Researcher Notes – Ransomware Groups Bring Driver-Based AV Killers to Kill AV and Blind EDR

Report Id: CW-20251102
Version: 1.0 (20.November.2025)

Executive Summary

In a recent investigation regarding the latest Qilin ransomware Tactics, Techniques, and Procedures (TTPs), we discovered that the Qilin ransomware group had deployed an Anti-Virus (AV) Killer tool dubbed KillaTeresa, named after the consistent filename observed across their operations. This tool operates with kernel-level capabilities, allowing it to terminate antivirus processes and blind Endpoint Detection Response (EDR) solutions, enabling attackers to operate effectively without being blocked by security vendors.

In the KillaTeresa execution flow, the AV-Killer tool drops two driver files onto the victim's disk. The first is a signed vulnerable driver that leverages the Bring-Your-Own-Vulnerable-Driver (BYOVD) technique. This driver is abused to gain access to kernel memory and modify critical kernel data structures, effectively disabling EDR visibility. The second is an unsigned malicious driver that *untrusts* files by setting an empty (Deny All) Discretionary Access Control List (DACL) and terminates antivirus processes.

Our observations reveal that KillaTeresa also has another variant, KillaStop, which utilizes a different vulnerable driver and targets more antivirus processes and EDR drivers but shares the same codebase as KillaTeresa. In this case, we have observed this variant in cases involving Makop, Akira, PandaLocker, and MedusaLocker ransomware.

This report is a continuation and provides a more detailed analysis than the previous report titled "Qilin Ransomware TTPs in 2025" [1], which briefly mentioned the malicious driver in the section "TA0005: Defense Evasion" but did not provide a detailed explanation of KillaTeresa's main malware or its exploitation of the signed vulnerable driver.

This report in a nutshell:

- Qilin, Makop, Akira, PandaLocker, and MedusaLocker ransomware bring driver-based AV-killer tools to terminate AV processes and blind EDR solutions;
- The AV-killer tools, named KillaTeresa and KillaStop, were packed with BabbleLoader - a packer widely used by other malware in the wild;
- KillaTeresa and KillaStop drop and load both a vulnerable driver and a malicious custom driver;
- The vulnerable driver contains two vulnerable Input/Output Control (IOCTL) functions that allow reading from and writing to physical memory through the Memory-mapped I/O (MMIO) space;
- KillaTeresa and KillaStop exploit these vulnerabilities to achieve read and write primitives. In this case, they obtained physical memory read and write primitives to modify kernel data structures to remove notification routines and callbacks installed by EDRs
- After blinding EDRs, blacklisted processes are then terminated.

[1] Qilin Ransomware TTPs in 2025

Techniques, Tactics and Procedures specific for this campaign:

Implants
AV-Killer variants: KillaTeresa and KillaStop

Victimology
Users in Malaysia, Brazil, India, Colombia, Mexico, and Ecuador

Kaspersky's products detect this threat as `Backdoor.Win32.Lilith.gen`, `Trojan.Win32.Arkmblk`, `Trojan.Win64.BabbleLoader`, `Rootkit.Win64.KillAV` and `Trojan.Win64.Agent.gen`. Kaspersky endpoint products' self-defense and anti-rootkit component protect against mentioned drivers.

This Report is composed of Kaspersky researchers' findings. Although it has been peer reviewed to ensure its quality, it's aimed towards sharing actionable, partial intelligence in a timely manner and therefore may not contain all the information we usually provide.
For more information please contact: crimewareintel@kaspersky.com

Technical Details

Background

In a recent investigation into the latest Qilin ransomware techniques, we found a tool that had been dropped and executed on multiple victim devices alongside the Qilin ransomware executable. The tool, which we named KillaTeresa, was found to be packed with *BabbleLoader* packer, which contained a large amount of junk code, junk strings, and a shellcode execution routine. The unpacked version of KillaTeresa is a PE file designed to terminate antivirus processes and disable visibility of EDR solutions.

In another case, we found that KillaTeresa has a variant we called KillaStop, which has been used by multiple ransomware groups, including Akira, Makop, MedusaLocker, PandaLocker, and an unknown crypto-miner.

The figure below shows the connections between ransomware operators and the AV-killer variants.

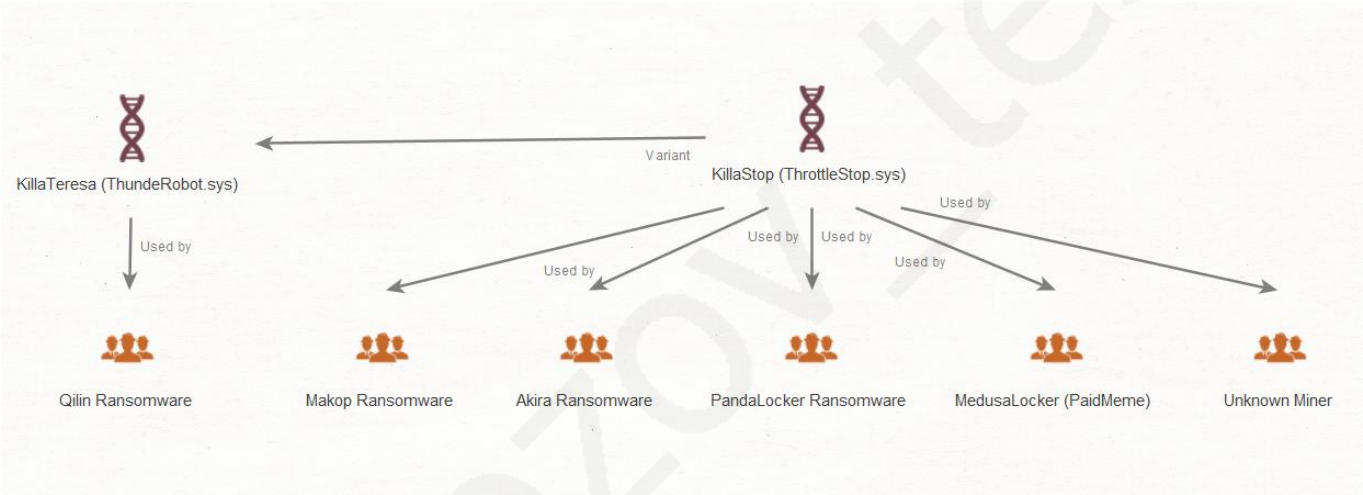

Fig. 1 Connections between ransomware operators and the AV-killer variants

Based on our analysis, both variants share the same code base but abuse different vulnerable drivers and KillaStop contains a more extensive list of antivirus processes and EDR drivers.

