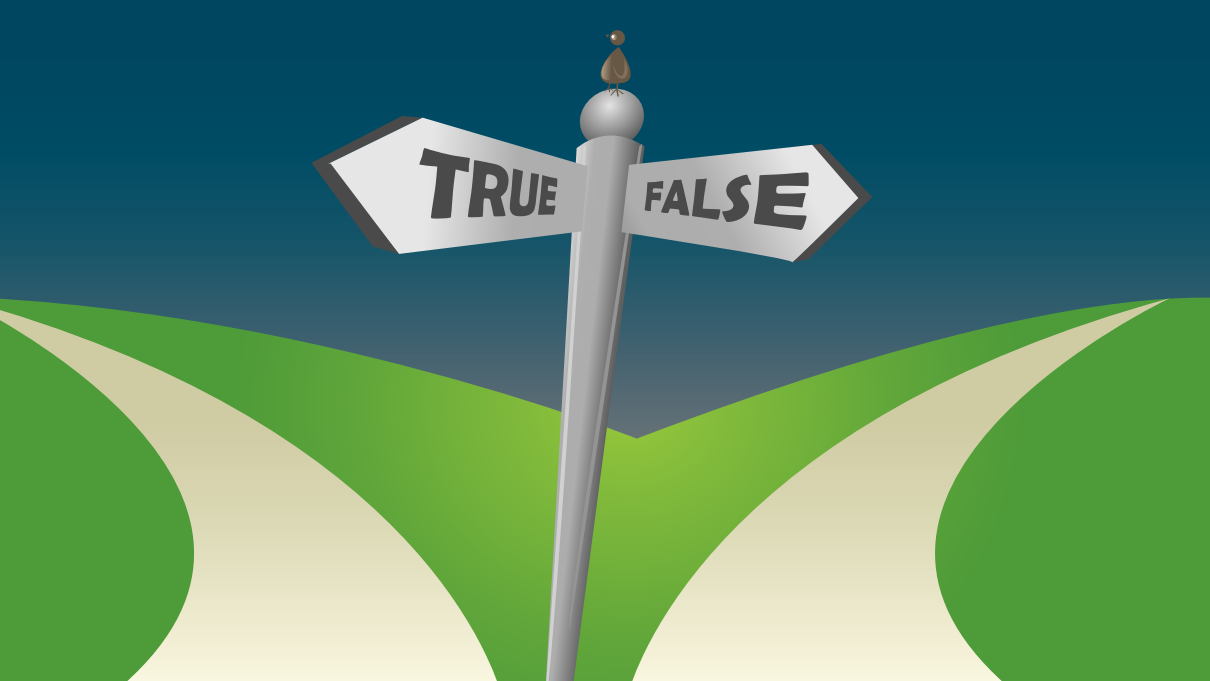
```
char data = *(char*)0xffffffff81a000e0;
array[data * 4096] = 0;
```

- Then check whether any part of `array` is cached

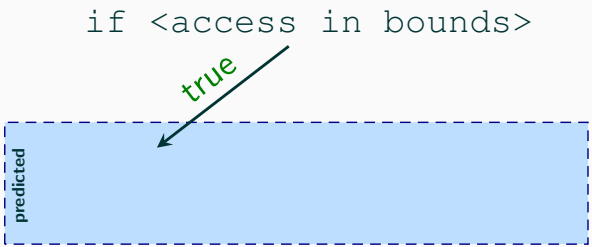
- Flush+Reload over all pages of the array

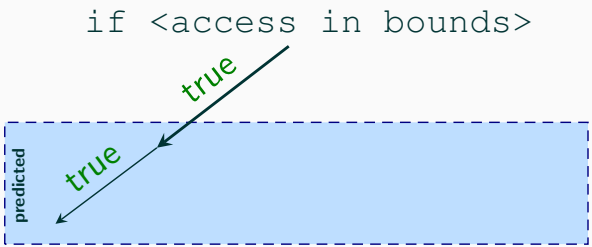
- Index of cache hit reveals data
- Permission check is in some cases not fast enough

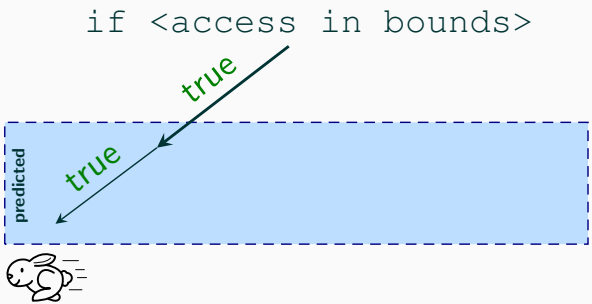
- Using out-of-order execution, we can read data at any address
- Privilege checks are sometimes too slow
- Allows to leak kernel memory
- Entire physical memory is typically also accessible in kernel address space

