The Journey Of Hunting ITW Windows LPE 0day

Quan Jin <twitter: @jq0904>
DBAPPSecurity
Black Hat USA 2022

Contents

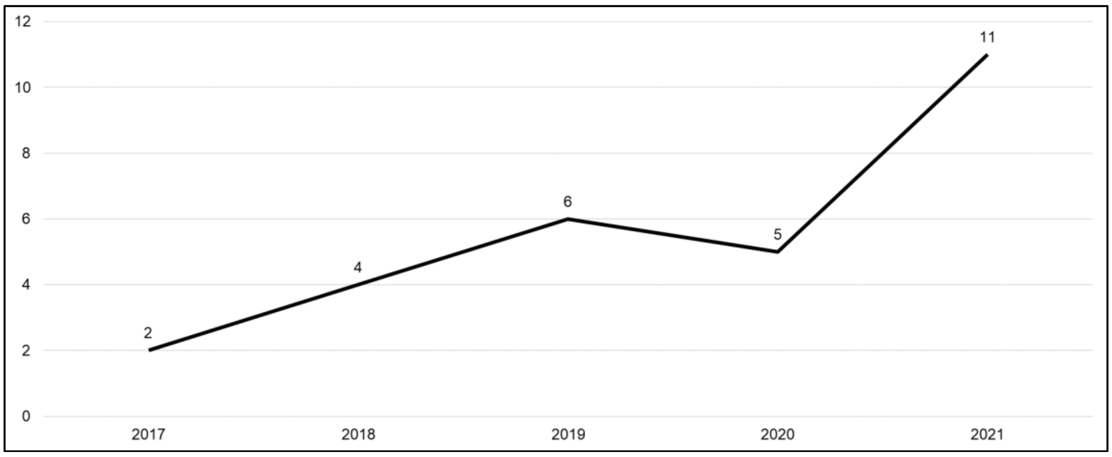
Abstract

In this paper, we will talk our story of how to hunt itw(in-the-wild) Windows LPE 0day during 2020 and 2021: why we think this is possible, how we study historical cases, how we use learned experience to develop a detection method, and how we improve the method to make it more accurate. By using this method, we successfully caught two itw Windows LPE 0days and an itw Windows LPE 1day.

We will also compare the advantages and disadvantages of several hunting methods, and give some insights into the trend of itw Windows LPE 0day in the future.

Background

As you can see in the picture, from 2017 to 2021, Microsoft disclosed a total of 28 itw Windows LPE 0days, most of them are Windows kernel vulnerabilities. These vulnerabilities are often used by top level groups and could cause great harm. For security vendors, it is very challenging to catch an itw Windows LPE 0day.



ITW Windows LPE 0day (2017-2021)

Starting in 2020, we began to think about the possibility of catching an itw Windows LPE 0day.

Is it possible to catch an itw LPE 0day?

In order to answer this question, we should think about other two questions:

1. How to obtain valuable data source?
2. How to develop effective detection methods?

For the first question, we have some private datasets. In addition, historical cases show that public platforms such as VirusTotal may have 0days. Therefore, by using private and public datasets, we can solve the first question.

For the second question, there are two methods for catching a 0day from millions of samples: dynamic detection or static detection.

a) Dynamic detection refers to simulation execution in a sandbox or real environment and picking out a sample through abnormal behavior (such as antivirus and sandbox)

b) Static detection refers to matching samples with static signatures (such as YARA)

Both methods have advantages and disadvantages. We have tried both methods within our capabilities. Based on the results, we think static detection is more suitable for us, we will detail the process later.

Next, I'll explain why we spend a lot of time studying historical itw Windows LPE 0days.

Learn from history (and now)

Why should we learn from history

There are three reasons:
1. Some exploit techniques are consistent over time
2. Thinking from the attacker's view allows for better defense
3. Historical cases have been carefully studied by the community

How we study historical cases

In order to learn from history, we've studied over 50 CVEs, almost all itw Windows LPE 0days and some 1days that from 2014 to 2021.

We carefully counted the discovered vendor, using organization, patch cycle, initial disclosure article, usage scenario, targeted system versions, vulnerability module, vulnerability type, exploit techniques, public analysis blogs, public exploits, the original sample(if have) and other information.

