[译]使用 COOP 绕过 CFI 保护

hellocaicaic**** / 2018-12-04 09:20:00 / 浏览数 2618

引言

CFI已经被确定为漏洞利用缓解的一个标准，并有了许多不同的实现，如Microsoft CFG2，Microsoft RFG3，PaX Team 的 RAP™4 和 Clang 的 CFI5。
在本系列文章中，我们将演示如何绕过现代CFI实施。

具体来说，在这篇文章中，我们将演示一种高级代码重用技术，伪造面向对象编程（ COOP ）来利用启用CFI保护的旧漏洞。

漏洞

CVE-2015-5122是Hacking Team用于利用Adobe Flash Player（ <= 18.0.0.203 ）的 UAF 漏洞。 可以在这里找到对漏洞本身的分析。
请注意，通过利用此漏洞，我们可以获得进程内存的完整读写原语。

我们的工作基于CVE-2015-5122在Metasploit中的利用实现，可在这里找到。 为了实现读/写原语，该漏洞覆盖了 vector 对象的 length 成员。
vector对象包含一个名为ExploitByteArray的类，该类包含提供完整读/写原语的write（ addr，data ）和read（ addr ）方法。 通过在Exploiter类中定义伪造的“magic” 方法并使用选取的地址覆盖其虚函数指针来获得代码执行。

metasploit实现首先调用VirtualProtect，以便将 spray 的栈迁移指令的内存页保护标志更改为READWRITE_EXECUTE，然后再使用 magic 方法调用可执行指令并开始执行ROP链。

```
// ■■■■■■■■■■■■ VirtualProtect ■■
eba.write(stack_address + 8 + 0x80 + 28, virtualprotect)
eba.write(magic_object, stack_address + 8 + 0x80); // overwrite vtable (needs to be restored)
eba.write(magic + 0x1c, stub_address)
eba.write(magic + 0x20, 0x10)
var args:Array = new Array(0x41)
Magic.call.apply(null, args);

// ■■■■■■■■■■■■■■■ROP■
eba.write(stack_address + 8 + 0x80 + 28, stub_address + 8)
eba.write(magic_object, stack_address + 8 + 0x80); // ■■■■ (■■■■■■)
eba.write(magic + 0x1c, stack_address + 0x18000)
Magic.call.apply(null, null);
eba.write(magic_object, magic_table);
eba.write(magic + 0x1c, magic_arg0)
eba.write(magic + 0x20, magic_arg1)
```

这种漏洞利用实现足以绕过ASLR和DEP保护，但无法绕过 CFI 保护。

例如，当 magic 函数返回到与原始调用指令在堆栈上 push 的地址不同的地址时，以及在栈迁移指令和ROP链执行时，基于Shadow Stack（ 后端 ）的CFI实现将很容易地检测。（ 译者注：Shadow Stack 为 Intel 在底层实现的 CET 中的一种功能 ）

CFI 约束

为了使现代CFI实现无法检测到这种利用，我们必须遵循Schuster等人描述的以下几个约束。

1. 间接调用/跳转到非已知的地址
2. 返回不符合规定的调用堆栈
3. 过度使用间接分支
4. 堆栈指针迁移（ 可能是暂时的迁移 ）
5. 注入新的代码指针或使用现有的代码指针
6. 间接调用/跳转/返回 到“关键函数”

我们添加了一个额外的约束，叫做执行细粒度策略的“关键函数”。

COOP（ 伪造面向对象编程 ）

伪造面向对象编程（ COOP ）是一种用于C ++应用程序的新的代码重用技术。
这种技术依赖于CFI实现方案不考虑C++语义的假设，因为它们不检查虚拟调用点是否调用了其对应的虚函数。
通过使用我们选择的vptr伪造vtable对象，我们可以在不触发CFI的情况下调用任何虚函数，因为CFI实现方案不会验证被调用的虚函数与调用者对象的类之间的任何连接。

首先，我们需要找到执行我们计划的操作的虚函数，这些函数称为“vfgadgets”。

其次，为了把它们组合在一起，我们需要找到一个名为Main Loop Gadget的特殊vfgadget（ML-G，如果它传递参数就叫ML-ARG-G）。
此vfgadget包含一个循环，该循环遍历对象列表并调用每个对象的虚函数。 在大型C++应用程序中找到这样的vfgadget是很容易的。
通过使用伪造成员对象列表填充伪造的ML-G对象，每个对象将包含一个包含不同vfgadget的伪vtable，我们可以执行应用程序代码的不同部分而不违反上述约束。
我们演示了两个vfgadgets的组合，按顺序执行 ATL::CComControl::CreateControlWindow和CLibrariesFolderBase::v_AreAllLibraries。

vfgadget 1

活动模板库（ATL）是一个C++模板库，用于简化Windows中COM对象的编程。
您可以在某些Windows库中找到这些模板，包括shell32.dll（版本6.1.7601.23403）。shell32.dll对象之一是CComControl，这是一个提供创建和管理ATL控件的方法的类

