Windows通知功能(WNF)妙用

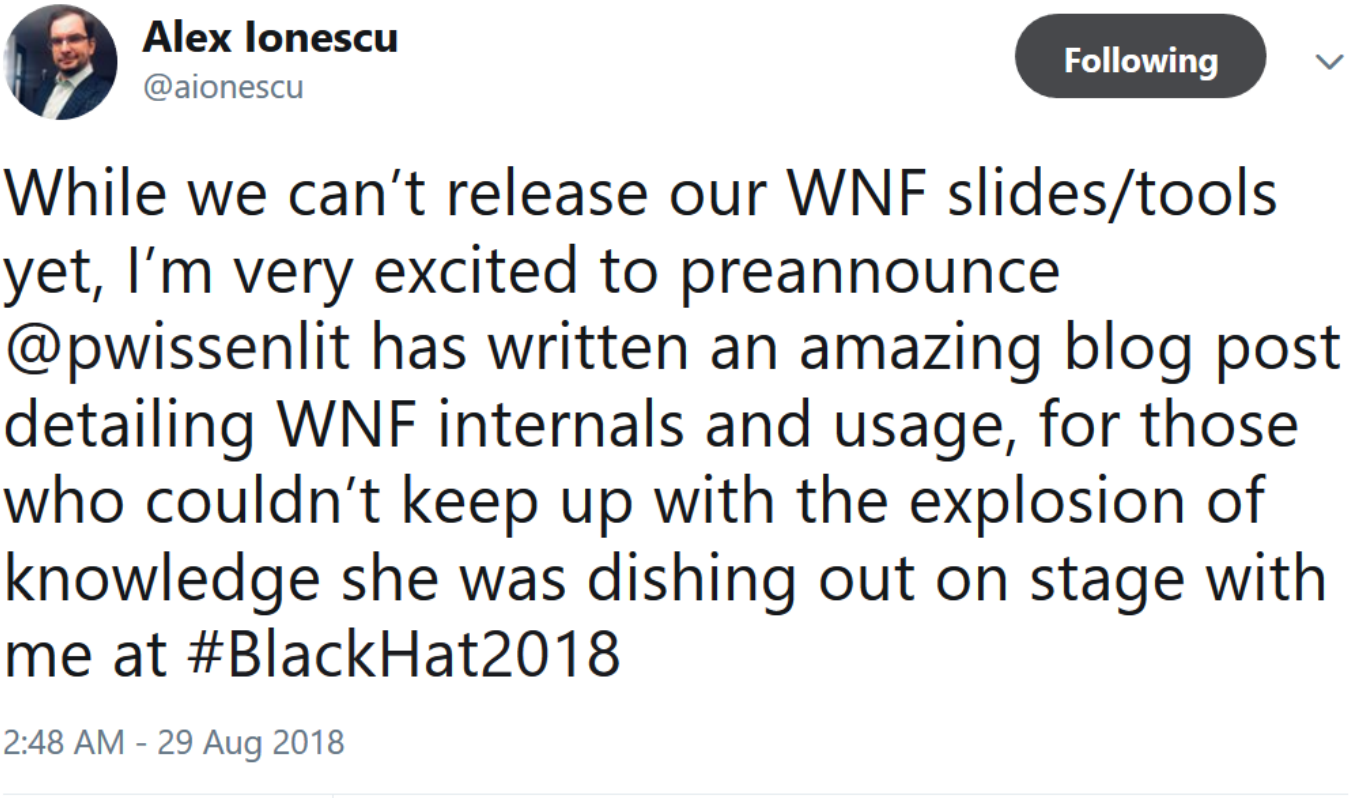
風見鶏 / 2018-10-30 07:30:00 / 浏览数 3566 技术文章 技术文章 顶(0) 踩(0)

前言

这篇博文简要介绍了Windows通知功能，并为Bruce Dang在2018年蒙特利尔侦查学院的研讨会上所做的一个精彩练习提供了一篇文章

序言

这篇文章主要是在Bruce Dang's Recon Montreal训练结束后几天写的，但是出于各种原因我决定推迟它的发表。我差点忘了这件事，直到Alex提醒我时我都还在犹豫。Bruce已经写了一个很好的关于WNF的帖子



介绍

不久前，Bruce Dang邀请五位BlackHoodies的女士参加他在Recon Montreal的Windows内核Rootkit培训。Barbie, Priya, Oryan, Aneal和我有机会来到这里接受为期四天的高强度工作。
在这篇博客文章中，我不会描述这门课的内容(相信我，它很棒)，但我会专注于我真正喜欢的练习之一:逆向和使用WNF!

数据输入

我根本不知道这个组件，在互联网上几乎没有关于它的信息。我所能得到的唯一输入是：

```
14. Reverse engineer the following Windows kernel functions.
   The result of this exercise will be used in the next exercise.

   • ExSubscribeWnfStateChange
   • ExQueryWnfStateData

15. Write a driver that gets notified when an application is using the microphone.
   Print its pid.

Hints:
   • check contentDeliveryManager_Utilities.dll for the WNF StateName.
   • some interesting info is available here:

http://redplait.blogspot.com/2017/08/wnf-ids-from-perfntcdll.html
```

