

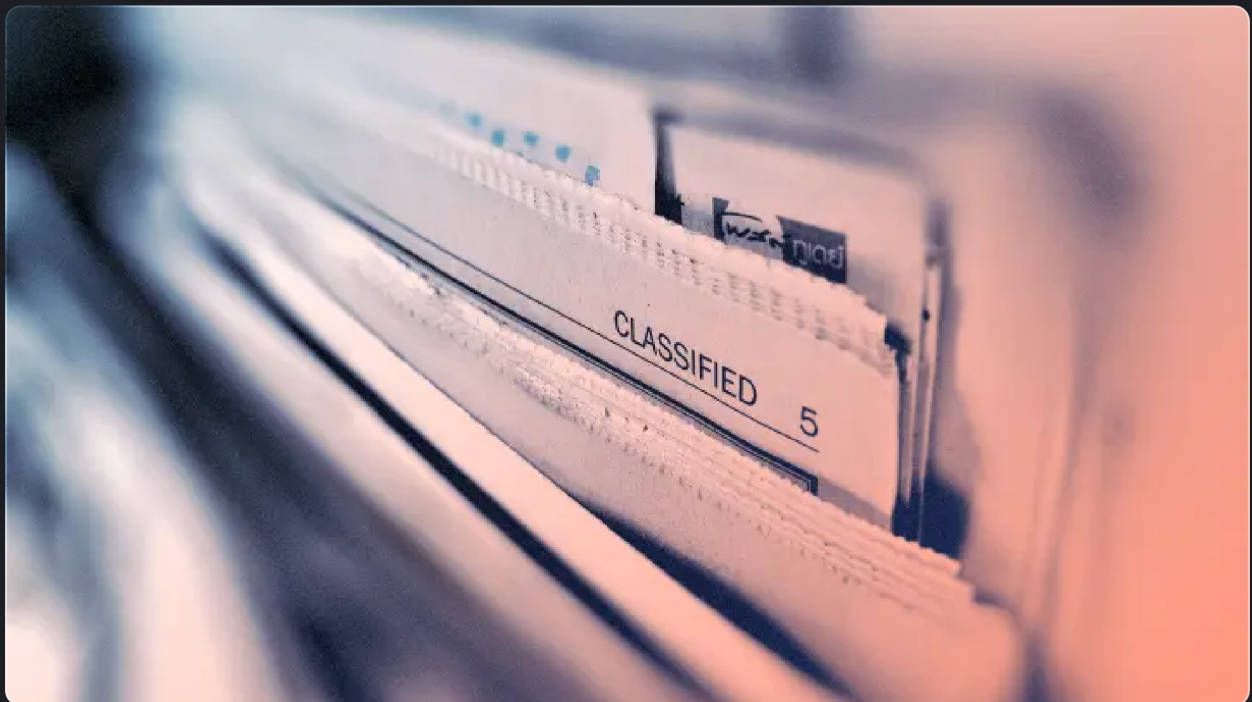
9 JANUARY 2024

JOHN UHLMANN • SAMIR BOUSSEADEN

Doubling Down: Detecting In-Memory Threats with Kernel ETW Call Stacks

With Elastic Security 8.11, we added further kernel telemetry call stack-based detections to increase efficacy against in-memory threats.

🕒 9 min read 🔖 Security research



Introduction

We were pleased to see that the [kernel call stack](#) capability we released in 8.8 was met with

extremely positive community feedback - both from the offensive research teams attempting to evade us and the defensive teams triaging alerts faster due to the additional context.

But this was only the first step: We needed to arm defenders with even more visibility from the kernel - the most reliable mechanism to combat user-mode threats. With the introduction of Kernel Patch Protection in x64 Windows, Microsoft created a shared responsibility model where security vendors are now limited to only the kernel visibility and extension points that Microsoft provides. The most notable addition to this visibility is the Microsoft-Windows-Threat-Intelligence Event Tracing for Windows(ETW) provider.

Microsoft has identified a handful of highly security-relevant syscalls and provided security vendors with near real-time telemetry of those. While we would strongly prefer inline callbacks that allow synchronous blocking of malicious activity, Microsoft has implicitly not deemed this a necessary security use case yet. Currently, the only filtering mechanism afforded to security vendors for these syscalls is user-mode hooking - and that approach is inherently fragile. At Elastic, we determined that a more robust detection approach based on kernel telemetry collected through ETW would provide greater security benefits than easily bypassed user-mode hooks. That said, kernel ETW does have some systemic issues that we have logged with Microsoft, along with suggested mitigations.