Here I would like to focus on a few key points:

Usage scenario:
a) Whether the sample is used as a standalone component, or as part of a chain
b) Whether the exploit was used in a fileless form (such as dll reflection), or was just contained in a drop file

These information will directly affect our selection of different detection methods.

Targeted system versions:
Many Windows LPE samples will check OS version before they are used, and can only trigger or exploit in some appropriate versions.

This information is especially useful when making a sandbox or reproduction environment.

Vulnerability module:
By counting the vulnerability modules of historical samples, we can conclude which component is most targeted, and which attack surface is most favored by attackers during a specific period of time.

Vulnerability type:
By counting the type of historical vulnerabilities, we can infer which type of vulnerability is the attacker most favored, this information can help us make right reproduction environment (for example, whether need to config Driver Verifier). This information can also tell us the popularity of different vulnerability types.

Exploit techniques:
I think this is the most important information. We count the exploit techniques for every itw Windows LPE 0day (which we can get the origin file or can find relative descriptions).

Based on the statistic, we have obtained some valuable conclusions. For example, “bServerSideWindowProc” method was popular from 2015 to 2016. The method of using “Previous Mode” to achieve arbitrary address read and write has become more and more popular since 2018. We also found that the method of using “HMValidateHandle” to leak kernel information is popular in the past five years.

Public analysis blogs & exploits:
The public blogs and exploits contain the research results of the community. Absorbing these existing knowledges is just like standing on the shoulders of giants, which is very helpful for us.

The original sample (if have):
We also pay great attention to collecting the original samples of each historical Windows LPE 0day. The files, hashes, and exploits of these original samples are the first-hand information, if we can detect them, we can also catch similar samples in the future.

Why should we learn from now

In addition to learning from history, we should also learn from the latest vulnerability and exploit techniques. The reasons are as follows:
1. A new disclosed vulnerability may have variants (such as CVE-2021-1732 and CVE-2022-21882)
2. A new targeted module will be fuzzed and audited by community (such as clfs.sys)
3. An attacker may have some similar vulnerabilities in use or wait to use (For example, Kaspersky discovered CVE-2021-28310 based on CVE-2021-1732)
4. A new exploit technique tends to be used by attackers soon (such as Pipe Attribute technique in “Scoop the Windows 10 pool!”and WNF technique in itw CVE-2021-31956 sample)

Next, I will describe how we compare different detection methods and choose one from them.

One road leads to Rome

Choose the right tool

As far as we know, there are three optional methods to catch an itw Windows LPE:
1. Antivirus (or something like it)
2. Sandbox (or something like it)
3. YARA (or something like it)

Antivirus is the most powerful tool. It was deployed in large-scale real-world environments. It can also detect threats in real time and have chance to extract encrypted privilege escalation component. Kaspersky have caught some itw LPE 0days with their antivirus product in the past few years. However, not every vendor's antivirus can be as good as Kaspersky. Also, antivirus is likely to be bypassed or detected. These increase the difficulty of developing an antivirus-based hunting method.

Sandbox is another tool to hunt itw 0day. Unlike antivirus, the sandbox environment is highly controllable and can be freely configured. In addition, the sandbox's behavior-based detection makes it accuracy. I had some successful experience on itw Office 0day hunting with the help of sandbox. Interested readers can refer to my previous speech on Bluehat Shanghai 2019.

However, I think sandbox is somehow not suitable for hunting Windows LPE 0days. Unlike Office, many LPE exploits have OS version check to avoid unexpected BSOD, which makes them more hidden to sandbox. You may think that we can solve this problem by making a few more environments, but the number of new PE samples is huge, each sample is delivered to a new environment means a huge resource overhead. Not every vendor has enough money to afford this.

In addition, sandbox-based detection methods have other disadvantages for Windows LPE samples：

a) Some samples require parameters (for example, a Pid), but the sandbox cannot provide valid parameters by default
b) Some samples only lead to BSOD without subsequent behavior, which is difficult to detect
c) There is a cycle between sandbox development and deployment, which will lead to missing the best detection cycle for some latest exploits

YARA is another method to hunt itw Windows LPE 0day. It has a very good effect on detecting samples which have certain signatures.

It almost has no technical barriers, no fear of various checks, and it is flexible in development and deployment. When a new exploit technique appears, we can quickly convert it to rules and feed it into the detection system. Finally, it's lower cost than antivirus and sandbox.

