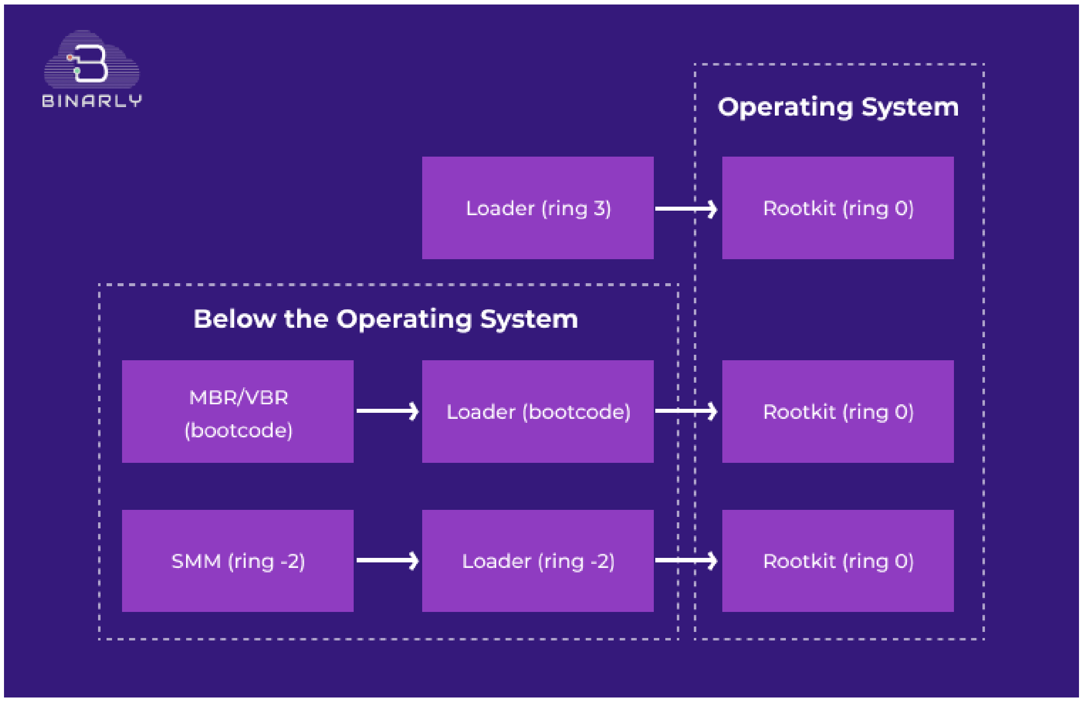
Design issues of modern EDRs:bypassing ETW-based solutions

binarly.io/posts/Design_issues_of_modern_EDRs_bypassing_ETW-based_solutions/index.html

As experts in firmware security, the Binarly team is frequently asked why endpoint solutions can't detect threats originating below the operating system such as firmware implant payloads. Unfortunately, the problem requires a more complex approach and the modern architecture of Endpoint Detection & Response (EDR) solutions are weak against generic attack patterns.

At Black Hat Europe 2021, Binarly researchers presented several attack vectors that weren't aimed at attacking a single solution, but instead exposed industry-wide problems. The technical details of two new UEFI bootloader-based pieces of malware (FinSpy and ESPecter, which behave similarly to classical bootkits, have been published recently. However, instead of infecting legacy bootstrap code (MBR/VBR), they attack the UEFI-based bootloader to persist below the operating system. These types of threats can influence the kernel-space before all the mitigations apply. This allows an attacker to install kernel-mode implants or rootkit code very early in the boot process.



Past publicly available ETW bypasses rely on hooking/unhooking techniques to alter the executable files loading with runtime changes such as reflective code injection (You're off the hook – Zero Nights 2016, Matrosov & Tang)

With that in mind, the Binarly research chose to focus on uncovering ETW design problems and uncover attacks that affect all the solutions relying on ETW telemetry. Firmware implants to deliver operating system payloads implementing these attacks will NOT be detected by modern endpoint solutions.

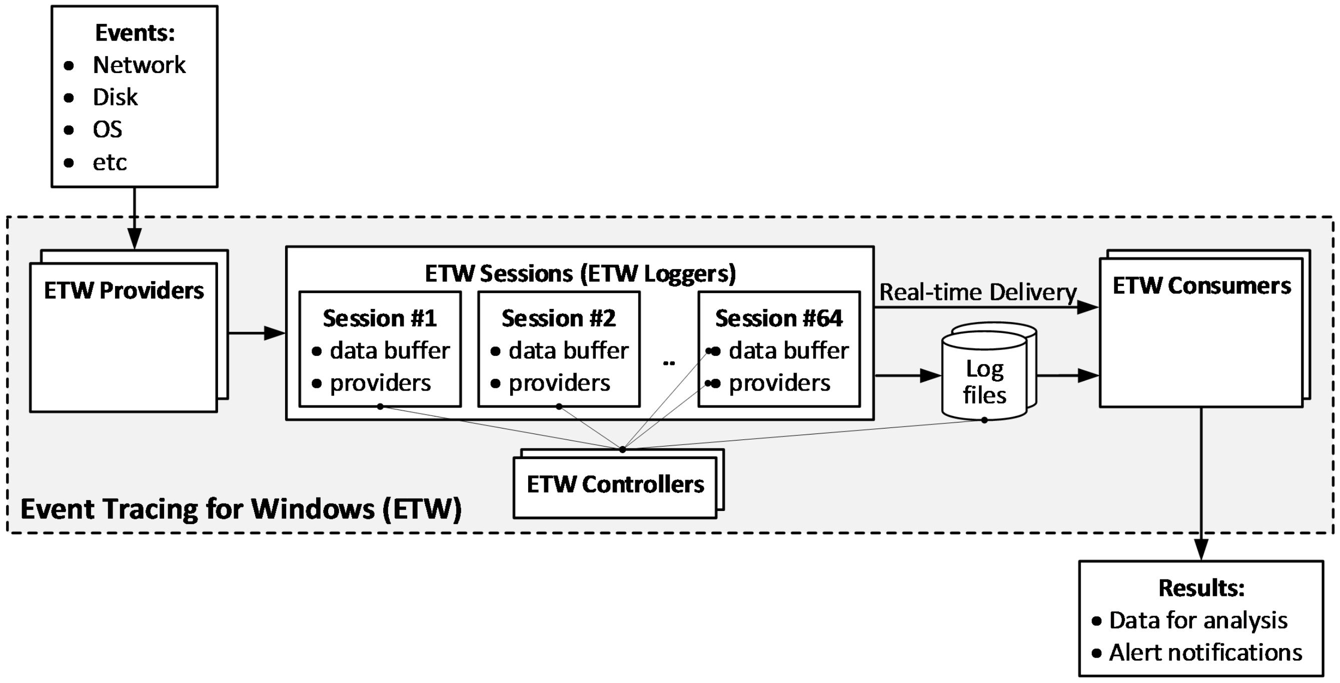
Design issues are the worst

Event Tracing for Windows (ETW) is a built-in feature, originally designed to perform software diagnostics, and nowadays ETW is widely used by Endpoint Detection & Response (EDR) solutions.

Attacks on ETW can blind a whole class of security solutions that rely on telemetry from ETW. Researching ways to disable ETW is of critical importance given that these attacks can result in disabling the whole class of EDR solutions that rely on ETW for visibility into the host events. Even more important is researching and developing ways to detect if the ETW has been tampered with and notify the security solutions in place.

This topic is very important for all the experts dealing with Windows security, malware detection and incident response.

Event Tracing for Windows (ETW) is a built-in Windows logging mechanism designed to observe and analyze application behavior. ETW was introduced quite a while ago (Windows XP) as a framework implemented in the kernel to troubleshoot OS components behavior and performance issues; since then, it has been expanded and improved significantly - in Windows 11, ETW can produce more than 50,000 event types from about 1,000 providers.



