Chasing the Silver Fox: Cat & Mouse in Kernel Shadows - Check Point Research

By shlomoo@checkpoint.com

Published: 2025-08-28 · Archived: 2026-04-05 13:25:28 UTC

Highlights:

- Check Point Research (CPR) uncovered an ongoing **in-the-wild** campaign attributed to the **Silver Fox APT** which involves the abuse of a **previously unknown vulnerable driver**, `amsdk.sys` (WatchDog Antimalware, version 1.0.600). This driver, built on the Zemana Anti-Malware SDK, was Microsoft-signed, not listed in the Microsoft Vulnerable Driver Blocklist, and not detected by community projects like LOLDrivers.
- The attackers leveraged this unknown vulnerable driver to **terminate protected processes** (**PP/PPL**) associated with modern security solutions, allowing EDR/AV evasion on fully updated Windows 10 and 11 systems without triggering signature-based defenses.
- A **dual-driver strategy** was employed to ensure compatibility across Windows versions: a **known vulnerable Zemana** driver for legacy systems, and the **undetected WatchDog** driver for modern environments. Both were embedded in a single self-contained loader which also included anti-analysis layers and the ValleyRAT downloader.
- Following CPR's disclosure, the vendor released a patched driver (`wamsdk.sys` , version 1.1.100). Although we promptly reported that the patch did **not fully mitigate** the arbitrary process termination issue, the attackers quickly adapted and incorporated a **modified version** of the patched driver into the ongoing campaign. By flipping a **single byte** in the **unauthenticated timestamp field**, they preserved the driver's valid Microsoft signature while generating a new file hash, effectively bypassing hash-based blocklists. This subtle yet efficient evasion technique mirrors patterns seen in earlier campaigns.
- The final payload delivered in all observed samples was **ValleyRAT**, a modular Remote Access Trojan attributed to the **Silver Fox APT** with infrastructure located in China.
- This campaign highlights a growing trend of **weaponizing signed-but-vulnerable drivers** to bypass endpoint protections and evade static detection.

Introduction

While Microsoft Windows has steadily strengthened its security model—through features like Protected Processes (PP/PPL) and enhanced driver verification—threat actors have adapted by shifting their tactics to exploit lower-level weaknesses that bypass these protections without triggering defenses. Among the most effective of these techniques is the abuse of vulnerable kernel-mode drivers, particularly those capable of arbitrary process termination. These drivers, when exploited, can disable or neutralize endpoint protection products, creating a clear path for malware deployment and persistence.

In this publication, we present our findings on a recently detected in-the-wild (ITW) campaign that leverages such a driver-based evasion technique. At the center of this operation is the **Silver Fox APT**, which used an unknown-to-be-vulnerable **WatchDog Antimalware** driver, (`amsdk.sys` , version 1.0.600), to terminate processes associated with security solutions and facilitate the delivery of the **ValleyRAT** backdoor. This driver, although built upon the same SDK (Zemana Anti-Malware SDK) as previously known vulnerable components, was not classified as vulnerable, was signed by Microsoft, and not detected by Microsoft's Vulnerable Driver Blocklist or community-driven sources like the LOLDrivers database.

Our research builds upon the vulnerable driver detection methodology we published in 2024, where we identified thousands of at-risk drivers, including those used in security solutions. One of the previously reported drivers from that research, the WatchDog Antimalware driver, is now confirmed as abused ITW in this campaign. The attackers used this driver to disable core EDR (Endpoint Detection and Response) and antivirus protections before delivering their final payload: **ValleyRAT**, a modular backdoor attributed to Silver Fox APT, with infrastructure located in China.

The campaign's architecture is centered around **all-in-one loader** samples, which combine anti-analysis features, embedded drivers, EDR/AV killer logic, and the ValleyRAT downloader into a single binary. These loaders are tailored to function across both legacy and modern systems (Windows 7 – Windows 10/11), using two different drivers to ensure compatibility. While one of the drivers—a legacy **Zemana-based** driver (`ZAM.exe`)—is already known and blocked, the second driver used in modern environments was previously unknown and therefore remained undetected.

We provide a comprehensive analysis of the observed campaign, detailing how the attackers:

- Exploited vulnerable drivers to terminate protected processes and bypass OS-level protections.
- Delivered multi-stage payloads using sophisticated loader construction.
- Leveraged a Microsoft-signed driver to evade trust-based detection mechanisms.
- Modified a patched version of the driver to avoid hash-based detection, without breaking digital signature validity.
- Ultimately delivered ValleyRAT as the final payload, providing remote access and control capabilities.

In addition, we discuss the implications of signed-but-exploitable drivers and the broader risks posed by attackers leveraging modified versions of previously patched components. We **reported** all relevant findings to **Microsoft's MSRC** and the **Watchdog company**, resulting in partial mitigations but the campaign continues to evolve.

Background & Key Findings

In late May 2025, we observed an ITW attack attributed to the Silver Fox APT group which targeted Windows systems with a custom loader designed to abuse kernel drivers to terminate security-related processes. The campaign marked a significant shift from using only known vulnerable drivers to deploying a previously unclassified, signed vulnerable driver that bypassed traditional detection mechanisms.

The abuse focused on two drivers, both derived from the **Zemana Anti-Malware SDK**. The first, Advanced Malware Protection driver (`ZAM.exe` , version 3.0.0.000), is long known for its weaknesses and is blocked by the

Microsoft Vulnerable Driver Blocklist. Its inclusion in the campaign served compatibility purposes for older systems like Windows 7.

More critically, the attackers deployed **WatchDog Antimalware** driver (`amsdk.sys` , version 1.0.600), which despite sharing the same SDK foundation, was **not publicly known to be vulnerable**, not on any blocklist, and was signed by Microsoft—enabling it to be loaded even on fully updated Windows 10/11 systems.

These drivers facilitated **arbitrary process termination**, including processes running under PP/PPL protection, which enabled the campaign's custom EDR/AV killer logic to disable a wide range of security solutions.

Each malware sample we analyzed was a **self-contained all-in-one loader**, composed of:

- Loader stub with an anti-analysis and persistence setup.
- Two embedded vulnerable drivers.
- Custom logic for terminating security processes based on a hardcoded list.
- A ValleyRAT downloader module configured to fetch and install the final payload.

Throughout the short monitoring window, we observed the campaign evolve to include multiple variants of these loaders and driver combinations. Some samples integrated new drivers still not known to be vulnerable, suggesting an ongoing effort to evade detections and bypass updated defenses.

Following our disclosure, Watchdog company released a patched version of the **WatchDog Antimalware** driver (`wamsdk.sys` , version 1.1.100). While this patch mitigated **local privilege escalation** (**LPE**) vectors, the updated driver still allowed **arbitrary process termination**, including protected processes, thereby failing to fully close the original attack vector. We disclosed this remaining issue to the vendor.

Soon afterwards, we identified a new sample abusing a modified version of the patched driver, once again attributed to the same APT group. The attackers altered **a single byte** in the **unauthenticated timestamp field** of the driver's Microsoft Authenticode signature. Because this field is not covered by the main signature digest, the driver remained validly signed and trusted by Windows, while presenting a new file hash and therefore bypassing hash-based blocklists. This subtle yet powerful evasion technique mirrors those seen in our earlier publication on large-scale legacy driver exploitation.

