What Cannot be Read, Cannot be Leveraged?

Revisiting Assumptions of JIT-ROP Defenses

Giorgi Maisuradze, Michael Backes, Christian Rossow
CISPA, Saarland University, Germany
Overview

- Code Reuse Attacks/Defenses
- JavaScript JIT Compilation
- Existing JIT Attacks/Defenses
- New JIT Gadget Creation
- New JIT Defense
Revisiting Code Reuse
Code Reuse

- Find useful code fragments
- Chain gadgets and functions together
- Trigger execution of the gadget chain
Code Reuse Defenses

- Find useful code fragments
- Chain gadgets and functions together
- Trigger execution of the gadget chain

Defenses:
- Randomize memory segments (ASLR)
- Fine-grained code randomization
**JIT-ROP [S&P’13]**

- Leak and collect code pointers
- Read code pages to find gadgets and function pointers
- Chain gadgets and functions together
- Trigger execution of the gadget chain
Execute-no-Read (XnR) Schemes

- Mark code pages non-readable
  [CCS’14, S&P’15, CCS’15, CODASPY’15, NDSS’16]
- Randomize code
- Hide code pointers
JIT code reuse without reading code?!
JIT Spraying [WOOT’10]

- Create JS functions containing operations on immediate values
- Trigger JIT-compilation of created functions
- Divert the CF in the middle of one of the constants
Defenses in Browsers

- **Constant Blinding (IE + Chrome)**
  - Store constant XORed with a random key to a register
  - XOR the register again with the key

- **NOP Insertion (IE)**
  - Randomly add random number of NOP instructions

- Only large constants (>2B) are blinded
2B Gadgets

The Devil is in the Constants [NDSS’15]

- Encode OPs in 2B unblinded constants
- Use intended return instruction from JIT-compiled code
- Find and use emitted gadgets

Limitations:

- Requires readable code to find gadgets
- Aligned return instruction
Gadgets via Control Flow Instructions
Emitting Gadgets via Control Flow Instructions

Displacement:
- Encodes the offset from a base address, e.g., from `rax` or `rip`
- Used in:
  - Instructions accessing the memory:
    ```
    mov rax, [rbx+0x12345678]; read memory at rbx+0x12345678
    ```
  - Control flow instructions (implicit base address is `rip`):
    ```
    jz 0x123456; jump at address rip+0x123456 if zero flag is set
    call 0x123456; call the function at address rip+0x123456
    ```

We will use JavaScript statements that are compiled to conditional jumps and direct calls to emit gadgets
Conditional Jumps in JS

- **JavaScript if statement:**
  - Emits conditional jumps after compilation
  - Displacement field denotes the size of the compiled if body
  - Change the size of if body ⇒ changes the value in the displacement field

- **Alternatives to if statement:**
  - while/for/switch/break...

*The size of the compiled if statement's body determines emitted Gadget*
Direct Calls in JS

- **Direct calls in JS:**
  - Displacement: the distance between the caller and the callee

- **Create a large JS function:**
  - Use \( i = i + j \) (16B) \( 0 \times 100 \times 000 \) times
  - Emitted values \([0x000005, 0xFFFFF5]\)
  - Prepend the function with \( i = 1 \) (8B) to change 5 into 13 (0xD)

- **Find callee/caller address:**
  - Compiler’s heap
  - Return address from stack
  - JS objects (e.g., from `Math.random`)
Applying Concepts to Modern Browsers
Gadgets in Chrome 51 (64-bit)

Goal:

Direct calls (**syscall**, **pop rcx**, **pop rdx**):
- Align LSHB of displacements to 0xd
- Emit direct calls (**j++** 0x80`000 times)

Conditional jumps (**pop r8**, **pop r9**):
- Stack two if statements (for **pop r8** and **pop r9**)
- Fill the inner (**pop r8**) with 0xc35841 bytes
- Add additional 0xed bytes to the outer **if** body
  - +0x13 bytes from **if** statement=0x100

**Gadget generation time = 1.3 seconds**
Limitations in Internet Explorer

Challenges in IE:

1. Limited JIT-compiled function size
   - Maximum function size 1MB
   - No 3B displacements in conditional jumps
   - No huge functions with direct calls

2. NOP insertion
   - Non-predictable offsets inside the function

Steps to make direct calls work:

1. Get code page at the correct distance from the callee
2. Fill the page with gadget emitting functions
3. Emit the direct call at the correct address, or recompile
Allocating Code Page at a Correct Distance

IE code-page size is 0x20`000 bytes

- Compile 200 JavaScript functions of size 0x10`000 bytes each to allocate 100 code pages
- Choose the code page at the correct distance
Gadgets in IE ➔ Emitting Gadgets

1. Deallocate the function
2. Fill the code page up to the correct distance from the callee
3. Compile gadget-emitting JS function
4. Check if the direct call was emitted at the correct place
5. If yes, then gadget is found, else deallocate the function and goto (3)
Gadgets in IE 11 (32-bit)

**Goal:**

<table>
<thead>
<tr>
<th>popa; ret</th>
<th>61c3 cd80c3</th>
</tr>
</thead>
</table>

**Direct calls** (`syscall, popa`):
- Use `i=Math.random()` 250 times (≈0x1`000 bytes)
- Check the address of emitted direct call

**Emitting gadgets:**
1. Find and fill up the correct code page (≈4 seconds)
2. Compile `syscall` and `popa` at correct places (≈4 seconds)
3. Recompile `syscall` until the gadget is emitted (≈2 seconds)

**Average gadget generation time** ≈ 32 seconds
Gadgets in Browsers

- Conditional jumps
  - Work in *Chrome* and *Firefox*
  - Only upto 2B gadgets possible in *IE*

- Direct calls
  - Work in *Chrome* and *IE*
  - Do not work in *Firefox*
Removing the Gadgets...
For each direct call:

(1) If the address of the callee is known, replace the call with the indirect call

(2) If the address is *not* known:
   (a) convert the direct call into the indirect one
   (b) blind the emitted relative address
Defense ➔ Conditional Jumps

For each conditional jump:

1) Convert a direct (relative) jump into an indirect one

2) Add the inverted conditional jump to jump into `<if_body>`
Cost of the defense in V8:

- **V8’s JS Benchmark Suite:**
  - 2% performance overhead on average

- **Microbenchmarks:**
  - 14% and 10% overhead on average

- **V8’s JS Benchmark Suite (Code-size):**
  - ≈26% overhead (1,123 kB → 1,411 kB)

<table>
<thead>
<tr>
<th></th>
<th>Original</th>
<th>Modified</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richards</td>
<td>36,263</td>
<td>35,555</td>
<td>1.95%</td>
</tr>
<tr>
<td>DeltaBlue</td>
<td>63,641</td>
<td>62,045</td>
<td>2.51%</td>
</tr>
<tr>
<td>Crypto</td>
<td>33,366</td>
<td>32,725</td>
<td>1.92%</td>
</tr>
<tr>
<td>RayTrace</td>
<td>77,198</td>
<td>75,488</td>
<td>2.21%</td>
</tr>
<tr>
<td>EarleyBoyer</td>
<td>44,900</td>
<td>43,700</td>
<td>2.67%</td>
</tr>
<tr>
<td>RegExp</td>
<td>6,525</td>
<td>6,414</td>
<td>1.71%</td>
</tr>
<tr>
<td>Splay</td>
<td>21,095</td>
<td>20,479</td>
<td>2.92%</td>
</tr>
<tr>
<td>NavierStokes</td>
<td>31,924</td>
<td>31,998</td>
<td>-0.23%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>32,255</td>
<td>31,662</td>
<td>1.96%</td>
</tr>
</tbody>
</table>
Summary

- Displacement fields of x86 instructions can be used to inject arbitrary 3 byte values in JIT-compiled code
- Predictable code output from JIT compilers allows adversaries to generate and reuse gadgets without reading the code
- Gadgets in displacements fields can be removed by converting direct jumps/calls into indirect ones

Thank you!