Using ETW to collect host telemetry has the following advantages:

- Available system-wide, in all recent Windows OSs without having to be installed, loading any kernel drivers or OS rebooting.
- Supports a standardized framework to produce and consume logging events.
- High-speed logging that lets applications consume events in real-time or from a disk file.

ETW is used to collect events in large-scale business solutions such as Docker, Amazon CloudWatch. MS SQL Server has been using ETW for more than 10 years.

One of the first examples of using ETW-based tools to analyze and reveal malware behavior was presented by Mark Russinovich in his talk "Malware Hunting with the Sysinternals Tools" about 10 years ago. Since then, developers of modern EDRs have leveraged ETW to monitor security related events and successfully detect and respond to cutting-edge malware.

As an example, Process Monitor from the SysInternals suite was leveraging *NT Kernel Logger* ETW session for network tracing. Its latest version is still relying on ETW for such visibility but a dedicated ETW session, *PROCMON TRACE*, was introduced. During the Black Hat Europe 2021 talk, the Binarly team demonstrated an ETW attack on Processor Monitor that resulted in blinding the SysInternals tool from any network activity.

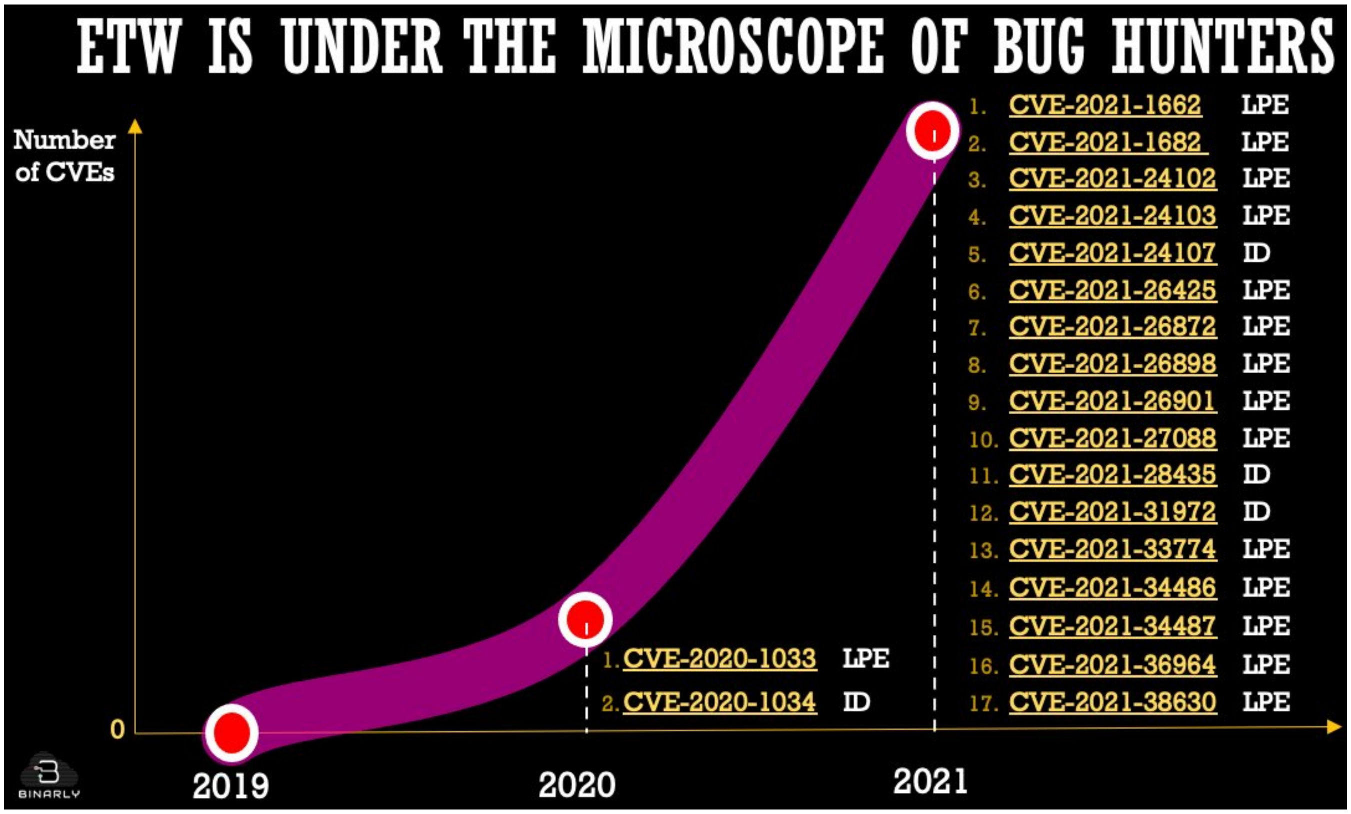
There are plenty of practical research projects demonstrating the ability of ETW to capture malicious activity or perform threat research and reverse engineering:

- DARPA has sponsored several ETW-based monitoring systems for malware detection
  - Project **Windows Low-Level System Monitoring Data Collection** obtains data from many ETW providers, including NT Kernel Logger to reveal and reconstruct various attacks such as browser exploit attacks and malicious file download.
  - Project **MARPLE** focuses on hardening enterprise security by automating the detection of APT threats. One of its modules, Holmes, collects host telemetry via ETW to produce detection signals for APT campaigns.
  - Project **APTShield** uses ETW for logging system call routines. This scheme helps to detect RATs by analyzing malicious behavior, such as key logging, screen grabbing, remote shell, audio recording and unauthorized registry manipulation.
- MITRE-built ETW-based Security Sensor to detect process injection, capture process creation and termination, file creation and deletion events for certain filenames and paths.

ETW is a hot topic in many academic papers focused mostly on detecting malicious behavior:

- Malware Characterization Using behavioral Components from George Mason University
- Detecting File-less Malicious Behavior of.NET C2 agents using ETW from University of Amsterdam
- Tactical Provenance Analysis for Endpoint Detection and Response Systems form University of Illinois
- Etc.

According to the MITRE CVE database, in 2021, there is an exponential rise in the number of ETW related vulnerabilities that received a CVE number. So it is safe to assume that ETW has caught the attention of Bug hunters:



As discussed, ETW is helpful for gathering telemetry to defend against attacks, but it has drawbacks.

One important drawback is that ETW has an opaque structure, including undocumented providers and providers that issue undocumented events - ETW event templates are stored in the PE resource section under WEVT_TEMPLATE resource id.

ETW can be leveraged by living-off-the-land malware:

- ETW can provide sniffer functionality for file & registry operations, process, thread & network activity
- ETW can provide keylogger functionality
- ETW can be used to flood the HDD in DDOS attacks, since events can be cached to disk in log files
- malware can use ETW to detect sandbox detonations
- some ETW providers are available only for certain Protected Process Light (PPL) processes; but malware can disable PPL on targeted processes via a kernel mode driver without causing BSOD – and next disable the "hidden" ETW providers

Unfortunately for the defenders, ETW can be BYPASSED!

Many ways to disable ETW logging are publicly available from passing a TRUE boolean parameter into a `nt!EtwpStopTrace` function to finding an ETW specific structure and dynamically modifying it or patching `ntdll!ETWEventWrite` or `advapi32!EventWrite` to return immediately thus stopping the user-mode loggers.