Variant	Filenames used	Abused driver	Associate signed driver's software
KillaTeresa	`k1ll4_t3r3sa_0x00013C402290187B.exe` / `b1td3f_0nly.exe`	`ControlCenter.sys`	ThunderRobot
KillaStop	`k1ll4_0x00081281324D4A6.exe` / `0th3r_av5.exe`	`ThrottleStop.sys`	ThrottleStop

So, as both variants share the same code base but abuse different drivers, this report will not cover the KillaStop variant in detail. Moreover, the vulnerable driver `ThrottleStop.sys` has already been discussed in another report [2], where an AV-killer (unrelated to KillaTeresa and KillaStop) used by MedusaLocker also leveraged it to terminate antivirus processes.

[2] AV Killer abusing vulnerable drivers, used by Ransomware operators

KillaTeresa's first stager: BabbleLoader packed

Both KillaTeresa and KillaStop have been packed by a packer called BabbleLoader [3]. Based on our research, multiple malware families in the wild have also been packed with BabbleLoader, such as Bumblebee, Amadey, Lumma stealer, Meduza stealer, and the most recent, CastleRAT [4]. The presence of BabbleLoader across multiple malware families shows that this packer is not exclusive to a single threat actor.

The table below shows the details of one of the KillaTeresa executables packed with BabbleLoader.

MD5	BAE7EBAFD73D4E511671275DA72CCEE6
SHA1	15DBBC1826D081A4EA2D214E61E38C45853D22B8
SHA256	8AF1D9FAA51A21DF56C11851264F9A29DAB637AB02C2B1EFBD26F762F433354F
Link time	2023-05-22 08:51:28 UTC
File type	PE32+ executable (GUI) x86-64, for MS Windows
Compiler	Microsoft Visual C/C++(2015 v.14.0)[-]
File size	1.43 MB
File name	k1ll4_t3r3sa_0x00023F58D5F05B8.exe

In this report, we do not provide a detailed explanation of BabbleLoader, as it has already been covered in an Intezer [5] report. However, there are several characteristics of BabbleLoader based on our analysis of the KillaTeresa sample:

1. It contains a large amount of junk code, performs pointless assembly instructions, and executes unnecessary Windows API function calls. This prevents analysis tools such as IDA Pro from generating clean decompiled code, making the analysis harder.

2. The junk API calls and junk strings differ in each generated malware sample, suggesting that BabbleLoader applies polymorphism to API calls and strings. Moreover, the polymorphic mechanism is also applied to the file metadata of generated samples.

3. It contains anti-debugger mechanisms such as the `INT3` instruction.

4. It checks the installed graphics adapters and the number of running processes, which is likely intended to detect sandbox environments.

5. It also contains anti-emulation [6] techniques, such as attempting to dynamically import non-existent functions from system DLLs. If these functions are resolved successfully, which only occurs in emulator environments using Virtual DLLs (VDLLs), the malware terminates the execution.

6. It implements the Halo's Gate [7] system calls execution technique by performing direct syscalls to evade user-mode API hooks placed by EDR and AV solutions.

[3] BabbleLoader
[4] CastleRAT report by RecordedFuture
[5] BabbleLoader
[6] Raspberry Robin and its new anti-emulation trick
[7] Halo's Gate technique

7. It resolves critical APIs such as `MapViewOfFile`, `CreateFileMapping`, `UnmapViewOfFile`, and `VirtualProtect`, which are then used to map a shellcode into a memory section. This shellcode is responsible for decrypting the Donut [8] shellcode.

8. Finally, it executes the shellcode that decrypts and loads the Donut shellcode at the end of the function.

KillaTeresa's second stage: Donut Shellcode

In the second stage, BabbleLoader executes shellcode generated with the open-source shellcode generator Donut. This behavior is observed in nearly all malware samples packed with BabbleLoader. According to the project repository on GitHub [9], Donut shellcode is designed to load and execute PE files directly from memory. In this context, it is used to run the final stage of the AV-Killer component (a portable executable file).

Based on the GitHub repository of TheWover's Donut, the tool supports the following features:

- Optional encryption of the embedded payload using the Chaskey block cipher with a randomly generated 128-bit key;
- Compression of input payloads using aPLib, LZNT1, Xpress, or Xpress Huffman via `RtlCompressBuffer`;
- Anti-detection features, such as bypassing AMSI (Antimalware Scan Interface), WLDP (Windows Lockdown Policy), and ETW (Event Tracing for Windows);
- Multiple output formats, including C, Python, Ruby, PowerShell, C#, Base64, Hex, and UUID strings.

In this context, to dump the final-stage component from memory, analysts can monitor the invocation of the `RtlDecompressBuffer` API and capture the resulting decompressed file.

KillaTeresa final stage: AV-killer

The final stage of KillaTeresa comes in the form of a PE file. This executable drops two drivers: a signed vulnerable driver (`rwdrv.sys`) and an unsigned malicious driver (`hlpdrv.sys`), both of which are embedded within the executable.

The signed driver's arbitrary physical memory read/write primitives are abused to access kernel memory, and these primitives are exposed via `DeviceIoControl`. The tool then leverages this access to terminate antivirus processes, disable EDR drivers, and apply deny-all DACLs to "untrust" targeted files.

The table below provides details of the drivers that the KillaTeresa variant drops and loads.

Filename	Symbolic Link	SHA256	Remark
`rwdrv.sys` (`ControlCenter.sys`)	`\\DosDevices\\CCDRV1`	`6A784B693DA28EB0EB3E8C6E233C0BBD`	Signed driver with arbitrary physical memory read/write (R/W) capability
`hlpdrv.sys`	`\\DosDevices\\KMHLPDRV`	`CF7CAD39407D8CD93135BE42B6BD258F`	Anti-virus process terminator

Meanwhile, for the KillaStop variant, the table below shows the details of the drivers that the KillaStop variant drops and loads.

