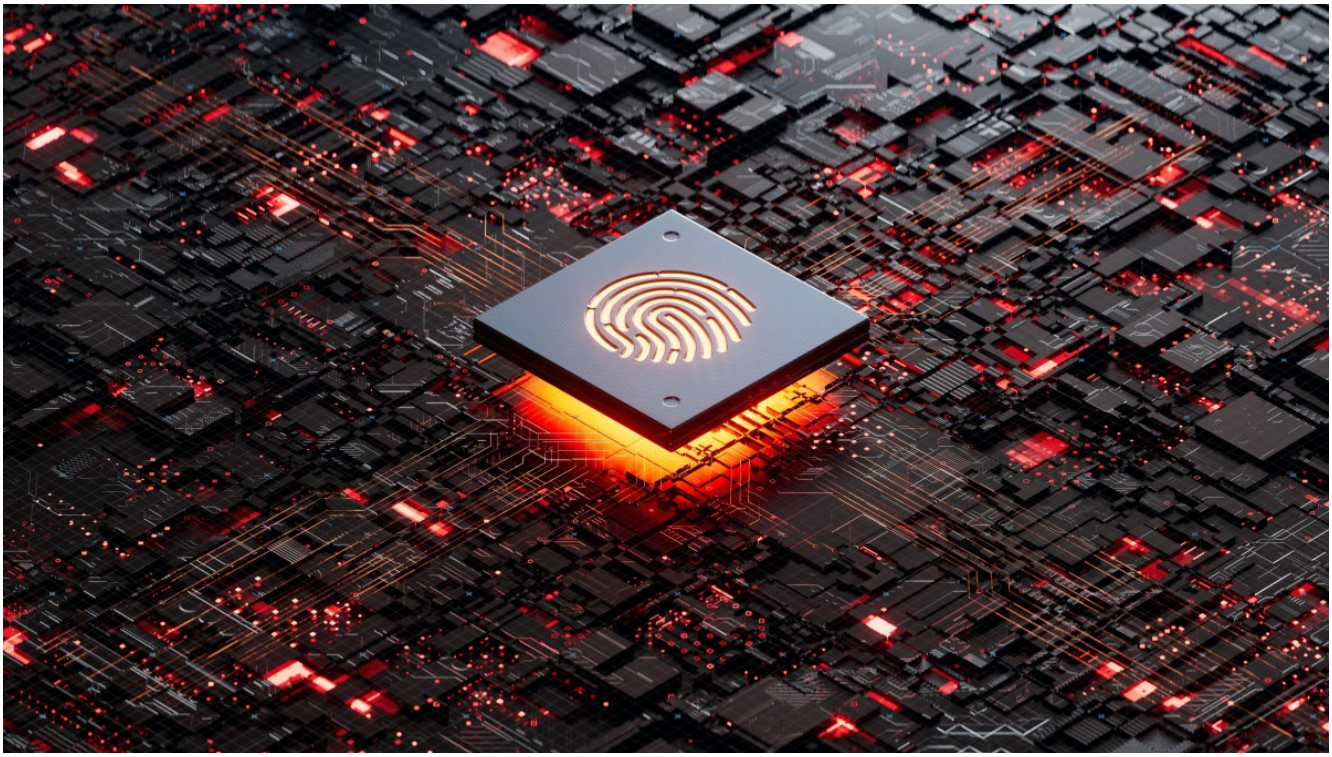
Signed driver malware moves up the software trust chain

news.sophos.com/en-us/2022/12/13/signed-driver-malware-moves-up-the-software-trust-chain/

December 13, 2022



While investigating suspicious activity on a customer network, Sophos X-Ops Rapid Response (RR) discovered a pair of files left behind on some compromised machines. The two files work together to terminate processes or services used by a variety of endpoint security product vendors.

Because the RR team kicked the attackers off the systems, preventing further damage, it isn't possible to know for certain which ransomware the attackers intended to deploy. However, techniques used and files left behind offer an intriguing clue about the group behind the attack.

In our post-attack analysis, SophosLabs determined that the pair of executable files — a cryptographically signed Windows driver (signed with a legitimate signing certificate) and an executable "loader" application designed to install the driver — were used in tandem in a failed attempt to disable endpoint security tools on the targeted machines.

