AN ANALYSIS OF SPECULATIVE TYPE CONFUSION VULNERABILITIES IN THE WILD

Ofek Kirzner
Adam Morrison
Blavatnik School of Computer Science
Tel Aviv University
SPECTRE VARIANT 1: BOUNDS CHECK BYPASS

Goal: Leak data from the victim address space

```java
foo(&valid_x - array1)
```

The secret is leaked

```java
void foo(long x) {
    // ...
    if (x < array1_len) {
        y = array1[x];
        z = array2[y * 4096];
    }
    // ...
}
```
SPECTRE VARIANT 1: BOUNDS CHECK BYPASS

Attacker – unprivileged user

```c
foo(&secret – array1)
```

The secret is leaked

Victim – the kernel

```c
void function_called_from_syscall(long x) {
    // ...
    if (x < array1_len) {
        y = array1[x];
        z = array2[y * 4096];
    }
    // ...
}
```

Read from kernel ➔ Read any physical address
MITIGATION IN THE LINUX KERNEL

A special API to ensure bounds checks are respected under speculation

```c
void function_called_from_syscall(long x) {
    // ...
    if (x < array1_len) {
        y = array1[x];
        z = array2[y * 4096];
    }
    // ...
}
```

```c
void function_called_from_syscall(long x) {
    // ...
    if (x < array1_len) {
        y = array_index_nospec(array1[x], array1_len);
        z = array2[y * 4096];
    }
    // ...
}
```
SPECTRE V1 IS MORE THAN BOUNDS CHECK BYPASS

Quoting from the Spectre paper [Kocher et al., 2019]:

Variant 1: Exploiting Conditional Branches. In this variant of Spectre attacks, the attacker mistrains the CPU's branch predictor into mispredicting the direction of a branch, causing the CPU to temporarily violate program semantics by executing code that would not have been executed otherwise.
SPECULATIVE TYPE CONFUSION

Misspeculation makes the victim execute with some variables holding values of the wrong type, and thereby leak memory content.
**SPECULATIVE TYPE CONFUSION - EXAMPLE**

**Speculation: Type confusion**

```c
void syscall_helper(struct Base* obj) {
    if (obj->type == TYPE1) {
        struct Type1* o = (struct Type1*) obj;
        leak(o->value);
    }
    if (obj->type == TYPE2) {
        ...
    }
}
```

```c
struct Base {
    enum Type type;
};
```

```c
struct Type1 {
    struct Base base;
    ...
    uint32_t value;
};
```
Observation: speculative type confusion may be much more prevalent than previously hypothesized.

We analyzed the Linux kernel, looking for speculative type confusion.

Found new types of speculative type confusion.
EBPF: SPECULATIVE TYPE CONFUSION
EBPF

Linux subsystem, enabling user-defined programs in kernel

eBPF bytecode

Verifier

Static verification

Bounds check bypass mitigations

Compiler

Native Code
EBPF VERIFIER VULNERABILITY

Flows considered by eBPF verifier

\[ r_0 = *(u64 *)(r0) \]

A: \texttt{if } r0 \texttt{!}== \texttt{0x0 goto B} \quad r6 = r9
B: \texttt{if } r0 \texttt{!}== \texttt{0x1 goto D} \quad r9 = *(u8 *)(r6)
C: \quad r1 = M[(r9 \texttt{&} 1) \texttt{*} 512]
D: ...

\[ r_0 \texttt{== 0} \]

\[ r_0 \texttt{== 1} \]

\[ r_0 = *(u64 *)(r0) \]

A: \texttt{if } r0 \texttt{!}== \texttt{0x0 goto B} \quad r6 = r9
B: \texttt{if } r0 \texttt{!}== \texttt{0x1 goto D} \quad r9 = *(u8 *)(r6)
C: \quad r1 = M[(r9 \texttt{&} 1) \texttt{*} 512]
D: ...

otherwise
EBPF VERIFIER VULNERABILITY

// r0 = ptr to an array entry (verified != NULL)
// r6 = ptr to stack slot (verified != NULL)
// r9 = scalar value controlled by attacker

r0 = *(u64 *)(r0)
A: if r0 != 0x0 goto B
   r6 = r9
B: if r0 != 0x1 goto D
   r9 = *(u8 *)(r6)
C: r1 = M[(r9 & 1) * 512]
D:...

