登录

Windows利用技巧：利用任意对象目录创建进行本地提权

Edvison / 2018-08-31 13:08:27 / 浏览数 2929 技术文章 技术文章 顶(0) 踩(0)

本文是Windows Exploitation Tricks: Exploiting Arbitrary Object Directory Creation for Local Elevation of Privilege的翻译文章。

前言

我们又回到了关于Windows利用技巧系列。 这次我将详细说明我将如何利用问题1550，它能使用CSRSS特权进程的某些行为创建任意对象目录。
我将再次说明如何利用特定漏洞，我希望读者能够更好地了解Windows操作系统的复杂性，并向微软提供有关非内存损坏利用技术的信息，以便他们能够以某种方式缓解这

漏洞概述

对象管理器目录与普通文件目录无关。 使用一组单独的系统调用（如NtCreateDirectoryObject而不是NtCreateFile）创建和操作目录。
即使它们不是文件目录，也很容易受到许多和你在文件系统上发现的相同类别问题的影响，包括特权创建和符号链接劫持攻击。

问题1550中的漏洞允许在用SYSTEM权限运行时在用户可控制的位置内创建目录。 该bug的根源在于Desktop Bridge应用程序的创建。
AppInfo服务负责创建新应用，调用未记录的API，CreateAppContainerToken来执行一些内部管理。
不幸的是，此API在用户的AppContainerNamedObjects对象目录下创建对象目录，以支持OS重定向BaseNamedObjects和RPC端点。

由于在不模拟用户的情况下调用API（在CreateProcess中调用它一般不是个大问题），因此使用服务的标识（即SYSTEM）创建对象目录。
由于用户可以将任意对象写入其AppContainerNamedObjects目录，所以他们可以删除对象管理器符号链接，并将创建的目录重定向到对象管理器命名空间中的任何位置。
该目录是使用显式安全描述符创建的，该描述符允许用户完全访问，这对于利用来说将变得非常重要。

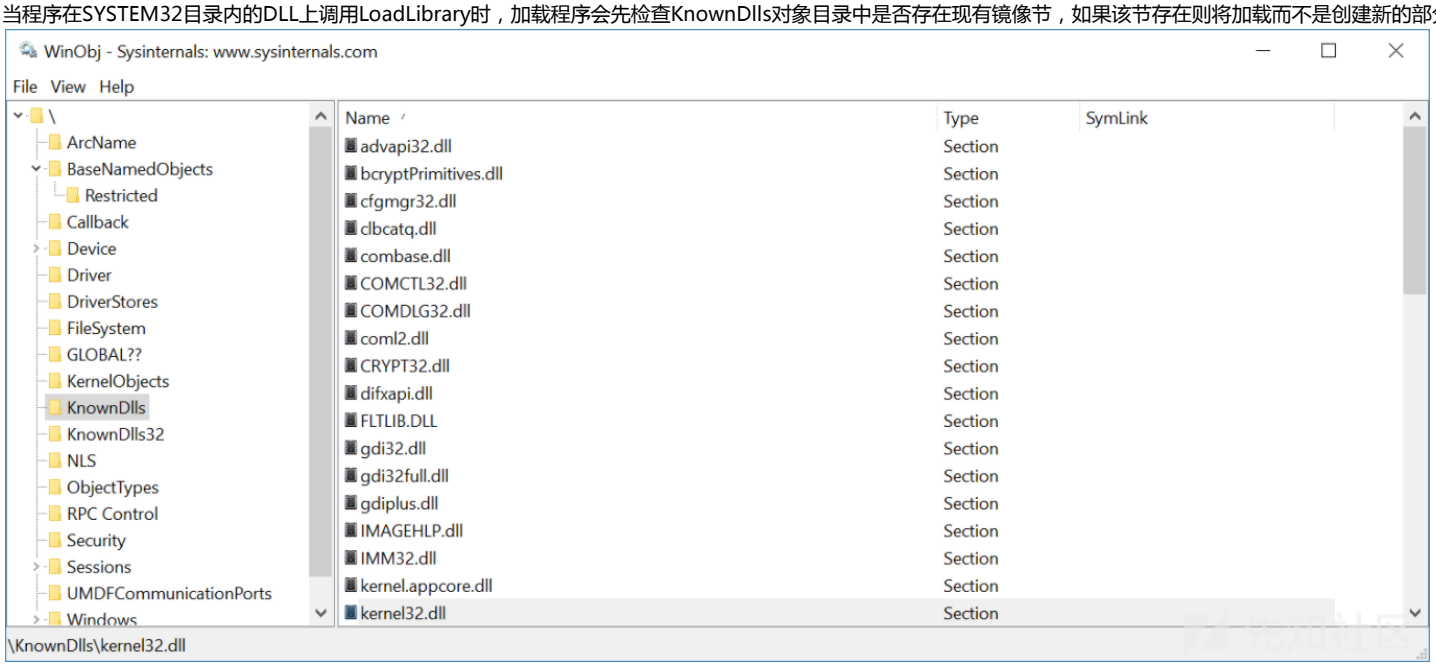
利用此漏洞的一个难点是，如果未在AppContainerNamedObjects下创建对象目录，但我们已经重定向其位置，那么执行令牌创建并在其操作过程中捕获目录句柄的基础N
可以创建目录，但很快就会被删除。
这种行为实际上是我报告的早期问题，它改变了系统调用的行为。这仍然可以通过在删除之前打开创建目录的句柄来实现，并且在实践中，只要你的系统具有多个处理器（基