Implementation

Endpoint telemetry is a careful balance between completeness and cost. Vendors don't want to balloon your SIEM storage costs unnecessarily, but they also don't want you to miss the critical indicator of compromise. To reduce event volumes for these new API events, we fingerprint each event and only emit it if it is unique. This deduplication ensures a minimal impact on detection fidelity.

However, this approach proved insufficient in reducing API event volumes to manageable levels in all environments. Any further global reduction of event volumes we introduced would be a blindspot for our customers. Instead of potentially impairing detection visibility in this fashion, we determined that these highly verbose events would be processed for detections on the host but would not be streamed to the SIEM by default. This approach reduces storage costs for most of our users while also empowering any customer SOCs that want the full fidelity of those events to opt into streaming via an advanced option available

in Endpoint policy and implement filtering tailored to their specific environments.

Currently, we propagate visibility into the following APIs -

- `VirtualAlloc`
- `VirtualProtect`
- `MapViewOfFile`
- `VirtualAllocEx`
- `VirtualProtectEx`
- `MapViewOfFile2`
- `QueueUserAPC` [call stacks not always available due to ETW limitations]
- `SetThreadContext` [call stacks planned for 8.12]
- `WriteProcessMemory`
- `ReadProcessMemory` (lsass) [planned for 8.12]

In addition to call stack information, our API events are also enriched with several behaviors:

API event	Description
<code>cross-process</code>	The observed activity was between two processes.
<code>native_api</code>	A call was made directly to the undocumented Native API rather than the supported Win32 API.
<code>direct_syscall</code>	A syscall instruction originated outside of the Native API layer.
<code>proxy_call</code>	The call stack appears to show a proxied API call to masking the true caller.
<code>sensitive_api</code>	Executable non-image memory is unexpectedly calling a sensitive API.
<code>shellcode</code>	Suspicious executable non-image memory is calling a sensitive API.
<code>image-hooked</code>	An entry in the call stack appears to have been hooked.

API event	Description
<code>image_indirect_call</code>	An entry in the call stack was preceded by a call to a dynamically resolved function.
<code>image_rop</code>	An entry in the call stack was not preceded by a call instruction.
<code>image_rwx</code>	An entry in the call stack is writable.
<code>unbacked_rwx</code>	An entry in the call stack is non-image and writable.
<code>allocate_shellcode</code>	A region of non-image executable memory suspiciously allocated more executable memory.
<code>execute_fluctuation</code>	The PAGE_EXECUTE protection is unexpectedly fluctuating.
<code>write_fluctuation</code>	The PAGE_WRITE protection of executable memory is unexpectedly fluctuating.
<code>hook_api</code>	A change to the memory protection of a small executable image memory region was made.
<code>hollow_image</code>	A change to the memory protection of a large executable image memory region was made.
<code>hook_unbacked</code>	A change to the memory protection of a small executable non-image memory was made.
<code>hollow_unbacked</code>	A change to the memory protection of a large executable non-image memory was made.
<code>guarded_code</code>	Executable memory was unexpectedly marked as PAGE_GUARD.
<code>hidden_code</code>	Executable memory was unexpectedly marked as PAGE_NOACCESS.
<code>execute_shellcode</code>	A region of non-image executable memory was executed in an unexpected fashion.
<code>hardware_breakpoint_set</code>	A hardware breakpoint was potentially set.

New Rules

In 8.11, Elastic Defend's behavior protection comes with many new rules against various popular malware techniques, such as shellcode fluctuation, threadless injection, direct syscalls, indirect calls, and AMSI or ETW patching.

These rules include:

Windows API Call via Direct Syscall

Identifies the call of commonly abused Windows APIs to perform code injection and where the call stack is not starting with NTDLL:

```
api where event.category == "intrusion_detection" and  
  
process.Ext.api.behaviors == "direct_syscall" and  
  
process.Ext.api.name : ("VirtualAlloc*", "VirtualProtect*",  
                        "MapViewOfFile*", "WriteProcessMemory")
```

VirtualProtect via Random Indirect Syscall

Identifies calls to the VirtualProtect API and where the call stack is not originating from its equivalent NT syscall NtProtectVirtualMemory:

```
api where  
  
process.Ext.api.name : "VirtualProtect*" and  
  
not _arraysearch(process.thread.Ext.call_stack, $entry, $entry.symbol_info:
```