The final payload across all samples was **ValleyRAT**, also known as "Winos". This RAT offers a full set of capabilities for remote surveillance, command execution, and data exfiltration. Its use, along with infrastructure hosted in China and targeted security process lists aligned with East Asian vendors, confirms the campaign's attribution to **Silver Fox APT**.

This campaign demonstrates how threat actors are moving beyond known weaknesses to weaponize unknown, signed drivers—a blind spot for many defense mechanisms. The exploitation of a Microsoft-signed, previously unclassified vulnerable driver, combined with evasive techniques such as signature manipulation, represents a sophisticated and evolving threat. Despite some mitigations by the vendor, the attacker's ability to quickly adapt—evidenced by the use of altered but validly signed driver variants—highlights the need for proactive, behavior-based detection methods and deeper scrutiny of signed kernel-mode drivers.

Infrastructure & Victimology

All detected command-and-control (C2) servers used in the final stage of the attack, specifically for deploying the ValleyRAT backdoor, are hosted within China, often leveraging public cloud or web services.

The victimology suggests a globally distributed targeting pattern. In most observed cases, the malware is delivered via `.rar` archives containing either a single executable (`.exe`) or a dynamic-link library (`.dll`) that is side-loaded through a legitimate application. The exact infection vector remains unidentified.

As detailed in **Appendix A – List of Processes to be Terminated**, the malware is configured to terminate processes associated with security and antivirus solutions commonly used in China. Combined with the geographic location of the C2 infrastructure, this strongly suggests that the primary targets are located in Asia, particularly within China.

Technical Analysis: All-In-One Loader

All samples we analyzed were deployed as self-contained, all-in-one loaders. The loader is composed of several parts, each with a specific role:

- **The loader stub** – Implements anti-analysis techniques and sets up persistence.
- **Two embedded vulnerable drivers** – Abused for arbitrary process termination.
- **EDR/AV killer logic** – Targets and disables security processes.
- **ValleyRAT downloader module and configuration** – Fetches and executes the final payload.

In most cases (~75 % of detected samples) these loaders are not packed. Occasionally, however, the attackers use unaltered versions of common public packers, such as UPX.

One such example is a 64-bit, **UPX-packed** PE with the internal name `Runtime Broker` . The sample retains its original compilation timestamp, **2025-06-03 06:38:30 UTC**, roughly two weeks after we first observed the campaign in the wild.

Figure 1: The all-in-one self-contained loader – UPX-packed, 64-bit PE.

Anti-analysis Techniques

Upon execution, the sample performs a few common anti-analysis checks, such as **Anti-VM** (detection of virtual environments), **Anti-Sandbox** (detection of execution within a sandbox), **hypervisor detection**, and others. If any of these checks fail, the execution is aborted, and a fake system error message is displayed.


Figure 2: Anti-VM – CPU vendor check.

The virtual environment and hypervisor detection routines include a defined exclusion that allows execution to continue if the computer name is set to any of the following values: `DESKTOP-T3N3M3Q`, `DESKTOP-03AMF90`, or `WIN-VMHH95J6C26`. We believe this exclusion is intended to prevent execution from being aborted on systems used by the attackers during malware development.

We observed that some of the detected samples include an additional anti-analysis check using the public service `http[://]ip-api[.]com/json`. This service is used to retrieve information about the infected machine's public IP address, including the **ISP** (Internet Service Provider) and **ORG** (Organization) fields.

If the ISP or ORG values match any entry from a predefined list (shown in the table below), the process is terminated, and a fake error message, "The program does not support your configuration.", is displayed.

Detected ISP + ORG
Microsoft Corporation
Beijing Qihu Technology Company Limited
Google LLC
Google Cloud (asia-northeast1)

Persistence Settings

To establish persistence, the loader creates a folder named `RunTime` under the system path `C:\Program Files\RunTime`. The all-in-one loader sample and the appropriate version of the vulnerable driver—selected based on the infected system's Windows OS version—are dropped into this folder with the filenames `RuntimeBroker.exe` and `Amsdk_Service.sys`, respectively.


Figure 3: Persistence settings – Dropping files to the created "RunTime" folder.

Subsequently, specific services are created to ensure the dropped files are executed automatically on system startup. The service named `Termaintor` is responsible for maintaining persistence for the previously dropped copy of the all-in-one loader (`RuntimeBroker.exe`).


Figure 4: The creation of the "Termaintor" service.

Figure 4: The creation of the “Termaintor” service.

The second created service, `Amsdk_Service`, is simply a configured registry key required to load the dropped vulnerable driver.

We believe the name `Termaintor` is a deliberate typographical error and also a possible indication that the EDR/AV killer logic was inspired by the publicly available PoC—a tool named “**Terminator**” that abuses the known vulnerable Zemana Anti-Malware driver.

Embedded Payloads

While most globally defined strings in the analyzed sample are simply Base64-encoded, the embedded payloads use a combination of hexadecimal encoding and Base64. These payloads are embedded directly as **encoded byte strings** within the `.rdata` section of the binary.


Figure 5: The encoded embedded payloads.
Figure 5: The encoded embedded payloads.

Two of the encoded payloads are different vulnerable drivers used by the EDR/AV killer logic. Only one of them is deployed on the infected system, depending on the detected Windows version.

The older driver is a 64-bit, validly signed **Advanced Malware Protection** driver, `ZAM.exe`, version 3.0.0.000. This driver is already known to be vulnerable and is detected by both LOLDrivers and the Microsoft Vulnerable Driver Blocklist. It is used **only if the infected system is running an older version of Windows** (e.g., Windows 7).

The newer driver is a 64-bit, validly signed **WatchDog Antimalware** driver, `amsdk.sys`, version 1.0.600. This driver was not previously known to be vulnerable and **bypasses** both LOLDrivers and Microsoft’s blocklist. It is used **only if the infected system is running a modern version of Windows** (e.g., Windows 10 or 11).

The final embedded payload is the encoded **ValleyRAT downloader** module, including its hardcoded configuration.

ValleyRAT Downloader

As previously mentioned, the ValleyRAT downloader stage is embedded within the all-in-one loaders in an encoded format combining Base64 and hexadecimal string. Once decoded, it results in a 64-bit, UPX-packed DLL that is converted into shellcode. This shellcode includes a stub responsible for in-memory reflective loading and is injected into an already running process, typically an instance of `svchost.exe`.

The internal name of the DLL, `上线模块.dll` (translated as “*Online module.dll*”), is preserved in the Export Directory, along with three exported functions: `load`, `run`, and a second `run`.


Figure 6: The Export Directory of the ValleyRAT downloader DLL.
Figure 6: The Export Directory of the ValleyRAT downloader DLL.

The exported functions provide alternative entry points for executing and loading the DLL, such as using a custom-supplied configuration instead of the hardcoded one, while ultimately triggering the same core logic as the `DllMain` function.


Figure 7: The exported functions of the ValleyRAT downloader DLL.