根据Microsoft的文档，该方法通过调用CWindowImpl :: Create初始化并创建一个窗口对象，并调用其他内部对象，如下所述。
对于每个窗口，它们都需要有一个WndProc函数 - 一个回调函数，它将处理发送到窗口的所有消息。

我们发现这个函数把 thunk 作为窗口的 WndProc 过程。 这个 thunk 通过用C ++
this指针覆盖第一个WndProc堆栈参数（应该是窗口的HANDLE），将Windows C回调调用转换为虚函数调用。

thunk 的数据及其反汇编代码：

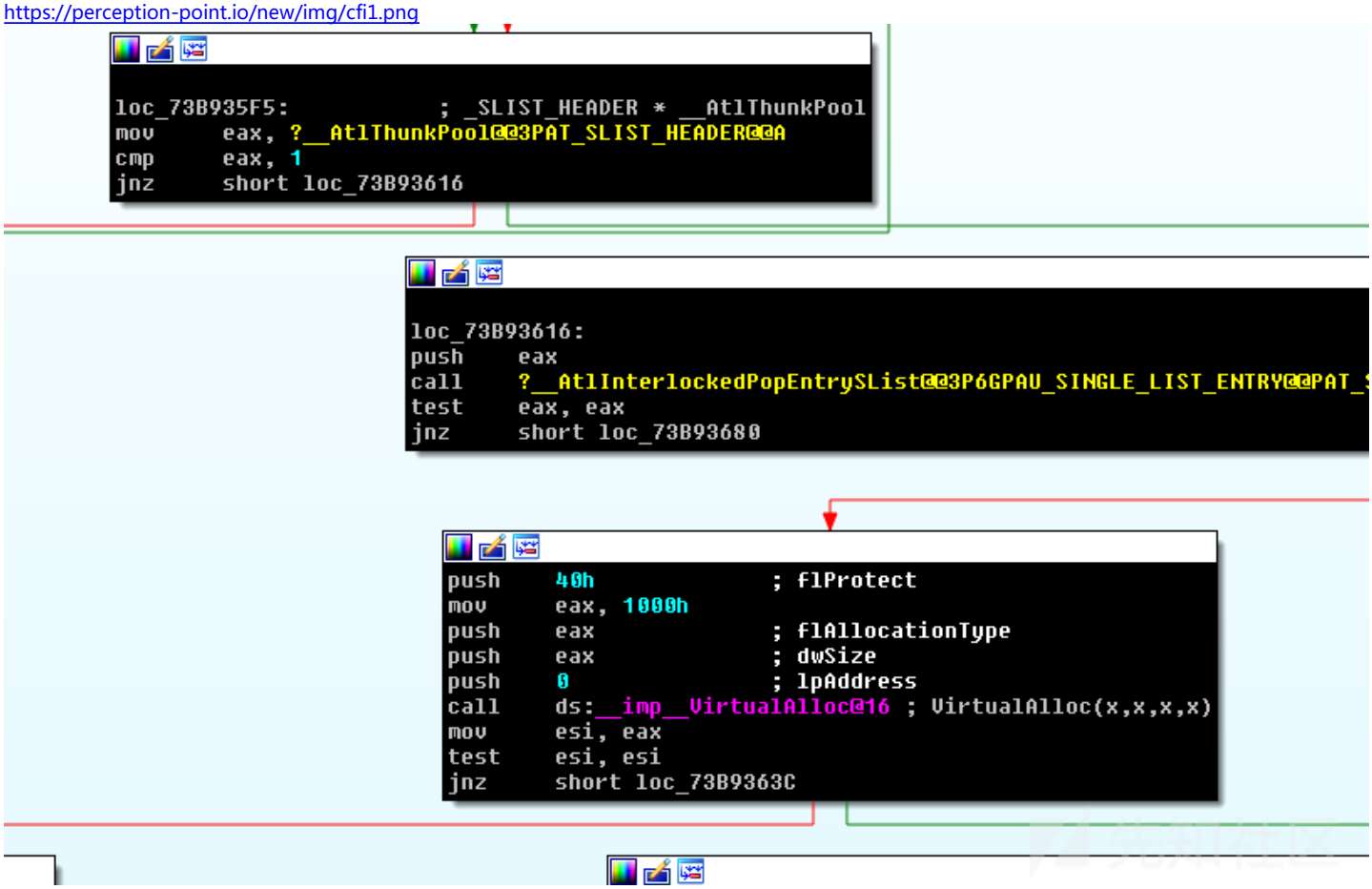
```
Thunk data:
c7 44 24 04 [DWORD thisPointer] e9 [DWORD WndProc]
disas:
mov DWORD PTR [esp+0x4], thisPointer
jmp WndProc
```