这是我发送给MSRC的原始PoC停止的点，所有PoC都创建了一个任意对象目录。 你可以在问题跟踪器中找到此PoC附加到初始错误报告。
现在让我们深入了解如何利用此漏洞从普通用户提权得到特权SYSTEM用户。

利用

利用的主要问题是找个位置，并在里面创建一个对象目录，然后利用该目录来提升我们的权限。 事实证明这比你想象的要难。
虽然几乎所有Windows应用程序都使用底层的对象目录，例如BaseNamedObjects，但应用程序通常会与漏洞无法用于修改的现有目录进行交互。

本利用的对象目录是KnownDlls（我在本系列的前一篇博客中简要提到过）。 此对象目录包含NAME.DLL形式的命名图像节对象列表。
当程序在SYSTEM32目录内的DLL上调用LoadLibrary时，加载程序会先检查KnownDlls对象目录中是否存在现有镜像节，如果该节存在则将加载而不是创建新的部分对象。



KnownDll只能由管理员写入（严格来说我们不会看到），因此如果你能删除此目录中的任意节对象，那么也能强制系统服务加载指定的DLL，例如使用Diagnostics Hub服务 我在上一篇博文中描述过，它会映射节，而不是磁盘上的文件。
但是，除了添加一个无助于利用的新子目录之外，该漏洞不能用于修改KnownDlls对象目录。 也许我们可以通过利用漏洞使用的其他功能来间接定位KnownDlls？

每当我研究产品的特定部分时，我总会记下有趣或意外的行为。在我研究Windows符号链接时发现了一个有趣的例：Win32
API支持名为DefineDosDevice的函数，此API的目的是允许用户定义新的DOS驱动器号。 API需要三个参数，一组标志，要创建的驱动器前缀（例如X

:)和映射该驱动器的目标设备。 API的主要用途和CMD的SUBST命令相似。

在现代版本的Windows上，此API在用户自己的DOS设备对象目录中创建一个对象管理器符号链接，该位置可由普通的低权限用户帐户写入。
但是，如果你看一下DefineDosDevice的实现，你会发现它没有在调用者的进程中实现。
相反，该实现在当前会话的CSRSS服务中调用RPC方法，特别是BASESRV.DLL中的BaseSrvDefineDosDevice方法。
调用特权服务的主要原因是它允许用户创建永久符号链接，当符号链接对象的所有句柄都关闭时，该链接不会被删除。
通常，要创建永久命名的内核对象，你需要SeCreatePermanentPrivilege权限，但普通用户没有该权限。
另一方面CSRSS会这样做，通过调用该服务，我们可以创建永久符号链接。