一些背景

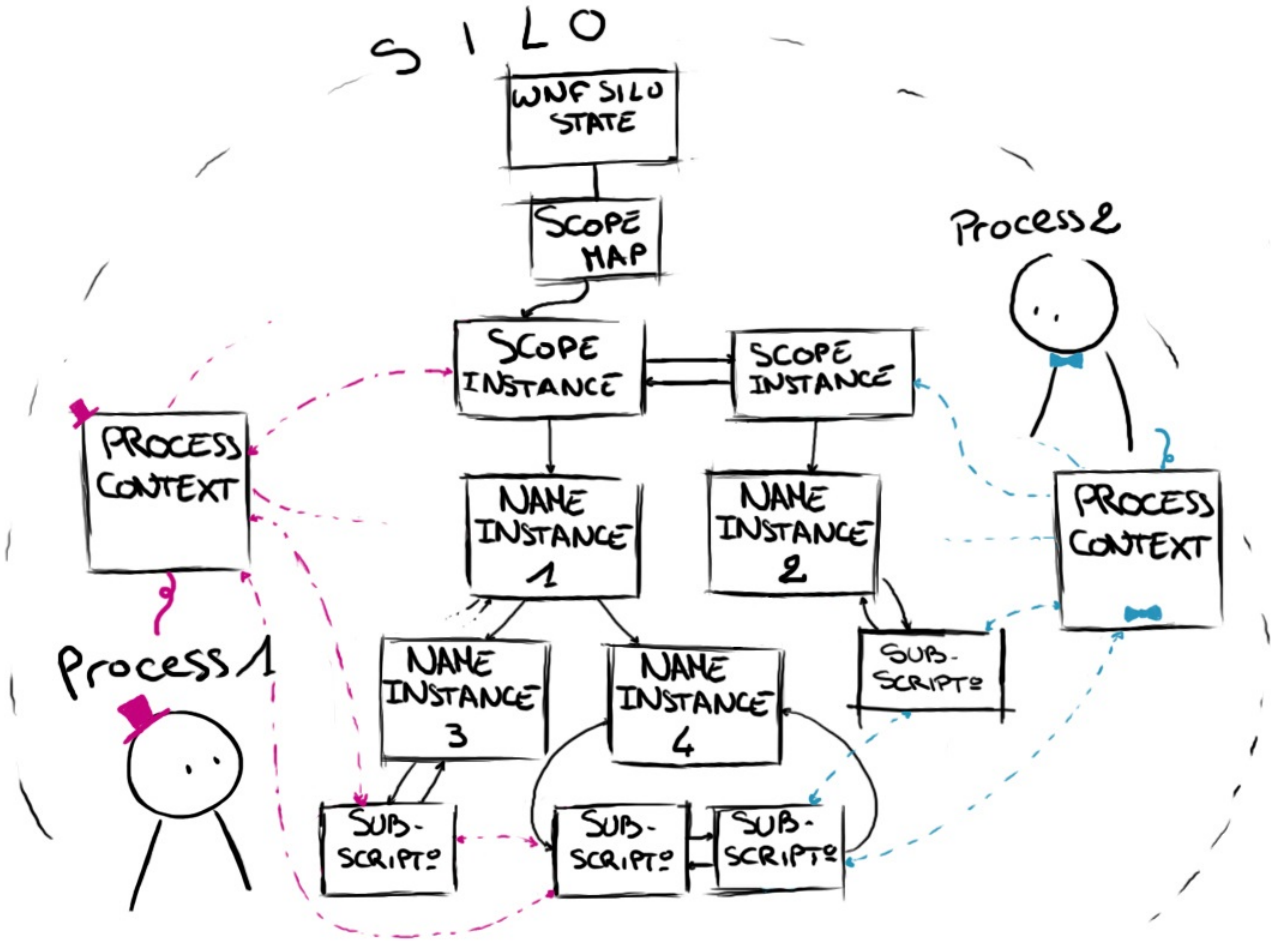
Windows通知功能(WNF)是一个(不是很知名的)内核组件，用于在系统中分发通知。它既可以在内核模式中使用，也可以在用户空间中使用，其中包含一组导出(但显然没有

- 应该注意，WNF状态名可以实例化到单个进程、模型(Windows容器)或整个机器。例如，如果应用程序在模型内运行，则只会在其容器内发生的模型作用域事件中通知它

在这篇博文中，我不会讨论在使用高级API时涉及到的用户空间机制:它们有点超出了练习的范围，解释可能会让这篇博文显得过于繁重。Alex Ionescu和我在BlackHat USA 2018年展会上就WNF的两种特性进行了深入的讨论，该视频和幻灯片将于2018年11月发布(该视频和幻灯片的发布尚待MSRC解决一些漏洞)。

数据结构

WNF中涉及到许多结构，下面是它们在内存中的关系的简化视图：



WNF状态名的事件或实例在内存中由WNF_NAME_INSTANCE结构表示。这些结构在二叉树中排序，并链接到事件发生的范围。作用域决定组件能够看到或访问哪些信息。