有关ATLThunk的更多信息，请访问这里。

请注意，此方法与 Windows 7 Build 7601 SP1相关，并且在以后的ATL版本中可能进行了更改。

这个vfgadget真正有趣的是thunk被写入了 VirtualAlloc 使用EXECUTE_READWRITE权限分配的内存页。
这个VirtualAlloc调用由ATL::_stdcallthunk::Init执行，如下面的屏幕截图所示（0x40是flProtect参数，意思是EXECUTE_READWRITE）：

https://perception-point.io/new/img/cfi1.png



有关此vfgadget的其他注意事项包括：

1. 创建的thunk指针稍后将保存在我们的CComControl伪造对象中
2. 只有一个检查可以阻止vfgadget调用ATL::_stdcallthunk::Init，并且我们对此检查有完全控制权，因为它只验证对象的一个成员为NULL。

总而言之，我们将创建伪造的CComControl对象，调用ATL::CComControl::CreateControlWindow
vfgadget，然后使用我们的读/写原语（ExploitByteArray）来读取创建的thunk地址。 最终给我们一个READWRITE_EXECUTE页面来存储我们的shellcode。

我们用来执行vfgadget的magic方法将3个参数传递给虚函数，但ATL::CComControl::CreateControlWindow只接收2个参数，这会导致堆栈损坏并导致进程崩溃。为了避免堆栈损坏，我们使用另一个接收3个参数的vfgadget并使用它来调用ATL::CComControl::CreateControlWindow。