[8] Donut GitHub repository link
[9] Donut GitHub repository link

Filename	Symbolic Link	SHA256	Remark
rwdrv.sys (ThrottleStop.sys)	\\DosDevices\\mgdsrv	6BC8E3505D9F51368DDF323ACB6ABC49	Signed driver with arbitrary physical memory read/write (R/W) capability
hlpdrv.sys	\\DosDevices\\KMHLPDRV	CF7CAD39407D8CD93135BE42B6BD258F	Anti-virus process terminator

Geo-fencing via Locale Checks

Upon dropping the drivers, KillaTeresa checks whether the system locale matches any of several specific values. If a match is found, execution terminates. Based on these locale strings, the following countries are excluded from execution, primarily covering the Commonwealth of Independent States (CIS) region:

Locale String	Country
ru-RU	Russia
ru-BY	Belarus
ru-KG	Kyrgyzstan
ru-MD	Moldova
ru-UA	Ukraine
kk-KZ	Kazakhstan
ky-KG	Kyrgyzstan
uz-Cyrl, uz-Cyrl-UZ	Uzbekistan
uz-Latn, uz-Latn-UZ	Uzbekistan
uz-Arab	Uzbekistan
az-Cyrl, az-Cyrl-AZ	Azerbaijan
az-Latn, az-Latn-AZ	Azerbaijan
ka-GE	Georgia
uk-UA	Ukraine
tg-Cyrl, tg-Cyrl-TJ	Tajikistan
tk-TM	Turkmenistan

Privilege Adjustments and Virtual-to-Physical Address Mapping (Superfetch)

Based on our investigation of the final stage, KillaTeresa attempts to enable the following three privileges:

- SeProfileSingleProcessPrivilege
- SeDebugPrivilege
- SeLoadDriverPrivilege

In this context, SeProfileSingleProcessPrivilege is required for Superfetch queries. With these privileges enabled, the payload queries Superfetch via RtlGetNativeSystemInformation to build a memory map that is later used by the vulnerable driver's physical read/write primitives. Both KillaTeresa and KillaStop leverage this memory map to calculate physical memory addresses for subsequent read and write operations.

SuperfetchMemoryRangesQuery

After creating the virtual-to-physical map, the malware obtains an array of physical spans. Each span is described by a base Page Frame Number (PFN) and a count, which indicate valid RAM pages and prevent invalid physical access.

```
typedef struct _PF_PHYSICAL_MEMORY_RANGE {
    ULONG_PTR BasePfn;
    ULONG_PTR PageCount; // pages [BasePfn .. BasePfn+PageCount-1]
} PF_PHYSICAL_MEMORY_RANGE, *PPF_PHYSICAL_MEMORY_RANGE;
```

SuperfetchPfnQuery

This process returns per-PFN identities (`MMPFN_IDENTITY`) for the selected PFNs. From each entry, the PFN is converted into a physical address (`PFN << 12`). The corresponding virtual address and physical address are then stored as an entry in a global Virtual-to-Physical Address Map.

Fig. 2 Creation of global VA-to-PA Mapping

rwdrv.sys: Signed Vulnerable Driver Used for Physical R/W

During KillaTeresa's execution, the malware drops the embedded signed vulnerable driver `rwdrv.sys` into the temporary directory and loads it. At this stage, it employs the Bring-Your-Own-Vulnerable-Driver (BYOVD) technique.

MD5	6A784B693DA28EB0EB3E8C6E233C0BBD
SHA1	99D26B558C423EA4BEDD0E1EC38B8E590DC3D98A
SHA256	0B12EB25DB68D8714BA52583597ED20E5FAB2F6E82DCD0BCB23161ACB4A9A126
Link time	2017-01-17 07:13:27 UTC

File type	PE32+ executable (native) x86-64, for MS Windows
Compiler	Microsoft Visual C/C++ (15.00.30729) [LTCG/C]
File size	11.54 KB
File name	rwdrv.sys

In this case, `rwdrv.sys` is a signed driver that exposes arbitrary physical memory read/write primitives and is known to be vulnerable. The underlying driver exploited is `ControlCenter.sys/ControlCenter64.sys` from ThundeRobot Control Center v2.0.0.10, with the vulnerability assigned CVE-2024-39251.

Combined with the previously created virtual-to-physical memory mapping, these primitives are leveraged to locate and modify critical kernel data structures, effectively disabling EDR visibility.

The table below lists the key properties of the vulnerable driver:

Description	Details
Symbolic Link	`\\.\CCDRV1`
Service Name	`Mgdsrv`
Phys. memory read IOCTL	`0x9C406104`
Phys. memory write IOCTL	`0x9C40A108`

IOCTL 0x9C406104: Arbitrary Physical Memory Read

In the vulnerable driver, this function accepts an input buffer (including the physical address, size, and driver's switch logic) and returns the requested bytes in the I/O Request Packet's output buffer. Once a call to `DeviceIoControl` succeeds, the malware receives the data from the driver's output buffer. This works by first mapping the physical address into virtual memory and reading the bytes from the virtual address before unmapping it.

The following figure shows the implementation of an arbitrary physical memory read function, where physical addresses are mapped into virtual memory using `MmMapIoSpace`, copied into the output buffer, and then unmapped via `MmUnmapIoSpace` once the operation is completed.

```
18
19  if ( a2 != 16 )
20    return 0xC000000DLL;
21  v9 = PHYSICAL_ADDRESS[1].HighPart * PHYSICAL_ADDRESS[1].LowPart;
22  if ( a4 < v9 )
23    return 0xC000000DLL;
24  v10 = v9;
25  MappedVA = MmMapIoSpace(*PHYSICAL_ADDRESS, v9, MmNonCached);// Maps Physical Memory and returns mapped Virtual Address
26  v12 = 0;
27  switch ( PHYSICAL_ADDRESS[1].LowPart )
28  {
29    case 1u:
30      qmemcpy(outputBuffer, MappedVA, (unsigned int)PHYSICAL_ADDRESS[1].HighPart);// Copies Data from Mapped Virtual Address
31      break;
32    case 2u:
33      HighPart = (unsigned int)PHYSICAL_ADDRESS[1].HighPart;
34      v17 = outputBuffer;
35      v18 = MappedVA;
36      while ( HighPart )
37      {
38        *v17++ = *v18++;
39        --HighPart;
40      }
41      break;
42    case 4u:
43      v13 = (unsigned int)PHYSICAL_ADDRESS[1].HighPart;
44      v14 = outputBuffer;
45      v15 = MappedVA;
46      while ( v13 )
47      {
48        *v14++ = *v15++;
49        --v13;
50      }
51      break;
52    default:
53      v12 = 1;
54      break;
55  }
56  MmUnmapIoSpace(MappedVA, v10);              // Once copy is completed, unmap that Virtual Memory
57  if ( v12 )
58    return 0xC000000DLL;
59  *a5 = a4;
0000098C arb_phy_read_sub_1158C:12 (158C)
```

Fig. 3 Arbitrary Physical Memory Read

IOCTL 0x9C40A108: Arbitrary Physical Memory Write

For arbitrary physical memory writes, the vulnerable driver uses `DeviceIoControl` with an input buffer that specifies the target physical address, the number of bytes to write, the data to be written, and the driver's switch logic. If the call succeeds, the driver writes the supplied data to the specified physical memory.