The C2 servers are defined in the embedded configuration. Notably, both the IP addresses and ports are stored in reverse order. For example: `156[.]234[.]58[.]194:52110` and `156[.]234[.]58[.]194:52111`.


Figure 8: The ValleyRAT downloader – Embedded configuration.

The communication between the ValleyRAT downloader and the C2 servers is encrypted using a simple XOR cipher with the key `363636003797e4383a36`. After decrypting the traffic, we found that the downloaded content includes the final payload: the ValleyRAT backdoor (also known as Winos).


Figure 9: The ValleyRAT downloader – Decrypted C2 traffic.

Technical Analysis: EDR/AV Killer

The EDR/AV killer routine is embedded directly within the all-in-one loaders.

Initially, depending on the Windows version of the infected system, one of the two embedded vulnerable drivers is dropped. As both drivers are based on the Zemana Anti-Malware SDK, the exploitation logic remains the same i.e. abusing their ability to terminate arbitrary processes.

The figure below shows the routine responsible for creating the service required to load the vulnerable driver. Specifically, the service `Amsdk_Service` (of type `SERVICE_KERNEL_DRIVER`) is created using the Windows API functions `RegCreateKeyW` and `RegSetValueExW`, followed by a call to the NT API `NtLoadDriver` to initiate the driver loading process.


Figure 10: The vulnerable driver loading process.

Once the driver is loaded, its created device `amsdk` is opened to enable communication with the driver, which is later used in the main EDR/AV killing routine (`KillEDRMain` function).


Figure 11: Opening the “amsdk” device of the abused driver.

The core EDR/AV killer logic, implemented in the `KillEDRMain` function, iterates over a Base64-encoded list of target processes to be terminated. This list, defined as `TERMINATE_PROCESS_LIST`, contains 192 unique process names (see **Appendix A – List of Processes to be Terminated**). When any of the listed processes is found running on the system, it is terminated by issuing a sequence of

IOCTLs, `0x80002010` (`IOCTL_REGISTER_PROCESS`) followed by `0x80002048` (`IOCTL_TERMINATE_PROCESS`), via the Windows API function `DeviceIoControl` that communicates directly with the driver's device.


Figure 12: EDR/AV killer logic – Process termination.

We provide further details about the drivers abused in this campaign, including descriptions of the vulnerabilities, their impact, and example PoC code, in the following section.

Technical Analysis: New Vulnerable Driver

While the all-in-one loaders described in the previous section can abuse two versions of vulnerable drivers (depending on the targeted Windows system version), both are based on the Zemana Anti-Malware SDK. In this section, we shift our focus to the one *not previously known to be vulnerable*, despite its exploitation and impact being almost identical to the known variant.

As mentioned earlier, the abused driver, **WatchDog Antimalware** driver version 1.0.600, is a 64-bit, validly signed Windows kernel device driver. It is still actively used and was originally part of the Watchdog Anti-Malware product.


Figure 13: Vulnerable valid-signed WatchDog Antimalware Driver.

Even though the internal name is `amsdk.sys` , the original PDB path still references `zam64.pdb` , suggesting reuse of the Zemana Anti-Malware SDK.


Figure 14: PDB path of the WatchDog Antimalware Driver.

We confirmed that the WatchDog Antimalware driver is indeed based on the Zemana Anti-Malware SDK through a detailed inspection of its code. Some parts of the code (not just the metadata of the compiled PE) were changed by the WatchDog Antimalware developers, particularly those responsible for the driver's device creation. These changes were likely made in response to the well-known vulnerabilities affecting drivers derived from the Zemana Anti-Malware SDK. In fact, there are publicly available PoCs (for over two years) exploiting "Zemana" drivers as EDR/AV killing tools, such as Terminator.

As this driver is signed by Microsoft (Microsoft Windows Hardware Compatibility Publisher), it can be loaded and used even on the latest fully updated Windows 10/11 systems. In addition, due to the diversity of the WatchDog Antimalware driver, common detection and prevention mechanisms, such as LOLDrivers and the Microsoft Vulnerable Driver Blocklist which cover "Zemana" drivers, are ineffective against the WatchDog Antimalware driver. As a result, there is no obstacle to its use.

Despite modifications to the codebase intended to mitigate known vulnerabilities in "Zemana" drivers, the WatchDog Antimalware driver remains vulnerable with similar impacts, including LPE, unrestricted raw disk read/write access, arbitrary process termination, and more.

Vulnerability Description

There are multiple vulnerabilities affecting the WatchDog Antimalware driver. First, the driver can terminate arbitrary processes without verifying whether the process is running as protected (PP/PPL), which is common for Anti-Malware services. As a result, it is a convenient candidate for the BYOVD (Bring Your Own Vulnerable Driver) technique where it is abused as an EDR/AV killer.

BYOVD attacks are not usually considered vulnerabilities in the traditional sense, as the attacker must first deploy and load the vulnerable driver on the targeted system. These are procedures that require Administrator privileges (elevation from Administrator to System does not cross a security boundary). However, if the driver is already present on the targeted system (e.g., as part of the Watchdog Anti-Malware product), even a non-privileged user can abuse it to disable security solutions.

A more critical driver vulnerability is its ability to cross the security boundary from a non-privileged user to System, resulting directly in LPE (Local Privilege Escalation). The root cause lies in the routine responsible for the driver's device creation.

`IoCreateDeviceSecure`, a more secure kernel function than `IoCreateDevice` because it lets you specify a DACL, is used. A strong DACL is set on the created device via the SDDL string `D:P(A;;GA;;;SY)(A;;GA;;;BA)`, granting access only to System and Administrators, but the `DeviceCharacteristics` do not explicitly include the `FILE_DEVICE_SECURE_OPEN` flag. Without this flag as part of the device characteristics, the strong DACL does not apply to the entire device namespace, allowing even non-privileged users to communicate with the device.

Explanation – Device Namespace & FILE_DEVICE_SECURE_OPEN

Every device has its own namespace, where names in the namespace are paths that begin with the device´s name. For a device named `\Device\DeviceName`, its namespace consists of any name with the form `\Device\DeviceName\anyfile`. The lack of the `FILE_DEVICE_SECURE_OPEN` flag can be abused to obtain a full access handle to the device itself, even by a non-privileged user, because the strong DACL is not propagated to the namespace, e.g., opening a handle to `\Device\DeviceName\anyfile` will return a handle for the device itself `\Device\DeviceName`.

Below, we can see the driver´s initialization routine, where the device object is created with a strong DACL, but without the `FILE_DEVICE_SECURE_OPEN` characteristics flag.


Figure 15: Driver initialization - Device creation. The converted DACL, part of the SDDL string:
Figure 15: Driver initialization – Device creation.

The converted DACL, part of the SDDL string:

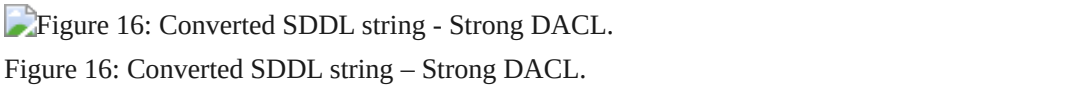
Figure 16: Converted SDDL string - Strong DACL.
Figure 16: Converted SDDL string – Strong DACL.