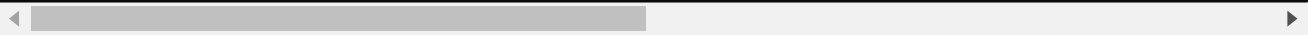


Image Hollow from Unbacked Memory

```
api where process.Ext.api.behaviors == "hollow_image" and  
  
process.Ext.api.name : "VirtualProtect*" and  
  
process.Ext.api.summary : "*.dll*" and
```

```
process.Ext.api.parameters.size >= 10000 and process.executable != null and  
process.thread.Ext.call_stack_summary : "*Unbacked*"
```

Below example of matches on `wwanmm.dll` module stomping to replace it's memory content with a malicious payload:

AMSI and WLDAP Memory Patching

Identifies attempts to modify the permissions or write to Microsoft Antimalware Scan Interface or the Windows Lock Down Policy related DLLs from memory to modify its behavior for evading malicious content checks:

```
api where  
  
(  
  (process.Ext.api.name : "VirtualProtect*" and  
    process.Ext.api.parameters.protection : "*W*") or  
  
  process.Ext.api.name : "WriteProcessMemory*"  
  ) and  
  
process.Ext.api.summary : ("* amsi.dll*", "* mpoav.dll*", "* wldp.dll*")
```

Evasion via Event Tracing for Windows Patching

Identifies attempts to patch the Microsoft Event Tracing for Windows via memory modification:

```
api where process.Ext.api.name : "WriteProcessMemory*" and  
  
process.Ext.api.summary : ("*ntdll.dll!Etw*", "*ntdll.dll!NtTrace*") and  
  
not process.executable : ("?:\\Windows\\System32\\lsass.exe", "\\Device\\Harddisk\\")
```

Windows System Module Remote Hooking

Identifies attempts to write to a remote process memory to modify NTDLL or Kernelbase modules as a preparation step for stealthy code injection:

```
api where process.Ext.api.name : "WriteProcessMemory" and  
  
process.Ext.api.behaviors == "cross-process" and  
  
process.Ext.api.summary : ("*ntdll.dll*", "*kernelbase.dll*")
```

Below is an example of matches on [ThreadLessInject](#), a new process injection technique that involves hooking an export function from a remote process to gain shellcode execution (avoiding the creation of a remote thread):

Conclusion

Until Microsoft provides vendors with kernel callbacks for security-relevant syscalls, Threat-Intelligence ETW will remain the most robust visibility into in-memory threats on Windows. At Elastic, we're committed to putting that visibility to work for customers and optionally directly into their hands without any hidden filtering assumptions.

[Stay tuned](#) for the call stack features in upcoming releases of Elastic Security.

Resources

Rules released with 8.11:

- [AMSI or WLDP Bypass via Memory Patching](#)
- [Call Stack Spoofing via Synthetic Frames](#)
- [Evasion via Event Tracing for Windows Patching](#)
- [Memory Protection Modification of an Unsigned DLL](#)
- [Network Activity from a Stomped Module](#)
- [Potential Evasion via Invalid Code Signature](#)
- [Potential Injection via an Exception Handler](#)
- [Potential Injection via Asynchronous Procedure Call](#)

- [Potential Thread Call Stack Spoofing](#)
- [Remote Process Injection via Mapping](#)
- [Remote Process Manipulation by Suspicious Process](#)
- [Remote Thread Context Manipulation](#)
- [Suspicious Activity from a Control Panel Applet](#)
- [Suspicious API Call from a Script Interpreter](#)
- [Suspicious API from an Unsigned Service DLL](#)
- [Suspicious Call Stack Trailing Bytes](#)
- [Suspicious Executable Heap Allocation](#)
- [Suspicious Executable Memory Permission Modification](#)
- [Suspicious Memory Protection Fluctuation](#)
- [Suspicious Memory Write to a Remote Process](#)
- [Suspicious NTDLL Memory Write](#)
- [Suspicious Null Terminated Call Stack](#)
- [Suspicious Kernel32 Memory Protection](#)
- [Suspicious Remote Memory Allocation](#)
- [Suspicious Windows API Call from Virtual Disk or USB](#)
- [Suspicious Windows API Call via Direct Syscall](#)
- [Suspicious Windows API Call via ROP Gadgets](#)
- [Suspicious Windows API Proxy Call](#)
- [VirtualProtect API Call from an Unsigned DLL](#)
- [VirtualProtect Call via NtTestAlert](#)
- [VirtualProtect via Indirect Random Syscall](#)
- [VirtualProtect via ROP Gadgets](#)
- [Windows API via a Callback Function](#)
- [Windows System Module Remote Hooking](#)



Twitter



Facebook



LinkedIn



Reddit

[Sitemap](#)



[Elastic.co](#)



[@elasticseclabs](#)

© 2025. Elasticsearch B.V. All Rights Reserved.