

Windows 本地提权在野 0day 狩猎之旅

Quan Jin <twitter: @jq0904>

DBAPPSecurity

Black Hat USA 2022

注：本文是作者在今年 Black Hat USA 2022 演讲议题的中文版

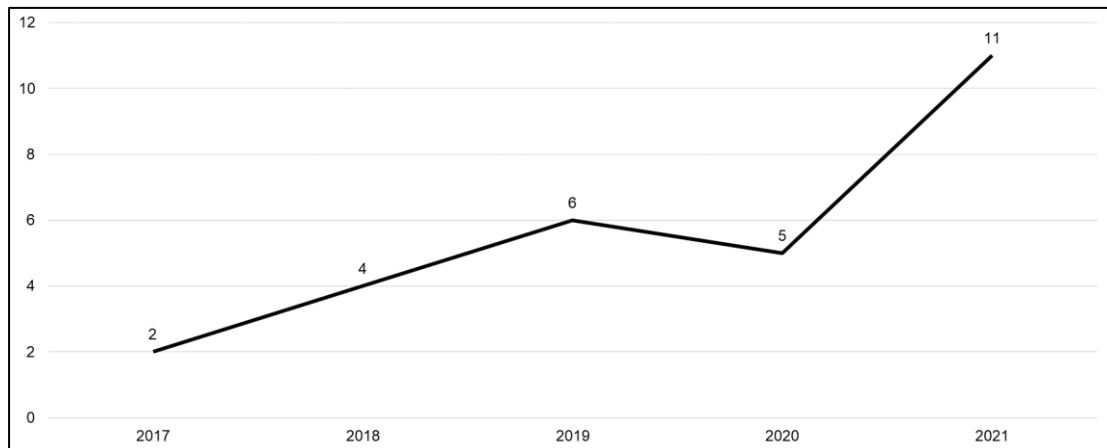
摘要

本文将讲述我们如何在 2020-2021 两年间狩猎 Windows 本地提权在野 0day 的故事：为什么我们觉得这件事可行，我们如何研究历史案例，我们如何用学到的经验开发一套检测系统，以及我们如何改进这套系统使之变得更准确。通过使用这套系统，我们成功捕获了两个 Windows 本地提权在野 0day 和一个 Windows 本地提权在野 1day。

此外，我们也将比较几种不同狩猎方法的优点和缺点，并给出一些对未来 Windows 本地提权在野 0day 的趋势预测。

背景

从下图可以看出，从 2017 年到 2021 年，微软一共披露了 28 个在野的 Windows 本地提权 0day，它们中的大部分是 Windows 内核漏洞。这些漏洞通常被顶级 APT 组织使用，并且会造成巨大危害。对安全厂商来说，捕获一个这样的在野 0day 是一件非常有挑战性的事情。



Windows 本地提权在野 0day (2017-2021)

是否有可能捕获一个 Windows 本地提权在野 0day?

为了回答这个问题，我们需要思考另外两个问题：

1. 如何获取有价值的数据源（可能含有在野 0day 的数据源）？
2. 如何开发一套有效的检测方法？

对于第一个问题，我们有一些自己的数据集。此外，历史案例表明像 VirusTotal 这类的公共

平台有可能出现 0day。因此，通过使用私有和公有数据集，我们可以解决第一个问题。

对于第二个问题，有两种方法可以从海量样本中捕获一个 0day：动态检测和静态检测。

- a) 动态检测是指的是在真实环境中进行检测，或在沙箱中进行模拟执行，并通过异常行为过滤出 0day 样本（典型代表是杀毒软件和沙箱）
- b) 静态检测是指使用静态特征匹配出样本（典型代表是 YARA）

两种方法都有其优点和缺点。我们对两种方法都进行了一些思考和尝试。测试结果表明，静态检测方法更适合我们。我们将在后文详细描述这个过程。

接下来，我将解释为什么我们花了大量时间研究历史上的 Windows 本地提权在野 0day。

从历史（和当下）中学习

为什么我们应当从历史中学习？

有三个原因：

- 1. 一些漏洞利用手法在一段时间内具有连续性
- 2. 从攻击者视角进行思考可以帮助我们更好地防御
- 3. 历史案例已经被整个安全社区仔细研究过（我们可以站在巨人的肩膀上）

我们如何学习历史案例？

为了从历史中学习，我们研究了超过 50 个 CVE 漏洞，涵盖了从 2014 年到 2021 年期间几乎所有的 Windows 本地提权在野 0day 和部分 1day。

我们仔细统计了这些漏洞的发现厂商、使用组织、修补周期、初始披露文章、使用场景、被攻击的系统版本、漏洞模块、漏洞类型、利用手法、公开的分析博客、公开的利用代码、原始样本（如果有）和其他信息。