By combining unrestricted access to the driver's device with its ability to perform privileged operations, all of the previously described vulnerabilities can be exploited. These capabilities of the WatchDog Antimalware driver are

triggered by issuing specific IOCTLs during communication with the device.

The most critical examples are summarized in the table below:

IOCTL Name	**IOCTL Code**	**Meaning + Impact**
IOCTL_REGISTER_PROCESS	0x80002010	Process registration – Mitigation bypass
IOCTL_OPEN_PROCESS	0x8000204C	Obtain full access process´ handle – LPE
IOCTL_TERMINATE_PROCESS	0x80002048	Arbitrary process termination – Disabling EDR/AV
IOCTL_SCSI_READ	0x80002014	Unrestricted raw disk read
IOCTL_SCSI_WRITE	0x80002018	Unrestricted raw disk write

Below is a simplified view of the `DispatchDeviceControl` callback function, which is responsible for handling IOCTL requests and executing the corresponding routines.


Figure 17: DispatchDeviceControl callback function – Handling IOCTL requests.

The arbitrary process termination capability, abused by the threat actor in the campaign we describe here, is triggered via the IOCTL `0x80002048` (`IOCTL_TERMINATE_PROCESS`), which directly invokes the `TerminateProcessById` function. Notably, before this action, the attacker-controlled process must first register itself (be added to the allowlist to bypass mitigation) by issuing IOCTL `0x80002010` (`IOCTL_REGISTER_PROCESS`).

The `TerminateProcessById` function blocks termination only if the target process is marked as critical (to prevent a system crash). However, it does not handle protected processes (PP/PPL), which are typically used by anti-malware services. As a result, security solution processes can be freely terminated.


Figure 18: The logic of TerminateProcessById function.

The PoC demonstrating the driver's **arbitrary process termination** capability is relatively simple and the exploit can be implemented as follows:

```
#define IOCTL_REGISTER_PROCESS 0x80002010

#define IOCTL_TERMINATE_PROCESS 0x80002048

// Loading the amsdk.sys driver

DWORD pidTerminate = 1337; // Pid of process to be killed, PP/PPL processes possible
```

DWORD pidRegister = GetCurrentProcessId();

HANDLE hDevice = CreateFileA("\\\\.\\amsdk\\anyfile", GENERIC_READ | GENERIC_WRITE, NULL, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

DeviceIoControl(hDevice, IOCTL_REGISTER_PROCESS, &pidRegister, sizeof(pidRegister), NULL, 0, NULL, NULL);

DeviceIoControl(hDevice, IOCTL_TERMINATE_PROCESS, &pidTerminate, sizeof(pidTerminate), NULL, 0, NULL, NULL);

#define IOCTL_REGISTER_PROCESS 0x80002010 #define IOCTL_TERMINATE_PROCESS 0x80002048 // Loading the amsdk.sys driver DWORD pidTerminate = 1337; // Pid of process to be killed, PP/PPL processes possible DWORD pidRegister = GetCurrentProcessId(); HANDLE hDevice = CreateFileA("\\\\.\\amsdk\\anyfile", GENERIC_READ | GENERIC_WRITE, NULL, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL); DeviceIoControl(hDevice, IOCTL_REGISTER_PROCESS, &pidRegister, sizeof(pidRegister), NULL, 0, NULL, NULL); DeviceIoControl(hDevice, IOCTL_TERMINATE_PROCESS, &pidTerminate, sizeof(pidTerminate), NULL, 0, NULL, NULL);

```
#define IOCTL_REGISTER_PROCESS 0x80002010
#define IOCTL_TERMINATE_PROCESS 0x80002048
// Loading the amsdk.sys driver
DWORD pidTerminate = 1337; // Pid of process to be killed, PP/PPL processes possible
DWORD pidRegister = GetCurrentProcessId();
HANDLE hDevice = CreateFileA("\\\\.\\amsdk\\anyfile", GENERIC_READ | GENERIC_WRITE, NULL, NULL, OPEN
DeviceIoControl(hDevice, IOCTL_REGISTER_PROCESS, &pidRegister, sizeof(pidRegister), NULL, 0, NULL, NU
DeviceIoControl(hDevice, IOCTL_TERMINATE_PROCESS, &pidTerminate, sizeof(pidTerminate), NULL, 0, NULL
```

The core of the vulnerability in the WatchDog Antimalware driver is not complex and can easily be addressed. Specifying the `FILE_DEVICE_SECURE_OPEN` device characteristic ensures the propagation of the strong DACL to the whole device namespace and prevents access by non-privileged users. An additional check to verify if the targeted process to be terminated is running as protected (PP/PPL) will mitigate the possibility of disabling Anti-Malware solutions.

Final Stages: The ValleyRAT Backdoor

All the samples we analyzed (see **Appendix B – IOCs**) deployed the well-known ValleyRAT backdoor (also known as Winos) as the final stage. This malware strain is strongly attributed to, and associated with, the well-known APT group Silver Fox.

Similar to the ValleyRAT downloader, the deployed ValleyRAT backdoor is delivered as a 64-bit DLL converted to shellcode (the DLL is preceded by stub shellcode responsible for in-memory reflective loading).

The internal name `登录模块.dll` (translated as "*LoginModule.dll*") of this DLL is preserved in the Export Directory, along with a single exported function, `run`. This exported function is almost identical to `DllMain`,

triggers the same main logic, and likely serves as an alternate execution mechanism.


Figure 19: The exported function “run” - ValleyRAT backdoor DLL.

Figure 19: The exported function “run” – ValleyRAT backdoor DLL.

One of the first executed functions to avoid detection is a callback invoked via the Windows API `EnumWindows`. The callback function, `EnumFunc`, is responsible for detecting processes with window titles associated with analysis tools, primarily those used for network analysis, commonly found in sandboxes or malware labs. If any of the defined window titles is detected, execution is delayed by a 20-second sleep, and the enumeration continues until none of the tools are detected.


Figure 20: The ValleyRAT “EnumFunc” callback function used to avoid detection.

Figure 20: The ValleyRAT “EnumFunc” callback function used to avoid detection.

The identical routine was already described in ValleyRAT Insights: Tactics, Techniques, and Detection Methods. As the ValleyRAT backdoor and its stages were thoroughly analyzed in several publicly available reports, a detailed analysis of the version deployed in this campaign is considered out of the scope for this publication.

As previously described, the attackers abuse two vulnerable drivers to terminate processes associated with security solutions. While the older one—Advanced Malware Protection driver, `ZAM.exe`, version 3.0.0.000—was classified as known-vulnerable by the Microsoft Vulnerable Driver Blocklist and by other common detection mechanisms such as those implemented by the LOLDrivers project, the newer one—WatchDog Antimalware driver, `amsdk.sys`, version 1.0.600—bypasses all of them.

As we saw, the all-in-one loaders used in this campaign are tailored to support not only the latest versions of Windows OS (Windows 10/11) but also older ones (e.g., Windows 7), meaning even legacy systems remain at risk.

The **Microsoft Vulnerable Driver Blocklist** uses advanced detection mechanisms, beyond simple hash-based checks, to protect against known vulnerable drivers. As it is built into the Windows OS, we **reported** the issue to **MSRC**.

