# The 1-Byte Constant Attack against JIT Compilers

Takahiro Shinagawa, Yuki Suzuki, Tomoyuki Nakayama, and Masanori Misono
*The University of Tokyo*
*shina@ecc.u-tokyo.ac.jp, {suzuki,nakayama,misono}@os.ecc.u-tokyo.ac.jp*

## 1 Introduction

JIT compilers in browsers are often subject to attacks because they must handle source code from untrusted hosts. One of the attacks is to exploit constants in the source code that are complied into immediate values in the binary code in order to generate arbitrary small pieces of code, called gadgets, and chain them with return-oriented programming (ROP) [2].

To mitigate this attack, some JIT compilers introduced *constant blinding* that encrypts constants so that they do not appear in the binary code as is [1]. Since such compilers blind only large constants for performance reasons, a recent study showed that using only small constants can mount a successful ROP attack [1]. However, it requires at least 2-byte constants (and existing `ret` instructions) to construct 3-byte gadgets that are necessary to issue a system call.

In this poster, we reveal for the first time that we can construct any 2-byte gadgets and a part of 3-byte gadgets from only 1-byte constants in JavaScript programs on the x64 architecture. The main idea is to exploit two 1-byte constants that are compiled into a single instruction and combine the generated 2-byte constant with the adjacent byte in the instruction. All current major JavaScript JIT compilers do not blind 1-byte constants and are vulnerable to this attack.

## 2 1-Byte Constant Attack

The goal of this attack is to generate the following gadgets that issue the ZwVirtualProtectMemory() system service in Windows to change the page permissions to be executable. The target JIT compiler in this work is ChakraCore that was used in the Microsoft browsers such as IE and Edge.

```
41 59 c3        pop %r9;  ret;
41 58 c3        pop %r8;  ret;
5a c3           pop %rdx; ret;
59 c3           pop %rcx; ret;
58 c3           pop %rax; ret;
0f 05 c3        syscall;  ret;
```

We need to generate a few 2-byte gadgets and two types of 3-byte gadgets starting with `41` and `0f`. To achieve this, we exploit WebAssembly to generate mov instructions that contain the above gadgets. First, the following Web Assembly code can generate arbitrary 2-byte gadgets by changing the 1-byte offset and assigned value.

```
(i32.store8 offset=0x58
    (get_local $i) (i32.const 0xc3))
```

This code is compiled into the following binary code.

```
c6 43 58 c3     movb $0xc3,0x58(%rbx)
```

We can find the 2-byte sequence "`58 c3,`" which is the binary code of the gadget "`pop %rax; ret`"

To generate 3-byte gadgets, we must carefully control the behavior of the JIT compiler. For example, to generate a 3-byte gadget starting with `41`, we must force the compiler to use the `%rcx` register and generate the following code.

```
c6 41 58 c3    movb $0xc3,0x58(%rcx)
```

We achieved this by accessing several global variables and controlling the register allocation scheme of the compiler.

Generating 3-byte gadget starting with `0f` is more difficult because the following complex instruction must be generated.

```
c6 44 0f 05 c3    movb $0xc3,0x5(%rdi,%rcx)
```

We achieved this by accessing global variables instead of local variables to force the compiler to use the above addressing mode, and access many other global variables carefully so that the compiler uses the `%rdi` and `%rcx` registers.

More specifically, we first access 13 global variables several times so that all registers are assigned to the global variables. Then, we find the global variables assigned to the `%rdi` register and do not access it in the last iteration. Finally, the following code generates the above gadget.

```
(i32.store8 offset=5
    (get_global $global$10) (i32.const 0xc3))
```

We will show the complete code in the poster.

## References

[1] M. Athanasakis et al. The devil is in the constants: Bypassing defenses in browser JIT engines. In *Proc. NDSS*, 2015.

[2] C. Rohlf et al. Attacking Client Side JIT Compilers. Black Hat USA, 2011.