Further analysis of the loader application provided strong evidence it was a variant of a malware that Mandiant named **BURNTCIGAR**. Prior research by other companies indicate that threat actors who used this tool in prior attacks later attempted to deploy ransomware that calls itself Cuba.

Sophos privately reported the existence of all the signed drivers in this research to Microsoft who have published an advisory as part of today's Patch Tuesday release.

Driver abuse on the rise

The use of device drivers as a way to sabotage or terminate security tools has been on the rise in 2022. Some of the previous attacks have employed a "bring your own vulnerable driver" (BYOVD) approach, in which the attackers leverage a Windows driver from a legitimate software publisher that contains security vulnerabilities. Two examples of BYOVD attacks include the Robinhood ransomware in 2020 and most recently the BlackByte ransomware. Recent versions of the operating system require that drivers must carry a legitimate cryptographic signature in order to load properly.

The driver recovered in this attack bears a close resemblance to one used in a previously documented attack. In that case, the driver was digitally signed using a stolen NVIDIA digital certificate (one of the files leaked by threat actors known as Lapsus$). Another closely related driver, used in still-earlier attacks, had been digitally signed using certificates from one or more Chinese companies with more dubious reputations.

The research by SophosLabs indicates that the threat actors are moving up the trust pyramid, attempting to use increasingly more well-trusted cryptographic keys to digitally sign their drivers. Signatures from a large, trustworthy software publisher make it more likely the driver will load into Windows without hindrance, improving the chances that Cuba ransomware attackers can terminate the security processes protecting their targets' computers.

Kernel attacks can disable many security products

In most ransomware incidents, attackers kill the target's security software in an essential precursor step before deploying the ransomware itself. In recent attacks, some threat actors have turned to the use of Windows drivers to disable security products.

Because drivers pose a uniquely challenging risk to security, Windows enables Driver Signature Enforcement by default. The policy ensures that all kernel-mode drivers need to be signed in order to be loaded. If the policy is enabled and the driver is not signed, Windows will not load the driver, throw error code 577, and display a message that it cannot verify the digital signature for this file.

To get around this security measure, attackers must use a signed driver, preferably one that's signed with a currently valid key from a trustworthy source. There are two general ways to accomplish this.

The easier of the two is the "Bring Your Own Vulnerable Driver" (BYOVD) attacks, in which the attackers deploy legitimate, commercial software drivers that also happen to have security vulnerabilities. Easier because the attackers don't need to create the driver from scratch or obtain a leaked code-signing certificate: They just abuse one that already exists, preferably one that has gone through Windows certification, signed by the Microsoft Windows Hardware Compatibility Publisher (WHCP) certificate.

The more challenging method is for the attackers to contrive a way to get their custom driver, created by (or specially made for) the attackers, signed by a trusted certificate. This is the method these attackers employ.

Most kernel driver attacks have typically taken the BYOVD form. Recent examples include BlackByte ransomware, which used a vulnerable graphics card overclocking driver, and another ransomware actor abusing a vulnerable anti-cheat driver created by the software publisher of the video game Genshin Impact.

While Sophos products were listed among the targets for the driver, the driver was ultimately unsuccessful at terminating them due to Sophos anti-tampering features.

Signed code is supposed to be an assurance

Microsoft understands that kernel-mode drivers can perform highly privileged operations that user-mode processes cannot leverage, potentially reducing the effectiveness of some antimalware, endpoint security, or EDR products. So the company has created a quality assurance process that only gives its stamp of approval to software Microsoft can scrutinize, and an operating system that only loads the software that carries this stamp.

A cryptographic code signature is not just an extra tab in the file's properties sheet. It's meant to indicate a few things: Primarily, it means the code belongs to the company named in the code signature data, appended to the original program. It's an assertion, supposedly with proof, that the program is unchanged from what was presented to the code-signing company at a specific date and time.

Digital signatures also indicate a degree of safety and trust, as well.

Code signing entities like Microsoft may also weed out malicious drivers, using any of a variety of methods to scrutinize drivers submitted for signature.