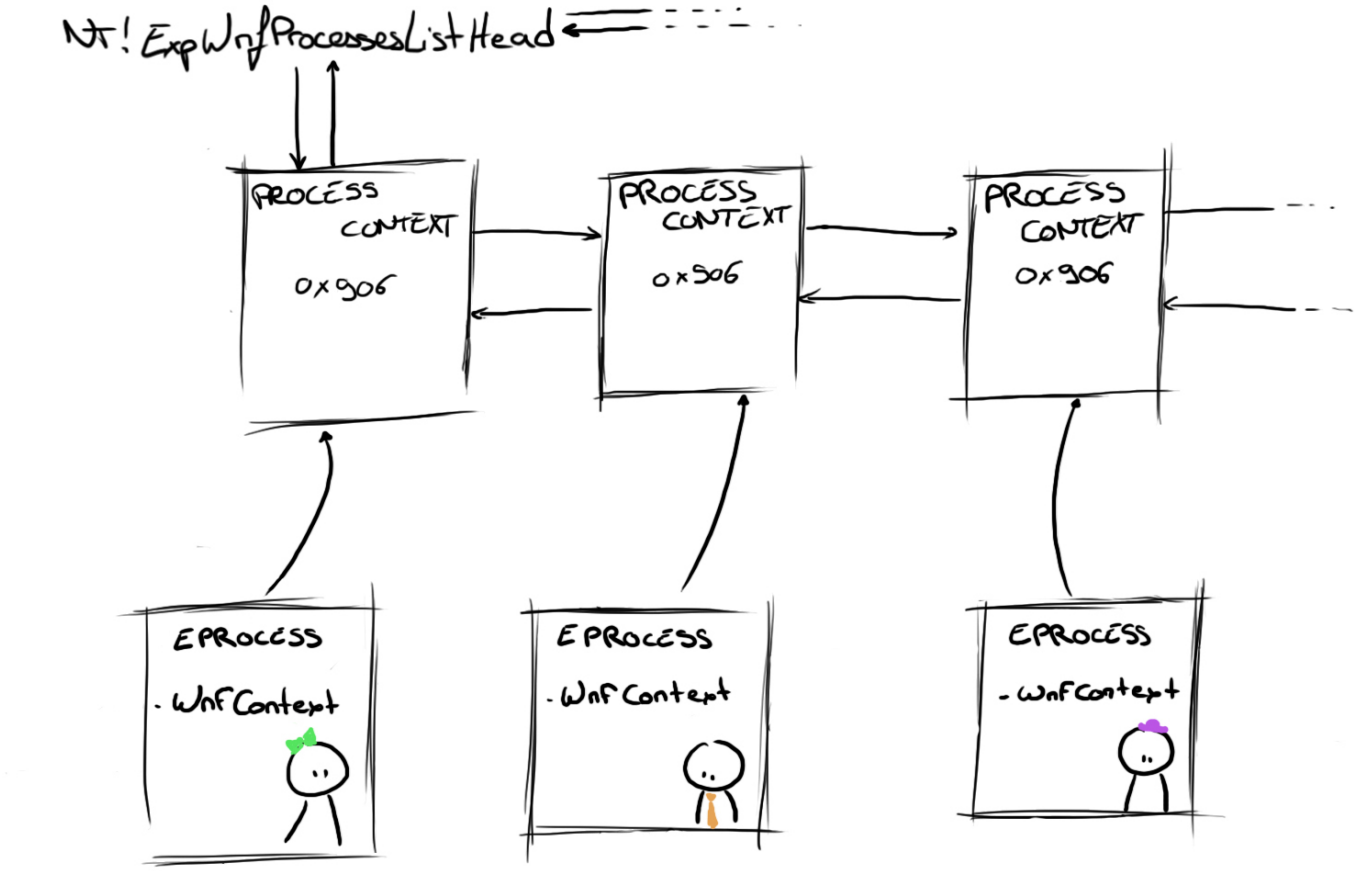
下面定义了五种可能的作用域类型:

```
typedef enum _WNF_DATA_SCOPE
{
 WnfDataScopeSystem = 0x0,
 WnfDataScopeSession = 0x1,
 WnfDataScopeUser = 0x2,
 WnfDataScopeProcess = 0x3,
 WnfDataScopeMachine = 0x4,
} WNF_DATA_SCOPE;
```

用WNF_SCOPE_INSTANCE结构标识的作用域按类型存储在双链接列表中，它们的头保存在特定模型的WNF_SCOPE_MAP中。

当组件认证WNF状态名时，将创建一个新的WNF_SUBSCRIPTION结构，并将其添加到属于关联的WNF_NAME_INSTANCE的链接列表中。如果认证者使用的是低级别的A

WNF_PROCESS_CONTEXT对象跟踪特定认证进程涉及的所有不同结构。它还存储用于通知流程的KEVENT。这个上下文可以通过EPROCESS对象访问，也可以通过爬行nt!



如果您想知道0x906指的是什么，那么这与WNF使用的所有结构都有一个描述结构类型和大小的小头(Windows文件系统相关数据结构中常见的情况)有关。

```
typedef struct _WNF_CONTEXT_HEADER
{
   CSHORT NodeTypeCode;
   CSHORT NodeByteSize;
} WNF_CONTEXT_HEADER, *PWNF_CONTEXT_HEADER;
```

这个头文件在调试时非常方便，因为很容易在内存中找到对象。下面是一些WNF结构的节点类型代码：

```
#define WNF_SCOPE_MAP_CODE          ((CSHORT)0x901)
#define WNF_SCOPE_INSTANCE_CODE     ((CSHORT)0x902)
#define WNF_NAME_INSTANCE_CODE      ((CSHORT)0x903)
#define WNF_STATE_DATA_CODE         ((CSHORT)0x904)
#define WNF_SUBSCRIPTION_CODE       ((CSHORT)0x905)
#define WNF_PROCESS_CONTEXT_CODE    ((CSHORT)0x906)
```

Reverse 时间

现在我们有了一些背景知识，让我们开始练习吧!第一部分实际上是颠倒下列功能，以便了解其目的:

- ExSubscribeWnfStateChange
- ExQueryWnfStateData

ExSubscribeWnfStateChange

```
NTSTATUS
ExSubscribeWnfStateChange (
   _Out_ptr_ PWNF_SUBSCRIPTION* Subscription,
   _In_ PWNF_STATE_NAME StateName,
   _In_ ULONG DeliveryOption,
   _In_ WNF_CHANGE_STAMP CurrentChangeStamp,
   _In_ PWNF_CALLBACK Callback,
   _In_opt_ PVOID CallbackContext
   );
```

ExSubscribeWnfStateChange允许在WNF引擎中注册新的认证。它将StateName作为参数之一，指定我们感兴趣的事件类型，并在触发通知时调用回调函数。它还返回一

在内部此函数仅将执行流传输到处理所有处理的私有对等体(ExpWnfSubscribeWnfStateChange)。

由于WNF状态名以不透明的格式存储，ExpWnfSubscribeWnfStateChange首先使用ExpCaptureWnfStateName解码ID的“clear”版本。

这个清晰的WNF状态名可以解码如下:

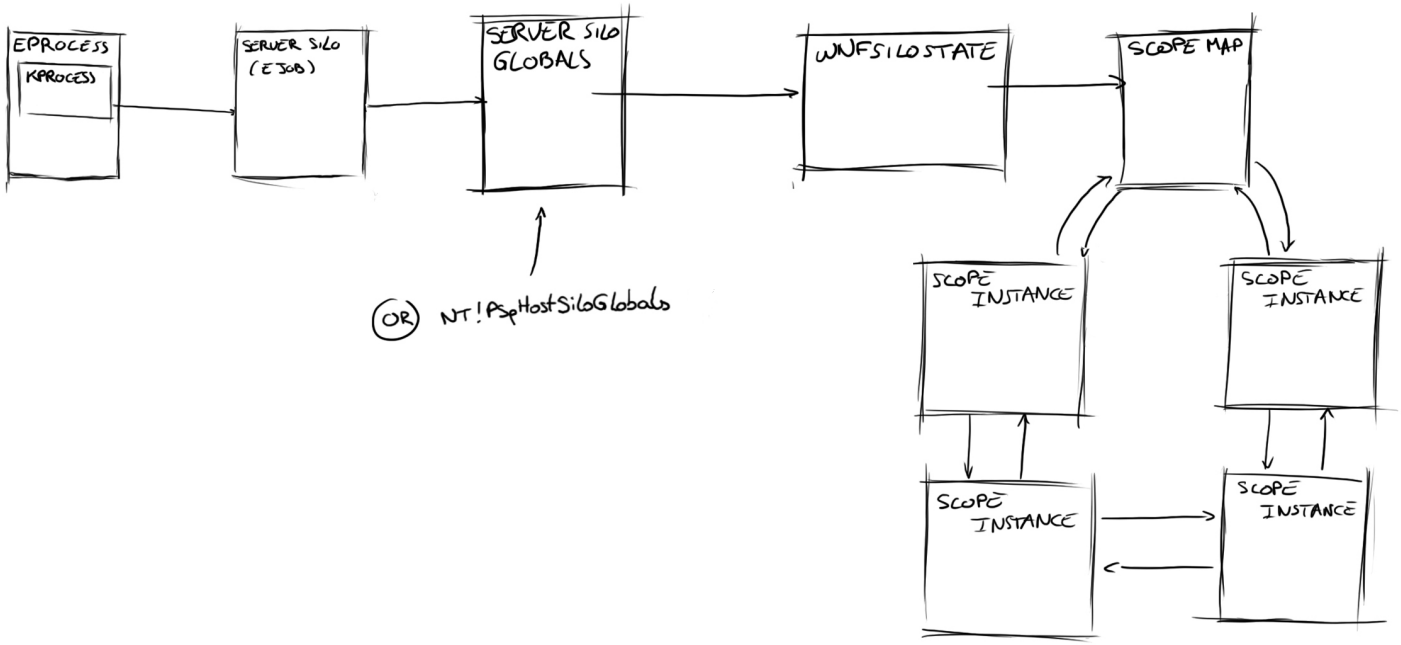
```
#define WNF_XOR_KEY 0x41C64E6DA3BC0074

ClearStateName = StateName ^ WNF_XOR_KEY;
Version = ClearStateName & 0xf;
LifeTime = (ClearStateName >> 4) & 0x3;
DataScope = (ClearStateName >> 6) & 0xf;
IsPermanent = (ClearStateName >> 0xa) & 0x1;
Unique = ClearStateName >> 0xb;
```