But it also has shortcomings, such as it can easily lead to false positives and false negatives. Therefore, if we use YARA to hunt itw LPE 0day, we need to be very familiar with historical cases, and we have done this before.

We considered the above hunting methods in combination with our own situation, and finally chose YARA as our main hunting method, which is more easy, more flexible and less expensive for Windows LPE 0day hunting.

Another reason we choose YARA is that, after writing some YARA rules, we back-tested some historical Windows LPE samples. To our surprise, YARA performed better than expected.

Build the right rule

Now we will describe how to transform learned experience into YARA rules.

Basicly, we have three principles:
1. Write rules according to the signatures of each stage of exploitation
2. Write rules for latest exploit techniques
3. Write rules for the most likey vulnerability

For the first idea, normally, a Windows kernel LPE exploit has the following stages:
a) Vulnerability Triggering
b) Heap Feng Shui
c) Kernel Information Leak
d) Arbitrary Address Read and Write
e) Control Flow Hijacking
f) Privilege Escalation

Our task is to write rules based on the common features of each stage. Here are some examples：

For kernel information leak, the idea is to match against common Windows kernel information leak techniques. Including but not limited to these:
- NtQuerySystemInformation
  - ➢ SystemBigPoolInformation
  - ➢ SystemModuleInformation
  - ➢ …
- Win32k Shared Info User Handle Table
- Descriptor Tables
- HMValidateHandle
- GdiSharedHandleTable

For arbitrary address read/write primitives, the idea is to match against the following points:
- SetWindowLong / SetWindowLongPtr
- SetWindowText / InternalGetWindowText / NtUserDefSetText
- GetMenuItemRect / SetMenuItemInfo / GetMenuBarInfo
- NtUpdateWnfStateData / NtQueryWnfStateDate
- GetBitmapBits / SetBitmapBits
- GetPaletteEntries / SetPaletteEntries
- CreatePipe / NtFsControlFile
- Previous Mode + NtReadVirtualMemory / WriteVirtualMemory

It should be noted that the above are just some possible ideas, not all ideas are suitable for YARA rules, and some ideas will lead to lots of false positives.

For the second idea, here I give two examples:

1. In July 2020, Paul Fariello (@paulfariello) and Corentin Bayet (@OnlyTheDuck) of Synacktiv presented a new Windows kernel heap overflow exploit technique at the SSTIC2020 conference. After studying their paper, we realized that the method of arbitrary address read with the help of Pipe Attribute in PagedPool is universal and may be used in the future. So we spent some time writing some YARA rules for this technique. It later turned out that these rules caught some high-value samples.
2. On June 8, 2021, Kaspersky wrote a blog, which disclosed an itw Windows LPE 0day (CVE-2021-31956). As mentioned in the blog, the sample achieved arbitrary address read and write with the help of Windows Notification Facility (WNF). In July and August 2021, Alex Plaskett(@alexjplaskett) of NCC Group published two blogs detailing the exploit techniques of CVE-2021-31956 and explaining the method of using WNF to construct arbitrary address read and write primitives. At the same time, YanZiShuang(@YanZiShuang) also wrote a blog discussing the method of exploiting vulnerabilities with the help of WNF. After studying these blogs, we realized that the method is universal. We again spent some time writing some YARA rules for this technique. As expected, we did catch some high-value samples.

For the third idea, I also give an example here. On April 13, 2021, Kaspersky wrote a blog and disclosed CVE-2021-28310, which is an itw 0day in Desktop Windows Manager. Less than a month later, ZDI published another blog, disclosing another vulnerability(CVE-2021-26900), which is also a vulnerability in Desktop Windows Manager. This made us realize that this type of vulnerability may appear again in the future, so we wrote several rules for Desktop Windows Manager vulnerabilities in hours. A few weeks later, we caught CVE-2021-33739.

Build the right rule is the first step. In order to catch an itw Windows LPE 0day, we need to build a whole system.

Build a workable system

Think about these questions:

1. When a sample is matched by a rule, how to notify us in time?
2. When we get a sample, how to quickly reproduce and classify it?
3. What skills should we master to debug different Windows LPE samples?

For the first question, if our YARA rules running on VirusTotal, we can use the notification mechanism on VirusTotal Hunting page, we can configure the “Notify by email“ item. When a new sample is matched, our email will receive a notification at once.