The following figure shows the implementation of this routine, where the driver maps the target physical address into virtual memory using `MmMapIoSpace`, performs the write operation via `memcpy`, and then unmaps the address with `MmUnmapIoSpace` once the operation is completed.

```
19  if ( a2 < 0x10 )
20    return 3221225485LL;
21  v7 = PHYSICAL_ADDRESS[1].LowPart * PHYSICAL_ADDRESS[1].HighPart;
22  if ( a2 < v7 + 16 )
23    return 3221225485LL;
24  vmmaped = (PHYSICAL_ADDRESS *)MmMapIoSpace(*PHYSICAL_ADDRESS, v7, MmNonCached);// mapping of physical to virtual
25  v9 = 0;
26  switch ( PHYSICAL_ADDRESS[1].LowPart )
27  {
28    case 1u:
29      qmemcpy(vmmaped, &PHYSICAL_ADDRESS[2], (unsigned int)PHYSICAL_ADDRESS[1].HighPart);// Writing operation
30      break;
31    case 2u:
32      HighPart = (unsigned int)PHYSICAL_ADDRESS[1].HighPart;
33      v14 = PHYSICAL_ADDRESS + 2;
34      v15 = vmmaped;
35      while ( HighPart )
36      {
37        *v15 = v14->LowPart;
38        v14 = (PHYSICAL_ADDRESS *)((char *)v14 + 2);
39        ++v15;
40        --HighPart;
41      }
42      break;
43    case 4u:
44      v10 = (unsigned int)PHYSICAL_ADDRESS[1].HighPart;
45      v11 = PHYSICAL_ADDRESS + 2;
46      v12 = (DWORD *)vmmaped;
47      while ( v10 )
48      {
49        *v12 = v11->LowPart;
50        v11 = (PHYSICAL_ADDRESS *)((char *)v11 + 4);
51        ++v12;
52        --v10;
53      }
54      break;
55    default:
56      v9 = 1;
57      goto LABEL_18;
58  }
59  _InterlockedOr(v16, 0);
60LABEL_18:
61  MmUnmapIoSpace(vmmaped, v7);                  // unmap after write is complete
62  if ( v9 )
63    return 3221225485LL;
64  *a5 = 0;
65  return 0;
66}
```

Fig. 4 Arbitrary Physical Memory Write

Environment Checks and Debug Messages

KillaTeresa does not work across all Windows versions, as its behavior depends on specific driver versions such as `ntoskrnl.exe` and `ci.sys`. For example, the routine `CI!CiValidateImageHeader` is located by starting from `SeGetCachedSigningLevel`. If the resolution fails or the hardcoded instruction byte sequence is not found, the malware displays a message box containing the string:

“CRITICAL ERROR!!! TELL CODER IF IT’S APPEARED!”

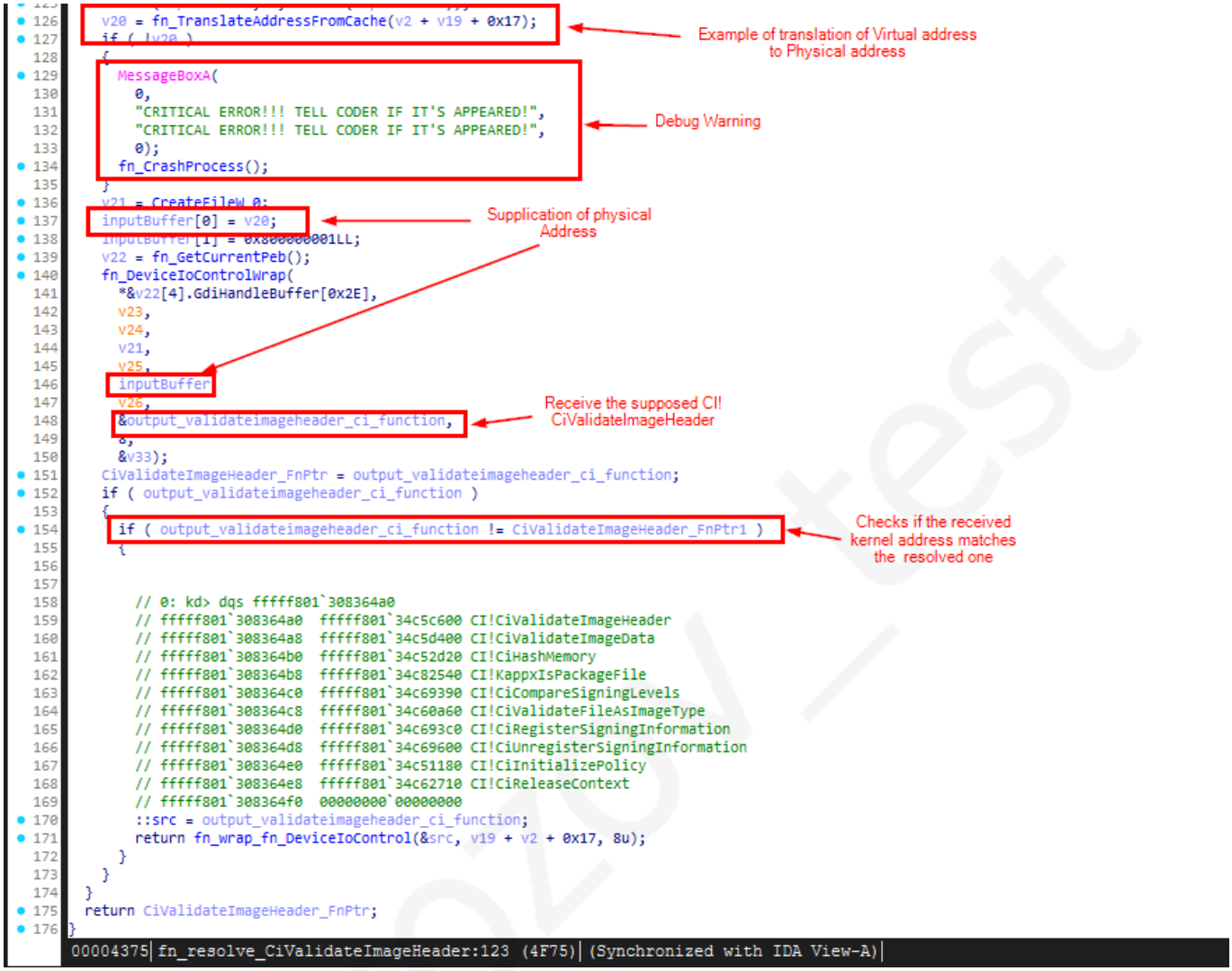

Fig. 5 Checking Manual Resolution against Resolved Function pointer of `CiValidateImageHeader`