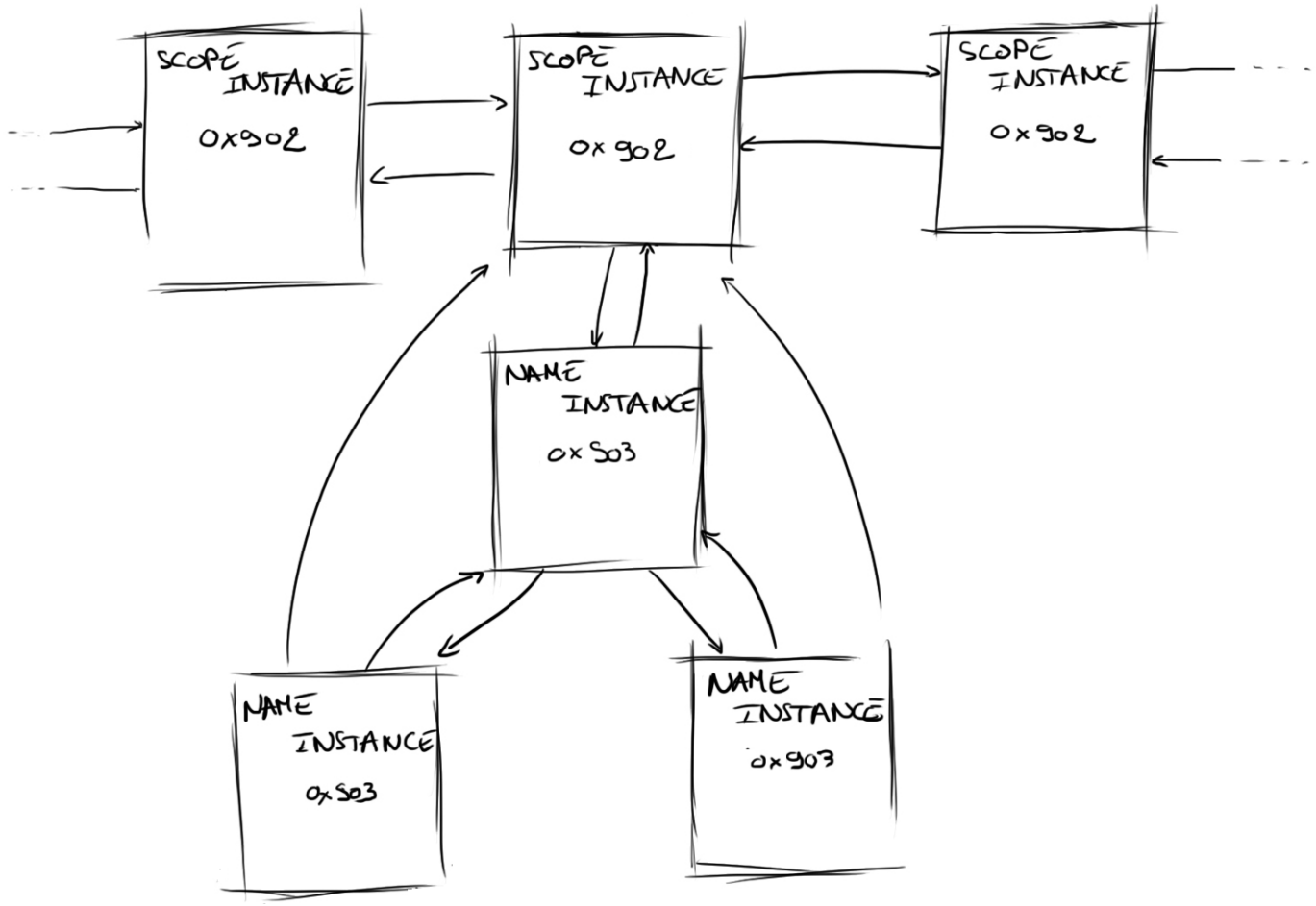
以一种更正式的方式，给出了以下结构：

```
typedef struct _WNF_STATE_NAME_INTERNAL
{
   ULONG64 Version : 4;
   ULONG64 Lifetime : 2;
   ULONG64 DataScope : 4;
   ULONG64 IsPermanent : 1;
   ULONG64 Unique : 53;
} WNF_STATE_NAME_INTERNAL, *PWNF_STATE_NAME_INTERNAL;
```

然后，ExpWnfSubscribeWnfStateChange调用ExpWnfResolveScopeInstance。后者检索服务器Silo全局(或nt!PspHostSiloGlobals(在不涉及服务器Silo的情况下)，并遍



从这个作用域实例结构中，ExpWnfSubscribeWnfStateChange(使用expwnfflood kupnameinstance)搜索匹配给定WNF状态名的WNF_NAME_INSTANCE:



如果没有找到匹配项，则使用ExpWnfCreateNameInstance创建一个新的WNF_NAME_INSTANCE。这个新实例被添加到从WNF_SCOPE_INSTANCE中根出的二叉树中。