As the primary risk in this detected in-the-wild campaign stems from the previously unknown vulnerable WatchDog Antimalware driver, we **reported** the campaign and the driver abuse to its **vendor**, the **Watchdog company**. As a result, the vendor released a patch: WatchDog Antimalware driver, `wamsdk.sys`, version 1.1.100. All Watchdog products are now deployed with this updated version of the driver.

We confirmed that the new driver mitigates the LPE risk by enforcing a strong DACL (allowing access only for SYSTEM and Administrator accounts) and by setting the `FILE_DEVICE_SECURE_OPEN` device characteristic, which ensures the propagation of the DACL across the entire device namespace.

Unfortunately, it does **not** mitigate the arbitrary process termination issue. It still lacks a check for whether the targeted process is running as protected (PP/PPL). As a result, this driver can still be abused similarly to how it was used in the described campaign to disable anti-malware solutions. We **reported** this to the vendor.

As anticipated, we already detected a sample in this campaign that abuses an altered variant of the released patched driver (WatchDog Antimalware Driver, `wamsdk.sys`, version 1.1.100).

This sample remains associated with the same campaign attributed to the **Silver Fox APT**, once again deploying the ValleyRAT backdoor as the final stage. We **reported** this new finding to the **Watchdog company**.

The patched driver (WatchDog Antimalware Driver, `wamsdk.sys`, version 1.1.100) is abused in a way where the attackers create a modified variant by altering just a single byte within the PE binary's `WIN_CERTIFICATE` structure—specifically in the **unauthenticated attributes** of the embedded Microsoft Authenticode signature. This byte is part of the **RFC 3161 timestamp (counter-signature)** applied by Microsoft's time-stamping authority. Because unauthenticated attributes are not covered by the primary signature's digest, this change does **not affect** the validity of the embedded Microsoft signature, and does not break the chain of trust. As a result, Windows continues to treat the driver as **validly signed and trusted**, even on the latest versions. However, the file's overall hash (e.g., SHA-256) is now different, which allows attackers to create a modified, uniquely hashed but still validly signed version of the original driver.

Figure 21: One-byte modification of the patched WatchDog Antimalware Driver (version 1.1.100).

Figure 21: One-byte modification of the patched WatchDog Antimalware Driver (version 1.1.100).

We recommend manually applying the latest version of the Microsoft Vulnerable Driver Blocklist, as it is usually auto-updated only once or twice a year. As we cannot ensure that all the vulnerable drivers abused in this campaign will be added to the Microsoft Vulnerable Driver Blocklist, we provided YARA rules (see **Appendix C – YARA**) to detect them and recommend monitoring and preventing their abuse.

Conclusion

We revealed a sophisticated campaign, attributed to the Silver Fox APT group, that exploits vulnerable signed drivers to bypass security protections and deploy the ValleyRAT backdoor. By abusing two vulnerable drivers, one previously known and one newly identified, attackers achieved arbitrary process termination, allowing them to disable anti-malware solutions and maintain stealth across multiple Windows versions, including the latest Windows 10 and 11.

The newly identified abuse of the WatchDog Antimalware driver demonstrates that even signed and seemingly trusted drivers can contain critical vulnerabilities. The attackers' technique of modifying unauthenticated attributes within the driver's digital signature to evade detection while preserving trustworthiness exposes the limitations of relying solely on hash-based or signature-based detection methods.

Our findings highlight the need for layered defense strategies, encompassing not only timely application of Microsoft's Vulnerable Driver Blocklist and custom detection rules like YARA signatures, but also robust behavior-based detection capable of heuristically identifying and blocking such threats. These combined measures are vital to detect and prevent the abuse of vulnerable drivers before attackers can escalate privileges or disable security software.

Our final point is that our research reinforces the need for ongoing efforts of security vendors and users to stay vigilant against the emerging abuse of legitimate drivers. Proactive identification, reporting, and patching of these vulnerabilities are critical to strengthening Windows systems against evolving threats leveraging Bring Your Own Vulnerable Driver (BYOVD) techniques.

Protections

Check Point Threat Emulation and Harmony Endpoint provide comprehensive coverage of attack tactics, filetypes, and operating systems and protect against the attacks and threats described in this report.

References

CPR – Breaking Boundaries: Investigating Vulnerable Drivers and Mitigating Risks: https://research.checkpoint.com/2024/breaking-boundaries-investigating-vulnerable-drivers-and-mitigating-risks/

CPR – Silent Killers: Unmasking a Large-Scale Legacy Driver Exploitation Campaign: https://research.checkpoint.com/2025/large-scale-exploitation-of-legacy-driver/

LOLDrivers: https://github.com/magicsword-io/LOLDrivers

Microsoft Vulnerable Driver Blocklist: https://learn.microsoft.com/en-us/windows/security/application-security/application-control/app-control-for-business/design/microsoft-recommended-driver-block-rules

Terminator PoC: https://github.com/ZeroMemoryEx/Terminator

Reverse Engineering Zemana AntiMalware/AntiLogger Driver: https://voidsec.com/reverse-engineering-terminator-aka-zemana-antimalware-antilogger-driver/

Watchdog Anti-Malware product: https://watchdog.com/solutions/anti-malware/

Zemana Anti-Malware SDK: https://zemana.com/us/sdk.html

Device Characteristics: https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/specifying-device-characteristics

UPX – the Ultimate Packer for eXecutables: https://github.com/upx/upx

Splunk – ValleyRAT Insights: https://www.splunk.com/en_us/blog/security/valleyrat-insights-tactics-techniques-and-detection-methods.html

Zscaler – Technical Analysis of ValleyRAT: https://www.zscaler.com/blogs/security-research/technical-analysis-latest-variant-valleyrat

Malpedia ValleyRAT: https://malpedia.caad.fkie.fraunhofer.de/details/win.valley_rat

Appendix A – List of Processes to be Terminated

Notice that most of the targeted processes are associated with security solutions typically deployed in China or the Asia region.