It, however, tries to cater for differences in offsets across certain Windows major and minor versions, which are obtained through `RtlGetNtVersionNumbers`.

Blinding EDRs via Callback & Notification Suppression

After `rwdrv.sys` is loaded, KillaTeresa clears six classes of kernel notification routines and callbacks. This is achieved by locating global kernel variables through pattern matching within kernel functions.

For example, the `nt!CallbackListHead` address is obtained by:

- Matching instruction patterns within `CmUnregisterCallback`, and
- Calculating the kernel address from the relative offset to `nt!CallbackListHead`.

This variable is an array of registry callbacks whose entries are cleared if the callback address falls within the memory range of blacklisted drivers. KillaTeresa applies the same approach to deprive blacklisted EDR and AV components of visibility into processes, threads, modules, handles, registry activity, and file I/O operations.

The following table lists the different callbacks and notification routines that KillaTeresa clears if they belong to its blacklist:

Callback / Notification Type	Registered With	Kernel Data Structure Located
[1] Process Notifications	`PsSetCreateProcessNotifyRoutine[Ex]`	`PspSetCreateProcessNotifyRoutine`
[2] Thread Notifications	`PsSetCreateThreadNotifyRoutine[Ex]`	`PspCreateThreadNotifyRoutine`
[3] Image-Load Notifications	`PsSetLoadImageNotifyRoutine[Ex]`	`PspLoadImageNotifyRoutine`
[4] Object Callbacks	`ObRegisterCallbacks()`	`PsProcessType` and `PsThreadType`
[5] Registry Callbacks	`CmRegisterCallback[Ex]`	`CallbackListHead`
[6] Minifilter Callbacks	`FltRegisterFilter`	`FltGlobals`

The following figure shows the corresponding callback and notification types.

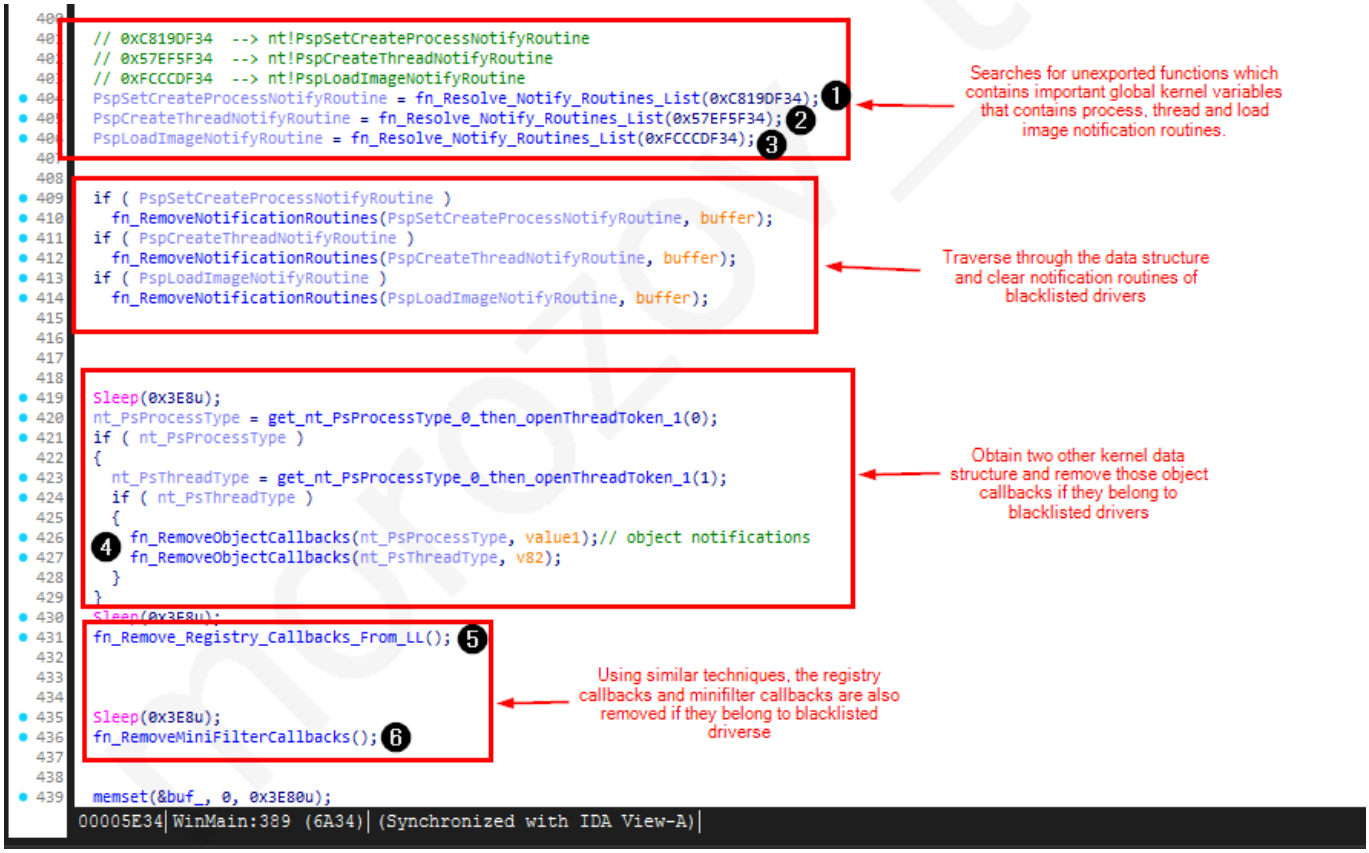

Fig. 6 KillaTeresa blinding EDR by clearing their notification/callbacks

In this context, our researchers assess that the final-stage payload contains code that may have been adapted from RealBlindingEDR, an open-source tool for disabling security solutions, based on observed code similarities [10].

