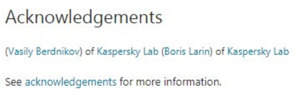
RESEARCH

The fourth horseman: CVE-2019-0797 vulnerability

The new zero-day in the Windows OS exploited in targeted attacks

By Vasily Berdnikov, Boris Larin on March 13, 2019. 10:00 am

In February 2019, our Automatic Exploit Prevention (AEP) systems detected an attempt to exploit a vulnerability in the Microsoft Windows operating system. Further analysis of this event led to us discovering a zero-day vulnerability in win32k.sys. We reported it to Microsoft on February 22, 2019. The company confirmed the vulnerability and assigned it CVE-2019-0797. Microsoft have just released a patch, crediting Kaspersky Lab researchers **Vasiliy Berdnikov** and **Boris Larin** with the discovery:



Acknowledgements

(Vasily Berdnikov) of Kaspersky Lab (Boris Larin) of Kaspersky Lab

See acknowledgements for more information.

This is the fourth consecutive exploited Local Privilege Escalation vulnerability in Windows we have discovered recently using our technologies. Just like with CVE-2018-8589, we believe this exploit is used by several threat actors including, but possibly not limited to, FruityArmor and SandCat. While FruityArmor is known to have used zero-days before, SandCat is a new APT we discovered only recently. In addition to CVE-2019-0797 and CHAINSHOT, SandCat also uses the FinFisher/FinSpy framework.

Kaspersky Lab products detected this exploit proactively through the following technologies:

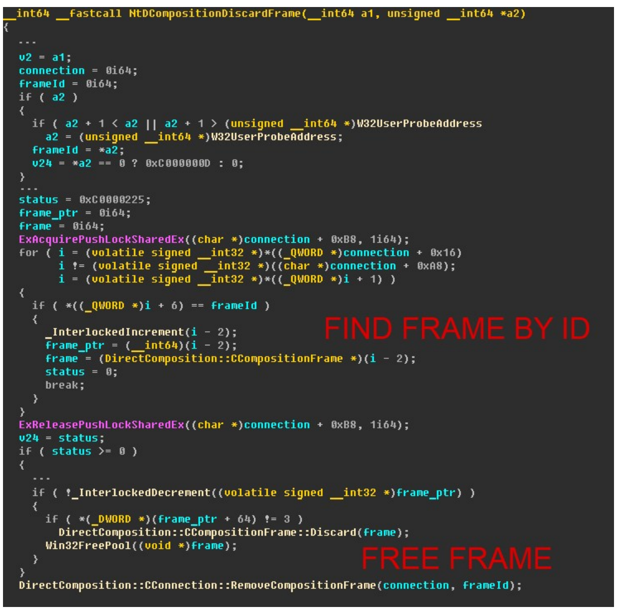
1. Behavioral detection engine and Automatic Exploit Prevention for endpoint products;
2. Advanced Sandboxing and Anti Malware engine for Kaspersky Anti Targeted Attack Platform (KATA).

Kaspersky Lab verdicts for the artifacts used in this and related attacks are:

- HEUR:Exploit.Win32.Generic
- HEUR:Trojan.Win32.Generic
- PDM:Exploit.Win32.Generic

Brief technical details – CVE-2019-0797

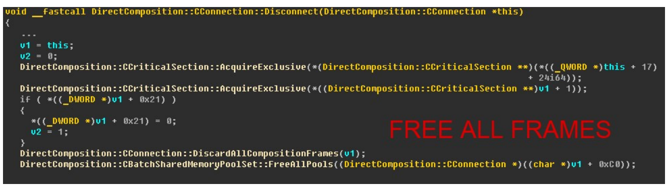
CVE-2019-0797 is a race condition that is present in the win32k driver due to a lack of proper synchronization between undocumented syscalls NtDCompositionDiscardFrame and NtDCompositionDestroyConnection. The vulnerable code can be observed below on screenshots made on an up-to-date system during initial analysis:



```
__int64 __fastcall NtDCompositionDiscardFrame(__int64 a1, unsigned __int64 *a2)
{
  ...
  v2 = a1;
  connection = 0i64;
  frameId = 0i64;
  if ( a2 )
  {
    if ( a2 + 1 < a2 || a2 + 1 > (unsigned __int64 *)W32UserProbeAddress
      a2 = (unsigned __int64 *)W32UserProbeAddress;
    frameId = *a2;
    v24 = *a2 == 0 ? 0xC000000D : 0;
  }
  ...
  status = 0xC0000225;
  frame_ptr = 0i64;
  frame = 0i64;
  ExAcquirePushLockSharedEx((char *)connection + 0xB8, 1i64);
  for ( i = (volatile signed __int32 *)*((_QWORD *)connection + 0x16)
        i != (volatile signed __int32 *)((char *)connection + 0xA8);
        i = (volatile signed __int32 *)*((_QWORD *)i + 1) )
  {
    if ( *((_QWORD *)i + 6) == frameId )
    {
      _InterlockedIncrement(i - 2);
      frame_ptr = (__int64)(i - 2);
      frame = (DirectComposition::CCompositionFrame *)(i - 2);
      status = 0;
      break;
    }
  }
  ExReleasePushLockSharedEx((char *)connection + 0xB8, 1i64);
  v24 = status;
  if ( status >= 0 )
  {
    ...
    if ( !_InterlockedDecrement((volatile signed __int32 *)frame_ptr) )
    {
      if ( *(_DWORD *)(frame_ptr + 64) != 3 )
        DirectComposition::CCompositionFrame::Discard(frame);
      Win32FreePool((void *)frame);
    }
  }
  DirectComposition::CConnection::RemoveCompositionFrame(connection, frameId);
```