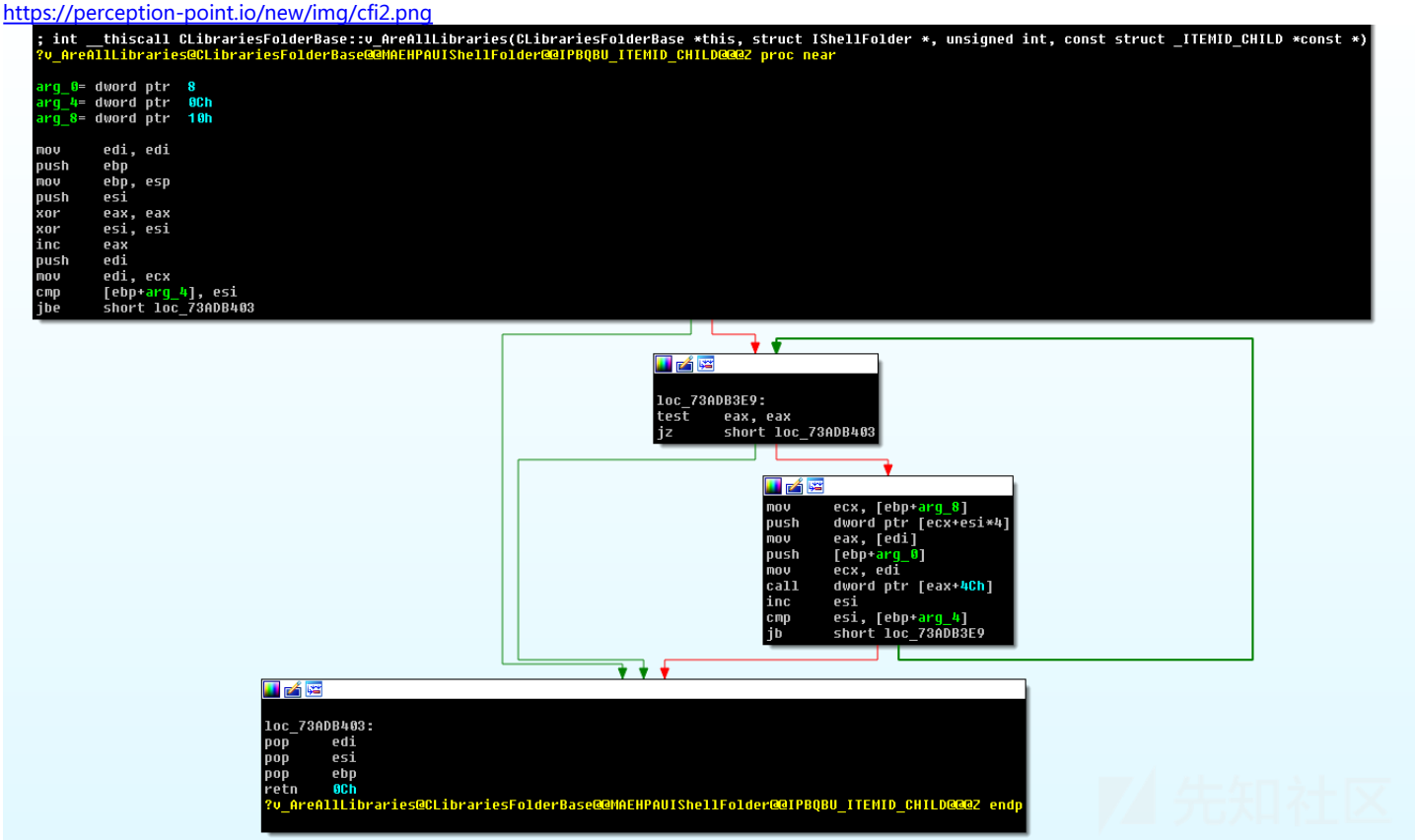
vfgadget 2

为了找到这样的vfgadget，我们使用IDA脚本搜索了shell32.dll，其中包含以下约束：

1. 这是一个虚函数（它从vtable交叉引用）
2. 它有一个涉及寄存器的间接调用
3. 它接收3个参数，像我们的Magic方法（这里会由"retn X"操作码检测）
4. 它将2个参数传递给间接被调用函数（通过从 push 的参数个数中减去 pop 的个数来检测）
5. 它小于0x30字节 - 毕竟，我们不需要长而复杂的vfgadget。

我们查看了脚本结果并选择了CLibrariesFolderBase::v_AreAllLibraries。
此vfgadget实际上是一个mainloop
gadget（ML-ARG-G），它在每次迭代中调用相同的虚函数（VTABLE的偏移量0x4c），并仅在函数返回成功的错误代码（0）时停止迭代。

