登录

Windows利用技巧系列之利用任意对象目录创建实现本地提权

mss**** / 2018-08-16 10:10:12 / 浏览数 3099 技术文章 技术文章 顶(0) 踩(0)

原文：https://googleprojectzero.blogspot.com/2018/08/windows-exploitation-tricks-exploiting.html

本文是Windows利用技巧系列的第二篇文章，在本文中，我们将为读者详细介绍如何利用Issue 1550漏洞，通过CSRSS特权进程来创建任意对象目录。我们之所以再次详细剖析特定漏洞的利用技巧，是为了帮助读者更好地认识Windows操作系统的复杂性，并向微软提

漏洞概述

对象管理器目录与普通文件目录是相互独立的，换句话说，它们会使用一组单独的系统调用（如NtCreateDirectoryObject而不是NtCreateFile）来创建和操作目录。虽然它

通过利用Issue 1550漏洞，攻击者不仅可以作为SYSTEM用户运行代码，同时，还能在处于用户控制下的位置内创建目录。该漏洞的根源，在于Desktop Bridge应用程序进程的创建过程。具体来说，是因为负责创建新应用程序进程的AppInfo服务，会调用一个未公开的API，即CreateAppContainerToken来执行一些内部管

由于这个API并非以用户的身份进行调用的（通常情况下，它是在CreateProcess中进行调用的，这样的话，就问题不大了），所以，这些对象目录实际上是以系统服务的身

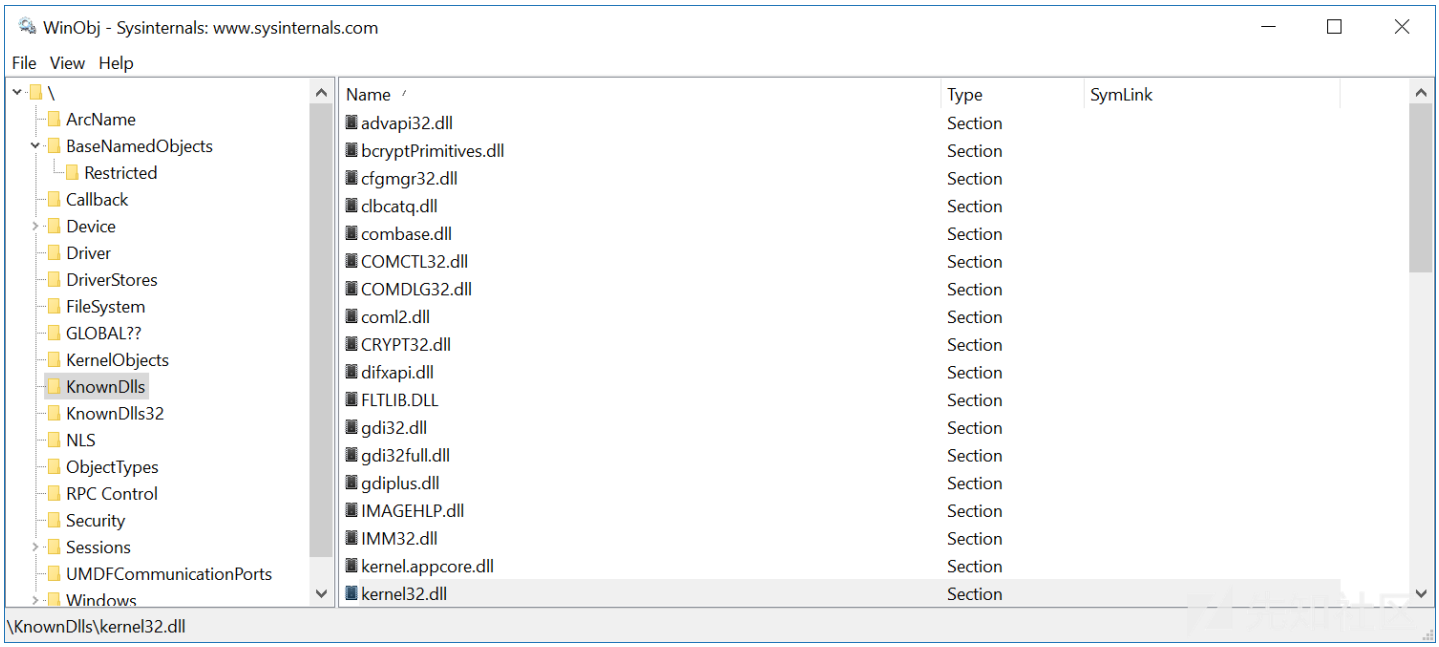
不过，该漏洞的一个利用难点是，如果没有在AppContainerNamedObjects下创建对象目录（比如，由于我们已对其位置进行了重定向），那么，完成令牌创建以及捕获目