Snippet of NtDCompositionDiscardFrame syscall (Windows 8.1)

On this screenshot with the simplified logic of the NtDCompositionDiscardFrame syscall you can see that this code acquires a lock that is related to frame operations in the structure DirectComposition::CConnection and tries to find a frame that corresponds to a given id and will eventually call a free on it. The problem with this can be observed on the second screenshot:

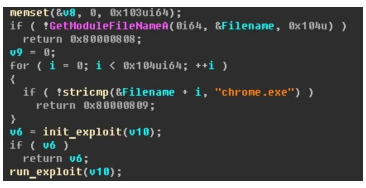


```
void __fastcall DirectComposition::CConnection::Disconnect(DirectComposition::CConnection *this)
{
  ...
  v1 = this;
  v2 = 0;
  DirectComposition::CCriticalSection::AcquireExclusive(*(DirectComposition::CCriticalSection **)(*((_QWORD *)this + 17)
                                                                                                + 24i64));
  DirectComposition::CCriticalSection::AcquireExclusive(*((DirectComposition::CCriticalSection **)v1 + 1));
  if ( *((_DWORD *)v1 + 0x21) )
  {
    *((_DWORD *)v1 + 0x21) = 0;
    v2 = 1;
  }
  DirectComposition::CConnection::DiscardAllCompositionFrames(v1);
  DirectComposition::CBatchSharedMemoryPoolSet::FreeAllPools((DirectComposition::CConnection *)((char *)v1 + 0xC0));
```

Snippet of NtDCompositionDestroyConnection syscall inner function (Windows 8.1)

On this screenshot with the simplified logic of the function DiscardAllCompositionFrames that is called from within the NtDCompositionDestroyConnection syscall you can see that it does not acquire the necessary lock and calls the function DiscardAllCompositionFrames that will release all allocated frames. The problem lies in the fact that when the syscalls NtDCompositionDiscardFrame and NtDCompositionDestroyConnection are executed simultaneously, the function DiscardAllCompositionFrames may be executed at a time when the NtDCompositionDiscardFrame syscall is already looking for a frame to release or has already found it. This condition leads to a use-after-free scenario.

Interestingly, this is the third race condition zero-day exploit used by the same group in addition to CVE-2018-8589 and CVE-2018-8611.



```
memset(&v8, 0, 0x103ui64);
if ( !GetModuleFileNameA(0i64, &Filename, 0x104u) )
  return 0x80000808;
v9 = 0;
for ( i = 0; i < 0x104ui64; ++i )
{
  if ( !stricmp(&Filename + i, "chrome.exe") )
    return 0x80000809;
}
v6 = init_exploit(v10);
if ( v6 )
  return v6;
run_exploit(v10);
```

Stop execution if module file name contains substring "chrome.exe"

The exploit that was found in the wild was targeting 64-bit operating systems in the range from Windows 8 to Windows 10 build 15063. The exploitation process for all those operating systems does not differ greatly and is performed using heap spraying palettes and accelerator tables with the use of GdiSharedHandleTable and gSharedInfo to leak their kernel addresses. In exploitation of Windows 10 build 14393 and higher windows are used instead of palettes. Besides that, that exploit performs a check on whether it's running from Google Chrome and stops execution if it is because vulnerability CVE-2019-0797 can't be exploited within a sandbox.

MICROSOFT WINDOWS | TARGETED ATTACKS | VULNERABILITIES AND EXPLOITS | ZERO-DAY VULNERABILITIES

Share post on:

Related Posts

Mokes and Buerak distributed under the guise of security certificates

Operation AppleJeus Sequel

Kaspersky Security Bulletin 2019. Statistics

IN THE SAME CATEGORY

Assessing the impact of protection from web miners

Agent 1433: remote attack on Microsoft SQL Server

How to steal a million (of your data)

On the IoT road: perks, benefits and security of moving smartly

How we hacked our colleague's smart home

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Enter your comment here

Name *

Email *

□ Save my name, email, and website in this browser for the next time I comment.

□ Notify me when new comments are added.

SUBMIT

□ I'm not a robot reCAPTCHA Privacy - Terms