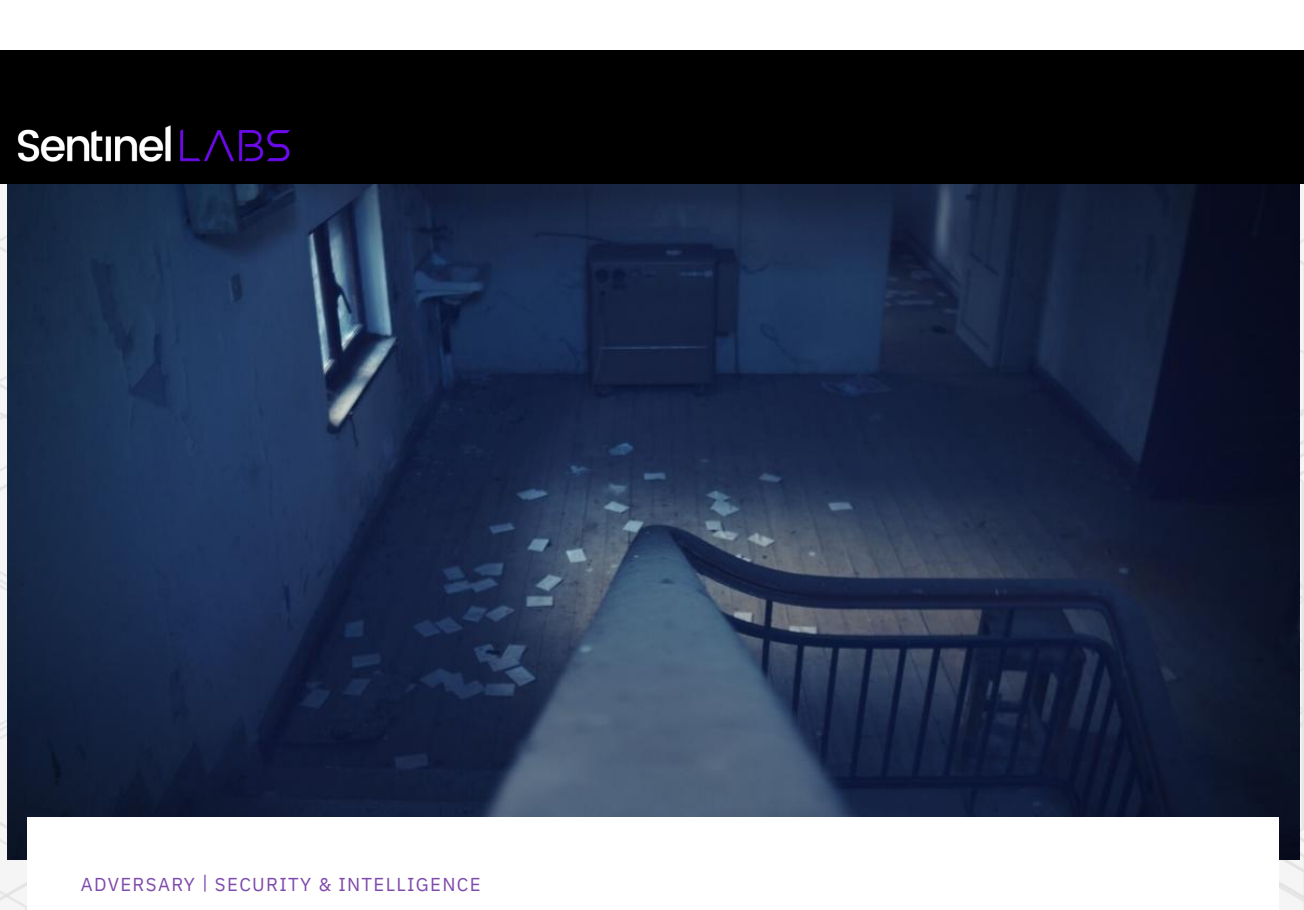
SentinelLABS



ADVERSARY | SECURITY & INTELLIGENCE

Driving Through Defenses | Targeted Attacks Leverage Signed Malicious Microsoft Drivers