这里我想重点说明其中的几个关键点。

使用场景：

- a) 样本是作为独立提权组件使用的，还是作为一个利用链的一部分？
- b) 利用代码是基于无文件形式使用的（例如 dll reflection），还是包含在一个落地文件中？

这些信息会直接影响我们对不同检测方案的选取。

被攻击的系统版本：

许多 Windows 本地提权样本在使用前会检测操作系统版本，并且只能在一些特定的版本中利用或触发。

这个信息在制作沙箱环境或漏洞复现环境时尤其有用。

漏洞模块：

通过统计历史样本的漏洞模块，我们可以得知哪个组件最多次被攻击，哪个攻击面在一段特定时间内最受攻击者青睐。

这些信息可以帮助我们预测接下来最可能出现的漏洞。

漏洞类型：

通过统计历史样本的漏洞类型，我们可以推断攻击者最喜欢使用的漏洞类型，这个信息可以帮助我们制作正确的复现环境（例如，是否需要配置 Driver Verifier）。这个信息也可以告诉我们不同漏洞类型的流行程度（例如竞争条件漏洞在最近几年逐渐增多）。

利用手法：

我认为这是最重要的信息。我们统计了大多数 Windows 本地提权在野 0day 的漏洞利用手法。基于这部分统计，我们得出了一些有价值的结论。例如，“bServerSideWindowProc”手法在 2015 年到 2016 年很流行；使用“Previous Mode”手法去实现任意地址读写的手法从 2018 年开始变得越来越流行；使用“HMValidateHandle”来泄露内核信息的手法在过去 5 年很流行。

公开的分析博客和利用代码：

公开的分析博客和利用代码包含整个社区的研究成果。吸收这些已存在的知识就像站在巨人的肩膀上，对我们帮助很大。

原始样本（如果有）：

我们仔细收集了历史 Windows 本地提权在野 0day 的原始样本（如果有）。这些原始样本的文件、哈希和利用手法都是第一手资料，如果我们能检测到这些样本，我们也能捕获到未来出现的相似样本。

为什么我们需要从当下学习？

除了从历史中学习，我们也应当从最新的漏洞和利用手法中学习。原因如下：

1. 一个新披露的漏洞可能会有变种（例如 CVE-2022-21882 漏洞是 CVE-2021-1732 的变种）
2. 一个最新被攻击的模块可能会被整个社区 fuzz 或审计（例如 clfs.sys）
3. 一个攻击者也许还有一些相似的漏洞正在使用或还未使用（例如，卡巴斯基基于 CVE-2021-1732 发现了 CVE-2021-28310）
4. 一种新的利用手法可能很快会被攻击者使用（例如“[Scoop the Windows 10 pool!](#)”这篇文章中的 Pipe Attribute 手法）

接下来，我将描述我们如何比较不同的检测方法，并从中选择一个作为主要检测方法。

一条大路通罗马

选择合适的工具

据我们所知，有三种可选的具体方法可以捕获一个在野的 Windows 本地提权漏洞：

1. 杀毒软件（或其他类似工具）
2. 沙箱（或其他类似工具）
3. YARA（或其他类似工具）

杀毒软件是最强有力的工具。它部署在大规模真实环境中，可以实时检测到威胁，并且有机会提取被加密的提权组件。在过去几年，卡巴斯基曾用他们的杀毒软件捕获了若干在野的 Windows 本地提权 0day。然而，不是每个厂商的杀毒软件都可以做到像卡巴斯基那样好。同时，杀毒软件经常会被绕过或被检测到，这些都增加了开发一套基于杀毒软件的本地提权漏洞狩猎方法的难度。

沙箱是另一个可以用来狩猎在野 0day 的工具。我曾有过一些使用沙箱成功捕获 Office 在野 0day 的经验。感兴趣的读者可以参考我之前在 Bluehat Shanghai 2019 上的[演讲](#)。

不像杀毒软件，沙箱环境高度可控，并且可以自由配置。此外，沙箱基于行为的检测使其结果十分准确。

然而，我认为沙箱在某种程度上不适合狩猎 Windows 本地提权 0day。不像 Office，许多本地提权漏洞利用会进行系统版本检测以避免不必要的蓝屏，这使得它们在面对沙箱时显得更隐蔽。你也许会觉得可以通过制作更多的环境来解决这个问题，但现今每日新增的 PE 文件数量巨大，每个样本都投递到一个新环境意味着巨大的资源开销，不是所有厂商都有足够的资金去供得起这种开销。