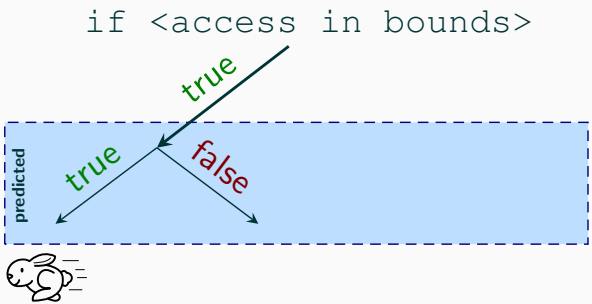

TRUE
FALSE

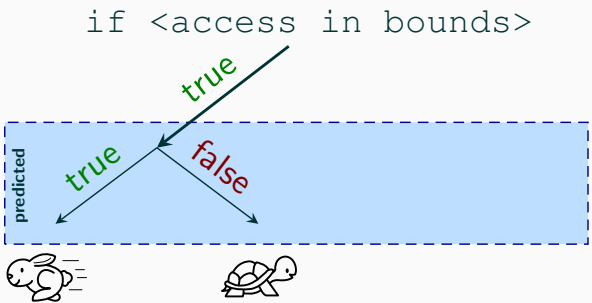
```
if <access in bounds>
```

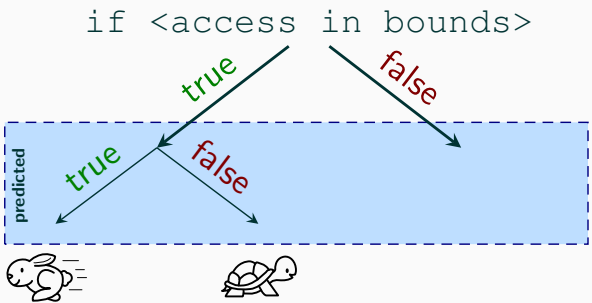

if <access in bounds>
true
predicted

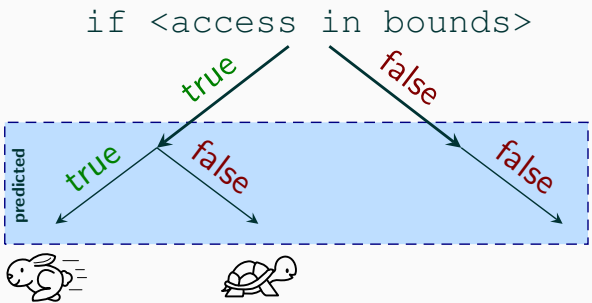

if <access in bounds>
true
predicted
true

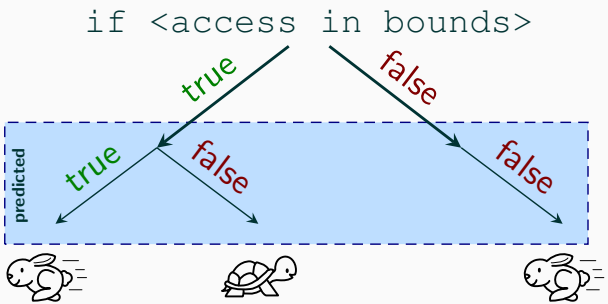

if <access in bounds>
true
predicted
true

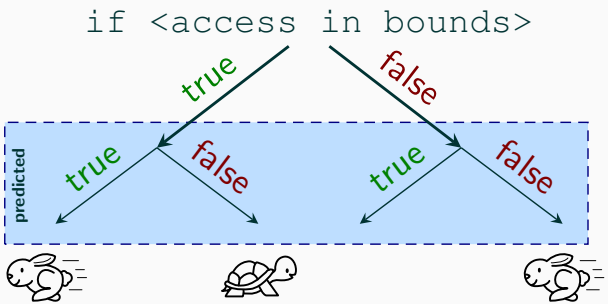

if <access in bounds>
true
predicted
true
false

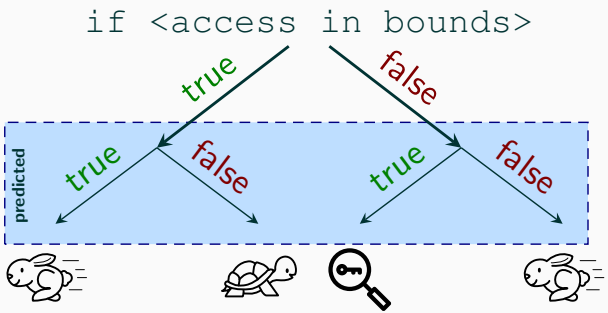

if <access in bounds>
true
predicted
true
false


if <access in bounds>
true
false
predicted
true
false


if <access in bounds>
true
false
predicted
true
false
false


if <access in bounds>
true
false
predicted
true
false
false


if <access in bounds>
true
false
predicted
true
false
true
false


if <access in bounds>
true
false
predicted
true
false
true
false

We are ready for the gory details of Spectre

```
index = 0;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 0;
```

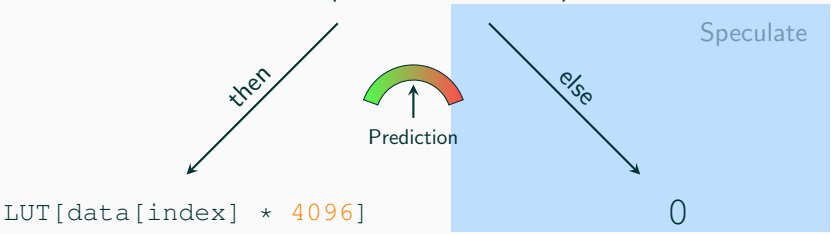
```
char* data = "textKEY";
```

```
if (index < 4)
```

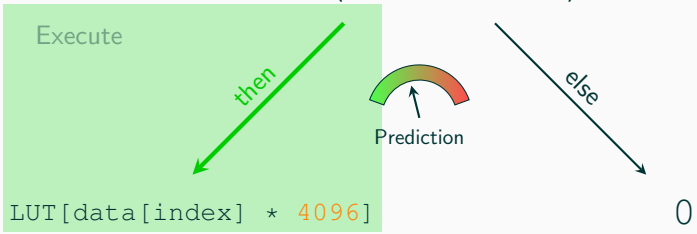

```
LUT[data[index] * 4096]
```

0

```
index = 0;
char* data = "textKEY";
if (index < 4)
```



```
index = 0;