For rules running on our own products, we built a similar notification interface like VT.

To answer the second question, we have counted the targetd OS version for each historical sample by studying historical cases, these information can be used here. In addition, considering that what we hunt may be a Nday, 1day or 0day, we need to make all three types of environments, and we need to update these environments over time. In order to minimize the reproduction time, we have made multiple environments of Windows 7, Windows 10, Windows 11, and covered both x86 and x64.

The third question depends on our experience on debugging different samples. For example, for those who have analyzed the Windows kernel heap overflow vulnerability, Windows kernel debugging and Driver Verifier are two basic skills. Besides, for those who have analyzed dwmcore vulnerability, it is necessary to use Windows remote debugging (because directly attaching the dwm process will cause system UI to become unresponsive). The more experience we have, the better we'll answer this question.

Test and improve the system

No method is perfect, to make our system more accurate, we have made the following tests and improvements:
1. Use the collected historical Windows LPE 0day samples to test the rules, eliminate false positives and false negatives, and improve the accuracy of the rules
2. Test the rules with the collected public pocs/exploits in the same way as above
3. For some cases where the public pocs/exploits cannot be collected, try to write the poc/exploit and test it
4. Apply the rules to a large number of samples for stress testing to eliminate the observed false positives and false negatives
5. Continue to convert latest exploit techniques into rules, write and test the rules and eliminate false positives and false negatives

One year after the system was deployed, we had caguht lots of Windows LPE vulnerabilities.

In next part, we will share three cases which hunt by our system:
1. CVE-2021-1732: an itw LPE 0day in Windows win32k subsystem
2. CVE-2021-33739: an itw LPE 0day in Windows Desktop Window Manager
3. Unknown CVE: an itw LPE 1day in Windows Common Log File System

Results

The Story of CVE-2021-1732

In December 10, 2020, we caught the first itw Windows kernel LPE 0day. Microsoft assigned CVE-2021-1732 to this vulnerability.

The itw sample was from our private dataset, we noticed it because it used HMValidateHandle to leak kernel information, which is a strong signature of Windows kernel LPE exploit. Further analysis showed the sample exploited a type confusion 0day in win32k module.

It is worth mentioning that the itw sample was used as an independent component.

When using the sample, you need to provide a Pid as a parameter, the Pid indicates the process which needs to be elevated. The targeted process will be terminated first, then restarted with system privilege. If you run the sample directly, it will also escalate itself to the system privilege, but will exit without any visible behavior.

Here are some highlights of this itw sample:

1. It targeted the latest version of Windows 10 1909 64-bits operating system at that time (The sample was compiled in May 2020)
2. It uses GetMenuBarInfo to built arbitrary address read primitive, which is novel
3. Before exploit, the itw sample detected specific antivirus and performed system version check

The rest of the details about this 0day can refer to our blog.

The Story of CVE-2021-33739

In May 22, 2021, we caught the second itw Windows LPE 0day. Microsoft assigned CVE-2021-33739 to this vulnerability.

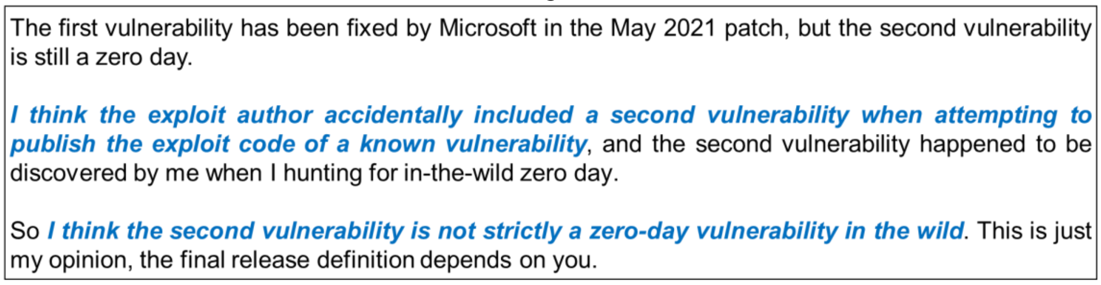
As I mentioned in the "Build the right rule" part, we will regularly predict the most likely vulnerability and write rules. Around May 2020, we wrote some rules for dwm vulnerability, after catching some dwm ndays, we caught an unfamiliar dwm sample on May 22, 2021. Further analysis showed there were an 1day exploit and another 0day in this sample.