实际上，我发送给MSRC的原始PoC的功能就到此为止了，该PoC所做的事情，无非就是创建了一个任意对象目录。读者可以在问题跟踪器中找到该PoC，它附加在原始漏洞

漏洞利用

要想利用该漏洞，关键问题是找到这样一个位置——我们能够在其中创建一个对象目录，并可以利用该目录来提升我们的权限。事实证明，这个问题要比我们想象的更难。

我们发现，一个有可能被滥用的对象目录是KnownDlls（我曾经在本系列的前一篇文章中简单提过它）。该对象目录包含了许多具有名称的映像节区（image section）对象，并且都是采取NAME.DLL形式进行命名。当应用程序调用LoadLibrary加载SYSTEM32目录内的DLL时，加载程序首先会检查映像节区是否已经存在于Know



严格来说，KnownDlls只允许管理员对其进行写操作（我们后面将会看到，实际上没有这么严格），因为如果您可以删除该目录中的任何节区对象的话，则可以强制系统服务Hub服务就可以达到这个目的，同时，它还能够映射节区，而非磁盘上的文件。但是，虽然该漏洞可以用来添加一个新子目录（这对于漏洞利用来说没有什么帮助），但是

每当我对某一产品的特定方面进行研究时，我总会记下值得注意或出乎意料的行为。例如，我在研究Windows符号链接时，一个行为就引起了我的注意。Win32 API提供了一个名为DefineDosDevice的函数，其的目是允许用户定义新的DOS驱动器号。该API需要三个参数，分别是一组标志，要创建的驱动器前缀（例如X:)和映射该驱SUBST命令非常类似。

在现代版本的Windows系统上，该API会在用户自己的DOS设备对象目录中创建一个对象管理器符号链接，我们知道，这是一个普通的低权限用户帐户可以写入的位置。但是

创建永久符号链接的能力固然值得我们关注，但是，如果我们只能在用户的DOS设备目录中创建驱动器号的话，那么这种能力也没有太大的用途。不过，我还注意到一个事实Control\ABC”，这样的话，它实际上会在用户的DosDevices目录之外创建一个符号链接，就这里来说，其路径为”\RPC Control\ABC”。之所以出现这种情况，是因为实现代码会将DosDevice前缀”\??”添加到设备名称，并将其传递给NtCreateSymbolicLink。内核将根据这个完整路径，找到

```
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;
}
```

如上所示，代码所做的第一件事就是构建设备名路径，然后尝试打开符号链接对象以便执行DELETE操作①。这是因为API支持重新定义现有的符号链接，因此，必须首先尝试

更值得关注的是中间的条件，即是否为DELETE操作打开了目标符号链接，这是调用NtMakeTemporaryObject所必需的。打开的句柄将传递给另一个函数②，即IsGlobalSy

```
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);
}
```

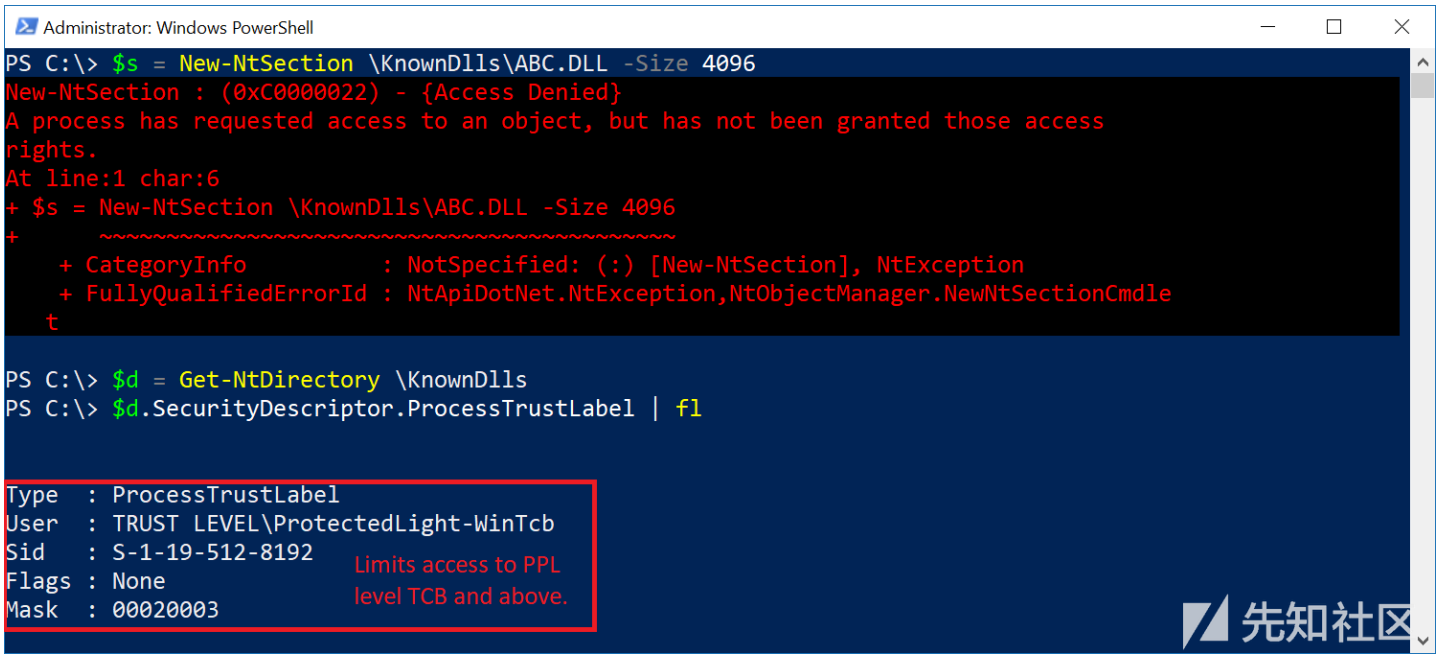
上述代码首先会检查打开的对象的名称是否以\GLOBAL??\开头。如果是的话，就将is_global标志设为TRUE。这样的话，就会导致启用身份切换的标志被清空，同时，设备
身份来重建链接。您猜怎么着，我们获得了一个允许我们在全局DOS设备目录下创建任意对象目录的漏洞。