SENTINELLABS / DECEMBER 13, 2022

Executive Summary

- SentinelOne has observed prominent threat actors abusing legitimately signed Microsoft drivers in active intrusions into telecommunication, BPO, MSSP, and financial services businesses.
- Investigations into these intrusions led to the discovery of POORTRY and STONESTOP malware, part of a small toolkit designed to terminate AV and EDR processes.
- We first reported our discovery to Microsoft's Security Response Center (MSRC) in October 2022 and received an official case number (75361). Today, MSRC released an associated advisory under ADV220005.
- This research is being released alongside Mandiant, a SentinelOne technology and incident response partner. Readers can find their blog here.

Targeting

In multiple recent investigations, SentinelOne's Vigilance DFIR team observed a threat actor utilizing a Microsoft signed malicious driver to attempt evasion of multiple security products. In subsequent sightings, the driver was used with a separate userland executable to attempt to control, pause, and kill various processes on the target endpoints. In some cases, the threat actor's intent was to ultimately provide SIM swapping services.

In 2022, the actors were involved in a variety of intrusions heavily targeting Business Process Outsourcing (BPO) and Telecommunications businesses. Additional targeting includes the Entertainment, Transportation, Managed Security Service Providers (MSSP), Financial, and Cryptocurrency sectors.

Notably, SentinelLabs observed a separate threat actor also utilizing a similar Microsoft signed driver, which resulted in the deployment of Hive ransomware against a target in the medical industry, indicating a broader use of this technique by various actors with access to similar tooling.

Background on the Driver Signing Process

Driver signing is an important security feature introduced as early as the first 64-bit Windows Operating Systems. It requires that drivers have valid cryptographic signatures in order to be loaded onto the system. That signature in turn identifies the developer as a pre-approved 'trustworthy' entity and their code inherits that trust. This validation was important in combating the scourge of kernel mode rootkits, malware designed to run with the highest privileges and thereby subvert attempts to detect or root them out. That battle has been going on for quite some time.

In order for a developer to have their driver signed by Windows Hardware Quality Labs (WHQL), they must first meet a specific set of criteria. This process was designed to rely on more strict checks to ensure that the requesting developer is real and not submitting malicious drivers.

A developer must first obtain an Extended Validation (EV) certificate and attach it to their account after registering for the Windows Hardware Developer Program. EV certificates require the developer to store the private key on a physical token and also include a series of checks to validate the requesting organization's identity. Once this is done, the developer must then submit their driver package through the Partner Portal for testing and signature. The drivers are run through a series of tests to ensure compatibility as well as any indications of malicious intent. If the driver passes the tests, it is then signed using Microsoft's "Windows Hardware Compatibility Publisher" certificate as shown below.

Microsoft's "Windows Hardware Compatibility Publisher" certificate Details

Abusing Trust With Signed Drivers

The main issue with this process is that most security solutions implicitly trust anything signed by only Microsoft, especially kernel mode drivers. Starting with Windows 10, Microsoft began requiring all kernel mode drivers to be signed using the Windows Hardware Developer Center Dashboard portal. Anything not signed through this process is not able to load in modern Windows versions. While the intent of this new requirement was to have stricter control and visibility over drivers operating at the kernel level, threat actors have realized if they can game the process they would have free reign to do what they want. The trick however, is to develop a driver that doesn't appear to be malicious to the security checks implemented by Microsoft during the review process.

This is not the first time a malicious kernel mode driver has been signed by Microsoft. In June 2021, GData published a blog on a malicious Netfilter rootkit signed through the same process described above. Microsoft subsequently acknowledged this and stated they "…will be sharing an update on how we are refining our partner access policies, validation and the signing process to further enhance our protections."

Analysis of a Malicious Microsoft-Signed Driver

A recent Microsoft-signed driver of interest belongs to a small toolkit designed to terminate AV and EDR processes. The toolkit contains simple protection mechanisms used to prevent its repurpose, reuse, and redistribution. The toolkit consists of two main components: a userland component (STONESTOP), and a kernel mode component (POORTRY). STONESTOP functions as both a loader/installer for POORTRY, as well as an orchestrator to instruct the driver with what actions to perform. POORTRY exposes an IOCTL interface that includes functionality to tamper with target processes supplied by the STONESTOP component.