When we first met the sample, we didn't know it was compiled based a publish exploit code. As usual, we reproduced the sample in a full-patched environment. The reproduced result clearly showed that there is a 0day in the sample, which is an UAF in dwmcore.

Then we tracked the relative source code on GitHub, which is an exploit of CVE-2021-26868. The itw sample just replaced the shellcode part. At that time, we are a little confused: How can an 1day sample contains a 0day?

After careful confirmation, we concluded that the author accidentally introduced a new bug when writing the exploit for CVE-2021-26868. If so, this 0day can not be classified to “itw 0day”.

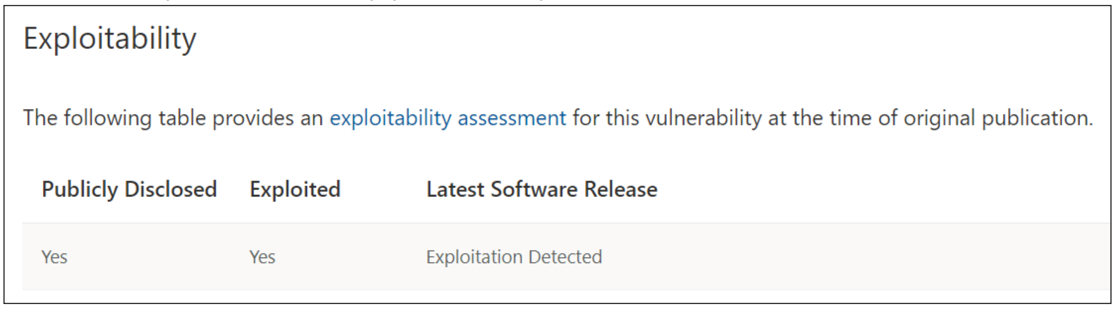
This is what we sent to MSRC before the bug was fixed:


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.

This is the exploit status finally published by MSRC:



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

So It's really an interesting case.

Let me talk more about CVE-2021-33739.

This vulnerability is caused by unbalanced reference count on CinteractionTrackerBindingManager Object in dwmcore.

In order to trigger the vulnerability, we only need to create a CInteractionTrackerBindingManagerMarshaler resource and a CinteractionTrackerMarshaler resource, and bind the same CinteractionTrackerMarshaler resource twice to CinteractionTrackerBindingManagerMarshaler resource, and do not release these resource manully.

```
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
```

Under normal condition (when resource1_id is different from resource2_id), the CinteractionTrackerBindingManager object will call ProcessSetTrackerBindingMode twice to add reference count by 2. Then the code will call RemoveTrackerBindings twice to sub reference count, and release the CinteractionTrackerBindingManager object normally when reference count is reduced to 0.

```
// 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
```