创建永久符号链接的能力当然很有趣，但如果我们仅限于在用户的DOS设备目录中创建驱动器号，那它也不是特别有用。
我还注意到，实现从未验证lpDeviceName参数是否为驱动器号。 例如，你可以指定名称"GLOBALROOT\RPC Control\ABC"，它实际上会在用户的DosDevices目录之外创建一个符号链接，特别是在这种情况下，路径为"\RPC Control\ABC"。
因为实现将DosDevice的前缀"\??"添加到设备名称并将其传递给NtCreateSymbolicLink。内核将遵循完整路径，找到GLOBALROOT，它是返回根的特殊符号链接，然后按
目前还不清楚这是否是固定行为，所以我更深入地研究了CSRSS的实施情况，CSRSS的缩写形式如下所示。

```
NTSTATUS BaseSrvDefineDosDevice(DWORD dwFlags,
                                LPCWSTR lpDeviceName,
                                LPCWSTR lpTargetPath) {
  WCHAR device_name[];
  snwprintf_s(device_name, L"\\??\\%s", lpDeviceName);
  UNICODE_STRING device_name_ustr;
  OBJECT_ATTRIBUTES objattr;
  RtlInitUnicodeString(&device_name_ustr, device_name);
  InitializeObjectAttributes(&objattr, &device_name_ustr,
                             OBJ_CASE_INSENSITIVE);

  BOOLEAN enable_impersonation = TRUE;
  CsrImpersonateClient();
  HANDLE handle;
  NTSTATUS status = NtOpenSymbolicLinkObject(&handle, DELETE, &objattr);①
  CsrRevertToSelf();

  if (NT_SUCCESS(status)) {
      BOOLEAN is_global = FALSE;

      // Check if we opened a global symbolic link.
      IsGlobalSymbolicLink(handle, &is_global); ②
      if (is_global) {
          enable_impersonation = FALSE; ③
          snwprintf_s(device_name, L"\\GLOBAL??\\%s", lpDeviceName);
          RtlInitUnicodeString(&device_name_ustr, device_name);
      }

      // Delete the existing symbolic link.
      NtMakeTemporaryObject(handle);
      NtClose(handle);
  }

  if (enable_impersonation) { ④
      CsrRevertToSelf();
  }

  // Create the symbolic link.
  UNICODE_STRING target_name_ustr;
  RtlInitUnicodeString(&target_name_ustr, lpTargetPath);

  status = NtCreateSymbolicLinkObject(&handle, MAXIMUM_ALLOWED,
                             objattr, target_name_ustr); ⑤

  if (enable_impersonation) { ⑥
      CsrRevertToSelf();
  }
  if (NT_SUCCESS(status)) {
      status = NtMakePermanentObject(handle); ⑦
      NtClose(handle);
  }
  return status;
}
```

我们可以看到代码所做的第一件事就是构建设备名路径，然后尝试打开DELETE访问①的符号链接对象。
这是因为API支持重新定义现有的符号链接，因此必须先尝试删除旧链接。如果我们遵循链接不存在的默认路径，我们将看到代码模拟调用者（在这种情况下为低权限用户）
没有什么太令人惊讶的，至于为什么我们可以创建任意符号链接，是因为所有代码都是在传递的设备名称前加上“\??”。
由于代码在执行任何重要操作时模拟调用者，因此我们只能在用户可以写入的位置创建链接。

更有趣的是中间条件，其中为DELETE访问打开目标符号链接，这是调用NtMakeTemporaryObject所必需的。
打开的句柄传递给另一个函数②，IsGlobalSymbolicLink，并根据该函数的结果设置禁用模拟的标志，然后使用全局DOS设备位置\GLOBAL再次重新创建设备名称？
对于前缀③，什么是IsGlobalSymbolicLink呢？ 我们再一次可以重新启动该功能并进行检查。

```
void IsGlobalSymbolicLink(HANDLE handle, BOOLEAN* is_global) {
  BYTE buffer[0x1000];
  NtQueryObject(handle, ObjectNameInformation, buffer, sizeof(buffer));
  UNICODE_STRING prefix;
  RtlInitUnicodeString(&prefix, L"\\GLOBAL??\\");
  // Check if object name starts with \GLOBAL??
  *is_global = RtlPrefixUnicodeString(&prefix, (PUNICODE_STRING)buffer);
}
```