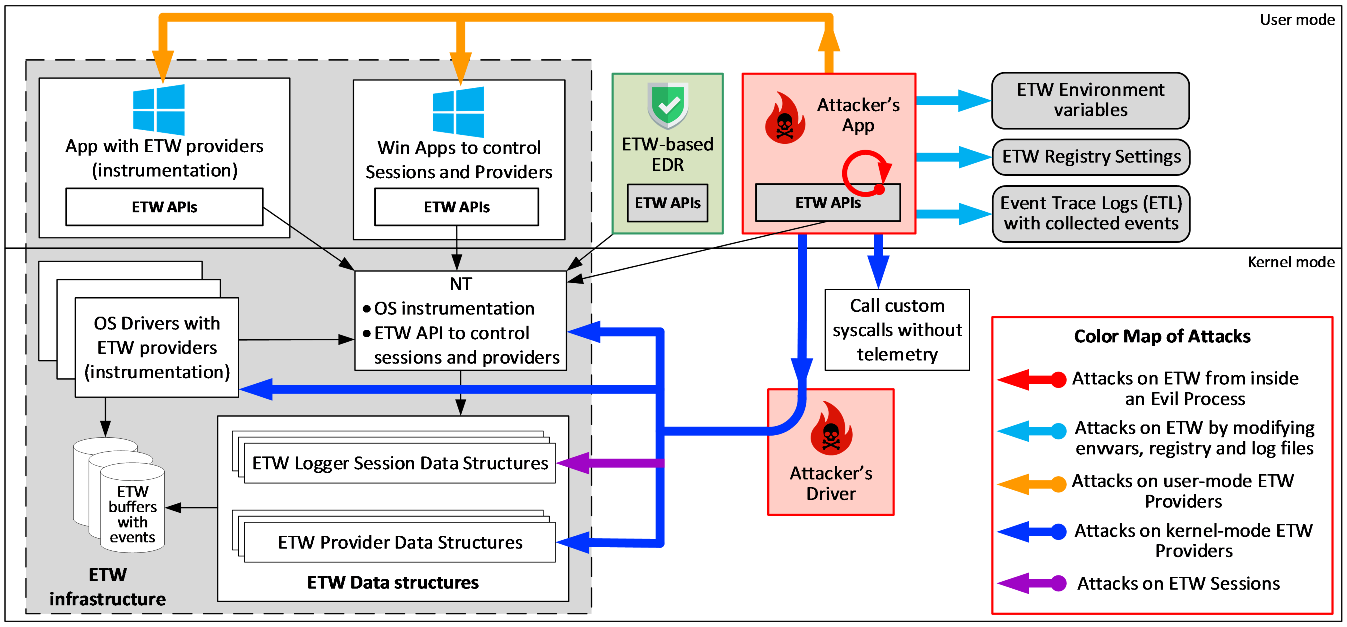
Over the past several years, there have been many examples of malware families that implemented mitigations to evade ETW-based logging.

- In March 2018, Kaspersky released a report on Slingshot, a complex and unknown cyber-espionage platform targeting African and Middle East countries.
  - **Slingshot**, the first-stage loader, renames the ETW-logs by appending .tmp extension to avoid leaving traces of its activity in system logs.
  - **Minisling**, a module in the platform, uses Microsoft-Windows-Kernel-General ETW provider to obtain the last reboot time and the Microsoft-Windows-Kernel-Power ETW provider to obtain the last unsuccessful attempt to turn the machine off.
- In 2019, **LockerGoga** Ransomware implemented functionality to disable ETW by turning off tracing from Microsoft-Windows-WMI-Activity provider via **wevtutil** tool.

- In August 2021, TrendMicro researchers published the details of a new campaign from **APT41** targeting South-China Seas countries where it was mentioned the use of 2 new shellcode loaders that can run payloads, uninstall themselves, disable ETW to evade detection, and also try out credentials.

MITRE has updated the Defense Evasion category to take into account these attacks by adding a new technique called "Indicator Blocking" as a separate sub-technique to Impair Defenses technique.

The ETW Threat Modeling is presented below:


User mode
App with ETW providers (instrumentation)
ETW APIs
Win Apps to control Sessions and Providers
ETW APIs
ETW-based EDR
ETW APIs
Attacker's App
ETW APIs
ETW Environment variables
ETW Registry Settings
Event Trace Logs (ETL) with collected events
Kernel mode
OS Drivers with ETW providers (instrumentation)
NT
• OS instrumentation
• ETW API to control sessions and providers
Call custom syscalls without telemetry
Attacker's Driver
ETW buffers with events
ETW Logger Session Data Structures
ETW Provider Data Structures
ETW Data structures
ETW infrastructure
Color Map of Attacks
Attacks on ETW from inside an Evil Process
Attacks on ETW by modifying envvars, registry and log files
Attacks on user-mode ETW Providers
Attacks on kernel-mode ETW Providers
Attacks on ETW Sessions

Threat Modeling ETW

There are 5 types of attacks, designated by a different color, targeting each component of the ETW architecture:

- Red shows attacks on ETW from inside an Evil Process
- Light blue shows attacks on ETW by modifying environment variables, registry and files
- Orange shows attacks on user-mode ETW Providers
- Dark blue shows attacks on kernel-mode ETW Providers
- Purple shows attacks on ETW Sessions

The Binarly team recapped the most important bypasses publicly available during their Black Hat Europe 2021 talk. The most up-to-date summary of the ETW attacks by category is presented below:

Type of Attack	Number of different techniques
Attacks from inside AN EVIL process	6
Attacks on ETW environment variables, registry, and files	3
Attacks on user-mode ETW providers	11
Attacks on kernel-mode ETW providers	7
Attacks on ETW sessions	9
Total Attacks	**36**

New attacks discovered by Binarly REsearch

Before introducing a new attack on Process Monitor presented by the Binarly team, let's talk a bit more about ETW Sessions. An ETW session is a global object identified by a unique name that allows multiple ETW consumers to subscribe and receive events from different ETW providers. To make matters more obfuscated, there is neither API nor documentation on how to identify the session a consumer subscribes to receive events from.

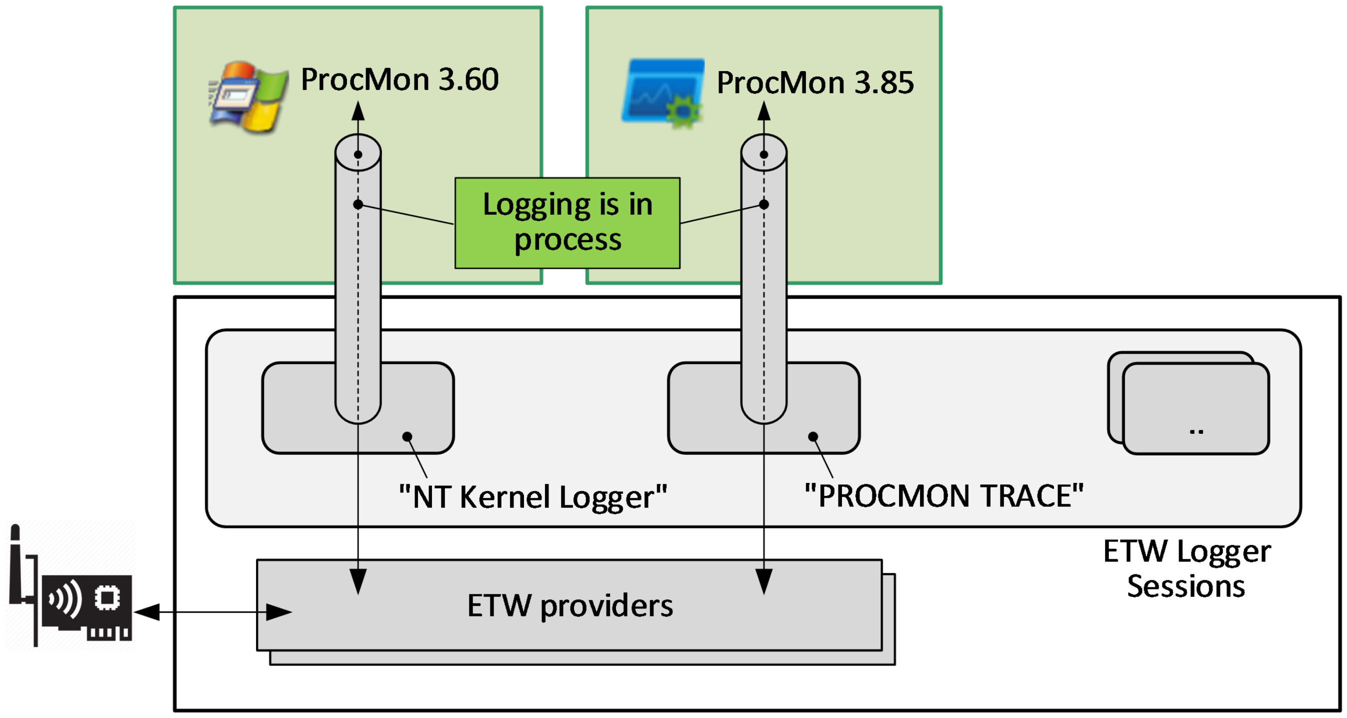
The EtwCheck (it will be released soon, stay tuned) is a powerful tool that can extract various kernel data including the in-memory WMI_LOGGER_CONTEXT structures for the corresponding sessions. These structures contain the sessions Security Descriptors, Flags and PIDs that can help identify the applications that subscribe to these sessions.