In a vulnerability scenario, the reference count of CinteractionTrackerBindingManager object will change different from normal condition, it will call ProcessSetTrackerBindingMode only once to add reference count by 1. But the code will still call RemoveTrackerBindings twice to sub reference count, in the first RemoveTrackerBindings call, the reference count of CinteractionTrackerBindingManager object will be reduced to 0, and the CinteractionTrackerBindingManager object will be freed in InternalRelease. In the second RemoveTrackerBindings call, when the code tries to get some data from the freed CinteractionTrackerBindingManager object, it will cause 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~~
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 !
```

The Story of a "Patched" 1day

In October 16, 2021, we caught a new itw Windows clfs 1day. The sample was from VirusTotal.

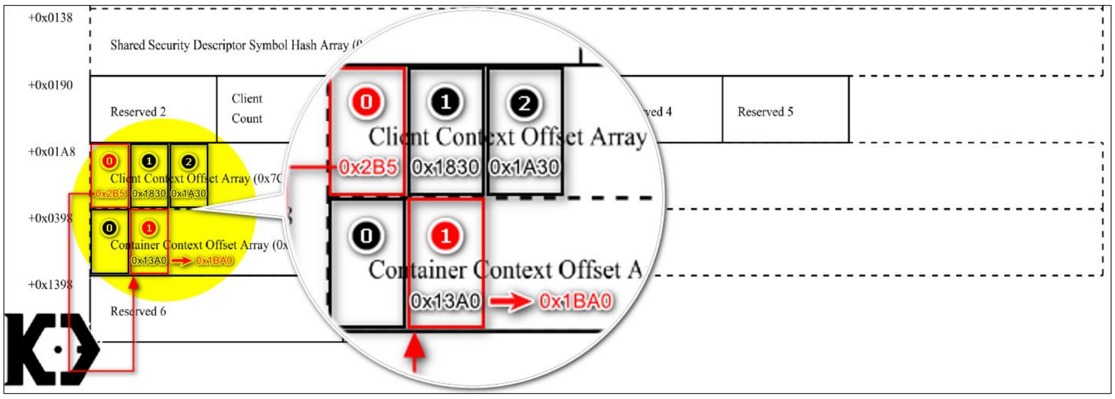
As I mentioned in the "Build the right rule" part, we will regularly write rules for latest exploit techniques. On October 16, 2021, the rule we wrote for Pipe Attribute hit a sample. Further testing revealed that the sample exploited a vulnerability which affected all supported Windows versions before September 2021.

Due to lack of information, we are unable to determine the CVE number of this vulnerability, it may be one of them or none of them:

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

The root cause of this 1day is the clfs module lacks some checks on the Client Context Offset. An attacker can take advantage of this to provide an invalid Client Context Offset.

The itw sample leveraged this to make the first Client Context Offset(0x2B5) point to the second Container Context Offset.



The picture is from "DeathNote of Microsoft Windows Kernel", KeenLab, 2016

It then use an 1-bit flip in FlushMetadata to change the second Container Context Offset from 0x13A0 to 0x1BA0, and makes the Container Context Offset point to a fake ClfsContainer object.

```
1: kd> .formats 13A0
Evaluate expression:
  Hex:     00000000`000013a0
  Binary:  00000000 00000000 00000000 00000000 00000000 00000000 00010011 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
```

With the help of the fake ClfsContainer, the exploit hijacked two virtual methods: CClfsContainer::Release and CClfsContainer::Remove, and built an arbitrary address write primitive based on that.

The normal virtual table of a ClfsContainer object:

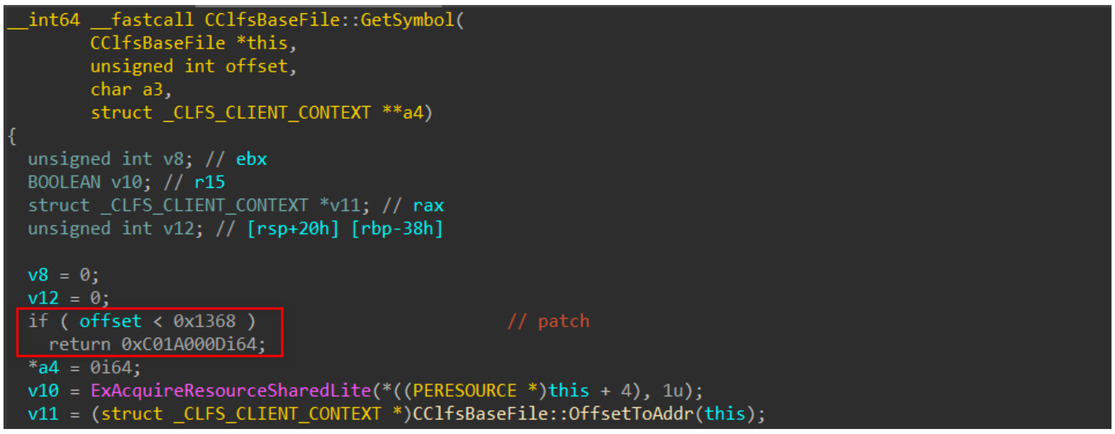
```
1: kd> dps fffff804`2e9354b8
fffff804`2e9354b8  fffff804`2e960c10 CLFS!CClfsContainer::AddRef
fffff804`2e9354c0  fffff804`2e94c060 CLFS!CClfsContainer::Release
fffff804`2e9354c8  fffff804`2e92b570 CLFS!CClfsContainer::GetSListEntry
fffff804`2e9354d0  fffff804`2e9489e0 CLFS!CClfsContainer::Remove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

The fake virtual table of the fake ClfsContainer object:

```
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
```

Apart from this, the itw sample built an arbitrary address read primitive using the "Pipe Attribute" method described in the "Scoop the Windows 10 pool!". In order to get the address of a Pipe Attribute, the exploit using another public method, it queried SystemBigPoolInformation to leaking the address of a Pipe Attribute object. With the kernel arbitrary address read and write primitives, the exploit successfully swapped the token of current process with system token, and spawned a shell with system privilege.