此外，基于沙箱的检测方法在狩猎 Windows 本地提权样本时还存在一些其他缺点：

1. 一些样本需要参数（例如，一个进程 id），但沙箱默认无法提供合理的参数
2. 一些样本只会导致蓝屏，而没有其他行为，这使得它们较难被检测到
3. 沙箱的开发和部署存在一个周期，这会导致错过一些最新漏洞样本的最佳检测周期

YARA 是第三种可以狩猎 Windows 本地提权在野 0day 的方法。它在检测含有特定特征的恶意样本方面有着很好的效果。

YARA 基本上没有技术壁垒，且 YARA 的规则无惧各种形式的检测（版本检测、绕过检测等）。YARA 的另一个优势是它在开发和部署上非常灵活。当一种新的漏洞利用手法出现，我们可以迅速将其转换为规则，并将其部署到检测系统。最后，YARA 规则的成本比杀毒软件和沙箱都低。

但 YARA 规则也有一些缺点，例如它可以轻易导致漏报和误报。因此，如果我们使用 YARA 去狩猎本地提权在野 0day，我们需要对历史案例非常熟悉，而这一点我们在之前已经完成。

以上，我们对比了三种方法在狩猎 Windows 本地提权 0day 方面的优缺点，基于这些比对并结合实际情况，我们最终选择了 YARA 作为我们的主要狩猎手法，对于 Windows 本地提权在野 0day 狩猎来说，它最容易上手、最灵活并且成本最低。

我们选择 YARA 的另一个原因是：在写了一些 YARA 规则后，我们回测了一些历史 Windows

本地提权样本。让我们意外的是，YARA 的表现超出我们的预期。

编写正确的规则

接下来我们将描述如何将之前学到的经验转换为 YARA 规则。

基本上，我们有三种思路：

1. 为漏洞利用各个阶段的特征编写规则
2. 为最新的漏洞和利用手法编写规则
3. 为最可能出现的漏洞编写规则

对于第一种思路（为漏洞利用各个阶段的特征编写规则），通常来说，一个 Windows 内核本地提权漏洞利用包含以下阶段：

- a) 漏洞触发
- b) 堆风水
- c) 内核信息泄露
- d) 任意地址读写
- e) 控制流劫持
- f) 权限提升

我们的任务就是基于每个阶段的通用特征写规则。以下是两个例子：

对于内核信息泄露阶段，思路是匹配通用的 Windows 内核信息泄露手法。包括但不限于下面这些：

- NtQuerySystemInformation
 - SystemBigPoolInformation
 - SystemModuleInformation
 - ...
- Win32k Shared Info User Handle Table
- Descriptor Tables
- HMValidateHandle
- GdiSharedHandleTable

对于任意地址读写原语，部分思路是匹配以下出现过的历史手法：

- SetWindowLong / SetWindowLongPtr
- SetWindowText / InternalGetWindowText / NtUserDefSetText
- GetMenuItemRect / SetMenuItemInfo / GetMenuBarInfo
- NtUpdateWnfStateData / NtQueryWnfStateData
- GetBitmapBits / SetBitmapBits
- GetPaletteEntries / SetPaletteEntries
- CreatePipe / NtFsControlFile
- Previous Mode + NtReadVirtualMemory / WriteVirtualMemory

需要指出的是，以上只是一些可能的思路，不是所有思路都适合 YARA 规则，其中一些思路

可能会导致大量误报。

对于第二种思路（为最新的漏洞和利用手法编写规则），这里我给出两个例子：

1. 2020 年 7 月，Synacktiv 公司的 Paul Fariello(@paulfariello)和 Corentin Bayet (@OnlyTheDuck)在 SSTIC 大会上提出了一种新的 Windows 内核堆溢出漏洞利用手法。在学习了他们的[白皮书](#)后，我们意识到在 PagedPool 中借助 Pipe Attribute 手法实现任意地址读取的利用方式具有通用性，在未来很可能被用到。因此我们花了一些时间为这种手法写了几条规则。事实证明这些规则后来捕获了一些高价值样本。
2. 2021 年 6 月 8 号，卡巴斯基写了一篇[博客](#)，披露了一个 Windows 本地提权在野 0day(CVE-2021-31956)。文章中提到样本使用 Windows Notification Facility (WNF) 实现了任意地址读写原语。随后在 2021 年 7 月和 8 月，NCC Group 公司的 Alex Plaskett(@alexjplaskett)发表了两篇[博客](#)，详细披露了 CVE-2021-31956 漏洞的利用细节，并解释了使用 WNF 构造任意地址读写原语的细节。与此同时，YanZiShuang(@YanZiShuang)也写了一篇[博客](#)讨论了借助 WNF 构造任意地址读写原语的手法。在学习了这些博客后，我们意识到这种方法是通用的，于是又花了一些时间为此写了几条规则。和预期的一样，我们确实捕获了一些高价值样本。