[10] RealBlindingEDR GitHub Repository

Blacklisted Drivers

In the KillaTeresa execution flow, the malware collects the addresses of registered notification routines and callbacks, and resolves their owning modules using `RtlGetNativeSystemInformation` (system module list). Each address is range checked, and if a match is found, the module (driver) name is obtained and compared against a blacklist. If listed, the associated notifications and callbacks are removed.

The following list contains security vendors whose drivers are blacklisted: Kaspersky, QAX, Qihoo 360, Doctor Web, Microsoft, Trend Micro, Antiy Zhijia, ESET, Avast, AVG, Bitdefender, Avira, Sophos, SentinelOne, Acronis, Panda Security, Webroot, Symantec, McAfee.

Meanwhile, KillaStop contains additional driver names that are not present in the KillaTeresa list. The following list presents security vendors with extra drivers included in KillaStop but absent in KillaTeresa (some of the vendors are present only in KillaStop): Kaspersky, Qianxin, Qihoo360, Huorong Internet Security, Sangfor, McAfee, CheckPoint Security, CallBack Technologies, ZoneAlarm, Cisco, Immunet, IObit Advanced SystemCare Security module, Fortinet FortiEDR / FortiClient, CrowdStrike, QuickHeal, NComputing vSpace.

hlpdrv.sys: Malicious unsigned driver

The second driver the malware drops into the temporary folder is the malicious unsigned driver named `hlpdrv.sys`.

MD5	`CF7CAD39407D8CD93135BE42B6BD258F`
SHA1	`CE1B9909CEF820E5281618A7A0099A27A70643DC`
SHA256	`BD1F381E5A3DB22E88776B7873D4D2835E9A1EC620571D2B1DA0C58F81C84A56`
Link time	`2025-03-03 11:11:32 UTC`
File type	`PE32+ executable (native) x86-64, for MS Windows`
File size	`12.0 KB`
File name	`hlpdrv.sys`

`hlpdrv.sys` is loaded after EDR notifications and callbacks are suppressed. It is used to *untrust* files by applying a deny-all DACL and to terminate processes by Process Identifier (PID), either unconditionally or after a successful untrust operation. If the untrust operation succeeds, the affected products cannot restart from the same executable because reopening the image is blocked, which gives the malware additional time to execute stealthily.

The following table describes the key properties and I/O control codes (IOCTLs) exposed by `hlpdrv.sys`:

Description	Details
Symbolic Link	`\\.\KMHLPDRV`
Service Name	`KMHLPSVC`
IOCTL - Untrust a file	`0x222000`
IOCTL - Kill PID	`0x222004`
IOCTL - Kill if untrusted	`0x222008`

IOCTL Breakdown

- **0x222000 (Untrust file):** Accepts a file path and opens with WRITE_DAC and sets a present, empty DACL, denying access to all non-privileged callers

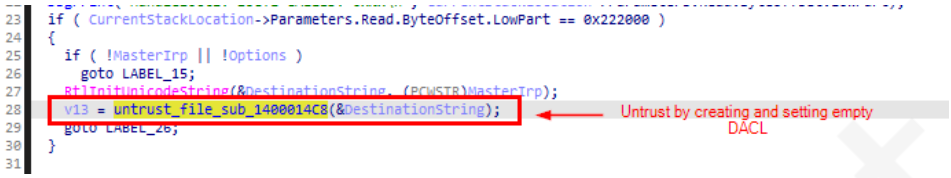

Fig. 7 IOCTL 0x222000

- **0x222004 (Terminate PID)**: Accepts a PID and terminates the process unconditionally

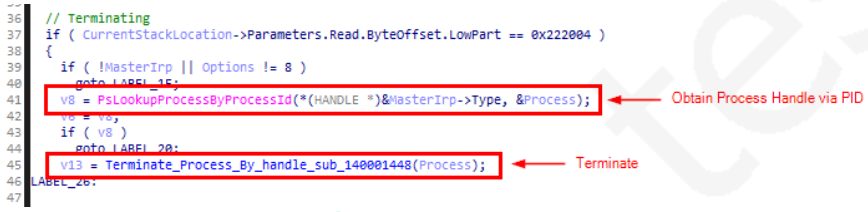

Fig. 8 IOCTL 0x222004

- **0x222008 (Terminate if untrusted)**: Resolves the process image path of the PID, untrusts it as above, and terminate only on success