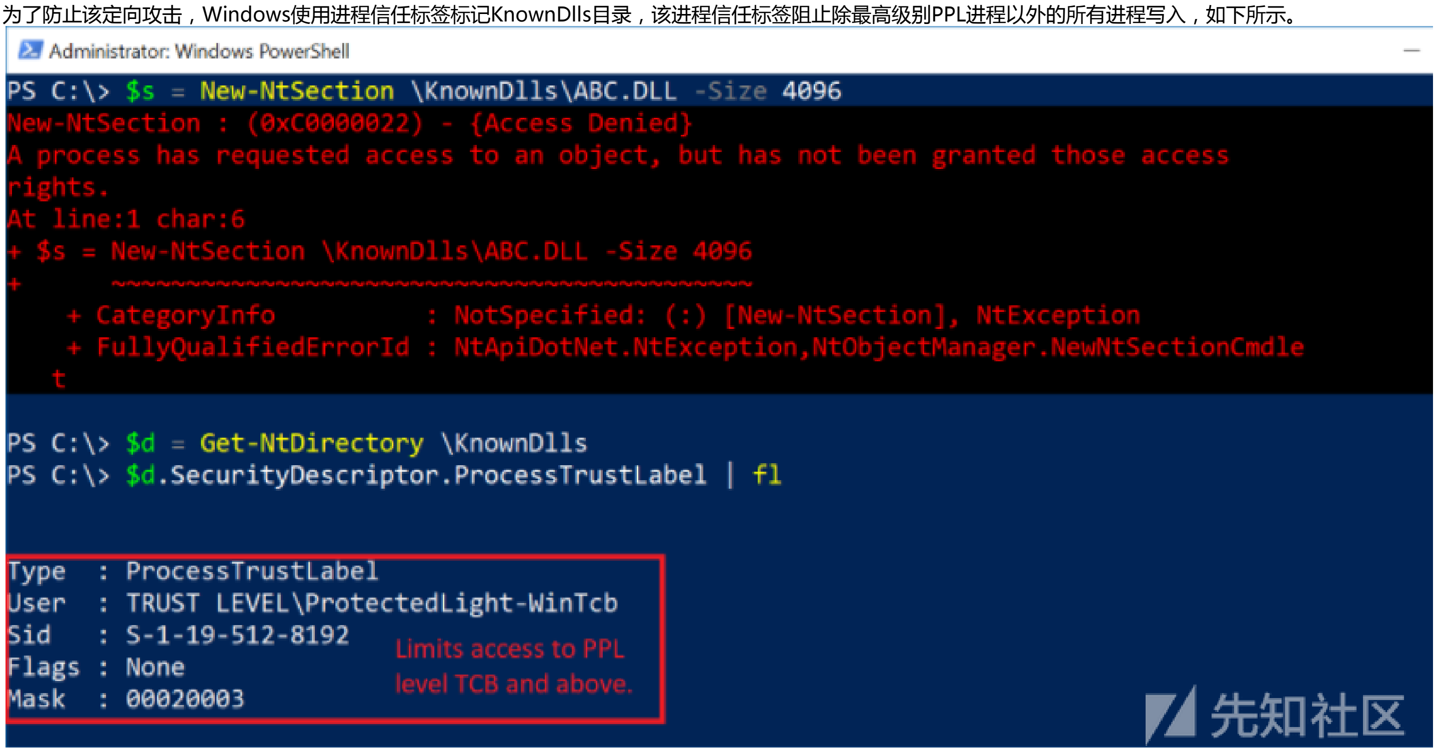
此代码会检查打开的对象的名称是否以\GLOBAL
??\开头。如果是，它将is_global标志设置为TRUE。使该标志模拟被清除并设备名称被重写。这意味着如果调用者具有对全局DOS设备目录内的符号链接的DELETE访问权限
。猜猜看，我们有一个漏洞可以让我们在全局DOS设备目录下创建一个任意对象目录。

如果不是为了重写路径，这并不是非常可利用的。 我们可以利用路径“\??\ABC”与“\GLOBAL
??\ABC”不同的特点，以构造在对象管理器命名空间中作为SYSTEM创建任意符号链接的机制。 这对我们有什么帮助？
如果你编写了一个指向KnownDlls的符号链接，那么在打开DLL加载程序请求的部分时，内核将会跟随它。
因此，即使我们无法在KnownDlls中直接创建新的节对象，我们也可以创建一个符号链接，该链接指向该目录之外的低权限用户可以创建节对象的位置。
我们现在可以利用劫持将任意DLL加载到特权进程内的内存中，并实现提权。