2345PCSafeBootAssistant.exe

2345SafeCenterCrashReport.exe

2345SafeCenterInstaller.exe

BrowserPrivacyAndSecurity.exe

SecurityHealthService.exe

remove_incompatible_applications.exe

securityhealthsystray.exe

uninstallation_assistant_host.exe

2345AdRtProtect.exe 2345Associate.exe 2345AuthorityProtect.exe 2345ExtShell.exe 2345ExtShell64.exe 2345FileShre.exe 2345HipsSet.exe 2345InstDll.exe 2345LSPFix.exe 2345LeakFixer.exe 2345MPCSafe.exe 2345ManuUpdate.exe 2345NetFlow.exe 2345NetRepair.exe 2345NightMode.exe 2345PCSafeBootAssistant.exe 2345ProtectManager.exe 2345RTProtect.exe 2345RtProtectCenter.exe 2345SFGuard.exe 2345SFGuard64.exe 2345SFWebShell.exe 2345SafeCenterCrashReport.exe 2345SafeCenterInstaller.exe 2345SafeCenterSvc.exe 2345SafeCenterUpdate.exe 2345SafeLock.exe 2345SafeSvc.exe 2345SafeTray.exe 2345SafeUpdate.exe 2345ScUpgrade.exe 2345Setting.exe 2345ShellPro.exe 2345ShortcutArrow.exe 2345SoftMgr.exe 2345SoftmgrDaemon.exe 2345SoftmgrSvc.exe 2345SysDoctor.exe 2345TrashRcmd.exe 2345Uninst.exe 2345UsbGuard.exe 2345VirusScan.exe 360AI.exe 360FileGuard.exe 360QMachine.exe 360Restore.exe 360Safe.exe 360SkinMgr.exe 360huabao.exe 360leakfixer.exe 360netcfg.exe 360netcfg64.exe 360realpro.exe 360rp.exe 360rps.exe 360sd.exe 360sdSetup.exe 360sdToasts.exe 360sdrun.exe 360sdsf.exe 360sdupd.exe 360speedld.exe 360tray.exe BGADefMgr.exe BrowserPrivacyAndSecurity.exe CertImporter-1684.exe Client.exe ConfigSecurityPolicy.exe DSMain.exe DlpUserAgent.exe DumpUper.exe Fetion.exe HipsDaemon.exe HipsMain.exe HipsTray.exe MSPCManager.exe MSPCManagerCore.exe MSPCManagerService.exe MipDlp.exe MpCmdRun.exe MpCopyAccelerator.exe MpDlpCmd.exe MpDlpService.exe MsMpEng.exe MultiTip.exe NewIDView.exe NisSrv.exe PCMAutoRun.exe QMAIService.exe QMDL.exe QMFloatWidget.exe QQPCExternal.exe QQPCMgrUpdate.exe QQPCPatch.exe QQPCRTP.exe QQPCSoftCmd.exe QQPCSoftMgr.exe QQPCTray.exe QQRepair.exe RMenuMgr.exe SecurityHealthHost.exe SecurityHealthService.exe SysCleanProService.exe SysInspector.exe VolSnapshotX64.exe ZhuDongFangYu.exe activeconsole anti-malware antimalware avpia.exe avpvk.exe callmsi.exe eCapture.exe eComServer.exe ecls.exe ecmd.exe ecmds.exe eeclnt.exe egui.exe eguiProxy.exe feedback.exe feedbackwin.exe kailab.exe kassistant.exe kassistsetting.exe kauthorityview.exe kavlog2.exe kcddltool.exe kcleaner.exe kcrm.exe kctrlpanel.exe kdf.exe kdinfomgr.exe kdownloader.exe kdrvmgr.exe kdumprep.exe kdumprepn.exe keyemain.exe kfixstar.exe kfloatmain.exe khealthctrlspread.exe kinst.exe kintercept.exe kislive.exe kismain.exe kldw.exe kmenureg.exe knewvip.exe knotifycenter.exe krecycle.exe kscan.exe kschext.exe kscrcap.exe ksetupwiz.exe kslaunch.exe kslaunchex.exe ksoftmgr.exe ksoftmgrproxy.exe ksoftpurifier.exe kteenmode.exe ktrashautoclean.exe kupdata.exe kwebx.exe kwsprotect64.exe kwtpanel.exe kxecenter.exe kxemain.exe kxescore.exe kxetray.exe kxewsc.exe mpextms.exe

packageregistrator.exe plugins-setup.exe plugins_nms.exe qmbsrv.exe rcmdhelper.exe rcmdhelper64.exe remove_incompatible_applications.exe restore_tool.exe safesvr.exe securityhealthsystray.exe smartscreen.exe sysissuehat.exe troubleshoot.exe uni0nst.exe uninstallation_assistant_host.exe upgrade.exe vssbridge64.exe webx.exe webx_helper.exe wmiav.exe wsctrlsvc.exe

```
2345AdRtProtect.exe
2345Associate.exe
2345AuthorityProtect.exe
2345ExtShell.exe
2345ExtShell64.exe
2345FileShre.exe
2345HipsSet.exe
2345InstDll.exe
2345LSPFix.exe
2345LeakFixer.exe
2345MPCSafe.exe
2345ManuUpdate.exe
2345NetFlow.exe
2345NetRepair.exe
2345NightMode.exe
2345PCSafeBootAssistant.exe
2345ProtectManager.exe
2345RTProtect.exe
2345RtProtectCenter.exe
2345SFGuard.exe
2345SFGuard64.exe
2345SFWebShell.exe
2345SafeCenterCrashReport.exe
2345SafeCenterInstaller.exe
2345SafeCenterSvc.exe
2345SafeCenterUpdate.exe
2345SafeLock.exe
2345SafeSvc.exe
2345SafeTray.exe
2345SafeUpdate.exe
2345ScUpgrade.exe
2345Setting.exe
2345ShellPro.exe
2345ShortcutArrow.exe
2345SoftMgr.exe
2345SoftmgrDaemon.exe
2345SoftmgrSvc.exe
2345SysDoctor.exe
2345TrashRcmd.exe
2345Uninst.exe
2345UsbGuard.exe
```

```
2345VirusScan.exe
360AI.exe
360FileGuard.exe
360QMachine.exe
360Restore.exe
360Safe.exe
360SkinMgr.exe
360huabao.exe
360leakfixer.exe
360netcfg.exe
360netcfg64.exe
360realpro.exe
360rp.exe
360rps.exe
360sd.exe
360sdSetup.exe
360sdToasts.exe
360sdrun.exe
360sdsf.exe
360sdupd.exe
360speedld.exe
360tray.exe
BGADefMgr.exe
BrowserPrivacyAndSecurity.exe
CertImporter-1684.exe
Client.exe
ConfigSecurityPolicy.exe
DSMain.exe
DlpUserAgent.exe
DumpUper.exe
Fetion.exe
HipsDaemon.exe
HipsMain.exe
HipsTray.exe
MSPCManager.exe
MSPCManagerCore.exe
MSPCManagerService.exe
MipDlp.exe
MpCmdRun.exe
MpCopyAccelerator.exe
MpDlpCmd.exe
MpDlpService.exe
MsMpEng.exe
MultiTip.exe
NewIDView.exe
NisSrv.exe
PCMAutoRun.exe
```

```
QMAIService.exe
QMDL.exe
QMFloatWidget.exe
QQPCExternal.exe
QQPCMgrUpdate.exe
QQPCPatch.exe
QQPCRTP.exe
QQPCSoftCmd.exe
QQPCSoftMgr.exe
QQPCTray.exe
QQRepair.exe
RMenuMgr.exe
SecurityHealthHost.exe
SecurityHealthService.exe
SysCleanProService.exe
SysInspector.exe
VolSnapshotX64.exe
ZhuDongFangYu.exe
activeconsole
anti-malware
antimalware
avpia.exe
avpvk.exe
callmsi.exe
eCapture.exe
eComServer.exe
ecls.exe
ecmd.exe
ecmds.exe
eeclnt.exe
egui.exe
eguiProxy.exe
feedback.exe
feedbackwin.exe
kailab.exe
kassistant.exe
kassistsetting.exe
kauthorityview.exe
kavlog2.exe
kcddltool.exe
kcleaner.exe
kcrm.exe
kctrlpanel.exe
kdf.exe
kdinfomgr.exe
kdownloader.exe
kdrvmgr.exe
```

```
kdumprep.exe
kdumprepn.exe
keyemain.exe
kfixstar.exe
kfloatmain.exe
khealthctrlspread.exe
kinst.exe
kintercept.exe
kislive.exe
kismain.exe
kldw.exe
kmenureg.exe
knewvip.exe
knotifycenter.exe
krecycle.exe
kscan.exe
kschext.exe
kscrcap.exe
ksetupwiz.exe
kslaunch.exe
kslaunchex.exe
ksoftmgr.exe
ksoftmgrproxy.exe
ksoftpurifier.exe
kteenmode.exe
ktrashautoclean.exe
kupdata.exe
kwebx.exe
kwsprotect64.exe
kwtpanel.exe
kxecenter.exe
kxemain.exe
kxescore.exe
kxetray.exe
kxewsc.exe
mpextms.exe
packageregistrator.exe
plugins-setup.exe
plugins_nms.exe
qmbsrv.exe
rcmdhelper.exe
rcmdhelper64.exe
remove_incompatible_applications.exe
restore_tool.exe
safesvr.exe
securityhealthsystray.exe
smartscreen.exe
```

```
sysissuehat.exe
troubleshoot.exe
uni0nst.exe
uninstallation_assistant_host.exe
upgrade.exe
vssbridge64.exe
webx.exe
webx_helper.exe
wmiav.exe
wsctrlsvc.exe
```