Windows checks the validity of driver signatures before it allows drivers to load, unimpeded. As a code integrity measure, Microsoft maintains a blocklist that, when the company adds a driver to it, "revoke[s the driver] in the Windows kernel in the most recent security updates," a spokesperson for Microsoft's MSRC explained. Drivers revoked in this way "will no longer be trusted by the Windows platform and will be blocked from running."

Windows also treats some older software, like the drivers that were later abused in the BYOVD attacks, as validly signed — at first. Any given driver past its shelf-life may contain exploitable vulnerabilities, but Microsoft doesn't go looking for them in the millions of expired third-party drivers in existence. A certificate (or driver) only gets revoked after someone notices malware signed by it (or the driver itself) — and alerts Microsoft.

So while attackers may be able to place that driver file on the hard drive, and create an associated service to load it, Windows will throw errors if the driver isn't signed, or is signed by a revoked certificate.

Bottom line: It's much harder to get the file past these integrity checks imposed by the code signers, and get a legitimately signed file, than to create the bespoke malware driver in the first place.

While there's little doubt that the leak of NVIDIA's signing certificate led to that now-revoked certificate being used to sign one version of the driver, somehow, the malware's creator managed to get other versions of the driver signed by the code-signing authorities for Microsoft and other software publishers.

Poking through the BURNTCIGAR ashtray

The executable files recovered from this attack included an .exe (we refer to this as the 'loader') and a .sys file (the driver). By searching for related drivers in public repositories, we also discovered an archive containing both the driver and loader, as well as a plain text file that contains a list of 186 program filenames used by a large number of endpoint security and EDR software packages — presumably a list of processes to target. No such text file was recovered during the incident we investigated.

When executed, the loader copies the driver to *%temp%*, giving it a five- to seven-character filename, then creates a service entry for the driver (in this instance, named *mEfEk.sys*) by executing a command:

```
"C:\\Windows\system32\cmd.exe /c sc create mEfEk binPath=
%TEMP%\mEfEk.sys type= kernel start= demand"
```

Then the loader starts the service.

```
"C:\\Windows\system32\cmd.exe /c sc start mEfEk"
```

This process is mirrored in a subroutine extracted from the reverse-engineered loader, shown below.

```
sub_4015F9();
sub_401942();
ConsoleWindow = GetConsoleWindow();
ShowWindow(ConsoleWindow, 0);
memset(Buffer, 0, sizeof(Buffer));
v9 = 0;
GetEnvironmentVariableA("TEMP", Buffer, 0x104u);
v4 = strlen(Buffer);
lpNewFileName = (LPCSTR)malloc(v4 + 112);
sprintf((char *const)lpNewFileName, "%s%s", Buffer, "\\nvnjfmp.sys");
system("sc create nvnjfmp binPath= %TEMP%\\nvnjfmp.sys type= kernel start= demand");
CopyFileA("nvnjfmp.sys", lpNewFileName, 0);
system("sc start nvnjfmp");
v11 = -1;
v13 = InteractWithDriver(0x222088u, &v11, &v10);
if ( v13 != 1 )
{
  MessageBoxA(0i64, "Failed to connect driver!", "System Error", 0x1010u);
  CurrentProcess = GetCurrentProcess();
  TerminateProcess(CurrentProcess, 0);
}
v11 = v10;
v13 = InteractWithDriver(0x222088u, &v11, &v10);
if ( v13 != 1 )
{
  MessageBoxA(0i64, "Failed to connect driver!", "System Error", 0x1010u);
  v6 = GetCurrentProcess();
  TerminateProcess(v6, 0);
}
v14 = v10;
```