Speculative flows are not verified
Unprivileged process can read arbitrary memory addresses at a rate of \(\sim 6.5\) KB/sec

Shadow gadget
Can both be taken

\[
\begin{align*}
A: & \quad \text{if } r0 \neq 0x0 \text{ goto } B \\
& \quad r6 = r9 \\
B: & \quad \text{if } r0 \neq 0x0 \text{ goto } D \\
& \quad r9 = *(u8 *)(r6)
\end{align*}
\]

Branch Prediction Unit

Mutually exclusive

\[
\begin{align*}
A: & \quad \text{if } r0 \neq 0x0 \text{ goto } B \\
& \quad r6 = r9 \\
B: & \quad \text{if } r0 \neq 0x1 \text{ goto } D \\
& \quad r9 = *(u8 *)(r6)
\end{align*}
\]

Manipulating the branch predictor (details in the paper)
COMPILER INTRODUCED SPECULATIVE TYPE CONFUSIONS
COMPILERS MIGHT CREATE SPECULATIVE TYPE CONFUSION

Innocent looking code is compiled in a way that introduces vulnerability.

Compiler reasoning:
Branches are mutually exclusive.
CAN WE FIND IT IN THE WILD?

Binary level analysis of Linux

Focused on system calls, which have well-defined user-controlled interface

The leakage mechanism is out of scope: aiming at finding speculative attacker-controlled memory dereference

<table>
<thead>
<tr>
<th>compiler</th>
<th>flags</th>
<th># vulnerable syscalls</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC 9.3.0</td>
<td>-Os</td>
<td>20</td>
</tr>
<tr>
<td>GCC 9.3.0</td>
<td>-O3</td>
<td>2</td>
</tr>
<tr>
<td>GCC 5.8.2</td>
<td>-Os</td>
<td>0</td>
</tr>
<tr>
<td>GCC 5.8.2</td>
<td>-O3</td>
<td>0</td>
</tr>
</tbody>
</table>

A pattern in syscalls the receive an optional untrusted user pointer (details in paper)
SPECULATIVE POLYMORPHIC TYPE
CONFUSION
SPECTRE V2 MITIGATIONS

Spectre v2 exploits misprediction of indirect branch target addresses

Retpolines: block indirect branch prediction

Optimization: restrict speculation to valid targets [Linux, Amit et al., 2019]

Might create speculative type confusion vulnerabilities

```
# %rax = branch target
cmp $0xXXXXXXXX, %rax # target1?
jz $0xXXXXXXXX
cmp $0xYYYYYYYY, %rax # target2?
jz $0xYYYYYYYY
...
jmp ${fallback} # jmp to retpoline thunk
```
SPECULATIVE POLYMORPHIC TYPE CONFUSION

```c
struct Common { void (*foo)(void*); };  
struct A { struct Common common; char* ptr; };  
struct B { struct Common common; long user_controlled_scalar; };  

void some_code_path(struct Common* common) {  
    /* ... */  
    common->foo(common);  
}

void foo_A(struct Common* common) {  
    char x = *((struct A*) common)->ptr;  
    B->user_controlled_scalar  
    leak(x);  
}
```
ANALYSIS

Analysis
- Linux code analysis - looking at ways in which polymorphism can lead to speculative type confusion

Results
- Thousands - flagged potentially vulnerable
- Hundreds - "array indexing" instances
- All - limited speculation window or limited control on user value

Conclusion
- Were a conditional branch-based mitigation used instead of retpolines, the kernel’s security would be on shaky ground
SUMMARY

Analysis
- Polymorphic type confusion
- Attacker-introduced: eBPF
- Compiler-introduced

Conclusion
Speculative type confusion is prevalent

Discussion
Mitigation is difficult and requires rethinking
(Discussion in paper)

ofekkir@gmail.com
mad@cs.tau.ac.il