https://perception-point.io/new/img/cfi2.png



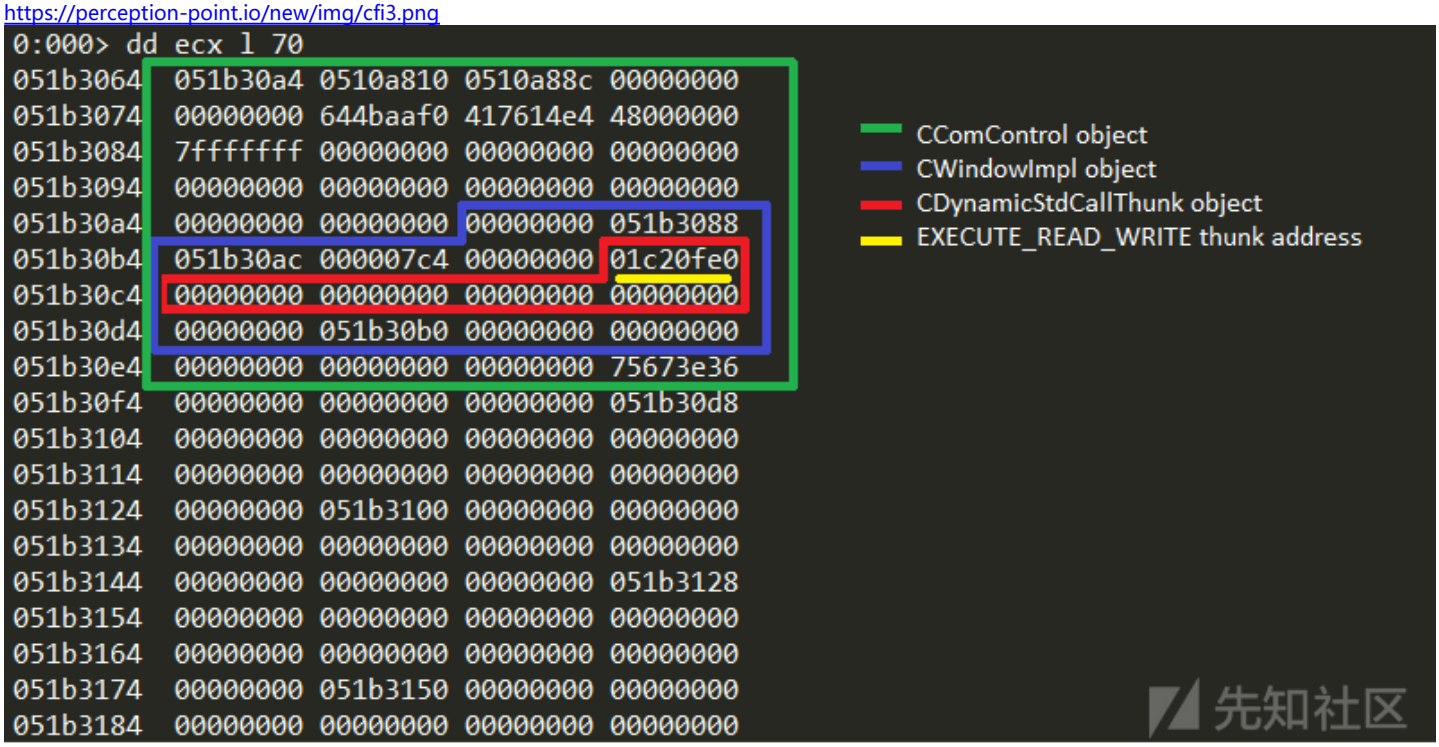
组合

伪造我们的对象，magic 方法指针 [vtable + 0x18] 将指向我们的ML-ARG-G，而ML-ARG-G内部调用vtable的偏移[vtable + 0x4c]将指向CreateControlWindow vfgadget：

```
// first, we save the current this pointer, to recover it later
original_this = eba.read(magic_object);

var magic_vtable:uint = magic_object+0x40;

// now, lets put a fake vptr at [magic_object]
eba.write(magic_object, magic_vtable);

// [vtable+0x18] will hold the first vfgadget that will be invoked
// it will be the ML-ARG-G we found in shell32.
eba.write(magic_vtable + 0x18, ml_arg_g);

// [vtable+0x4c] will hold the second vfgadget that will be invoked
// It will be the createWindow vfgadget we found in shell32
eba.write(magic_vtable+0x4C, createWindow_g)

Now, invoking the Magic method will perform our COOP flow:
```

```
eba.write(magic + 0x1c, 0x0)
eba.write(magic + 0x20, magic_object+0x100)
var args:Array = new Array(0x41)
Magic.call.apply(null, args);
eba.write(magic_object, original_this);
```