We observed three versions of the driver toolkit that the threat actors have used in separate time periods. The first version of POORTRY was not signed directly by Microsoft. The second version was VMProtected and signed through the WHQL signing process. The third version was also signed through the WHQL signing process, but was protected with an unidentified packer.

The major operational differences between the toolkit versions are the use of software packing as an anti-analysis technique, the handling of the information about the processes designated for termination, and additional capabilities related to file tampering.

The POORTRY sample of the first version of the toolkit we observed was not packed. While in an infinite loop punctuated by one-second pauses between iterations, STONESTOP:

- reads process names from an external configuration file named, for example, `poyuo.pdata`.
- maps the process names to PIDs.
- masks the PIDs and sends the masked PIDs to POORTRY. STONESTOP masks PIDs using a value derived from a handshake protocol conducted between the userland component and the driver.

Before reading from the file, STONESTOP verifies the file's integrity against a predefined MD5 hash and terminates if the verification fails.

```
md5Hash = (char *)MD5_file("poyuo.pdata", 32);
if ( stricmp(md5Hash, "e63b30d765ce65e0b9674558ec22c2ba") )
{
  printf("%s\n", md5Hash);
  MessageBoxA(0i64, "Failed to verify 32 bit md5!", "System Error", 0x1010u);
  hSelf = GetCurrentProcess();
  TerminateProcess(hSelf, 0);
}
```

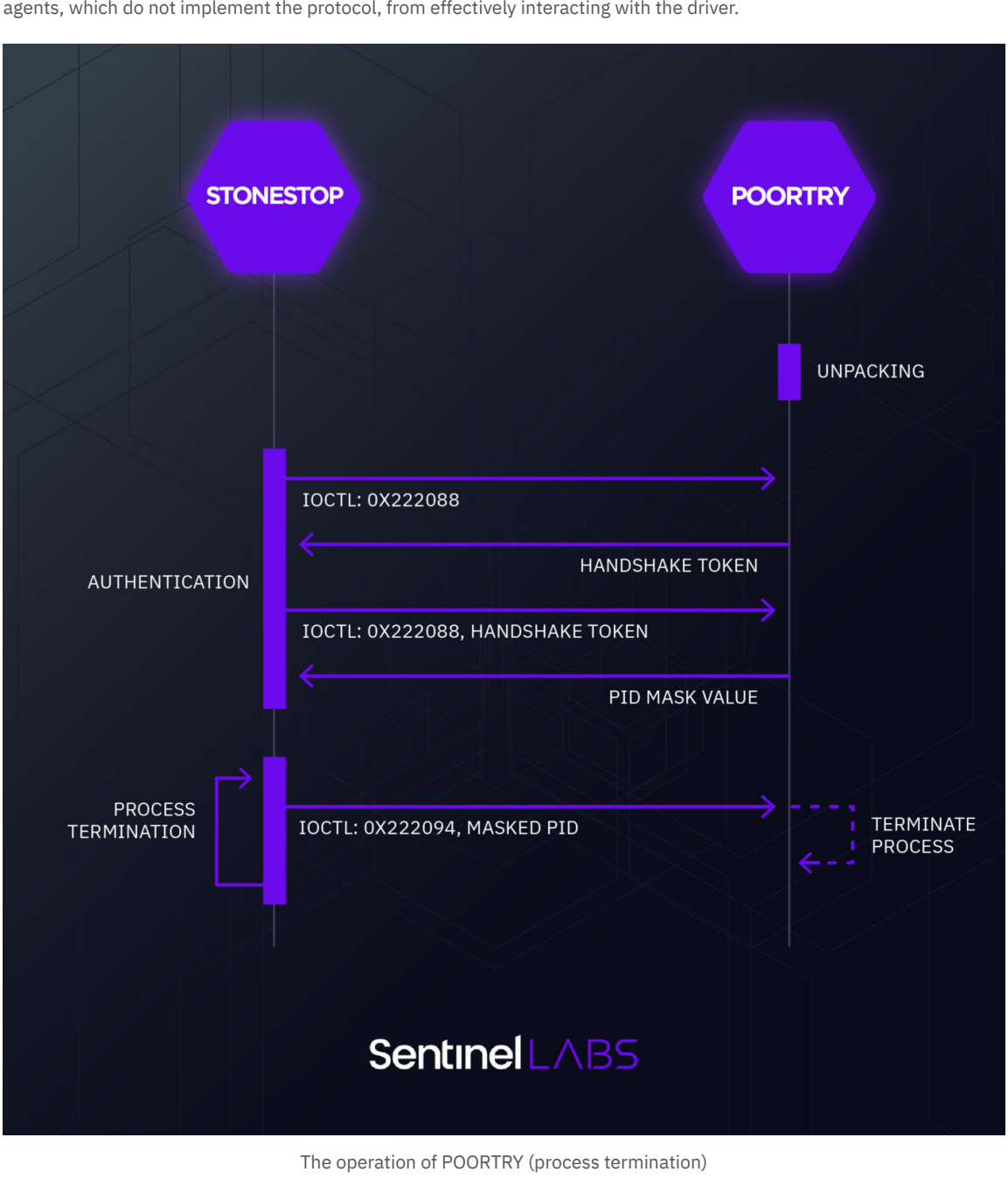
STONESTOP integrity check of configuration file