BURNTCIGAR

loader creating and starting a service to install the driver

This application works in conjunction with the driver to kill any service or process named like any of the ones listed in the plaintext file. 29 of the names in the list relate to Sophos.

```
{
  KillProcess("savadminservice.exe", v14);
  KillProcess("savservice.exe", v14);
  KillProcess("sedservice.exe", v14);
  KillProcess("alsvc.exe", v14);
  KillProcess("sophoscleanm64.exe", v14);
  KillProcess("sophosfs.exe", v14);
  KillProcess("sophosfilescanner.exe", v14);
  KillProcess("sophoshealth.exe", v14);
  KillProcess("mcsagent.exe", v14);
  KillProcess("mcsclient.exe", v14);
  KillProcess("sophossafestore64.exe", v14);
  KillProcess("sophossafestore.exe", v14);
  KillProcess("sspservice.exe", v14);
  KillProcess("swc_service.exe", v14);
  KillProcess("swi_service.exe", v14);
  KillProcess("sophosui.exe", v14);
  KillProcess("sophosntpservice.exe", v14);
  KillProcess("hmpalert.exe", v14);
  KillProcess("sophoslivequeryservice.exe", v14);
  KillProcess("sophososquery.exe", v14);
  KillProcess("sophosfimservice.exe", v14);
  KillProcess("swi_fc.exe", v14);
  KillProcess("swi_filter.exe", v14);
  KillProcess("sophosmtrextension.exe", v14);
  KillProcess("sdcservice.exe", v14);
  KillProcess("sophoscleanup.exe", v14);
  KillProcess("sophos", v14);
  KillProcess("ui.exe", v14);
  KillProcess("savapi.exe", v14);
  KillProcess("sophoscleanup.exe", v14);
  KillProcess("sophosntpservice.exe", v14);
  KillProcess("sophoscleanm64.exe", v14);
  KillProcess("sophosfs.exe", v14);
  KillProcess("sophoshealth.exe", v14);
  KillProcess("sophossafestore64.exe", v14);
  KillProcess("sophosfimservice.exe", v14);
  KillProcess("sophosfilescanner.exe", v14);
  KillProcess("mpcmdrun.exe", v14);
  KillProcess("msmpeng.exe", v14);
  KillProcess("NisSrv.exe", v14);
  KillProcess("MpCopyAccelerator.exe", v14);
  Sleep(0x3E8u);
}
```

Processes the malware attempts to prevent from running

The driver and the loader interact by accessing a symbolic link named “\\\\.\\KApcHelperLink1”. The program contains at least two hardcoded IOCTL codes, 0x222088 and 0x222094, used to interact with the driver. The loader uses IOCTL code 0x222094 to pass the process name, and signal the driver to kill a process.

The figure below shows the decompiled function responsible for interacting with the driver. First the BURNTCIGAR loader retrieves a handle to **\\\.\\KApcHelperLink1**, then it communicates with the driver via DeviceIoControl. The IOCTL code is passed as a function parameter. This distinctive “KApcHelper” string appeared in build paths of some of the drivers that weren’t packed with VMprotect. One of the samples we found had been uploaded to VirusTotal was named **KApcHelper_x64.sys**.

```
__int64 __fastcall InteractWithDriver(DWORD dwIoControlCode, void *lpInBuffer, void *a3)
{
  DWORD BytesReturned; // [rsp+40h] [rbp-10h] BYREF
  unsigned int v5; // [rsp+44h] [rbp-Ch]
  HANDLE hDevice; // [rsp+48h] [rbp-8h]

  BytesReturned = 0;
  v5 = 0;
  hDevice = CreateFileA("\\\\.\\KApcHelperLink1", 0xC0000000, 3u, 0i64, 3u, 0x80u, 0i64);
  if ( hDevice == (HANDLE)-1i64 )
    return 0i64;
  v5 = DeviceIoControl(hDevice, dwIoControlCode, lpInBuffer, 0x104u, a3, 0x104u, &BytesReturned, 0i64);
  CloseHandle(hDevice);
  return v5;
}
```

BURNTCIGAR stubs out a process

In February 2022, Mandiant wrote that BURNTCIGAR was "a utility … which terminates processes associated with endpoint security software to allow their ransomware and other tools to execute uninhibited." Mandiant reported, at that time, that adversaries bypassed security on targeted systems, using a legitimate-but-vulnerable driver (originally published by antivirus vendor Avast) as its BYOVD.

In August 2022, Palo Alto Networks (PAN) noted a change in tactics. Specifically, PAN wrote that "[t]he change of tactics by Tropical Scorpius (aka UNC2596, aka 'the Cuba attackers') is to make use of the expired legitimate NVIDIA certificate, as well as use of their own driver targeting security products for termination." Instead of the vulnerable Avast driver, BURNTCIGAR was now installing the *ApcHelper.sys* driver signed with a leaked, revoked NVIDIA certificate.