The default number of simultaneously running sessions is 64. The NT kernel supports a maximum of 8 System Logger Sessions with hardcoded unique names. Some examples of System Session include NT Kernel Logger, Global Logger, and Circular Kernel Context Logger.

NT Kernel Logger session receives telemetry from different providers implemented in the ntoskrnl.exe and OS core drivers. One important key point, which will be leveraged in the attack on Process Monitor, is that Windows supports only one active NT Kernel Logger session at any time but any application with admin privileges can start/stop this session.

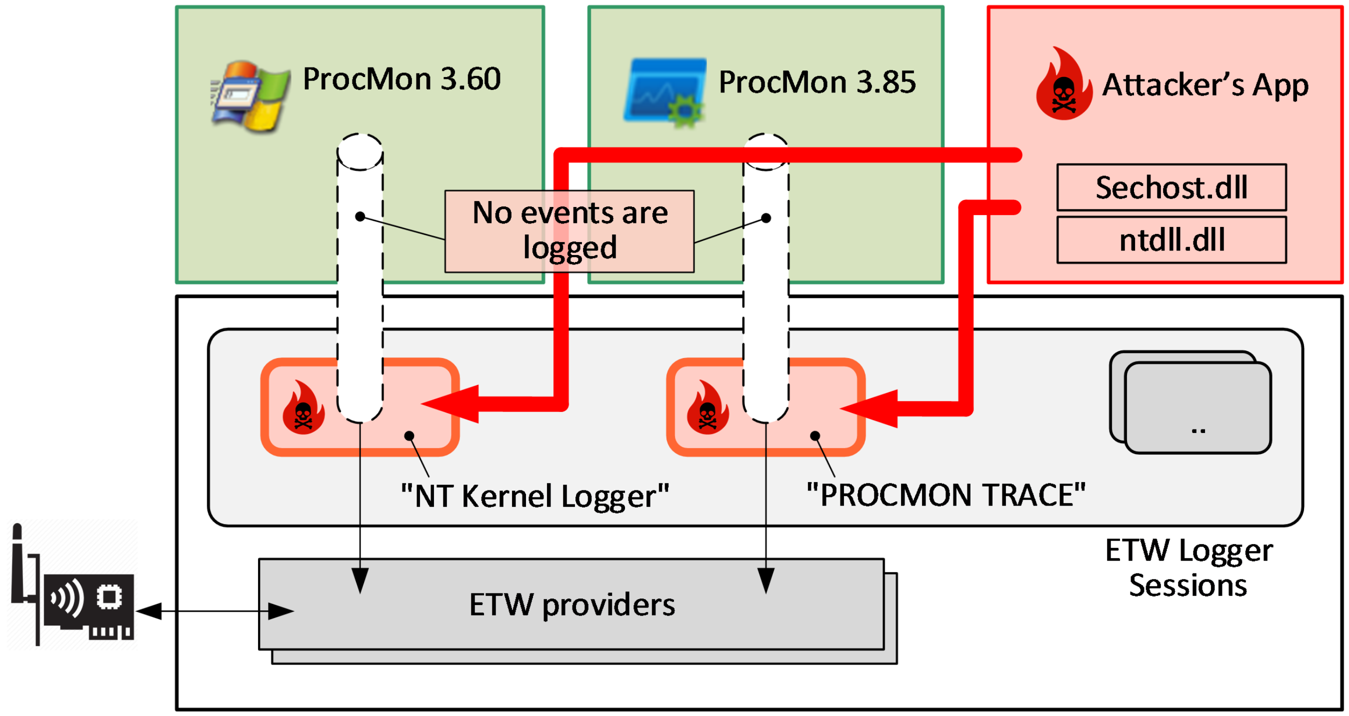
Process Monitor is free, and very popular for malware analysis which uses absolutely the same technology as many EDRs. To receive network events Process Monitor launches an ETW session as follows:

- Process Monitor version up to 3.60 uses **NT Kernel Logger** session.
- Process Monitor version 3.85 uses a session called **PROCMON TRACE**.



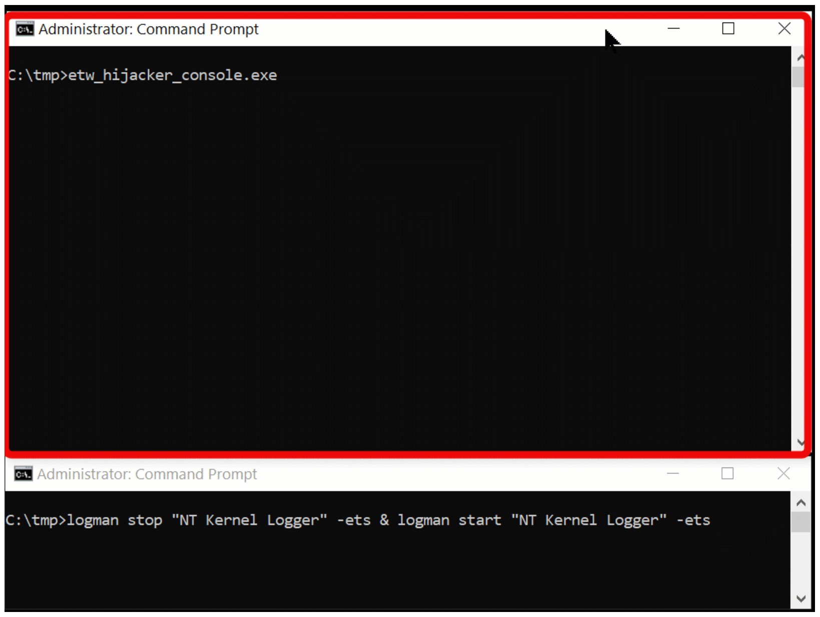
Process Monitor uses ETW to sniff network events

From an attacker application, the running session that Processor Monitor relies on is stopped and a fake one is started. As a result, Process Monitor stops receiving network events and relaunching it does not fix the problem.