把这些全总结在一起，我们可以使用下面的步骤来利用漏洞：

1. 使用此漏洞创建目录“\GLOBAL ??\KnownDlls”
2. 在新目录中创建一个符号链接，其中包含要劫持的DLL的名称，例如TAPI32.DLL。 此链接的目标无关紧要。
3. 在用户的DOS设备目录中创建一个名为“GLOBALROOT”的新符号链接，指向“\GLOBAL ??”。
   当调用者通过用户的DOS设备目录访问它时，将覆盖真正的GLOBALROOT符号链接对象。
4. 调用DefineDosDevice，指定设备名称“GLOBALROOT\KnownDlls\TAPI32.DLL”以及用户可以在其中创建节对象的位置的目标路径。 这将导致以下情况：
   a. CSRSS打开符号链接“\??\GLOBALROOT\KnownDlls\TAPI32.DLL”，然后打开“\GLOBAL ??\KnownDlls\TAPI32.DLL”。
   由于这是由用户控制的，所以能成功打开，并且该链接被视为全局，禁用模拟。
   b. CSRSS将路径重写为“\GLOBAL ??\GLOBALROOT\KnownDlls\TAPI32.DLL”，然后调用NtCreateSymbolicLinkObject而不进行模拟。
   这使得之后真正的GLOBALROOT被链接，并创建具有任意目标路径的符号链接“\KnownDlls\TAPI32.DLL”。
5. 在目标位置为任意DLL创建映像节对象，然后通过获取服务来使用TAPI32.DLL的路径调用LoadLibrary，强制将其加载到特权服务（如诊断中心）中。
6. 实现提权

这实际上是第二次使用DefineDosDevice API进行利用，它是Protected Process Light （PPL）旁路管理员。
PPL进程仍然使用KnownDlls，因此如果你能添加新条目，那么也可以将代码注入受保护进程。
为了防止该定向攻击，Windows使用进程信任标签标记KnownDlls目录，该进程信任标签阻止除最高级别PPL进程以外的所有进程写入，如下所示。



那么我们的漏洞利用如何运作？ CSRSS实际上作为最高级别的PPL运行，因此允许写入KnownDlls目录。
一旦模拟被删除，就能使用进程的身份，这样就能获取完全访问权限。

如果你想测试这个漏洞我已经在这里将新的PoC附加到问题tracker。

总结

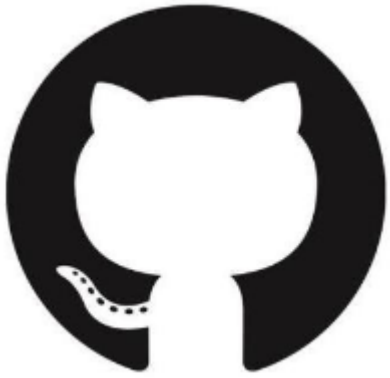
你可能想知道我是否向Node报告了DefineDosDevice的行为？ 我没有，主要是因为它本身并不是一个漏洞。
即使在PPL管理员的情况下，MSRC也不会考虑可用的安全边界（比如这个例子）。
当然，Windows开发人员可能会尝试在将来更改此行为，假设它不会导致兼容性的重大回归。 这个功能自Windows早期就已存在，至少从Windows XP开始，因此可能存在依赖它的东西。 通过详细描述这个漏洞，我想给MS提供尽可能多的信息，以便在将来解决这种开发技术。

我确实向MSRC报告了这个漏洞，并在2018年6月的补丁中得到修复。 Microsoft如何修复此漏洞？
开发人员添加了一个新的API，CreateAppContainerTokenForUser，它在创建新的AppContainer令牌期间模拟令牌。
通过在令牌创建期间模拟，代码可确保仅使用用户的权限创建所有对象。
由于它是一个新的API，因此必须更改现有代码才能使用它，因此你仍然可以在旧的CreateAppContainerToken的代码中找到可利用的漏洞。

在利用任何平台上的漏洞时，有时也需要深入了解不同组件的交互方式。
在这种情况下，虽然最初的漏洞显然是一个安全问题，但目前尚不清楚如何进行全面利用。
在逆向工程中遇到的有趣行为的日志总是值得记录下来的，即使某些东西本身不是安全漏洞，但它有可能对利用另一个漏洞有用。

上一篇：『功守道』软件供应链安全大赛·PE... 下一篇：Pwn2Own 2018 Safa...

1. 3 条回复


chybeta 2018-09-01 16:12:14

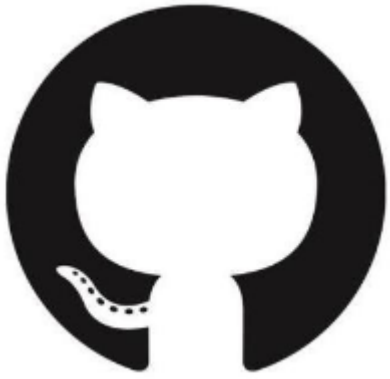
与 https://xz.aliyun.com/t/2586 重复了。

0 回复Ta


Edvison 2018-09-04 14:44:17

@chybeta 才发现= =不过这系列其他的还没翻译吧

0 回复Ta


chybeta 2018-09-04 23:15:03

@Edvison 没有

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