该函数的下一步是调用ExpWnfSubscribeNameInstance来创建一个新的认证对象。正如前面解释的那样，这个对象将保存引擎触发通知所需的所有信息。

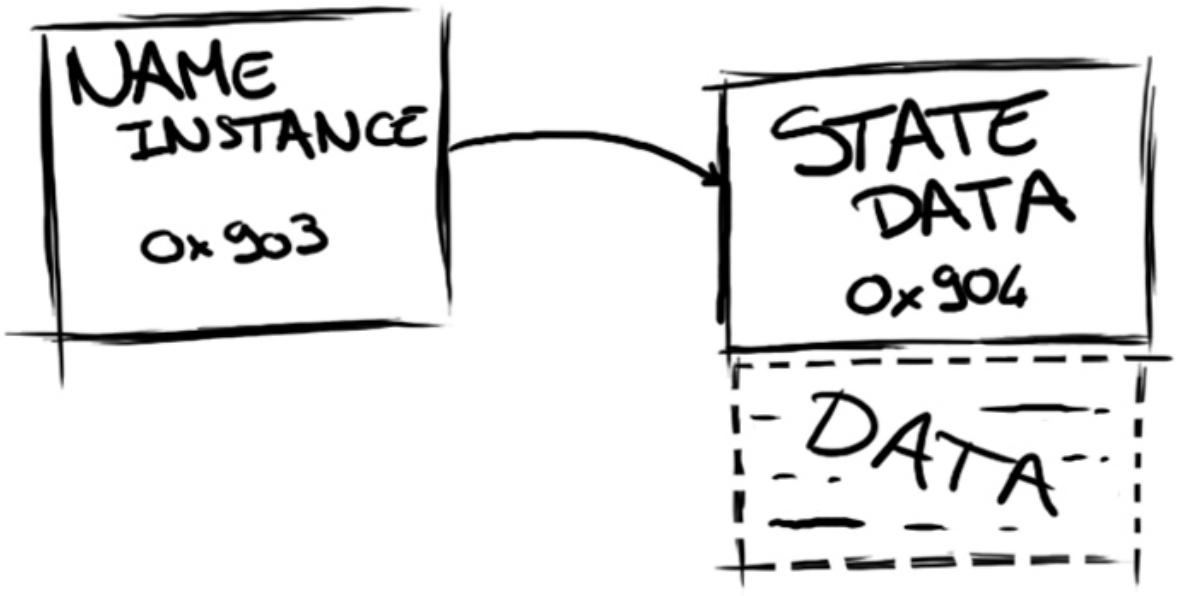
最后，ExpWnfSubscribeWnfStateChange调用ExpWnfNotifySubscription将新认证插入到挂起队列并触发通知。

ExQueryWnfStateData

```
NTSTATUS
ExQueryWnfStateData (
   _In_ PWNF_SUBSCRIPTION Subscription,
   _Out_ PWNF_CHANGE_STAMP ChangeStamp,
   _Out_ PVOID OutputBuffer,
   _Out_ OPULONG OutputBufferSize
);
```

这个函数非常简单，因为它只执行两个操作。首先，它使用ExpWnfAcquireSubscriptionNameInstance从认证中检索WNF_NAME_INSTANCE。然后，它使用ExpWnfRe

对于记录，所有WNF状态名都将其数据存储在内存中WNF_STATE_DATA结构下。这个结构包含各种元数据，例如数据大小和它被更新的次数(称为ChangeStamp)。指向W



Alex还希望我指出，WNF状态名可以标记为persistent，这意味着数据(和更改标签)将在重新引导时被保留(显然是通过使用辅助数据存储)。关于这一点的更多细节将在我们

编写代码

基本上，有了对该功能的逆向，我们应该能够注册一个新的认证，并作为任何其他使用WNF的合法应用程序而被通知。

然而，我们仍然缺少一个元素:找到麦克风输入所需的WNF状态名

我将只详细介绍驱动程序与WNF交互相关的部分。如果您对Windows上的驱动程序开发感兴趣，您可能想看看Windows驱动程序工具包文档及其示例，或者更好的是，直

寻找正确的WNF状态名

作为检索WNF状态名的提示，Bruce提供了博客文章的链接和库的名称(contentdeliverymanager_utility .dll)

在他们的博客中，Redplait定义了WNF使用的几个状态名。不幸的是，我们正在寻找的那个没有被列出。然而，这仍然给了我们一个好的开始，因为我们现在知道了WNF状

一个简单的方法就是在contentDeliveryManager_Utilities中为博客的WNF状态名之一grep.dll，希望其他id会在附近...幸运的是，这个工作得很好!通过对rda中匹配模式的

```
.rdata:00000001800E7A30                 dq offset WNF_SEB_GEOLOCATION
.rdata:00000001800E7A38                 dq offset aWnf_seb_geoloc ; "WNF_SEB_GEOLOCATION"
.rdata:00000001800E7A40                 dq offset aGeolocationSer ; "Geolocation service should be started"
.rdata:00000001800E7A48                 dq offset WNF_SEB_DEV_MNF_CUSTOM_NOTIFICATION_RECEIVED
.rdata:00000001800E7A50                 dq offset aWnf_seb_dev_mn ; "WNF_SEB_DEV_MNF_CUSTOM_NOTIFICATION_REC"...
.rdata:00000001800E7A58                 dq offset aOemCustomNotif ; "OEM custom notification received"
.rdata:00000001800E7A60                 dq offset WNF_SEB_MOB_OPERATOR_CUSTOM_NOTIFICATION_RECEIVED
.rdata:00000001800E7A68                 dq offset aWnf_seb_mob_op ; "WNF_SEB_MOB_OPERATOR_CUSTOM_NOTIFICATIO"...
.rdata:00000001800E7A70                 dq offset aMoCustomNotifi ; "MO custom notification received"
.rdata:00000001800E7A78                 dq offset WNF_SEB_CACHED_FILE_UPDATED
```

我们现在只需要找一个特定的麦克风(还记得这个练习吗?:p)

```
.rdata:00000001800E3680  dq offset WNF_AUDC_CAPTURE
.rdata:00000001800E3688  dq offset aWnf_audc_captu ; "WNF_AUDC_CAPTURE"
.rdata:00000001800E3690  dq offset aReportsTheNu_0 ; "Reports the number of, and process ids "...
// “Reports the number of, and process ids of all applications currently capturing audio.
//  Returns a WNF_CAPTURE_STREAM_EVENT_HEADER data structure”
```

应该注意，包含在Windows性能分析器中的perf_nt_c.dll库中同样的表也可以用。

订阅事件

要认证一个新事件，我们只需在驱动程序中调用ExSubscribeWnfStateChange，该驱动程序可以从上面找到的WNF状态名称。这个函数是导出的，但没有在任何标头中定义