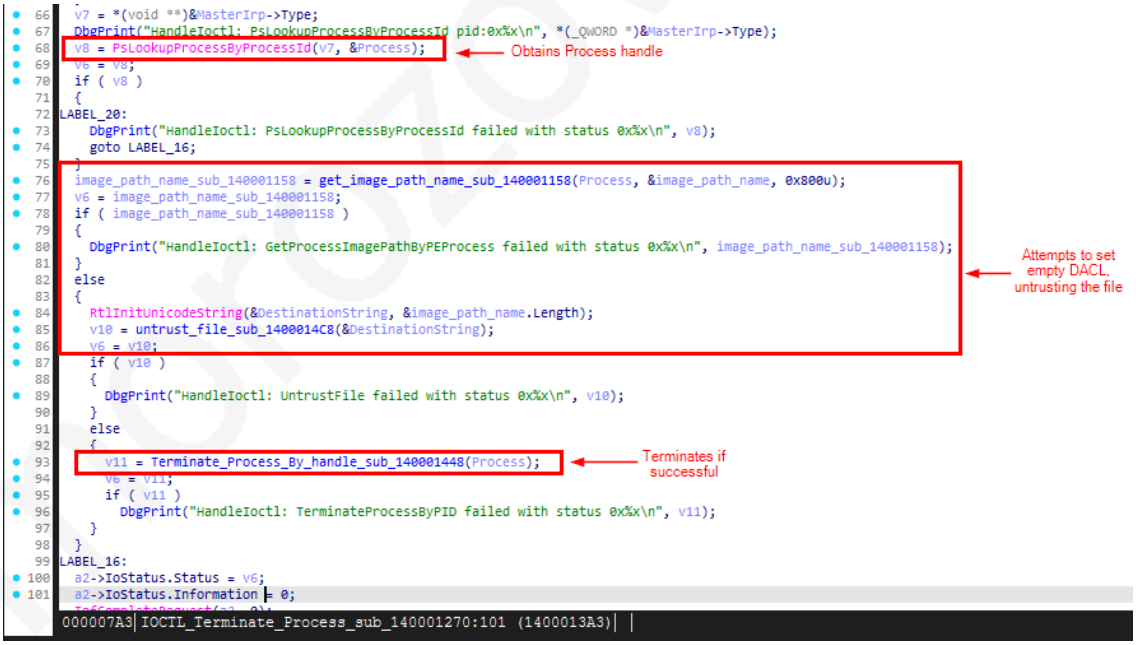

Fig. 9 IOCTL 0x222008

Terminating Blacklisted Processes

The malware enumerates the blacklisted processes and matches them against its list. If a match is found, it invokes the driver with IOCTL `0x222008`, ensuring that termination occurs only if the image is successfully untrusted. The following list contains security vendors whose processes are blacklisted: Microsoft, Kaspersky, Sophos, ESET, SentinelOne, Sangfor Technologies, Panda Security, Trend Micro, Webroot, Bitdefender, Apache, Symantec, Trellix, Akamai Guardian Core.

In line with the driver list, KillaStop also includes additional process names that are absent from KillaTeresa's list. The following list presents security vendors with extra processes included in KillaStop but absent in KillaTeresa (some of the vendors are present only in KillaStop): Microsoft, ESET Inspect, SentinelOne, Sangfor, Huorong, Qianxin, Qihoo 360, McAfee Endpoint Security, FSecure, ZoneAlarm, Immunet, IObit Advanced SystemCare, Fortinet, CrowdStrike, Quick Heal Antivirus.

Overview of the Malware

In summary, the final stage of Killateresa and KillaStop first checks for configured locale in the system. They then proceed to adjust required privilege level before loading vulnerable driver. The Virtual and Physical mapping is created with information obtained from two superfetch calls. With the arbitrary read and write primitives, Killateresa and KillaStop then removes notification routines and callbacks installed by EDRs before terminating blacklisted process list.

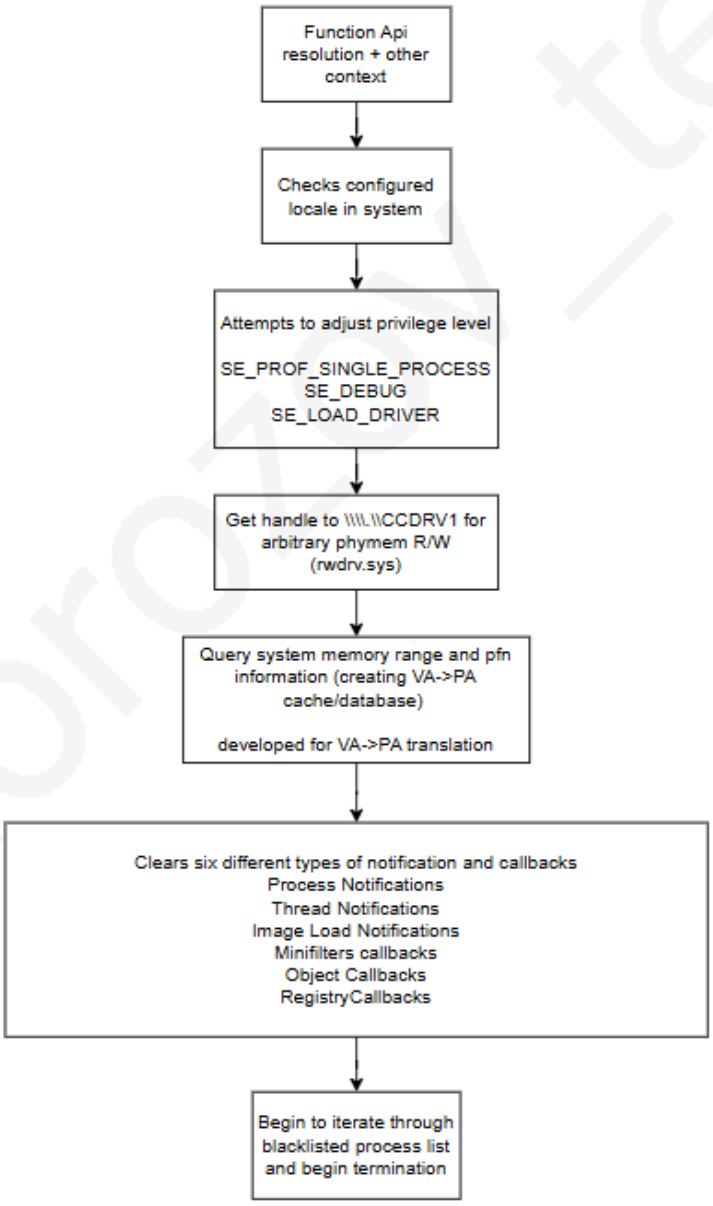

Fig. 10 Overview of KillaTeresa

Victims

Based on our investigation, the KillaTeresa AV-Killer was first discovered in March 2025 in Malaysia, where it was deployed by Qilin ransomware during one of their intrusions. Since then, another variant of KillaTeresa, named KillaStop, has been identified in victims in India, Colombia, Mexico, and Ecuador in separate ransomware cases involving PandaLocker, MedusaLocker, and Makop. The targeting of this AV-Killer tool appears to align solely with ransomware operations, with no evidence of sector-specific victimology for either KillaTeresa or KillaStop.

Attribution

Our research shows that the KillaTeresa AV-Killer has only been observed in a Qilin ransomware case. However, the variant we named KillaStop has been used by multiple ransomware operators, suggesting that it represents a second version of the tool now being shared across different groups. For this reason, we are not confident in attributing these AV-Killer tools (KillaTeresa and KillaStop) to a specific threat actor, given the widespread use of the KillaStop variant among various cybercriminal operators.

Conclusions

Based on our observations, ransomware operators continue to upgrade their toolsets to carry out successful attacks against victims. The use of BYOVD techniques has been around for quite some time, and to this day it is still widely employed by many ransomware operators to terminate antivirus processes.

In this case, multiple ransomware operators deployed an AV-killer tool, packed with BabbleLoader, to disable antivirus processes and evade EDR detection. The malware abused a new vulnerable driver to load a malicious one, effectively turning the tool into driver-based AV killer.