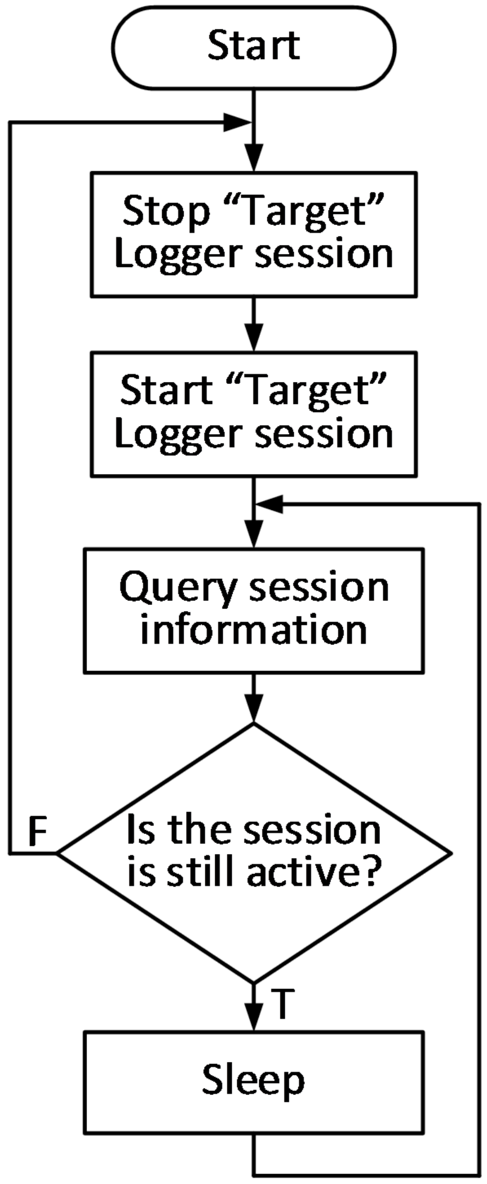
ETW Hijacker blinds Process Monitor by stopping ETW sessions

To demonstrate this attack, Binarly team has developed ETW Hijacker (it will be released soon, stay tuned) which functionality is based on the following control flow:



```
C:\tmp>etw_hijacker_console.exe
```

```
C:\tmp>logman stop "NT Kernel Logger" -ets & logman start "NT Kernel Logger" -ets
```

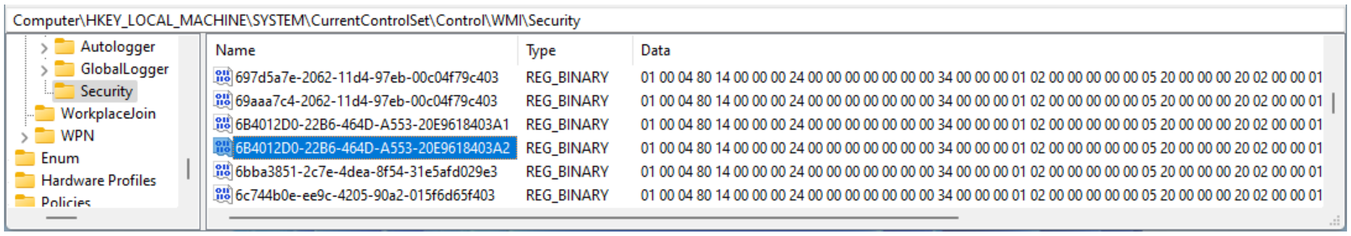

Start
Stop "Target" Logger session
Start "Target" Logger session
Query session information
Is the session is still active?
F
T
Sleep

Block diagram of ETW Hijacker

In the second attack introduced during the talk, the targeted application is Windows Defender and its secure ETW sessions:

Logger ID	Logger Name	Instance GUID	Registry Path
4	DefenderApiLogger	{6B4012D0-22B6-464D-A553-20E9618403A**2**}	HKLM\SYSTEM\CurrentControlSet\Control\WMI\Autologger\DefenderApiLogger
5	DefenderAuditLogger	{6B4012D0-22B6-464D-A553-20E9618403A**1**}	HKLM\SYSTEM\CurrentControlSet\Control\WMI\Autologger\DefenderAuditLogger

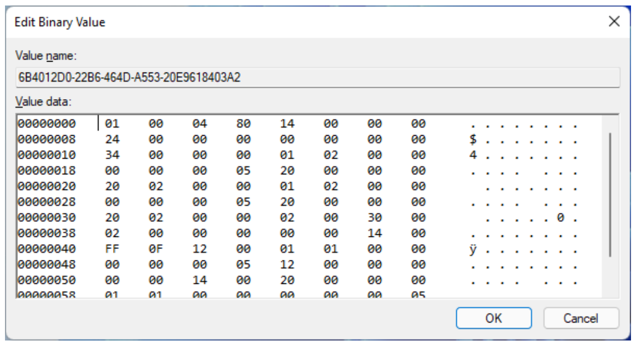
Each ETW session has an associated Security Descriptor, which is located in the registry in the `HKLM\SYSTEM\CurrentControlSet\Control\WMI\Security` key. The binary content of the security descriptor associated with `DefenderApiLogger` is written in the `6B4012D0-22B6-464D-A553-20E9618403A2` value and the binary content of the security descriptor associated with `DefenderAuditLogger` is written in the `6B4012D0-22B6-464D-A553-20E9618403A1` value under `HKLM\SYSTEM\CurrentControlSet\Control\WMI\Security` key.



Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\WMI\Security

Name	Type	Data
697d5a7e-2062-11d4-97eb-00c04f79c403	REG_BINARY	01 00 04 80 14 00 00 00 24 00 00 00 00 00 00 00 34 00 00 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 01
69aaa7c4-2062-11d4-97eb-00c04f79c403	REG_BINARY	01 00 04 80 14 00 00 00 24 00 00 00 00 00 00 00 34 00 00 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 01
6B4012D0-22B6-464D-A553-20E9618403A1	REG_BINARY	01 00 04 80 14 00 00 00 24 00 00 00 00 00 00 00 34 00 00 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 01
6B4012D0-22B6-464D-A553-20E9618403A2	REG_BINARY	01 00 04 80 14 00 00 00 24 00 00 00 00 00 00 00 34 00 00 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 01
6bba3851-2c7e-4dea-8f54-31e5afd029e3	REG_BINARY	01 00 04 80 14 00 00 00 24 00 00 00 00 00 00 00 34 00 00 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 01
6c744b0e-ee9c-4205-90a2-015f6d65f403	REG_BINARY	01 00 04 80 14 00 00 00 24 00 00 00 00 00 00 00 34 00 00 00 01 02 00 00 00 00 00 05 20 00 00 00 20 02 00 00 01

As it can be seen in the next figure, the Security Descriptor stored in the registry matched the corresponding Security Descriptor in kernel memory (WMI_LOGGER_CONTEXT.SecurityDescriptor.Object).

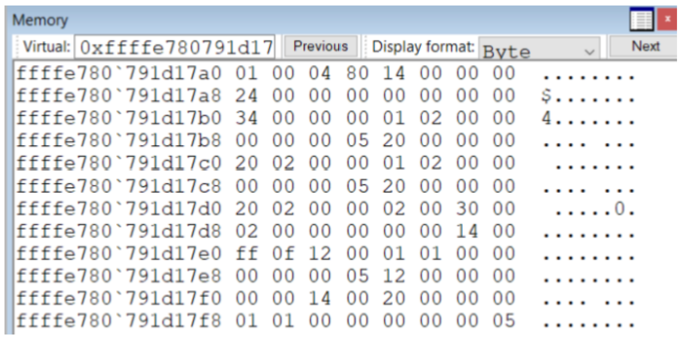


Edit Binary Value

Value name:
6B4012D0-22B6-464D-A553-20E9618403A2

Value data:

```
00000000  01  00  04  80  14  00  00  00    ........
00000008  24  00  00  00  00  00  00  00    $.......
00000010  34  00  00  00  01  02  00  00    4.......
00000018  00  00  00  05  20  00  00  00    .... ...
00000020  20  02  00  00  01  02  00  00     .......
00000028  00  00  00  05  20  00  00  00    .... ...
00000030  20  02  00  00  02  00  30  00     .....0.
00000038  02  00  00  00  00  00  14  00    ........
00000040  FF  0F  12  00  01  01  00  00    ÿ.......
00000048  00  00  00  05  12  00  00  00    ........
00000050  00  00  14  00  20  00  00  00    .... ...
00000058  01  01  00  00  00  00  00  05
```