Appendix B – IOCs

All-in-one self-contained loaders:

SHA-256	Final Stage
d24fffc34e45c168ea4498f51a7d9f7f074d469c8d4317e8e2205c33a99b5364	ValleyRAT/Winos
fc97ad46767a45f4e59923f96d15ec5b680a33f580af7cc4e320fb9963933f26	ValleyRAT/Winos
09587073acbfec909eea69aa49774b3fdaa681db9cec7cb20a4143050897c393	ValleyRAT/Winos
2f0e34860194ccd232f7c8c27fefe44c96b63468e8581f93c38767725255f945	ValleyRAT/Winos
57f37bc0519557cf3f4c375fd04900a4d5afb82e3b723c6b9d0f96dc08eea84d	ValleyRAT/Winos
b26aecc21da159c0073ecde31cc292d87c8674af8c312776d2cc9827e5c1ad6a	ValleyRAT/Winos
baccea051dc6bb1731fa2bc97c5e0cc2cd37463e83bf73a400451ad7ba00a543	ValleyRAT/Winos
9e72b958b4ad9fdf64b6f12a89eb2bae80097a65dc8899732bce9dafda622148	ValleyRAT/Winos
35ccb9c521c301e416a3ea0c0292ae93914fe165eb45f749c16de03a99f5fa8e	ValleyRAT/Winos
5f23694d44850c1963b38d8eab638505d14c5605e9623fb98e9455795fa33321	ValleyRAT/Winos

C2 Servers (ValleyRAT/Winos):

IP Address	Port	AS	Country
47.239.197.97	52116, 52117	Alibaba Cloud	HK (China)
8.217.38.238	8888	Alibaba Cloud	HK (China)

IP Address	Port	AS	Country
156.234.58.194	52110, 52111	Yancy Limited	HK (China)
156.241.144.66	52139, 52160	AROSS-AS	HK (China)
1.13.249.217	9527, 9528	Shenzhen Tencent Computer Systems Company Limited	China

Vulnerable drivers abused in the campaign:

All of them are based on the Zemana Anti-Malware SDK.

SHA-256	Note
12b3d8bc5cc1ea6e2acd741d8a80f56cf2a0a7ebfa0998e3f0743fcf83fabb9e	Used for WIN 10/11
0be8483c2ea42f1ce4c90e84ac474a4e7017bc6d682e06f96dc1e31922a07b10	Used for WIN 10/11
9c394dcab9f711e2bf585edf0d22d2210843885917d409ee56f22a4c24ad225e	Used for older WIN OS

Appendix C – YARA

Detects 64-bit, valid-signed WatchDog Antimalware driver, `amsdk.sys`, version 1.0.600 (bypassing LOLDrivers and Microsoft Vulnerable Driver Blocklist):

rule watchdog_antimalware_driver_64bit_ver10600

description = "Detects 64-bit, valid-signed WatchDog Antimalware driver, version 1.0.600"

author = "Jiri Vinopal @ Check Point Research"

hash = "12b3d8bc5cc1ea6e2acd741d8a80f56cf2a0a7ebfa0998e3f0743fcf83fabb9e"

uint16(0) == 0x5a4d and uint16(uint32(0x3c)) == 0x4550 and

// Detect 64-bit Windows drivers

uint16(uint32(0x3C) + 0x5c) == 0x0001 and uint16(uint32(0x3C) + 0x18) == 0x020b and

// Detect OriginalFilename "amsdk.sys" and FileVersion "1.0.600"

pe.version_info["OriginalFilename"] == "amsdk.sys" and pe.version_info["FileVersion"] == "1.0.600" and

// Detect only signed drivers, not a real verification

pe.number_of_signatures > 0 and for all i in (0..pe.number_of_signatures -1):

(pe.signatures[i].verified)

import "pe" rule watchdog_antimalware_driver_64bit_ver10600 { meta: description = "Detects 64-bit, valid-signed WatchDog Antimalware driver, version 1.0.600" author = "Jiri Vinopal @ Check Point Research" hash = "12b3d8bc5cc1ea6e2acd741d8a80f56cf2a0a7ebfa0998e3f0743fcf83fabb9e" condition: // Detect PE uint16(0) == 0x5a4d and uint16(uint32(0x3c)) == 0x4550 and // Detect 64-bit Windows drivers uint16(uint32(0x3C) + 0x5c) == 0x0001 and uint16(uint32(0x3C) + 0x18) == 0x020b and // Detect OriginalFilename "amsdk.sys" and FileVersion "1.0.600" pe.version_info["OriginalFilename"] == "amsdk.sys" and pe.version_info["FileVersion"] == "1.0.600" and // Detect only signed drivers, not a real verification pe.number_of_signatures > 0 and for all i in (0..pe.number_of_signatures -1): (pe.signatures[i].verified) }

```
import "pe"

rule watchdog_antimalware_driver_64bit_ver10600
{
    meta:
        description = "Detects 64-bit, valid-signed WatchDog Antimalware driver, version 1.0.600"
        author = "Jiri Vinopal @ Check Point Research"
        hash = "12b3d8bc5cc1ea6e2acd741d8a80f56cf2a0a7ebfa0998e3f0743fcf83fabb9e"
    condition:
        // Detect PE
        uint16(0) == 0x5a4d and uint16(uint32(0x3c)) == 0x4550 and
        // Detect 64-bit Windows drivers
        uint16(uint32(0x3C) + 0x5c) == 0x0001 and uint16(uint32(0x3C) + 0x18) == 0x020b and
        // Detect OriginalFilename "amsdk.sys" and FileVersion "1.0.600"
        pe.version_info["OriginalFilename"] == "amsdk.sys" and pe.version_info["FileVersion"] == "1.
        // Detect only signed drivers, not a real verification
        pe.number_of_signatures > 0 and for all i in (0..pe.number_of_signatures -1):
            (pe.signatures[i].verified)
}
```