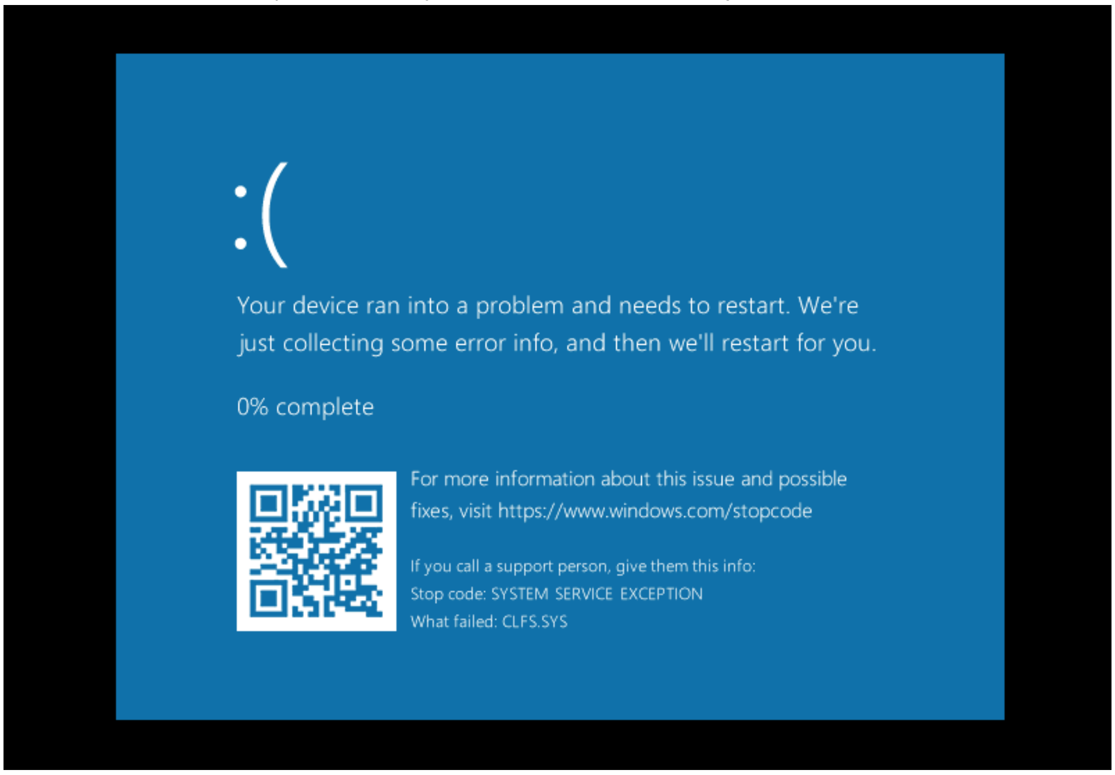
Let's take a look at how Microsoft fixed this vulnerability. They only checked the value of Client Context Offset to make sure it couldn't be less than 0x1368!



```
__int64 __fastcall CClfsBaseFile::GetSymbol(
        CClfsBaseFile *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 *)CClfsBaseFile::OffsetToAddr(this);
```

What if we construct a Client Context Offset that is greater than 0x1368, and make the Client Context Offset point directly to a CclfsContainer object?



:(

Your device ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

0% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

If you call a support person, give them this info:
Stop code: SYSTEM SERVICE EXCEPTION
What failed: CLFS.SYS

We reported this variant to MSRC at December 2021, Microsoft fixed this case in April 2022 and assigned CVE-2022-24481 to it.

Suggestions and insights

Some detection suggestions on Windows LPE vulnerabilities:

- Choose the most suitable method within your capability
- Carefully study historical cases is always a good thing
- Keep an eye out for new variants of a new itw vulnerabillity

Some insights into the future trends of itw Windows LPE 0day:

- More vulnerabilities in clfs may appear in the future
- “Pipe Attribute” method will be using again in the future
- ITW exploits which use the following techniques may appear in the future:
    - 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

Reference

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/KangD1W2/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/*