As ransomware evolves, threat actors will continue to develop new methods to disable or bypass security products in order to ensure the execution of their malware on protected machines.

Appendix I – Indicators of Compromise

Note: *The indicators in this section are valid at the time of publication. Any future changes will be directly updated in the corresponding .ioc file.*

File hashes

```
KillaTeresa
A4B84D632FEC031076C83576E3A27A6C    k1ll4_t3r3sa_0x00013C402290187B.exe
BAE7EBAFD73D4E511671275DA72CCEE6    k1ll4_t3r3sa_0x00023F58D5F05B8.exe
C7CCFB9D57E917AB0915930E01ACAE97    b1td3f_0nly.exe

KillaStop
73212A972F1457ED1DA3DED3095B2B1A    0th3r_av5.exe
3FD73115A166157E731E8B538155AB4F    0th3r_av5.exe
B5EF31B97601ED7409779B9F1C32C921    k1ll4_0x00081281324D4A6.exe

Unpacked samples
3402561ECDDA9CE866486826F3E4B095
30DFEA917D92572CCE61685E057ABEEE
DE3841E81A32F390DD270D895E72178B
3E6E519B8DCF2B9A07CE8F16EB9CDF0F

Malicious driver
CF7CAD39407D8CD93135BE42B6BD258F    hlpdrv.sys

Vulnerable driver (Legitimate)
6A784B693DA28EB0EB3E8C6E233C0BBD    rwdrv.sys (ControlCenter.sys)
6BC8E3505D9F51368DDF323ACB6ABC49    rwdrv.sys (ThrottleStop.sys)
```

File paths

```
C:\Users\[username]\AppData\Local\Temp\2\hlpdrv.sys
C:\Users\[username]\AppData\Local\Temp\2\rwdrv.sys
```

Yara Rules

```
import "pe"

rule CW_KillaTeresa_KillaStop_unpacked {
  meta:
    description   = "Detecting Final Payload of KillaTeresa and KillaStop"
    author        = "Kaspersky"
    copyright     = "Kaspersky"
    distribution  = "DISTRIBUTION IS FORBIDDEN. DO NOT UPLOAD TO ANY MULTISCANNER OR SHARE
ON ANY THREAT INTEL PLATFORM"
    version       = "1.0"
    last_modified = "2025-09-02"
    hash          = "BAE7EBAFD73D4E511671275DA72CCEE6"

  strings:
    $s1 = "CRITICAL ERROR!!! TELL CODER IF IT'S APPEARED!" ascii
    $s2 = "hlpdrv.sys" wide
    $s3 = "rwdrv.sys" wide

    $service1 = "KMHLPSVC" wide
    $service2 = "mgdsrv" wide
```

```
    $symbolic1 = "\\\\.\\KMHLPDRV" wide
    $symbolic2 = "\\\\.\\CCDRV1" wide

    /*
    .text:000000000000694D                or      ecx, 20h
    .text:0000000000006950                lea     edx, [rdi-41h]
    .text:0000000000006953                cmp     dx, 19h
    .text:0000000000006957                cmovbe  eax, ecx
    .text:000000000000695A                add     eax, r10d
    .text:000000000000695D                imul    r10d, eax, 0DEADFACEh
    */

    $hash_algo = { 83 C9 20 8D 57 BF 66 83 FA 19 0F 46 C1 41 03 C2 44 69 D0 CE FA AD DE }

  condition:
    pe.is_pe and 7 of them

}

rule CW_BabbleLoader_packed {
  meta:
    description   = "Rule to detect BabbleLoaded packed malware"
    author        = "Kaspersky"
    copyright     = "Kaspersky"
    distribution  = "DISTRIBUTION IS FORBIDDEN. DO NOT UPLOAD TO ANY MULTISCANNER OR SHARE
ON ANY THREAT INTEL PLATFORM"
    version       = "1.0"
    last_modified = "2025-05-13"

  strings:
    /*
        .text:00000000000019F8 49 FF C0          inc     r8
        .text:00000000000019FB 41 0F BE 10       movsx   edx, byte ptr [r8]
        .text:00000000000019FF 41 8A 00          mov     al, [r8]
        .text:0000000000001A02 8B CA             mov     ecx, edx
        .text:0000000000001A04 83 C9 20          or      ecx, 20h
        .text:0000000000001A07 2C 41             sub     al, 41h ; 'A'
        .text:0000000000001A09 3C 19             cmp     al, 19h
        .text:0000000000001A0B 0F 47 CA          cmova   ecx, edx
        .text:0000000000001A0E 41 03 C9          add     ecx, r9d
        .text:0000000000001A11 44 69 ?? ?? ?? ?? imul    r9d, ecx, 5A764DAEh
        .text:0000000000001A18 41 80 38 00       cmp     byte ptr [r8], 0
        .text:0000000000001A1C 75 DA             jnz     short loc_19F8
        */
    $hex1 = { 49 ff c0 41 0f be 10 41 8a 00 8b ca 83 c9 20 2c 41 3c 19 0f 47 ca 41 03 c9 44
69 C9 [4] 41 80 38 00 75 da }

    // Shellcode - 2nd stage
    $sc1 = {48 83 EC 20 65 48 8B 04 25 60 00 00 00}
    $sc2 = {48 8B 98 F8 0F 00 00}

  condition:
    pe.is_pe and
    $hex1 or (all of ($sc*))
}
```

```
rule CW_KillaTeresa_KillaStop_MalDriver {
  meta:
    description   = "Rule to detect the malicious untrust and terminate driver"
    author        = "Kaspersky"
    copyright     = "Kaspersky"
    distribution  = "DISTRIBUTION IS FORBIDDEN. DO NOT UPLOAD TO ANY MULTISCANNER OR SHARE
ON ANY THREAT INTEL PLATFORM"
    version       = "1.0"
    last_modified = "2025-09-12"

  strings:
    $ = "HandleCreate: Device opened." ascii
    $ = "\\Device\\KMHLPDRV" wide
    $ = "Driver initialized successfully." ascii
    $ = "HandleIoctl: TerminateProcessByPID failed with status 0x%x" ascii
    $ = "HandleIoctl: UntrustFile failed with status 0x%x" ascii
    $ = "HandleIoctl: GetProcessImagePathByPEProcess failed with status 0x" ascii
    $ = "File access rights removed successfully." ascii

  condition:
    pe.is_pe and
    all of them
}
```


CDCBBD92CCF5DABB8B3DDB874D644DA86B1F42A3