SophosLabs' retrohunt for driver code matching the ones in the files we analyzed also uncovered at least three variants of the driver signed by code signing certificates that belong to companies based in China, first seen between July and September. In the case of one of those three, the code signing authority revoked the certificate used to sign the driver.

The certificate used to sign the other two has not been revoked, but it belongs to a prolific publisher of software that Sophos flags as a "PUA," or Potentially Unwanted Application. PUAs occupy a fuzzy middle ground between legitimate software and malware. Sophos products will detect anything signed by this publisher's certificate as a PUA, and will prevent its execution or installation. Out of the 41 programs signed with this publisher's key uploaded in the past 90 days to VirusTotal, only one has zero detections by antimalware products; 38 of them are classified as either malware or PUA software by 10 or more security companies.

The crew behind BURNTCIGAR have once again shifted their tactics, moving from signing their AV-killer driver with stolen certificates, to valid certificates of dubious origin, to legitimate certificates of good repute.

The sequence of discoveries

SHA-256 hash	signed by	uploaded to VT

9b1b15a3aacb0e786a608726c3abfc94968915cedcbd239ddf903c4a54bfcf0c	2022-07-27 21:05:11	Zhuhai liancheng Technology Co., Ltd.
42b22faa489b5de936db33f12184f6233198bdf851a18264d31210207827ba25	2022-08-09 14:07:45	Windows Hardware Compatibility Publisher
6839fcae985774427c65fe38e773aa96ec451a412caa5354ad9e2b9b54ffe6c1	2022-08-21 15:30:50	Windows Hardware Compatibility Publisher
7f4555a940ce1156c9bcea9a2a0b801f9a5e44ec9400b61b14a7b1a6404ffdf6	2022-08-22 15:38:20	Windows Hardware Compatibility Publisher
d7c81b0f3c14844f6424e8bdd31a128e773cb96cccef6d05cbff473f0ccb9f9c	2022-09-20 02:36:50	NVIDIA Corporation
5f6fec8f7890d032461b127332759c88a1b7360aa10c6bd38482572f59d2ba8b	2022-09-24 06:42:44	Beijing JoinHope Image Technology Ltd.
0440ef40c46fdd2b5d86e7feef8577a8591de862cfd7928cdbcc8f47b8fa3ffc	2022-09-30 16:50:19	Zhuhai liancheng Technology Co., Ltd.
274340f7185a0cc047d82ecfb2cce5bd18764ee558b5227894565c2f9fe9f6ab	2022-10-21 02:45:30	Windows Hardware Compatibility Publisher
0d10c4b2f56364b475b60bd2933273c8b1ed2176353e59e65f968c61e93b7d99	2022-10-21 02:47:56	Windows Hardware Compatibility Publisher
c8f9e1ad7b8cce62fba349a00bc168c849d42cfb2ca5b2c6cc4b51d054e0c497	2022-10-21 20:45:56	Windows Hardware Compatibility Publisher
8e035beb02a411f8a9e92d4cf184ad34f52bbd0a81a50c222cdd4706e4e45104	2022-10-27 01:19:05	Windows Hardware Compatibility Publisher

Table: The list of drivers, sorted by the date they were uploaded to VirusTotal, with their code-signers

Two of the drivers we encountered were signed with digital certificates from a Chinese company, Zhuhai Liancheng Technology Co., Ltd. One driver signed with this key was used on July 27, 2022, the earliest of this batch. The Zhuhai Liancheng Technology (ZLT) certificate is distinctive because Sophos detects any file signed with this certificate as a PUA.

In August, three WHCP-signed samples were uploaded to VirusTotal a few weeks apart.