char* data = "textKEY";
if (index < 4)
```



```
index = 1;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 1;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

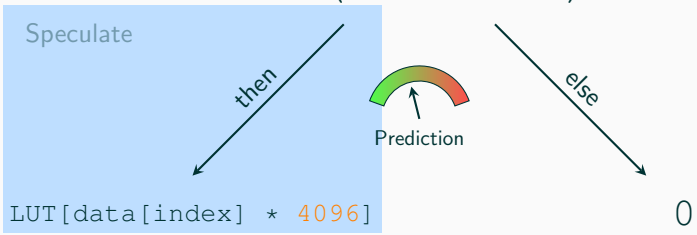

```
LUT[data[index] * 4096]
```

0

```
index = 1;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```



```
index = 1;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

```
index = 2;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 2;
```

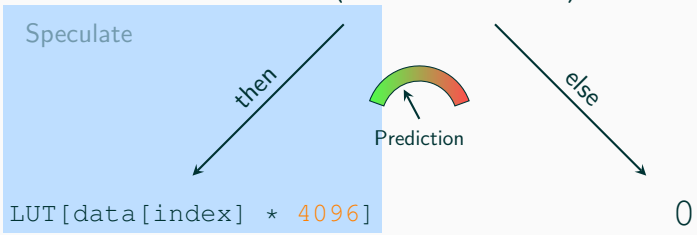
```
char* data = "textKEY";
```

```
if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 2;
char* data = "textKEY";
if (index < 4)
```



```
index = 2;
char* data = "textKEY";
if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 3;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 3;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

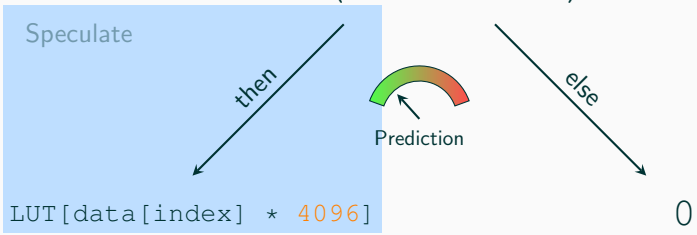

```
LUT[data[index] * 4096]
```

0

```
index = 3;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```



```
index = 3;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

```
index = 4;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 4;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

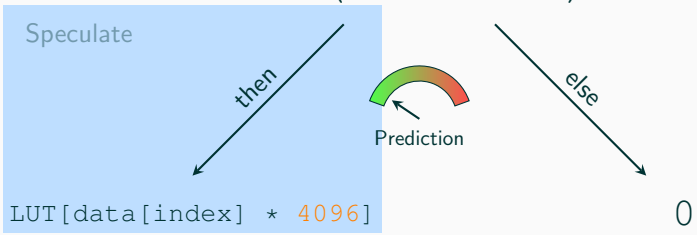