这里唯一需要做的是调用具有正确参数的函数。

```
NTSTATUS
CallExSubscribeWnfStateChange (
   VOID
   )
{
   PWNF_SUBSCRIPTION wnfSubscription= NULL;
   WNF_STATE_NATE stateName;
   NTSTATUS status;

   stateName.Data = 0x2821B2CA3BC4075; // WNF_AUDC_CAPTURE

   status = ExSubscribeWnfStateChange(&wnfSubscription, &stateName, 0x1, NULL, &notifCallback, NULL);
   if (NT_SUCCESS(status)) DbgPrint("Subscription address: %p\n", Subscription_addr);

   return status;
}
```

定义回调

正如我们前面看到的，ExSubscribeWnfStateChange在其参数中包含一个回调，每次触发事件时都调用该回调。此回调将用于获取和处理与通知相关的事件数据。

```
NTSTATUS
notifCallback (
   _In_ PWNF_SUBSCRIPTION Subscription,
   _In_ PWNF_STATE_NAME StateName,
   _In_ ULONG SubscribedEventSet,
   _In_ WNF_CHANGE_STAMP ChangeStamp,
   _In_opt_ PWNF_TYPE_ID TypeId,
   _In_opt_ PVOID CallbackContext
   );
```

要获得回调中的数据，我们必须调用ExQueryWnfStateDataName。同样，这个函数是导出的，但没有在任何标头中定义，所以我们必须自己定义它。

```
NTSTATUS
ExQueryWnfStateData (
   _In_ PWNF_SUBSCRIPTION Subscription,
   _Out_ PWNF_CHANGE_STAMP CurrentChangeStamp,
   _Out_writes_bytes_to_opt_(*OutputBufferSize, *OutputBufferSize) PVOID OutputBuffer,
   _Inout_ PULONG OutputBufferSize
   );
[...]
```

我们需要调用这个API两次:一次是为了获得为数据分配缓冲区所需的大小，另一次是为了实际检索数据

```
NTSTATUS
notifCallback(...)
{
   NTSTATUS status = STATUS_SUCCESS;
   ULONG bufferSize = 0x0;
   PVOID pStateData;
   WNF_CHANGE_STAMP changeStamp = 0;

   status = ExQueryWnfStateDataFunc(Subscription, &changeStamp, NULL, &bufferSize);
   if (status != STATUS_BUFFER_TOO_SMALL) goto Exit;

   pStateData = ExAllocatePoolWithTag(PagedPool, bufferSize, 'LULZ');
   if (pStateData == NULL) {
       status = STATUS_UNSUCCESSFUL;
       goto Exit;
   }

   status = ExQueryWnfStateDataFunc(Subscription, &changeStamp, pStateData, &bufferSize);
   if (NT_SUCCESS(status)) DbgPrint("## Data processed: %S\n", pStateData);

   [...] // do stuff with the data

Exit:
   if (pStateData != nullptr) ExFreePoolWithTag(pStateData, 'LULZ');
   return status;
}
```

卸载驱动程序时的清理

如果你盲目地尝试上面的代码，你会得到一个丑陋的蓝屏，因为我痛苦地知道，我第一次尝试这个练习和卸载我的驱动!我们需要提前删除这个订阅。

为此，我们可以在驱动卸载例程中调用ExUnsubscribeWnfStateChange(并确保PWNF_SUBSCRIPTION wnfSubscription被构造为全局变量)

```
PVOID
ExUnsubscribeWnfStateChange (
   _In_ PWNF_SUBSCRIPTION Subscription
   );

VOID
DriverUnload (
   _In_ PDRIVER_OBJECT DriverObject
   )
{
  [...]
  ExUnsubscribeWnfStateChange(g_WnfSubscription);
}
```