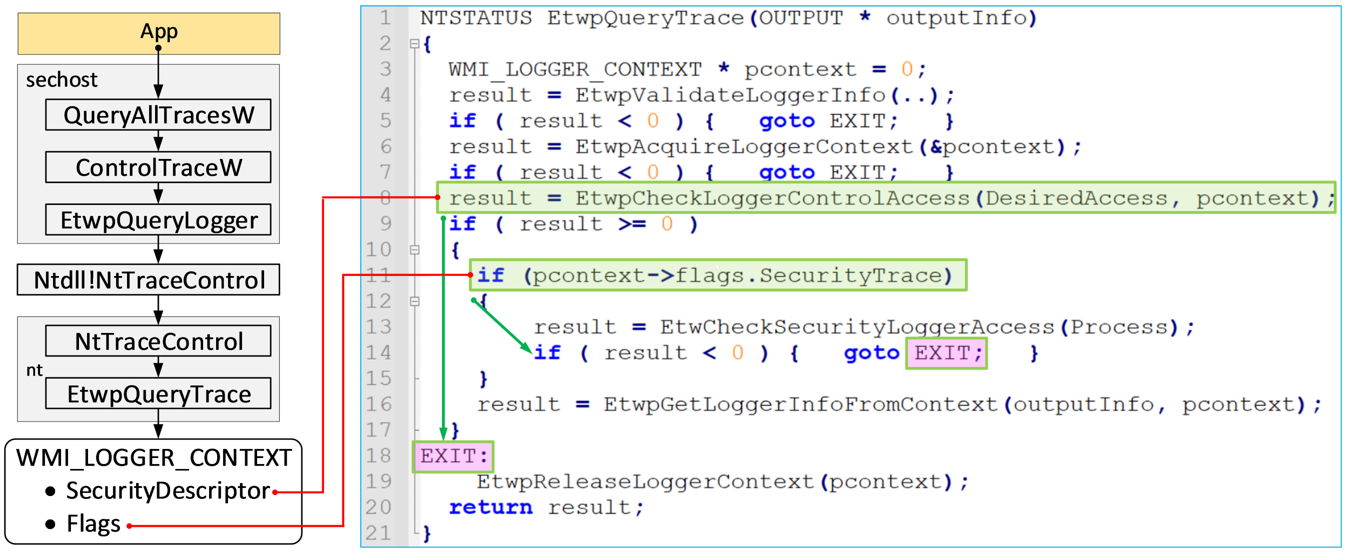
```
ffffe780`791d17a0 01 00 04 80 14 00 00 00  ........
ffffe780`791d17a8 24 00 00 00 00 00 00 00  $.......
ffffe780`791d17b0 34 00 00 00 01 02 00 00  4.......
ffffe780`791d17b8 00 00 00 05 20 00 00 00  .... ...
ffffe780`791d17c0 20 02 00 00 01 02 00 00   .......
ffffe780`791d17c8 00 00 00 05 20 00 00 00  .... ...
ffffe780`791d17d0 20 02 00 00 02 00 30 00   .....0.
ffffe780`791d17d8 02 00 00 00 00 00 14 00  ........
ffffe780`791d17e0 ff 0f 12 00 01 01 00 00  ........
ffffe780`791d17e8 00 00 00 05 12 00 00 00  ........
ffffe780`791d17f0 00 00 14 00 20 00 00 00  .... ...
ffffe780`791d17f8 01 01 00 00 00 00 00 05  ........
```

Security descriptor for DefenderApiLogger in Registry and in the kernel memory

One simple way to blind Windows Defender is to zero out registry values corresponding to its ETW sessions:

```
reg add "HKLM\System\CurrentControlSet\Control\WMI\Autologger\DefenderApiLogger" /v "Start" /t REG_DWORD
/d "0" /f
```

Windows provides QueryAllTracesW API to retrieve the properties and statistics for all ETW sessions for which the caller has permissions to query. After calling this function the execution control goes to the kernel and finally to EtwpQueryTrace function (its corresponding pseudocode shown below). As it can be seen in the pseudocode, the EtwpQueryTrace function includes several security checks to prevent returning information about a secure session to unprivileged applications. This is the reason why event apps with admin privileges can't query the Windows Defender ETW sessions. First, the access rights of the caller are checked by comparing the session security descriptor with the process token. Second, it checks whether the queried session has its SecurityTrace flag set and implement one more check based on the PPL mechanism in the EtwCheckSecurityLoggerAccess function.

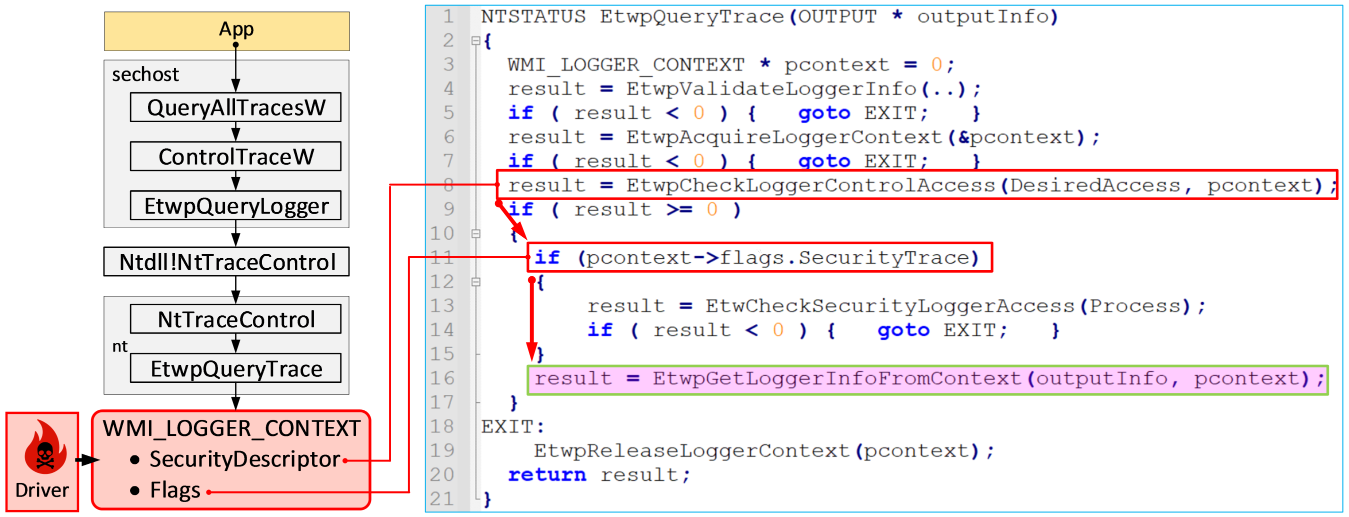


```
NTSTATUS EtwpQueryTrace(OUTPUT * outputInfo)
{
  WMI_LOGGER_CONTEXT * pcontext = 0;
  result = EtwpValidateLoggerInfo(..);
  if ( result < 0 ) {   goto EXIT;   }
  result = EtwpAcquireLoggerContext(&pcontext);
  if ( result < 0 ) {   goto EXIT;   }
  result = EtwpCheckLoggerControlAccess(DesiredAccess, pcontext);
  if ( result >= 0 )
  {
    if (pcontext->flags.SecurityTrace)
    {
      result = EtwCheckSecurityLoggerAccess(Process);
      if ( result < 0 ) {   goto EXIT;   }
    }
    result = EtwpGetLoggerInfoFromContext(outputInfo, pcontext);
  }
EXIT:
    EtwpReleaseLoggerContext(pcontext);
  return result;
}
```

EtwpQueryTrace function checks two fields: Security Descriptor and Flags. SecurityTrace

By default, users cannot query information about Defender ETW sessions since they are running with high privilege and have SecurityTrace flag enabled. Both session parameters, the security descriptor and SecurityTrace flag, are stored in the WMI_LOGGER_CONTEXT structure.