对于第三种思路（为最可能出现的漏洞编写规则），我这里也给出一个例子。2021 年 4 月 13 号，卡巴斯基写了一篇[博客](#)披露了 CVE-2021-28310，这是一个位于 Windows 桌面窗口管理器组件(Desktop Window Manager, 简称为 dwm)的在野 Windows 本地提权 0day。不到一个月后，ZDI 发表了另一篇[博客](#)，披露了另一个漏洞 (CVE-2021-26900)，这也是一个位于 dwm 组件的漏洞。这使我们意识到这种类型的漏洞在未来还会出现，所以我们在几个小时内为 dwm 漏洞写了几条规则。几周后，我们捕获了 CVE-2021-33739 这个 0day 漏洞。

编写出规则只是第一步。为了捕获一个 Windows 本地提权在野 0day，我们需要去构建一整个系统。

构建可行的系统

思考如下三个问题：

1. 当一个样本命中了规则，如何及时通知我们？
2. 当一个命中的样本推送给我们，如何快速复现和分类它？
3. 为了调试不同的 Windows 本地提权 0day，我们需要掌握什么技巧？

对于第一个问题，如果 YARA 运行在 VirusTotal 上，我们可以使用其 Hunting 页面的通知机制，我们可以在“Notify by email”一栏中填写用来接收通知的邮箱。这样，当一个新的样本被命中，我们的邮箱会在第一时间接收到通知。对于运行在我们自有产品上的规则，我们构建了一套类似 VirusTotal 的通知接口。

对于第二个问题，我们已经统计了每个历史案例中被攻击的系统版本，这些信息可以用在此处。此外，考虑到我们捕获的可能是 Nday、1day 或者 0day，因此我们需要制作这三种不同的环境，并且需要随时间流逝更新这三种环境。为了最小化复现时间，我们还制作了多种操作系统环境，包括 Windows 7、Windows 10 和 Windows 11，并且涵盖了 x86 和 x64。

第三个问题取决于我们调试不同样本的经验。举个例子，对于那些已经分析过 Windows 内核堆溢出漏洞的人来说，Windows 内核调试和 Driver Verifier 配置是两种基本技巧。此外，对于那些分析过 dwmcore 模块漏洞的人来说，使用 Windows 远程调试是必须的（因为直接附加到 dwm.exe 进程会导致系统 UI 无法响应）。我们积累的经验越多，就可以越好地回答这个问题。

测试和改进系统

没有方法是完美的，为了使系统更准确，我们做了如下测试和改进：

1. 使用收集到的历史 Windows 本地提权 0day 样本去测试规则，消除误报和漏报
2. 使用收集到的公开 poc/exploit 对规则进行测试，消除误报和漏报
3. 尝试编写 poc/exploit（对于一些无法收集到公开 poc/exploit 的漏洞案例），并对规则进行测试，消除误报和漏报
4. 将规则应用到大规模样本进行压力测试，消除观察到的漏报和误报
5. 持续将最新的漏洞和利用手法转化为规则，编写、测试规则并消除误报和漏报

这套系统部署一年以后，我们捕获了大量 Windows 本地提权漏洞样本。

在接下来的章节，我们将分享这套系统捕获的三个案例：

1. CVE-2021-1732：一个位于 win32k 子系统的 Windows 本地提权在野 0day
2. CVE-2021-33739：一个位于桌面窗口管理器（Desktop Window Manager）的 Windows 本地提权在野 0day
3. 未知 CVE：一个位于通用日志文件系统（Common Log File System）的 Window 本地提权在野 1day

首先来看第一个案例。

案例分享

CVE-2021-1732 的故事

2020 年 12 月 10 日，我们捕获了第一个 Windows 本地提权在野 0day，微软为这个漏洞分配的编号为 CVE-2021-1732。

这个在野样本是从我们的私有数据集中捕获的，我们注意到它是因为它使用了 HMValidateHandle 手法泄露内核信息，这是 Windows 内核提权漏洞利用中的一个强特征。进一步的分析表明这个样本利用了一个 win32k 模块中的类型混淆漏洞。

值得一提的是，这个在野样本是作为一个独立组件使用的。当使用这个样本时，用户需要提供一个进程 id 作为命令行参数，该进程 id 指出了需要被提权的进程。目标进程会先被结束，然后以 system 权限重新启动。如果直接运行该在野样本，它也会将自身提权到 system 权限，但会以没有任何可见行为的方式退出。

以下是这个在野样本的一些亮点：

1. 它针对的是当时最新的 Windows 10 1909 x64 操作系统（样本编译于 2020 年 5 月）
2. 它使用 GetMenuBarInfo 来构造任意地址读取原语，该手法此前未公开过
3. 在漏洞利用之前，它检测了特定杀毒软件，并对系统版本进行了检测