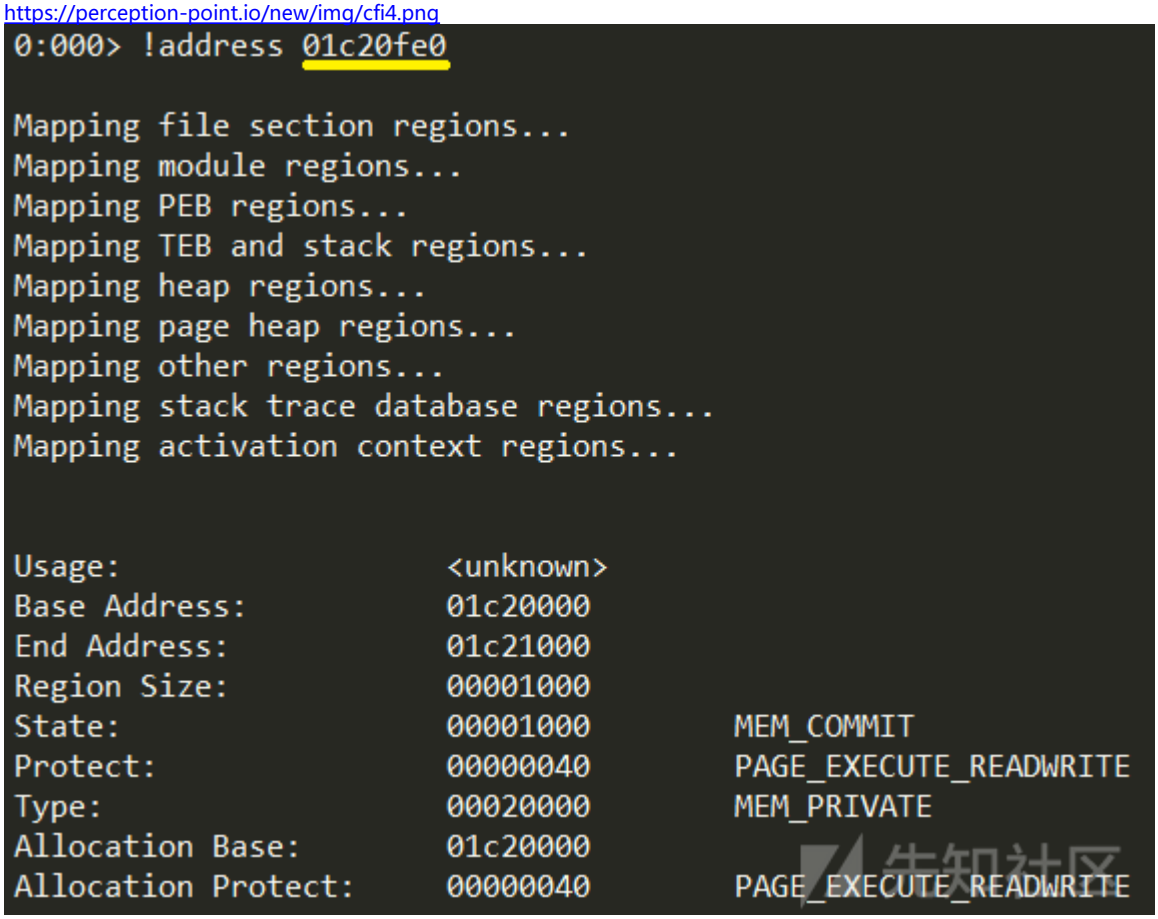
https://perception-point.io/new/img/cfi3.png



当我们的COOP流程结束时，我们可以读取创建的READWRITE_EXECUTE分配的内存页指针。 它将存储在"this"指针的0x5c偏移处：

```
// createWindow allocated a page with EXECUTE_READWRITE protection, and stored a pointer to it on magic_object+5C
var allocated_address:uint = eba.read(magic_object+0x5c)

// get page base address
allocated_address = allocated_address&0xFFFFF000
```

https://perception-point.io/new/img/cfi4.png



现在，我们可以简单地将我们的shellcode写入allocated_address，将此地址放入magic vtable offset中，并再次调用magic方法来实现代码执行。

进一步思考

有几种方法可以使用COOP技术实现漏洞利用。 在研究期间，我们还在flash DLL（18.0.0.203）中找到了5个vfgadgets，它们执行以下操作：

1. 在我们选择的任意路径下创建2个子目录。
2. 将文件写入名为“digest.s”的路径。
3. 调用MoveFileEx API，因为我们可以完全控制源和目标参数（这会将文件“digest.s”重命名为“atl.dll”，这是最终vfgadget所必需的）。
4. 使用可控路径参数调用SetCurrentDirectory API。 我们可以使用它将进程的当前目录设置为包含我们的有效负载文件的路径。
5. 将LoadLibrary调用为“atl.dll”，它将加载我们的有效负载dll。

将所有这些vfgadgets组合在一起是可能的，但由于COOP的微妙特性需要更多时间。
如前所述，我们发现了一个简单的vfgadget，它为我们提供了一个带有READWRITE_EXECUTE保护的内存页面，所以我们决定在这种情况下采用更简单的路径。

结论

我们已经成功地证明了COOP如何通过符合上述约束来规避现代CFI实现方案

1. Microsoft CFG未检测到“magic”方法的原始函数的重定向，因为目标是二进制文件的合法函数。
2. 没有违反后端CFI策略，因为没有覆盖返回地址
3. 没有使用ROP链
4. 没有更改堆栈指针
5. COOP操作对象指针，无需操作代码中的指针
6. VirtualAlloc是从绕过EMET关键函数保护的合法偏移中调用的。

为了缓解COOP攻击，CFI实现必须考虑语言语义和上下文状态。
如前所述，在这种特殊情况下，应用应该验证虚拟调用点与相关对象的虚函数或任何其他类型的语义是否匹配的细粒度策略应来检测攻击。

原文

https://perception-point.io/new/breaking-cfi.php

上一篇：2018鹏程杯初赛 pwn - b... 下一篇：Discuz x3.4前台SSRF
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

RSS 关于社区 友情链接 社区小黑板