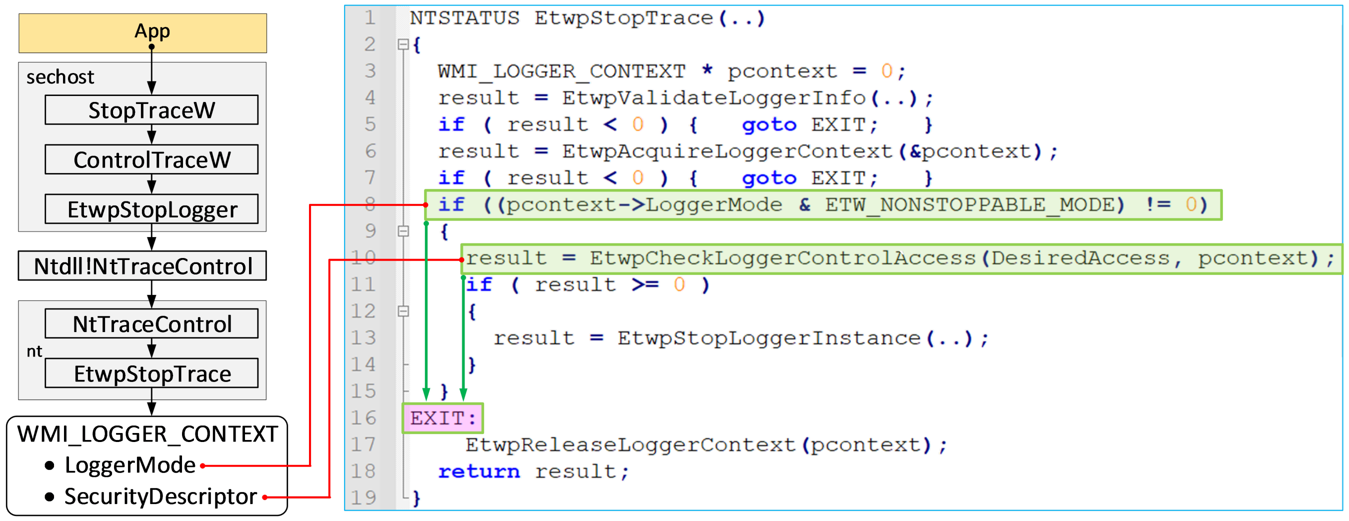
Malware can load a driver that patches the aforementioned values in the targeted WMI_LOGGER_CONTEXT structure to make the execution flow bypass the security checks and execute EtwpGetLoggerInfoFromContext function.



```
NTSTATUS EtwpQueryTrace(OUTPUT * outputInfo)
{
  WMI_LOGGER_CONTEXT * pcontext = 0;
  result = EtwpValidateLoggerInfo(..);
  if ( result < 0 ) {   goto EXIT;   }
  result = EtwpAcquireLoggerContext(&pcontext);
  if ( result < 0 ) {   goto EXIT;   }
  result = EtwpCheckLoggerControlAccess(DesiredAccess, pcontext);
  if ( result >= 0 )
  {
    if (pcontext->flags.SecurityTrace)
    {
      result = EtwCheckSecurityLoggerAccess(Process);
      if ( result < 0 ) {   goto EXIT;   }
    }
    result = EtwpGetLoggerInfoFromContext(outputInfo, pcontext);
  }
EXIT:
    EtwpReleaseLoggerContext(pcontext);
  return result;
}
```

Malware can patch the WMI_LOGGER_CONTEXT structure to allow querying Defender ETW sessions

Now, moving to stopping secure ETW sessions, Windows provides StopTraceW function to stop the specified ETW session. After calling this function the execution control goes to the kernel and finally to EtwpStopTrace function (its corresponding pseudocode shown below). As it can be seen in the pseudocode, the EtwpStopTrace function includes several security checks to prevent stopping the targeted secure session by unprivileged applications. This is the reason why even apps with admin privileges can't stop the Windows Defender ETW sessions. First, the session "stoppable" characteristics is checked by querying the LoggerMode field in the WMI_LOGGER_CONTEXT structure. Second, the access rights of the caller are checked by comparing the session security descriptor with the process token.

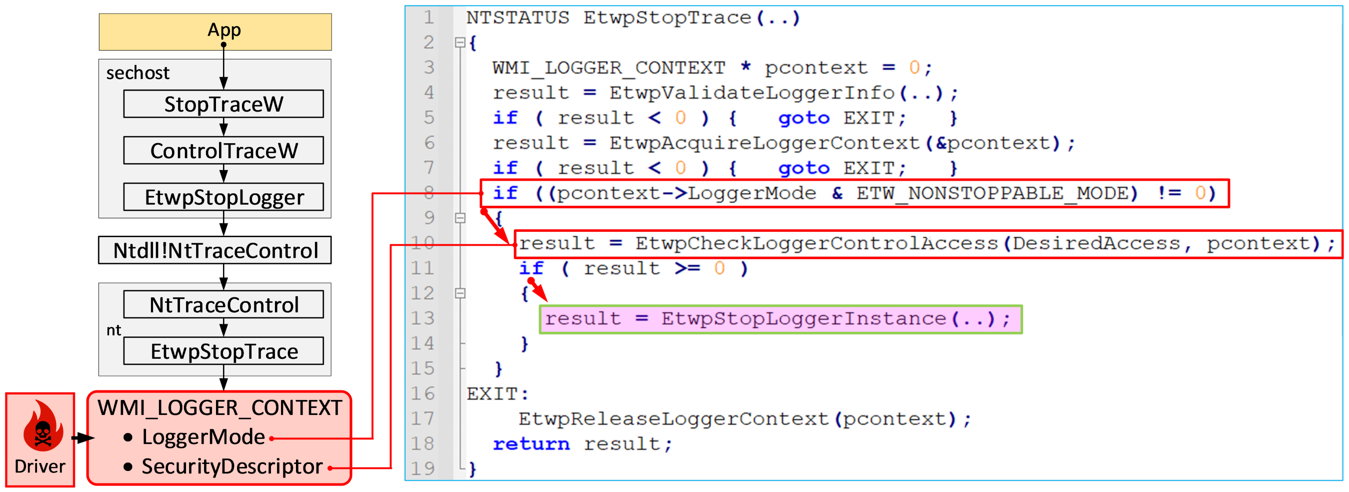


```
NTSTATUS EtwpStopTrace(..)
{
  WMI_LOGGER_CONTEXT * pcontext = 0;
  result = EtwpValidateLoggerInfo(..);
  if ( result < 0 ) {   goto EXIT;   }
  result = EtwpAcquireLoggerContext(&pcontext);
  if ( result < 0 ) {   goto EXIT;   }
  if ((pcontext->LoggerMode & ETW_NONSTOPPABLE_MODE) != 0)
  {
    result = EtwpCheckLoggerControlAccess(DesiredAccess, pcontext);
    if ( result >= 0 )
    {
      result = EtwpStopLoggerInstance(..);
    }
  }
EXIT:
    EtwpReleaseLoggerContext(pcontext);
  return result;
}
```

EtwpStopTrace function checks LoggerMode and Security Descriptor

By default, users cannot stop the Defender ETW sessions since they are running with high privilege and have been marked as non-stoppable. Both session parameters, the security descriptor and LoggerMode field, are stored in the WMI_LOGGER_CONTEXT structure.