关于该 0day 的其他细节可以参考我们的[博客](#)。

下面来看第二个案例。

CVE-2021-33739 的故事

2021 年 5 月 22 日，我们捕获了第二个 Windows 本地提权在野 0day，微软为这个漏洞分配的编号为 CVE-2021-33739。

正如我在“编写正确的规则”章节所提到，我们会经常预测最可能出现的漏洞并编写规则。在 2020 年 5 月前后，我们为 dwm 漏洞编写了一些规则，在捕获了一些 dwm 组件的 Nday 漏洞后，我们在 2021 年 5 月 22 日捕获了一个不熟悉的 dwm 漏洞样本。进一步的分析表明这个样本中含有一个 1day 漏洞利用和另一个 0day 漏洞。

最初遇到这个样本时，我们并不知道它是基于一份公开代码编译的。和往常一样，我们在一个全补丁环境中复现了这个样本。复现结果清楚地表明样本中存在一个 0day，这是一个位于 dwmcore.dll 模块的 UAF 漏洞。

随后我们在 Github 上追踪到了相关源码，那是一份 CVE-2021-26868 漏洞的利用代码。在野样本仅仅只是替换了源码中的 shellcode 部分。此时我们有点疑惑：一个 1day 样本中怎么可能包含一个 0day？

在仔细确认后，我们得出结论：作者在编写 CVE-2021-26868 漏洞的利用代码时无意间引入了一个新的 0day 漏洞。如果是这样的话，这个 0day 就不能被归类为“在野 0day”。

这是漏洞修复前我们发给 MSRC 的邮件（部分）：

The first vulnerability has been fixed by Microsoft in the May 2021 patch, but the second vulnerability is still a zero day.

I think the exploit author accidentally included a second vulnerability when attempting to publish the exploit code of a known vulnerability, and the second vulnerability happened to be discovered by me when I hunting for in-the-wild zero day.

So *I think the second vulnerability is not strictly a zero-day vulnerability in the wild*. This is just my opinion, the final release definition depends on you.

这是 MSRC 最终公布的该漏洞的在野利用状态：

Exploitability

The following table provides an [exploitability assessment](#) for this vulnerability at the time of original publication.

Publicly Disclosed	Exploited	Latest Software Release
Yes	Yes	Exploitation Detected

所以这确实是一个很有趣的案例。

现在让我来讲述更多关于 CVE-2021-33739 的细节。

这个漏洞的成因是非预期的操作导致 dwmcore 模块的 CInteractionTrackerBindingManager 对象出现引用计数不平衡，从而引发 UAF。

为了触发这个漏洞，我们只需要创建一个 CInteractionTrackerBindingManagerMarshaler 资源 和 一个 CInteractionTrackerMarshaler 资源，然后将同一个 CInteractionTrackerMarshaler 资源绑定到 CInteractionTrackerBindingManagerMarshaler 资源两次，并且不要手动释放这些资源。

```
DWORD dwDataSize = 12;  
DWORD* szBuff = (DWORD*)malloc(4 * 3);  
szBuff[0] = 0x02; // resource1_id is DirectComposition::CInteractionTrackerMarshaler  
szBuff[1] = 0x02; // resource2_id is DirectComposition::CInteractionTrackerMarshaler  
szBuff[2] = 0xffff; // new_entry_id
```

在正常情况下（当 resource1_id 和 resource2_id 不一样时），CInteractionTrackerBindingManager 对象会调用 ProcessSetTrackerBindingMode 两次，以将引用计数增加 2，随后代码会调用 RemoveTrackerBindings 两次以将引用计数减 2，并且在后面引用计数减为 0 时将 CInteractionTrackerBindingManager 对象正常释放。

```
// reference count starts from 0  
CResourceFactory::Create +1 ..... ref_count = 1  
CResourceTable::CreateEmptyResource +1 ..... ref_count = 2  
CComposition::Channel_CreateResource -1 ..... ref_count = 1  
CInteractionTrackerBindingManager::ProcessSetTrackerBindingMode +1 ..... ref_count = 2  
CInteractionTrackerBindingManager::ProcessSetTrackerBindingMode +1 ..... ref_count = 3  
CResourceTable::DeleteHandle -1 ..... ref_count = 2  
CInteractionTrackerBindingManager::RemoveTrackerBindings -1 ..... ref_count = 1  
CInteractionTrackerBindingManager::RemoveTrackerBindings -1 ..... ref_count = 0  
// release object when reference count is reduced to 0
```