POORTRY's use and repeated reading of a configuration file indicates that the toolkit is intended for convenient reconfiguration during runtime for reuse against different target processes and environments. The time intervals between the repeated reading from the configuration file allows for reconfiguring POORTRY (i.e., editing the configuration file) during runtime.

However, this comes at the cost of a filesystem footprint that makes the toolkit easier to detect. Furthermore, the integrity verification of the configuration file ensures that it has not been tampered with before executing STONESTOP, for example, during transfer to the victim environment and staging.

The POORTRY variant of the second version of the toolkit that we observed was packed using VMProtect. The STONESTOP variant of this toolkit does not feature an external configuration file but embeds target process names instead. This version of the toolkit is specifically tailored to the target environment. Although it has no reconfiguration capabilities during runtime, it does offer a smaller file system footprint.

The operation of the second version of POORTRY can be broken down into three phrases: unpacking, authentication, and process termination. The driver is packed to protect the toolkit from both reuse and reverse engineering. The authentication consists of a simple four-way handshake that STONESTOP initializes with a pre-determined DWORD value. If the authentication succeeds, target processes are terminated in an infinite loop by sending POORTRY a specific IOCTL along with a masked target PID. The handshake protocol and the use of masked PIDs keeps other userland agents, which do not implement the protocol, from effectively interacting with the driver.



The operation of POORTRY (process termination)

The table below lists the IOCTLs for process manipulation that STONESTOP may issue to POORTRY. In addition to terminating processes, POORTRY offers the capability to also suspend and resume processes.

IOCTL	Description
0x222088	Used for authentication via a 4-way handshake that STONESTOP initializes with a pre-determined DWORD value. POORTRY then responds with a corresponding token. STONESTOP then sends this token back to POORTRY. The final response from the kernel mode component contains a value that is used to mask the target PID, by multiplying it by this value.
0x222094	Will kill processes by their PID. The PID is unmasked by the value generated from the previous IOCTL, then used to identify the EPROCESS object for the target process. Once this is identified, a Kernel APC routine that will call ZwTerminateProcess to terminate the process is queued for the target process.
0x22209c	Will suspend processes by PID. The PID is unmasked, and is then used to locate the target process' EPROCESS block. The EPROCESS block is then passed to PsSuspendProcess to perform the suspension.
0x2220a0	Used to resume processes by PID. Similar to the previous IOCTL, the PID will be unmasked, then is used to find the target EPROCESS structure, and finally is passed to PsResumeProcess.

The third POORTRY variant was packed with an unidentified packer. While the process termination flow is almost identical to version 2, version 3 does not contain the PID masking feature / protection mechanism. It also does not feature the full authentication handshake but instead a shortened version without the handshake token.

