WarpAttack: Bypassing CFI through Compiler-Introduced Double-Fetches

Jianhao Xu*,†, Luca Di Bartolomeo†, Flavio Toffalini†, Bing Mao* and Mathias Payer†

jianhao xu@smail.nju.edu.cn,
{lucadi Bartolomeo, flavio.toffalini}@epfl.ch,
maobing@nju.edu.cn, mathias.payer@nebelwelt.net
Compiler-Introduced Double Fetch

A Linux kernel case (2012):

```c
/* commit: 8135cf8b092723dbfccc611fe6fdcb3a36c9951c5 */
switch (op->cmd) {
    case XEN_PKT_OP_conf_read:
        op-12 /* the corresponding assembly code */
        cmp DWORD PTR [r13+0x4],0x5
        break;
    case XEN_PKT_OP_create:
        ja 0x3358 <xen_pciblk_do_op+952>
        mov eax,DWORD PTR [r13+0x4]
        jmp QWORD PTR [rax*8+off_77D0]
    default:
        op-
    }
```
Compiler & Correctness-Security Gap

- Compiler

![Diagram of source code to binary conversion]

- Correctness-Security Gap

```c
// Attempt to scrub the sensitive data saved on stack
memset(secret, 0, sizeof(secret));
return;
```

Correctness ≠ Security

- Dead Store
- Sensitive Data Scrubbing
## Compiler-Introduced Security Issues

- **WarpAttack key insight**

<table>
<thead>
<tr>
<th>Compiler Correctness</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler-Introduced Double-Fetches</td>
<td>Concurrency bugs Or Benign data race</td>
</tr>
<tr>
<td></td>
<td>A weakness of control flow guards</td>
</tr>
</tbody>
</table>

**WarpAttack exploits a misalignment between compiler implementations and CFI assumptions**
Control Flow Integrity (CFI)

CFI guard control flow

- Inserts run-time checks
- Practical and (reasonably) fine-grained

CFI is getting more and more important
Bound-Checked Indirect Jumps

**CFI: no need to protect**

```c
switch(a) {
  case 0:
  case 1:
    ...
  default:
    ...
}
```

```Assembly
mov r1, [A]
cmp r1, imm
...  
mov r1, [r3+r1*4]
add r1, r3
jmp r1
```
Bound-Checked Indirect Jumps

Compilers are not aware of security boundaries

```assembly
mov r1, [A]
cmp r1, imm
...
mov r1, [r3+r1*4]
add r1, r3
jmp r1
```

```assembly
mov r1, [A]
cmp r1, imm
...
mov r2, [A]
mov r2, [r3+r2*4]
add r2, r3
jmp r2
```
WarpAttack: Threat model

Adversarial Capabilities

- Arbitrary read-write
- Thread control
- One triggerable gadget sample

Defensive Assumptions

- Non-Executable Memory
- Randomization
- Control Flow Protection

The only requirement beyond CFI’s Threat model.
WarpAttack

mov r1, [A]
cmp r1, imm
; may mov [A'], r1
ja 0xdead

mov r2, [A]/[A']
mov r2, [r3+ r2*4]
add r2, r3
jmp r2

; 0xdead: default branch

; 0xdeadbeef: malicious target

A: Checked Object
A': Spilled Object
B: Controlled Object

Memory

Attacker

Rewrite A: ensure not default branch
Rewrite A'/A': let the jump fetch address from B
Rewrite B: address to the malicious target

Jump Table
# Challenges and Solutions

## Gadget Code Detection:

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler-Introduced?</td>
<td><img src="image" alt="BIN file" /></td>
</tr>
<tr>
<td>Data Dependency</td>
<td><img src="image" alt="Search icon" /> <img src="image" alt="Filter icon" /></td>
</tr>
</tbody>
</table>

- **Compiler-Introduced?**
  - Indicates whether the gadget code is compiler-introduced.
- **Data Dependency**
  - Represents the dependency of data within the gadget code.
# Challenges and Solutions

## Proof-of-Concept Exploit:

<table>
<thead>
<tr>
<th>Challenges</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short time window</td>
<td>0.45% success rate in 20 seconds</td>
</tr>
<tr>
<td>Crashes when wrong</td>
<td>On Firefox 106.0.1</td>
</tr>
</tbody>
</table>
How WarpAttack affects real world

Vulnerable code in the wild

- All C/C++ programs potentially affected
- 1,600+ victim gadgets in 6 programs

Acknowledgements from

- GCC and clang

Other than X86/64
- ARM 32/64*
- RISCV 32/64
- MIPS 32/64

Only X86/64 has two variants, Others have just stack spilling variant

WarpAttack affects many programs, compilers and architectures
WarpAttack Conclusion

- CFI assumption
- Attack method
- Real world Impact
- Proof-of-Concept

Thanks!
• Backup slides
Double Fetch

Victim thread
- Check([A])
- var := [A]
- Use(var)
- Error()

Memory state
- [A] = 12
- [A] = 0xdeadbeef

Attacker thread
- [A] := 0xdeadbeef
Mitigations

Avoiding Gadget code generation
- GCC `-fno-switch-tables`
- Clang ‘O1’

Protecting Indirect Jump
- CFI checks for switch jump tables

Monitoring for Attack Behavior
- Characteristics like spawning several threads, constantly writing a certain memory site
- Crashes