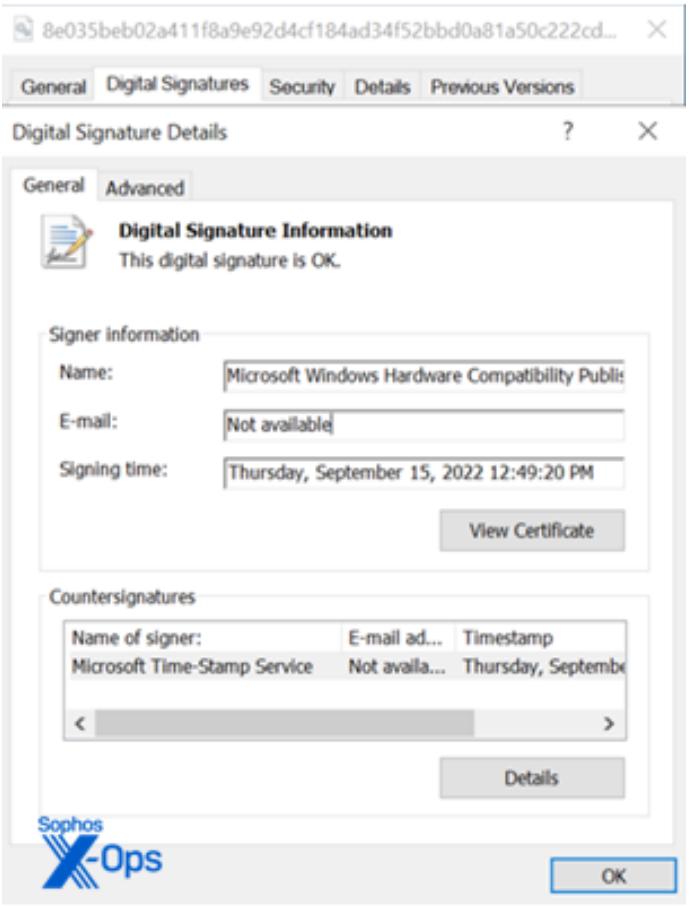
The next month, the attackers used a driver signed with the leaked, revoked NVIDIA certificate on September 20, then one signed by a certificate from Beijing JoinHope Image Technology, Ltd. (aka JHI) four days later. Six days after the JHI-signed driver was used, someone uploaded another build of the driver to VirusTotal, again signed by ZLT. Our telemetry indicates that this same driver was used in an attack that took place on October 21, 2022, nearly a month after it was first uploaded.

Four different drivers were uploaded in October, all of which were signed with the WHCP certificate.

All the driver variants found through hunting use the same symbolic link name, **KApcHelperLink1**, as did the driver found in our incident. The drivers also share other characteristics, such as the validity date range for the code signing. Interestingly, all drivers signed with the WHCP certificate were packed with the VMProtect packer.

Packer may conceal malicious driver code

VMProtect is a commercial runtime packer that protects software from reverse-engineering through a plethora of obfuscation techniques. While there are legitimate uses for packers in commercial software, malware executables commonly use packers (both legitimate commercial ones, and packers created specifically for malware) to complicate analysis. Packers are less commonly used in drivers.



The signature of the driver

Through a combination of static and dynamic analysis of the loader and driver, we were able to extract multiple technical artifacts that allowed us to hunt for potential unpacked variants of the driver, as well as similar versions of BURNTCIGAR.

We installed the driver on our analysis system and confirmed that the driver creates a symbolic link (named “KApcHelperLink1”) to the device **\device\KApcHelper1.**

If the symbolic link name looks familiar, it’s because it’s pretty close to the name of the driver module investigated by PAN’s Unit42: *ApcHelper.sys*. Pivoting our searches off of these terms, we found eleven total drivers, signed by four different signing keys. The earliest known sample (so far) of Microsoft-signed driver was uploaded to VirusTotal on August 9.