Additional functionality in the 3rd version of the toolkit includes the ability to tamper with files. The file-tampering functionality features the capability to overwrite files, as well as delete files from disk.

The table below lists the IOCTLs for file manipulation that STONESTOP may issue to POORTRY. The 3rd version of POORTRY offers functionality to delete files from disk, as well as overwrite files already present on disk.

IOCTL	Description
0x222184	Will delete files by path. The target file path is sent in cleartext without any type of masking or encoding from STONESTOP to POORTRY.
0x22218c	Used to overwrite target files. This routine expects two parameters: a destination file path to overwrite, and a source path to read from. The source file's contents are read and used to overwrite the destination path.

Detection Opportunity

Because EDR vendors are somewhat forced into trusting signed drivers by Microsoft, it can be difficult to distinguish between legitimate benign examples and malicious ones that slip through the security checks. Fortunately, there are some small breadcrumbs within the Authenticode signature that vendors can extract to allow for more granular control over malicious drivers submitted through the Microsoft signing process.

Microsoft provides code to extract vendor specific metadata from the SPC_SP_OPUS_INFO fields in the Authenticode signatures. If leveraged against the two malicious drivers in our instance, we are able to extract the original requesting publishers from them:

```
8e035beb02a411f8a9e92d4cf184ad34f52bbd0a81a50c222cdd4706e4e45104
大连纵梦网络科技有限公司
c8f9e1ad7b8cce62fba349a00bc168c849d42cfb2ca5b2c6cc4b51d054e0c497
Luck Bigger Technology Co., Ltd
```

This information can prove useful for security vendors as it would allow them to potentially blocklist Microsoft signed drivers based on the originating publisher.

Our Supplier Theory

We are highly confident that the malicious drivers mentioned above, as well as the one from June 2021, were used by different threat actors. This raises an important question: Is the driver signing process being exploited by a supplier(s) and offered as a service available to various threat actors willing to pay?

A competing theory is that multiple threat actors have compromised legitimate driver developers and surreptitiously used their EV certificate to sign and submit the malicious drivers using their developer account. However, this scenario is less likely due to the requirement that EV private keys be stored on a physical hardware token intended to help prevent digital theft.

Other evidence supporting the 'supplier' theory stems from the similar functionality and design of the drivers. While they were used by two different threat actors, they functioned in very much the same way. This indicates they were possibly developed by the same person then subsequently sold for use by someone else.

Conclusion

Code signing mechanisms are an important feature in modern operating systems. The introduction of driver signing enforcement was key in stemming the tide of rootkits for years. The receding effectiveness of code signing represents a threat to security and verification mechanisms at all OS layers. We hope that Microsoft will take steps to consider further enhancements to bolster the security of their signing process to help maintain the implicit trust placed in Microsoft-signed drivers.

In the meantime, we continue to monitor the abuse of signed drivers in collaboration with Mandiant researchers. We hope defenders will prioritize these TTPs to bolster their defenses.

Indicators of Compromise

Version 1

MD5	04a88f5974caa621cee18f34300fc08a
SHA-1	a804ebec7e341b4d98d9e94f6e4860a55ea1638d
SHA-256	9b1b15a3aacb0e786a608726c3abfc94968915cedcbd239ddf903c4a54bfcf0c

Version 2

MD5	6fcf56f6ca3210ec397e55f727353c4a
SHA-1	6debce728bcff73d9d1d334df0c6b1c3735e295c
SHA-256	8e035beb02a411f8a9e92d4cf184ad34f52bbd0a81a50c222cdd4706e4e45104

Version 3

MD5	909f3fc221acbe999483c87d9ead024a
SHA-1	b2f955b3e6107f831ebe67997f8586d4fe9f3e98
SHA-256	c8f9e1ad7b8cce62fba349a00bc168c849d42cfb2ca5b2c6cc4b51d054e0c497

ADVERSARY

SENTINELLABS

SentinelLabs is an open venue for our threat researchers and vetted contributors to reliably share their latest findings with a wider community of defenders. We are hunters, reversers, exploit developers, and tinkerers shedding light on the world of malware, exploits, APTs, and cybercrime across all platforms.