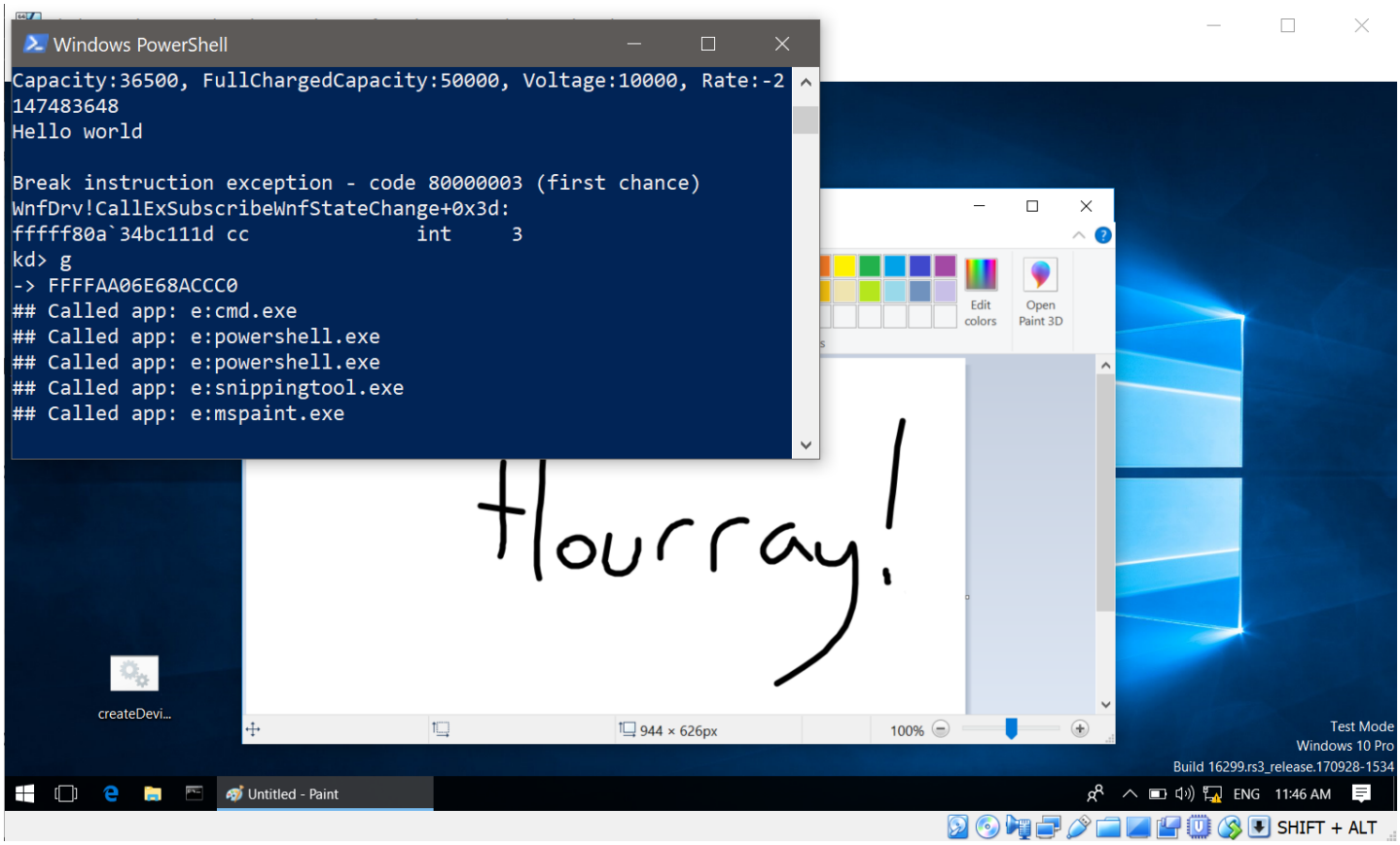
惊人的失败

我们现在要做的就是启动驱动程序，启用Cortana，使用它然后等待一会儿，等待事件触发。

然而!并没有什么!

我的实验结果完全失败了，因为我忘记了我的虚拟机上没有声卡(可能是我无法启动任何与声音相关的应用程序的原因吧)。最重要的是，由于我的主机配置，我根本无法让它

但是，为了确保我的驱动程序正确工作，我不得不选择另一个事件，并开始使用WNF_SHEL_DESKTOP_APPLICATION_STARTED。此通知在桌面应用程序启动时发出。作为



与WNF保持同步

我在前面展示了一种查找WNF名称的简单方法，方法是在包含该表的一个dll中搜索IDA中的名称。更可靠和可扩展的方法是通过解析DLL来查找表并转储它来检索WNF状态

我忘记在BlackHat杂志提到过，我现在想做一些广告。:^这个脚本可以用来区分两个dll，并快速获得表中的差异，以及从单个DLL中转储表数据。这个输出是Alex和我在我

```
$ python .\StateNamediffer.py -h
usage: StateNamediffer.py [-h] [-dump | -diff] [-v] [-o OUTPUT] [-py]
file1 [file2]

Stupid little script to dump or diff wnf name table from dll
```

```
positional arguments:
   file1
   file2 Only used when diffing

optional arguments:
   -h, --help show this help message and exit
   -dump Dump the table into a file
   -diff Diff two tables and dump the difference into a file
   -v, --verbose Print the description of the keys
   -o OUTPUT, --output OUTPUT Output file (Default: output.txt)
   -py, --python Change the output language to python (by default it's c)
```

输出示例(一旦11月的限制解除，我将发布脚本)

```
typedef struct _WNF_NAME
{
   PCHAR Name;
   ULONG64 Value;
} WNF_NAME, *PWNF_NAME;

WNF_NAME g_WellKnownWnfNames[] =
{
   {"WNF_A2A_APPURIHANDLER_INSTALLED", 0x41877c2ca3bc0875}, // An app implementing windows.AppUriHandler contract has been ins
   {"WNF_AAD_DEVICE_REGISTRATION_STATUS_CHANGE", 0x41820f2ca3bc0875}, // This event is signalled when device changes status of
   {"WNF_AA_CURATED_TILE_COLLECTION_STATUS", 0x41c60f2ca3bc1075}, // Curate tile collection for all allowed apps for current A
   {"WNF_AA_LOCKDOWN_CHANGED", 0x41c60f2ca3bc0875}, // Mobile lockdown configuration has been changed
   [...]
}
```

结论和感谢

多亏了这个练习，我有机会深入研究一个我根本不知道的内核组件，这个组件玩起来很有趣。我学到了很多东西，我真的很喜欢尝试弄明白如何使用WNF。我非常高兴在Re

本文翻译自： https://blog.quarkslab.com/playing-with-the-windows-notification-facility-wnf.html

上一篇：《阿里云安全报告》中文版开启全文下... 下一篇：搞定PatchGuard：利用KP...

1. 0 条回复
   - 动动手指，沙发就是你的了！