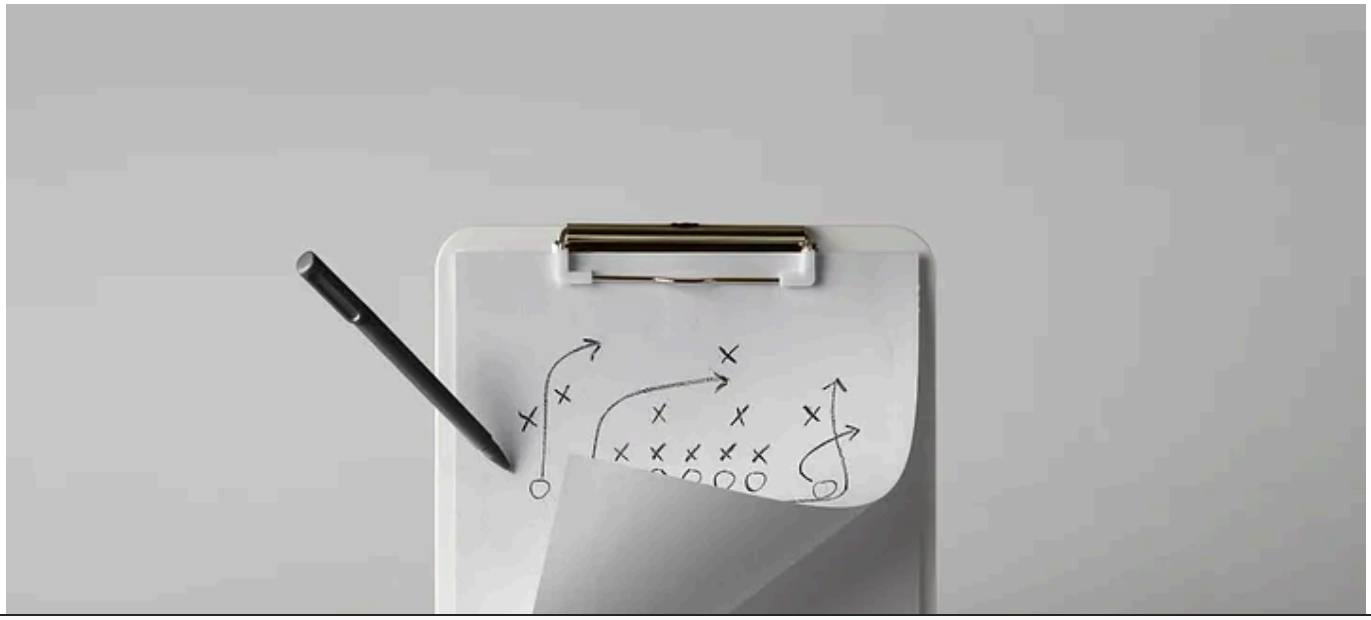
Tampering with Windows Event Tracing: Background, Offense, and Defense



with event log tampering tradecraft is foundational to our success. We continually evaluate our assumptions regarding the integrity of our event data sources, document our blind spots, and adjust our implementation. The goal of this blog post is to share our knowledge with the community by covering ETW background and basics, stealthy event log tampering techniques, and detection strategies.

Introduction to ETW and event logging

The ETW architecture differentiates between event *providers*, event

```
UserNotPresentTraceSession        Trace   Running
Diagtrack-Listener                Trace   Running
MSDTC_TRACE_SESSION               Trace   Running
WindowsUpdate_trace_log           Trace   Running
```

List all providers that a trace session is subscribed to

```
> logman query "EventLog-Application" -ets

Name:                 EventLog-Application
Status:               Running
Root Path:            %systemdrive%\PerfLogs\Admin
```

```
Properties:             65
Filter Type:            0

...

Provider:
Name:                   Microsoft-Windows-PowerShell
Provider Guid:          {A0C1853B-5C40-4B15-8766-3CF1C58F985A}
Level:                  255
KeywordsAll:            0x0
KeywordsAny:            0x9000000000000000 (Microsoft-Windows-
PowerShell/Operational,Microsoft-Windows-PowerShell/Admin)
Properties:             65
Filter Type:            0
```

keywords allow filtering by event category. A keyword corresponds to a specific bit value. *All* indicates that, for a given keyword matched by `KeywordsAny`, further filtering should be performed based on the specific bitmask in `KeywordsAll`. This field is often set to zero. More information on *All* vs. *Any* can be found here.