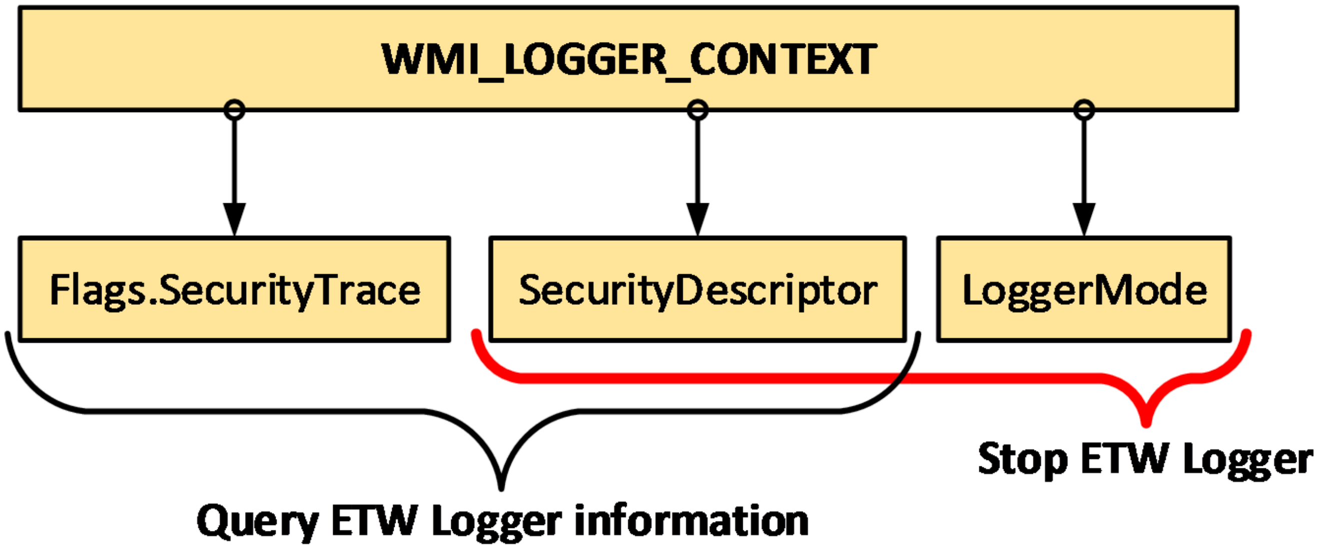
Malware can load a driver that patches the aforementioned values in the targeted WMI_LOGGER_CONTEXT structure to make the execution flow bypass the security checks and execute EtwpStopLoggerInstance function.



```
NTSTATUS EtwpStopTrace(..)
{
  WMI_LOGGER_CONTEXT * pcontext = 0;
  result = EtwpValidateLoggerInfo(..);
  if ( result < 0 ) {   goto EXIT;   }
  result = EtwpAcquireLoggerContext(&pcontext);
  if ( result < 0 ) {   goto EXIT;   }
  if ((pcontext->LoggerMode & ETW_NONSTOPPABLE_MODE) != 0)
  {
    result = EtwpCheckLoggerControlAccess(DesiredAccess, pcontext);
    if ( result >= 0 )
    {
      result = EtwpStopLoggerInstance(..);
    }
  }
EXIT:
    EtwpReleaseLoggerContext(pcontext);
  return result;
}
```

Malware can patch the WMI_LOGGER_CONTEXT structure to allow stopping Defender ETW sessions

To summarize the attack to query information about the target secure ETW session and then stop it, a malware driver has to patch three fields in the corresponding WMI_LOGGER_CONTEXT structure in memory.

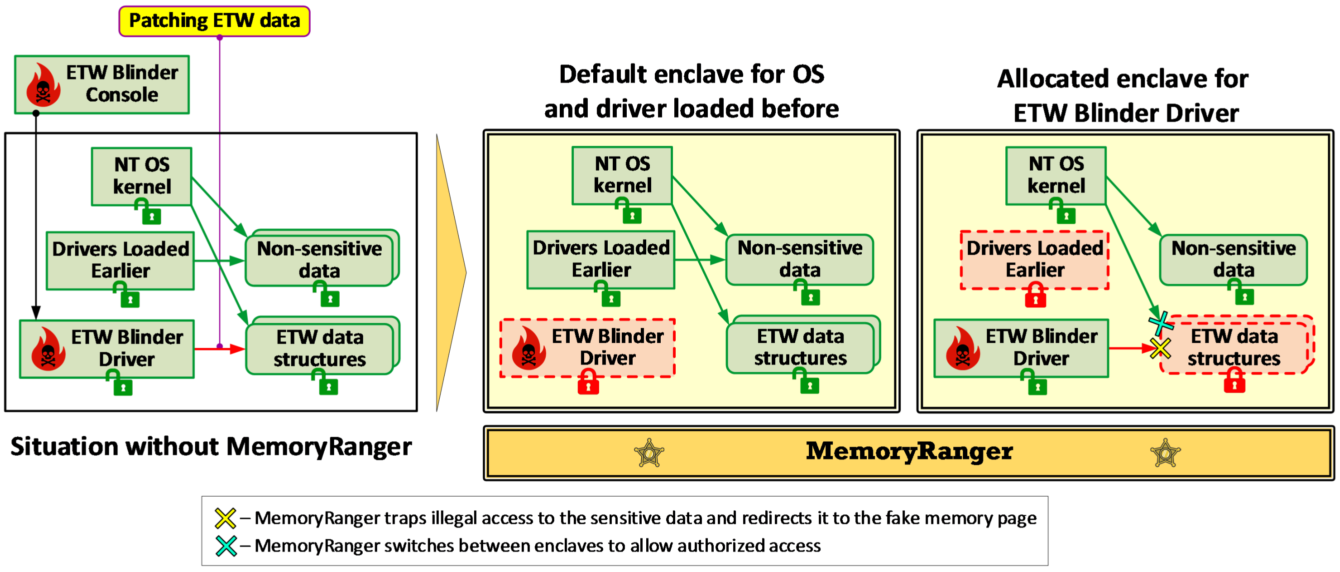


Summary of the Attack

The demo on querying and stopping the Windows Defender ETW sessions presented during the talk can be found on Binarly YouTube channel.

Windows PatchGuard is a software protection utility designed to forbid the kernel from being patched in order to prevent rootkit infections. However, we can see from the demo that PatchGuard does not protect kernel ETW structures from illegal write access.

To mitigate this risk, MemoryRanger can be used. Memory Ranger is a hypervisor-based utility designed to prevent attacks on kernel memory. After loading, MemoryRanger allocates a default enclave for the OS and previously loaded drivers. MemoryRanger traps the loading of the ETW Blinder driver and moves it to the isolated kernel enclave in run-time with different memory access restrictions. Using Extended Page Table (EPT), MemoryRanger can trap access to the sensitive data and return fake data to the attacker.


Patching ETW data
ETW Blinder Console
NT OS kernel
Drivers Loaded Earlier
Non-sensitive data
ETW Blinder Driver
ETW data structures
Situation without MemoryRanger
Default enclave for OS and driver loaded before
Allocated enclave for ETW Blinder Driver
MemoryRanger
– MemoryRanger traps illegal access to the sensitive data and redirects it to the fake memory page
– MemoryRanger switches between enclaves to allow authorized access

MemoryRanger can prevent patching ETW session structures

The demo on how MemoryRanger can prevent patching the WMI_LOGGER_CONTEXT structure presented at BlackHat Europe 2021 can be found on Binarly YouTube channel.

In conclusion, ETW was originally designed to perform software diagnostics, but nowadays it is widely used by various EDRs and cybersecurity solutions. It is crucial to understand the attacks on ETW because these attacks can disable the whole class of security solutions. During the talk we presented two new attacks on Process Monitor and Windows Defender and introduced two new tools that can help in identifying (EtwCheck) and preventing attacks on ETW (MemoryRanger).

Going on step further, let's assume that an attack originates in the firmware, sets persistence, executes the next-stage payload to move up in the kernel where it can implement one of the attacks shown in this presentation. This will be devastating, due to its stealth-ness and resilience to OS reinstallation or HDD replacing. That's why, today, security solutions MUST receive signals from below and above the OS to be able to respond effectively to such threats.