```
LUT[data[index] * 4096]
```

0

```
index = 4;
```

```
char* data = "textKEY";
```

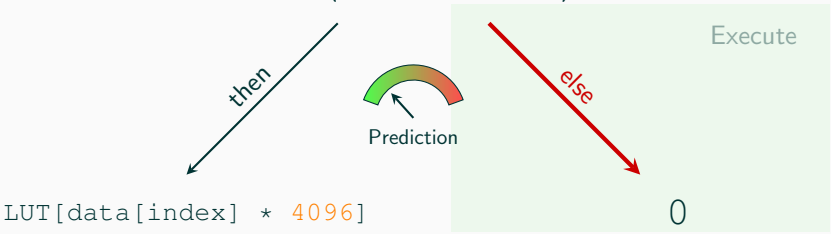
```
if (index < 4)
```



```
index = 4;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```



```
index = 5;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 5;
```

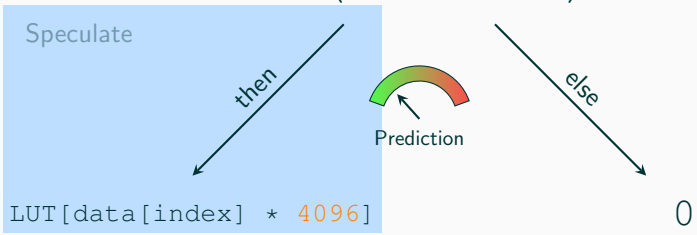
```
char* data = "textKEY";
```

```
if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

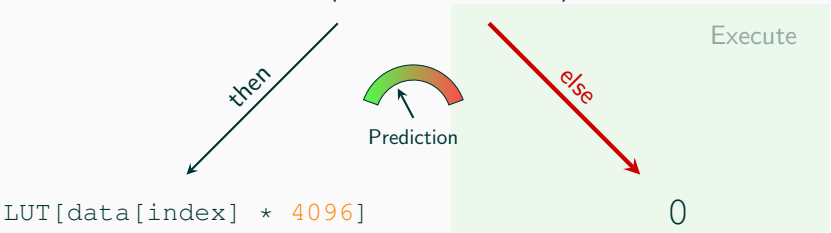
```
index = 5;
char* data = "textKEY";
if (index < 4)
```



```
index = 5;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```



```
index = 6;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 6;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

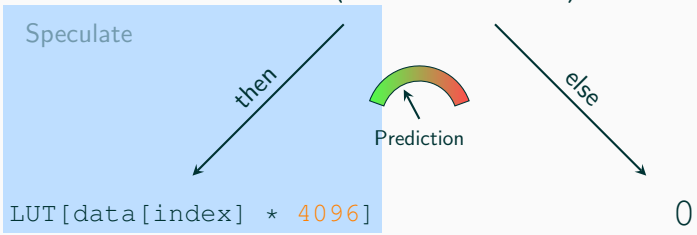

```
LUT[data[index] * 4096]
```

0

```
index = 6;
```

```
char* data = "textKEY";
```

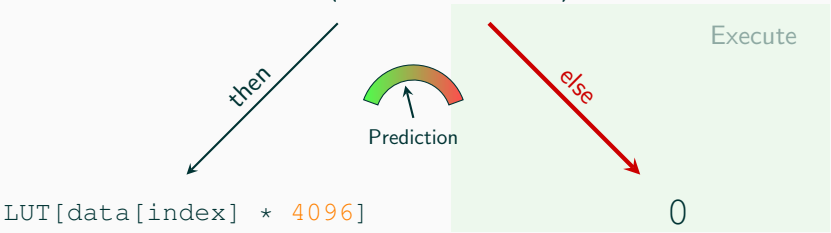
```
if (index < 4)
```



```
index = 6;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```



Embedded Security

Chapter 3: Meltdown and Spectre

Michael Schwarz, Moritz Lipp

20.03.2018

www.iaik.tugraz.at