- **KeywordsAny**: Enables filtering based on any combination of the keywords specified. This can be thought of as a logical OR where KeywordsAll is a subsequent application of a logical AND. The low 6 bytes refer to keywords specific to the provider. The high two bytes are

```
0x040 - EVENT_ENABLE_PROPERTY_ENABLE_KEYWORD_0
0x080 - EVENT_ENABLE_PROPERTY_PROCESS_START_KEY
0x100 - EVENT_ENABLE_PROPERTY_EVENT_KEY
0x200 - EVENT_ENABLE_PROPERTY_EXCLUDE_INPRIVATE
```

From a detection perspective, EVENT_ENABLE_PROPERTY_SID, EVENT_ENABLE_PROPERTY_TS_ID, EVENT_ENABLE_PROPERTY_PROCESS_START_KEY are valuable fields to collect. For example, EVENT_ENABLE_PROPERTY_PROCESS_START_KEY generates a value that uniquely identifies a process. Note that Process IDs

Enumerating all registered ETW providers

The `logman query providers` command lists all registered ETW providers, supplying their name and GUID. An ETW provider is registered if it has a binary manifest stored in the `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers\{PROVIDER_GUID}` registry key. For example, the `Microsoft-Windows-PowerShell` provider has the following registry values:

Notably, the PowerShell provider appears to support logging to the event log based on the existence of the reserved keywords in the high nibble of the defined keywords. Not all ETW providers are designed to be ingested into the event log; rather, many ETW providers are intended to be used solely for low-level tracing, debugging, and more recently-developed security telemetry purposes. For example, Windows Defender Advanced Threat Protection relies heavily upon ETW as a supplemental detection data source.

Viewing all providers that a specific process is sending events to

Another method for discovering potentially interesting providers is to view

- 05F95EFE-7F75–49C7-A994–60A55CC09571
  Microsoft.Windows.Kernel.KernelBase
- 072665FB-8953–5A85–931D-D06AEAB3D109
  Microsoft.Windows.ProcessLifetimeManage
- 7AF898D7–7E0E-518D-5F96-B1E79239484C
  Microsoft.Windows.Defender

Event provider internals

- Microsoft-Windows-PowerShell provider GUID: `{A0C1853B-5C40-4b15-8766-3CF1C58F985A}`
- Event ID: `PSEventId.ScriptBlock_Compile_Detail - 4104`
- Channel value: `PSChannel.Operational - 16`
  Again, the usage of a channel value indicates that the provider is intended to be used with the event log. The operational channel definition for the PowerShell ETW manifest can be seen here. When an explicit channel value is not supplied, Message Compiler (`mc.exe`) will assign a default value starting at 16. Since the operational channel was

The following properties should be noted:

- Operational channel events (as indicated by `0x8000000000000000` in the MatchAnyKeyword value) are captured.
- All logging levels are captured.
- Events should be captured even if an event keyword value is zero as indicated by the `EVENT_ENABLE_PROPERTY_ENABLE_KEYWORD_0` flag.

This information on its own does not explain why AMSI events are not

The `EVENT_DESCRIPTOR` context gives us the relevant information:

- Event ID: `1101 (0x44D)`
  This events details can be extracted from a recovered manifest as seen here.
- Channel: `16 (0x10)` referring to the operational event log channel
- Level: 4 (Informational)
- Keyword: `0x8000000000000001` (AMSI/Operational OR Event1). These

After running the above command, reboot, and the AMSI event log will begin to populate.

AMSI event became to be misconfigured and whether or not the misconfiguration was intentional.

ETW tampering techniques

If the goal of an attacker is to subvert event logging, ETW provides a stealthy mechanism to affect logging without itself generating an event log trail. Below is a non-exhaustive list of tampering techniques that an attacker can use to cut off the supply of events to a specific event log.

configured autologger. Removing a provider registration from an autologger will cause events to cease to flow to the respective trace session.

Example: The following PowerShell code disables Microsoft-Windows-PowerShell event logging:

```
Remove-EtwTraceProvider -AutologgerName EventLog-Application -Guid
'{A0C1853B-5C40-4B15-8766-3CF1C58F985A}'
```

value is set to 0. An attacker could swap out `EVENT_ENABLE_PROPERTY_ENABLE_KEYWORD_0` for `EVENT_ENABLE_PROPERTY_IGNORE_KEYWORD_0`, resulting in a value of `0x11`, which would result in all events where the keyword is 0 to not be logged. For example, PowerShell eventing supplies a 0 keyword value with its events, resulting in no logging to the PowerShell event log.

Example: The following PowerShell code disables Microsoft-Windows-PowerShell event logging:

`HKLM\SYSTEM\CurrentControlSet\Control\WMI\Autologger\AUTOLOGGER_NAME\{PROVIDER_GUID}` registry key. Note that modifying `EnableProperty` is just one specific example and that an attacker can alter ETW providers in other ways, too.

ETW provider removal from a trace session

Tampering category: Ephemeral
Minimum permissions required: SYSTEM
Detection artifacts: Unfortunately, no file, registry, or event log artifacts are associated with this event. While the technique example below indicates that

Alternative detection artifacts/ideas:

- Event ID 12 within the Microsoft-Windows-Kernel-EventTracing/Analytic log indicates when a trace session is modified, but it doesn't supply the provider name or GUID that was removed, so it would be difficult to confidently determine whether or not something suspicious occurred using this event.
- There have been several references thus far to the ETW PowerShell cmdlets housed in the `EventTracingManagement` module, which itself is a

- Use this not this: Logging / Event Tracing
- Writing an Instrumentation Manifest
- Event Tracing Functions
- Configuring and Starting an AutoLogger Session
- Event Tracing
- TraceLogging