同样，除非用于重写路径，否则，这个漏洞没有太大的利用价值。我们可以活用路径"\??\ABC"不同于"\GLOBAL??\ABC"这一事实，设法以SYSTEM身份在对象管理器命名空

综上所述，我们可以通过下列步骤来利用该漏洞：

1. 使用该漏洞创建目录“\GLOBAL??\KnownDlls”
2. 使用要劫持的DLL的名称（如TAPI32.DLL）在新目录中创建一个符号链接。注意，该链接的目标并不重要。
3. 在用户的DOS设备目录中创建一个名为“GLOBALROOT”的新符号链接，让其指向“\GLOBAL??”。当调用者通过用户的DOS设备目录访问它时，就会覆盖真正的GLOBAL
4. 调用DefineDosDevice，指定一个设备名称“GLOBALROOT\KnownDlls\TAPI32.DLL”，同事，将一个用户可以在其中创建节区对象的位置作为目标路径。这将导致以下
5. CSRSS打开符号链接“\??\GLOBALROOT\KnownDlls\TAPI32.DLL”，进而导致打开“\GLOBAL??\KnownDlls\TAPI32.DLL”。由于它是处于该用户的控制之下，因此打开
6. CSRSS将路径重写为“\GLOBAL??\GLOBALROOT\KnownDlls\TAPI32.DLL”，然后在不进行身份切换的情况下调用NtCreateSymbolicLinkObject。这样的话，会使用真
7. 在任意DLL的目标位置创建映像节区对象，然后强制将其加载到一个特权服务(如Diagnostics Hub)中，方法是让该服务使用一个指向TAPI32.DLL的路径调用LoadLibrary。
8. 实现提权。

实际上，针对DefineDosDevice API的滥用还有另一种用途，那就是绕过Protected Process Light（PPL）保护。PPL进程仍然在使用KnownDlls，因此，如果您可以向该目录中添加内容的话，就可以将代码注入该受保护进程中。为了防御这种攻击，Windows使用



那么，我们的漏洞利用的是如何得逞的呢？
实际上，CSRSS是作为最高级别的PPL运行的，因此它具有KnownDlls目录的写权限。一旦身份切换被废除，该进程的身份就会一直被沿用，也就是说，一直拥有全部的访问

如果你想测试这个漏洞利用的话，可以从这里下载新的PoC代码。

结束语

您可能想知道我是否MSRC报告了DefineDosDevice的这种行为？我没有，主要是因为它本身并不是一个漏洞。即使能够从Administrator权限提升到PPL权限，MSRC也不会
XP，因此，可能有些东西会依赖于它。我希望通过详细描述这个漏洞，给MS提供尽可能多的信息，以帮他们在将来克服这种漏洞利用技术。

我确实向MSRC报告了这个漏洞，并且，该漏洞已经在2018年6月的补丁中得到修复。那么，Microsoft是如何修复该漏洞的呢？开发人员添加了一个新的API，CreateAppC

无论利用哪种平台上的漏洞，有时都需要深入了解不同组件的交互方式。在这个例子中，虽然最初的漏洞显然是一个安全问题，但尚不清楚如何进行充分利用。在逆向工程中

上一篇：实战web缓存中毒 下一篇：CFG防护机制的简要分析

1. 0 条回复
   - 动动手指，沙发就是你的了！

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板