在漏洞场景中，CInteractionTrackerBindingManager 对象的引用计数变化与正常情况下不同。（dwmcore 内的）代码只会调用 ProcessSetTrackerBindingMode 一次将引用计数加 1，但代码仍然会调用 RemoveTrackerBindings 两次以将引用计数减 2。在第一次调用 RemoveTrackerBindings 时，CInteractionTrackerBindingManager 对象的引用计数会被减至 0，此时 CInteractionTrackerBindingManager 会被 InternalRelease 释放。在第二次调用 RemoveTrackerBindings 时，当函数内的代码尝试去获取已被释放的 CInteractionTrackerBindingManager 对象的成员数据时，就会造成 UAF。

```
// reference count starts from 0
CResourceFactory::Create +1 ..... ref_count = 1
CResourceTable::CreateEmptyResource +1 ..... ref_count = 2
CComposition::Channel_CreateResource -1 ..... ref_count = 1
CInteractionTrackerBindingManager::ProcessSetTrackerBindingMode +1 ..... ref_count = 2
CInteractionTrackerBindingManager::ProcessSetTrackerBindingMode +1 ..... ref_count = 2
CResourceTable::DeleteHandle -1 ..... ref_count = 1
CInteractionTrackerBindingManager::RemoveTrackerBindings -1 ..... ref_count = 0
// release object when reference count is reduced to 0
CInteractionTrackerBindingManager::RemoveTrackerBindings // UAF in this call !
```

接下来我们来看第三个案例。

一个“已被修补”的 1day 的故事

2021 年 10 月 16 日，我们捕获了一个新的 Windows 通用日志文件系统（CLFS）1day。这个样本来自 VirusTotal。

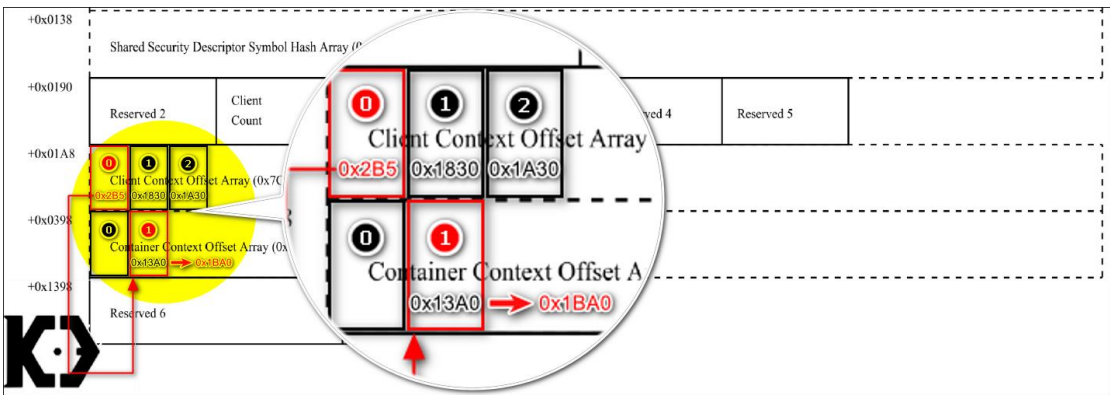
正如我在“编写正确的规则”章节所提到，我们会经常为最新的漏洞利用手法编写规则。2021 年 10 月 16 日，我们为 Pipe Attribute 编写的规则命中了一个样本。进一步的测试表明这个样本利用了一个 1day 漏洞，高楼的影响所有 2021 年 9 月前受支持的 Windows 操作系统版本。

由于缺乏相关信息，我们无法确定该漏洞对应的 CVE 编号，它可能是以下三个编号中的一个，也可能不是它们中的任何一个：

- CVE-2021-36963
- CVE-2021-36955
- CVE-2021-38633

这个 1day 漏洞的根因是 clfs 模块对 Client Context Offset 缺乏一些必要的校验。攻击者可以借此提供一个非法的 Client Context Offset。

在野样本借助这个漏洞让一个 blf 文件的第一个 Client Context Offset(0x2B5)指向第二个 Container Context Offset。



图片来自“DeathNote of Microsoft Windows Kernel”, KeenLab, 2016

随后借助 CClfsLogFcbPhysical::FlushMetadata 种的一个 1 比特翻转指令将第二个

Container Context Offset 从 0x13A0 改为了 0x1BA0，从而使该 Container Context Offset 指向了一个假的 ClfsContainer 对象。

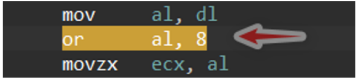
```
1: kd> .formats 13A0
Evaluate expression:
Hex:      00000000`000013a0

Binary:   00000000 00000000 00000000 00000000 00000000 00000000 0001011 10100000

1: kd> ? 13 | 8
Evaluate expression: 27 = 00000000`0000001b

1: kd> .formats 1BA0
Evaluate expression:
Hex:      00000000`00001ba0

Binary:   00000000 00000000 00000000 00000000 00000000 00000000 00011011 10100000
```