Analysis of the BURNTCIGAR loader

In this screenshot of our analysis of the code flow of allx7_64.exe, the BURNTCIGAR loader variant we found onVirusTotal, the loader uses same IOCTL code (as well as the same symbolic link name).

```
*((_DWORD *)DriverObject->DriverSection + 26) |= 0x20u;
((void (__fastcall *)(__int64 (__fastcall *)()))qword_140004068)(nullsub_1);
((void (__fastcall *)(__int64 (__fastcall *)(), _QWORD))qword_140004060)(sub_140001800, 0i64);
RtlInitUnicodeString(&DestinationString, L"\\device\\KApcHelper1");
result = IoCreateDevice(DriverObject, 0, &DestinationString, 0x22u, 0x100u, 0, &DeviceObject);
if ( result >= 0 )
{
  RtlInitUnicodeString(&SymbolicLinkName, L"\\dosdevices\\KApcHelperLink1");
  v3 = IoCreateSymbolicLink(&SymbolicLinkName, &DestinationString);
  if ( v3 >= 0 )
  {
    // DeviceIoCtrlResponse function responds to same IOCTL codes as used by BURNTCIGAR
    memset64(DriverObject->MajorFunction, (unsigned __int64)DeviceIoCtrlResponse, 0x1Bui64);
    DriverObject->DriverUnload = 0i64;
    DriverObject->MajorFunction[14] = (PDRIVER_DISPATCH)sub_140001B70;
    return 0;
  }
  else
  {
    _mm_lfence();
    IoDeleteDevice(DeviceObject);
    return v3;
  }
}
```

Similar code structure of BURNTCIGAR variant found on VirusTotal

In this screenshot of decompiled driver code, taken from the later driver signed by Zhuhai Liancheng Technology Co. Ltd., the routine creates the device **\\device\\KApcHelper1** and symbolically links it to **\\dosdevices\\KApcHelperLink1**, the same symbolic link name used by the BURNTCIGAR variant in the incident we investigated to communicate with the driver.

Driver variant using the same symbolic link name and device name as driver used in our incident

Connecting The Dots

We found several compelling similarities between the driver in this latest incident and earlier samples referenced in the PAN and Mandiant reports, as well as additional samples found through retrohunts on VirusTotal:

- The naming scheme of the drivers as well as the installed symbolic link names are the same or similar. In PAN's report, the driver is called ApcHelper.sys. The driver used in our incident is called KApcHelper.sys. We have not found any potential false positives through telemetry review so far.
- Our incident and the incident from PAN's report are both linked to Cuba ransomware, with high confidence.
- While not all samples have identical PE file metadata, we noticed that the Microsoft-signed driver from the incident we analyzed (SHA256: 8e035beb02a411f8a9e92d4cf184ad34f52bbd0a81a50c222cdd4706e4e45104) shares the same PE Header timestamp (2022:06:02 10:09:08+00:00) as the drivers signed by Zhuhai Liancheng Technology Co., Ltd. (ZLT) and the Beijing JHI driver.

In incidents investigated by Sophos, threat actors tied to Cuba ransomware used the BURNTCIGAR loader utility to install a malicious driver signed using Microsoft's certificate. The signature data embedded in those signed drivers shared the following common characteristics:

The WHCP certificate "Valid From" and "Valid To" fields:

```
Valid From: 2022-06-07 18:08:06
Valid To: 2023-06-01 18:08:06
```

The "Valid Usage" field:

```
Valid Usage WHQL Crypto, 1.3.6.1.4.1.311.10.3.5.1, Code Signing
```

The WHCP certificate used to sign the driver was issued by an authority called **Microsoft Windows Third Party Component CA 2014**, with "Valid To" and "Valid From" fields:

```
Valid From  2014-10-15 20:31:27
Valid To    2029-10-15 20:41:27
```

Sophos detects the malicious driver as **Troj/Agent-BJJB**, **Mal/Rootkit-VC**, or **Mal/Rootkit-VF**, and the malicious loader as **Troj/AVKill-Q**. Indicators of compromise for files relating to this investigation were shared prior to publication with Microsoft, to assist them with protection, and have been published to the SophosLabs Github.

For more information, there's additional research from Mandiant and SentinelOne from their independent, parallel investigations.

Acknowledgments

Sophos X-Ops acknowledges the analysis contributions of Rapid Responders Sergio Bestulic, Harinder Bhathal, Matthew Everts, Johnathan Fern, Jason Jenkins, Melissa Kelly, Elida Leite, Stephen McNally, Duke Thornley, Rajat Wason, Robert Weiland, and Carlos Zena; and SophosLabs researchers Richard Cohen, Andreas Klopsch, Prashant Kumar, Andrew Ludgate, Nathan Mante, Rajesh Nataraj, Simon Porter, Felix Weyne, and Michael Wood. Sophos also thanks Microsoft for rapidly engaging with us to help protect computer users.