Detects 64-bit, valid-signed WatchDog Antimalware Driver, `wamsdk.sys`, version 1.1.100 (bypassing LOLDrivers and Microsoft Vulnerable Driver Blocklist):

rule watchdog_antimalware_driver_64bit_ver11100

description = "Detects 64-bit, valid-signed WatchDog Antimalware driver, version 1.1.100"

author = "Jiri Vinopal @ Check Point Research"

hash = "5af1dae21425dda8311a2044209c308525135e1733eeff5dd20649946c6e054c"

hash = "0be8483c2ea42f1ce4c90e84ac474a4e7017bc6d682e06f96dc1e31922a07b10"

uint16(0) == 0x5a4d and uint16(uint32(0x3c)) == 0x4550 and

// Detect 64-bit Windows drivers

uint16(uint32(0x3C) + 0x5c) == 0x0001 and uint16(uint32(0x3C) + 0x18) == 0x020b and

// Detect OriginalFilename "wamsdk.sys" and FileVersion "1.1.100"

pe.version_info["OriginalFilename"] == "wamsdk.sys" and pe.version_info["FileVersion"] == "1.1.100" and

// Detect only signed drivers, not a real verification

pe.number_of_signatures > 0 and for all i in (0..pe.number_of_signatures -1):

(pe.signatures[i].verified)

import "pe" rule watchdog_antimalware_driver_64bit_ver11100 { meta: description = "Detects 64-bit, valid-signed WatchDog Antimalware driver, version 1.1.100" author = "Jiri Vinopal @ Check Point Research" hash = "5af1dae21425dda8311a2044209c308525135e1733eeff5dd20649946c6e054c" hash = "0be8483c2ea42f1ce4c90e84ac474a4e7017bc6d682e06f96dc1e31922a07b10" condition: // Detect PE uint16(0) == 0x5a4d and uint16(uint32(0x3c)) == 0x4550 and // Detect 64-bit Windows drivers uint16(uint32(0x3C) + 0x5c) == 0x0001 and uint16(uint32(0x3C) + 0x18) == 0x020b and // Detect OriginalFilename "wamsdk.sys" and FileVersion "1.1.100" pe.version_info["OriginalFilename"] == "wamsdk.sys" and pe.version_info["FileVersion"] == "1.1.100" and // Detect only signed drivers, not a real verification pe.number_of_signatures > 0 and for all i in (0..pe.number_of_signatures -1): (pe.signatures[i].verified) }

```
import "pe"

rule watchdog_antimalware_driver_64bit_ver11100
{
    meta:
        description = "Detects 64-bit, valid-signed WatchDog Antimalware driver, version 1.1.100"
        author = "Jiri Vinopal @ Check Point Research"
        hash = "5af1dae21425dda8311a2044209c308525135e1733eeff5dd20649946c6e054c"
        hash = "0be8483c2ea42f1ce4c90e84ac474a4e7017bc6d682e06f96dc1e31922a07b10"
    condition:
        // Detect PE
        uint16(0) == 0x5a4d and uint16(uint32(0x3c)) == 0x4550 and
        // Detect 64-bit Windows drivers
        uint16(uint32(0x3C) + 0x5c) == 0x0001 and uint16(uint32(0x3C) + 0x18) == 0x020b and
        // Detect OriginalFilename "wamsdk.sys" and FileVersion "1.1.100"
        pe.version_info["OriginalFilename"] == "wamsdk.sys" and pe.version_info["FileVersion"] == "1
        // Detect only signed drivers, not a real verification
        pe.number_of_signatures > 0 and for all i in (0..pe.number_of_signatures -1):
            (pe.signatures[i].verified)
}
```

Detects 64-bit, valid-signed Advanced Malware Protection driver, `ZAM.exe`, version 3.0.0.000 (detected by LOLDrivers and Microsoft Vulnerable Driver Blocklist):

rule zam_advanced_malware_protection_driver_64bit_ver300000

description = "Detects 64-bit, valid-signed Advanced Malware Protection driver, version 3.0.0.000"

author = "Jiri Vinopal @ Check Point Research"

hash = "9c394dcab9f711e2bf585edf0d22d2210843885917d409ee56f22a4c24ad225e"

uint16(0) == 0x5a4d and uint16(uint32(0x3c)) == 0x4550 and

// Detect 64-bit Windows drivers

uint16(uint32(0x3C) + 0x5c) == 0x0001 and uint16(uint32(0x3C) + 0x18) == 0x020b and

// Detect OriginalFilename "ZAM.exe" and FileVersion "3.0.0.000"

pe.version_info["OriginalFilename"] == "ZAM.exe" and pe.version_info["FileVersion"] == "3.0.0.000" and

// Detect only signed drivers, not a real verification

pe.number_of_signatures > 0 and for all i in (0..pe.number_of_signatures -1):

(pe.signatures[i].verified)

import "pe" rule zam_advanced_malware_protection_driver_64bit_ver300000 { meta: description = "Detects 64-bit, valid-signed Advanced Malware Protection driver, version 3.0.0.000" author = "Jiri Vinopal @ Check Point Research" hash = "9c394dcab9f711e2bf585edf0d22d2210843885917d409ee56f22a4c24ad225e" condition: // Detect PE uint16(0) == 0x5a4d and uint16(uint32(0x3c)) == 0x4550 and // Detect 64-bit Windows drivers uint16(uint32(0x3C) + 0x5c) == 0x0001 and uint16(uint32(0x3C) + 0x18) == 0x020b and // Detect OriginalFilename "ZAM.exe" and FileVersion "3.0.0.000" pe.version_info["OriginalFilename"] == "ZAM.exe" and pe.version_info["FileVersion"] == "3.0.0.000" and // Detect only signed drivers, not a real verification pe.number_of_signatures > 0 and for all i in (0..pe.number_of_signatures -1): (pe.signatures[i].verified) }

```
import "pe"

rule zam_advanced_malware_protection_driver_64bit_ver300000
{
    meta:
        description = "Detects 64-bit, valid-signed Advanced Malware Protection driver, version 3.0.
        author = "Jiri Vinopal @ Check Point Research"
        hash = "9c394dcab9f711e2bf585edf0d22d2210843885917d409ee56f22a4c24ad225e"
    condition:
        // Detect PE
        uint16(0) == 0x5a4d and uint16(uint32(0x3c)) == 0x4550 and
        // Detect 64-bit Windows drivers
```

```
        uint16(uint32(0x3C) + 0x5c) == 0x0001 and uint16(uint32(0x3C) + 0x18) == 0x020b and
        // Detect OriginalFilename "ZAM.exe" and FileVersion "3.0.0.000"
        pe.version_info["OriginalFilename"] == "ZAM.exe" and pe.version_info["FileVersion"] == "3.0.
        // Detect only signed drivers, not a real verification
        pe.number_of_signatures > 0 and for all i in (0..pe.number_of_signatures -1):
            (pe.signatures[i].verified)
}
```

Source: https://research.checkpoint.com/2025/silver-fox-apt-vulnerable-drivers/