在假的 ClfsContainer 对象的帮助下（攻击者已提前构造好该伪造的对象），利用代码劫持了两个虚函数：CClfsContainer::Release 和 CClfsContainer::Remove，并且基于此在 CClfsBaseFilePersisted::RemoveContainer 构造了一个一次写入两个任意地址的原语。

一个 ClfsContainer 对象的正常虚函数表如下：

```
1: kd> dps fffff804`2e9354b8
fffff804`2e9354b8 fffff804`2e960c10 CLFS!CClfsContainer::AddRef
fffff804`2e9354c0 fffff804`2e94c060 CLFS!CClfsContainer::Release
fffff804`2e9354c8 fffff804`2e92b570 CLFS!CClfsContainer::GetSListEntry
fffff804`2e9354d0 fffff804`2e9489e0 CLFS!CClfsContainer::Remove
```

假的 ClfsContainer 对象伪造的虚函数表如下：

```
0: kd> dps 0000003a`b777f1e8
0000003a`b777f1e8 00000000`00000000
0000003a`b777f1f0 fffff804`2f0cc390 nt!HalpDmaPowerCriticalTransitionCallback
0000003a`b777f1f8 00000000`00000000
0000003a`b777f200 fffff804`2ef95f70 nt!XmXchgOp
```

此外，在野样本还使用“[Scoop the Windows 10 pool!](#)”这篇文章里面提到的“Pipe Attribute”方法构造了一个任意地址读取原语。为了获得一个 Pipe Attribute 对象的地址，它还使用了另一个公开的方法：通过查询 SystemBigPoolInformation 来泄露一个 Pipe Attribute 对象的地址。

有了内核任意地址读写原语后，利用代码成功将当前进程的访问令牌替换为 system 进程的令牌，并以 system 权限弹出了一个 cmd 窗口。

```
管理员: C:\Users\test\Desktop\sample.exe
cmd.exe spawned
Microsoft Windows [版本 10.0.19042.1165]
(c) Microsoft Corporation。保留所有权利。

C:\Users\test\Desktop>whoami
nt authority\system

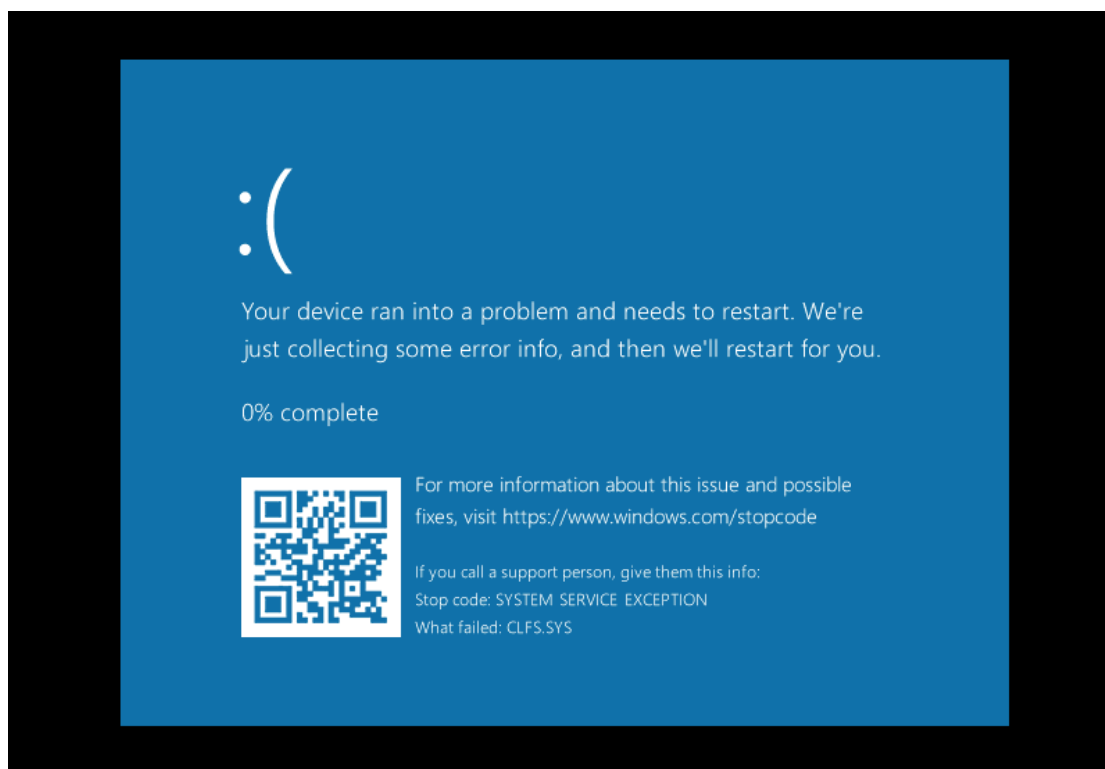
C:\Users\test\Desktop>
```

让我们来看一下微软是如何修补这个漏洞的，微软的开发人员只检查了 Client Context Offset 的下限，以确保其不能小于 0x1368。

```
_int64 __fastcall CCIFSBaseFile::GetSymbol(
    CCIFSBaseFile *this,
    unsigned int offset,
    char a3,
    struct _CLFS_CLIENT_CONTEXT **a4)
{
    unsigned int v8; // ebx
    BOOLEAN v10; // r15
    struct _CLFS_CLIENT_CONTEXT *v11; // rax
    unsigned int v12; // [rsp+20h] [rbp-38h]

    v8 = 0;
    v12 = 0;
    if ( offset < 0x1368 ) // patch
        return 0xC01A000Di64;
    *a4 = 0i64;
    v10 = ExAcquireResourceSharedLite(*((PERESOURCE *)this + 4), 1u);
    v11 = (struct _CLFS_CLIENT_CONTEXT *)CCIFSBaseFile::OffsetToAddr(this);
```

修补方案并没有对 Client Context Offset 的上限做检测。如果我们构造一个值大于 0x1368 的 Client Context Offset，并且使其直接指向一个内存中的 Container Context 对象会怎么样？答案是一个新的 0day 漏洞。



我们在 2021 年 12 月将该变种报告给了 MSRC，微软在 2022 年 4 月修复了这个漏洞，并且为其分配了 CVE-2022-24481 这个编号。

以上就是关于三个案例的分享。最后，让我们给出一些对 Windows 本地提权漏洞检测的建议，并分享一些对未来 Windows 本地提权在野 0day 的趋势预测。

写在最后

关于 Windows 本地提权漏洞的一些检测建议：

- 选择自己能力范围内最适合的方法
- 仔细研究历史案例总是一件好事
- 留意那些新出现的在野漏洞的变种

对未来 Windows 本地提权漏洞 0day 的趋势预测：

- 未来会出现更多 clfs 模块的漏洞
- “Pipe Attribute”这种手法在未来会继续被使用
- 未来会有更多在野利用使用以下手法：
 - [Arbitrary address read/write with the help of WNF](#), POC2021
 - [Arbitrary address read/write with the help of ALPC](#), Blackhat Asia 2022
 - [Arbitrary address read/write with the help of I/O Ring](#), TyphoonCon 2022

引用

1. <https://github.com/synacktiv/Windows-kernel-SegmentHeap-Aligned-Chunk-Confusion>
2. <https://securelist.com/puzzlemaker-chrome-zero-day-exploit-chain/102771/>

3. <https://research.nccgroup.com/2021/07/15/cve-2021-31956-exploiting-the-windows-kernel-ntfs-with-wnf-part-1/>
4. <https://research.nccgroup.com/2021/08/17/cve-2021-31956-exploiting-the-windows-kernel-ntfs-with-wnf-part-2/>
5. <https://vul.360.net/archives/83>
6. <https://securelist.com/zero-day-vulnerability-in-desktop-window-manager-cve-2021-28310-used-in-the-wild/101898/>
7. <https://www.zerodayinitiative.com/blog/2021/5/3/cve-2021-26900-privilege-escalation-via-a-use-after-free-vulnerability-in-win32k>
8. <https://ti.dbappsecurity.com.cn/blog/index.php/2021/02/10/windows-kernel-zero-day-exploit-is-used-by-bitter-apt-in-targeted-attack/>
9. <https://github.com/KangDIW2/CVE-2021-26868>
10. <https://i.blackhat.com/Asia-22/Friday-Materials/AS-22-Xu-The-Next-Generation-of-Windows-Exploitation-Attacking-the-Common-Log-File-System.pdf>
11. <https://windows-internals.com/one-i-o-ring-to-rule-them-all-a-full-read-write-exploit-primitive-on-windows-11/>
12. <https://github.com/oct0xor/presentations/blob/master/2019-02-Overview%20of%20the%20latest%20Windows%20OS%20kernel%20exploits%20found%20in%20the%20wild.pdf>
13. <https://research.checkpoint.com/2020/graphology-of-an-exploit-volodya/>
14. <https://research.checkpoint.com/2020/graphology-of-an-exploit-playbit/>