Peeling back the curtain with call stacks

elastic.co/security-labs/peeling-back-the-curtain-with-call-stacks

Subscribe

12 September 2023•Samir Bousseaden

In this article, we'll show you how we contextualize rules and events, and how you can leverage call stacks to better understand any alerts you encounter in your environment.

13 min read Security operations, Security research, Detection science

Introduction

Elastic Defend provides over 550 rules (and counting) to detect and stop malicious behavior in real time on endpoints. We recently added kernel call stack enrichments to provide additional context to events and alerts. Call stacks are a win-win-win for behavioral protections, simultaneously improving false positives, false negatives, and alert explainability. In this article, we'll show you how we achieve all three of these, and how you can leverage call stacks to better understand any alerts you encounter in your environment.

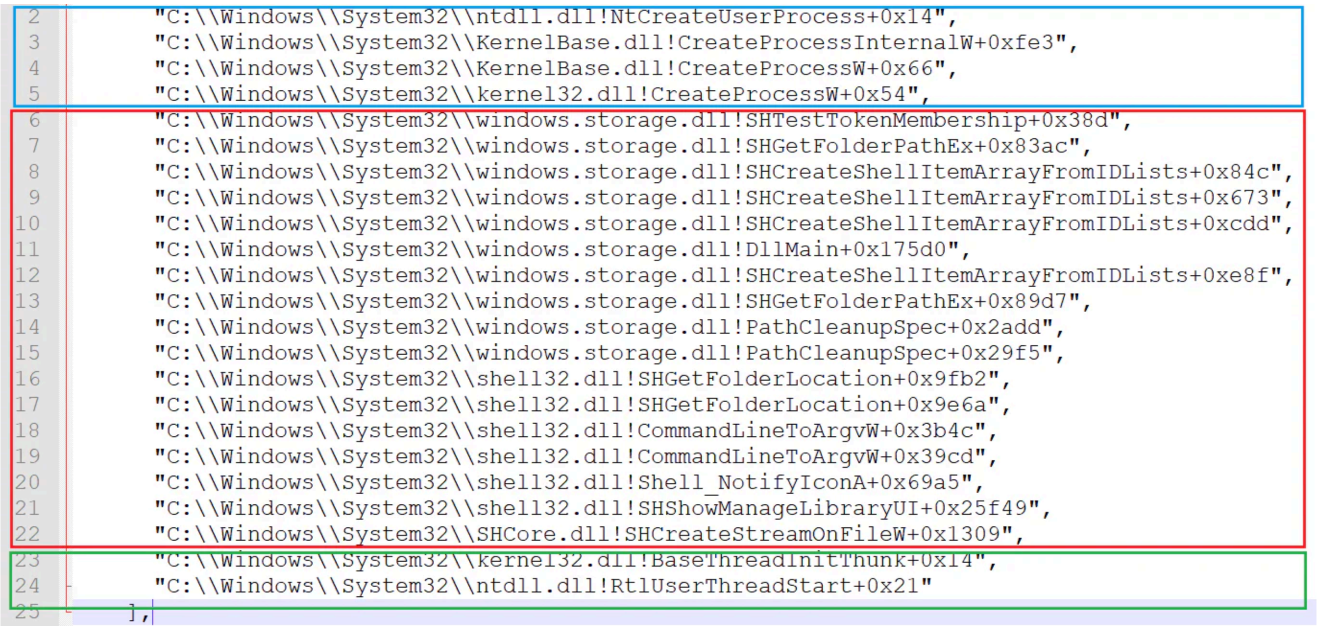
What is a call stack?

When a thread running function A calls function B, the CPU automatically saves the current instruction’s address (within A) to a thread-specific region of memory called the stack. This saved pointer is known as the return address - it's where execution will resume once the B has finished its job. If B were to call a third function C, then a return address within B will also be saved to the stack. These return addresses can be retrieved through a process known as a stack walk, which reconstructs the sequence of function calls that led to the current thread state. Stack walks list return addresses in reverse-chronological order, so the most recent function is always at the top.

In Windows, when we double-click on **notepad.exe**, for example, the following series of functions are called:

- The green section is related to base thread initialization performed by the operating system and is usually identical across all operations (file, registry, process, library, etc.)
- The red section is the user code; it is often composed of multiple modules and provides approximate details of how the process creation operation was reached
- The blue section is the Win32 and Native API layer; this is operation-specific, including the last 2 to 3 intermediary Windows modules before forwarding the operation details for effective execution in kernel mode

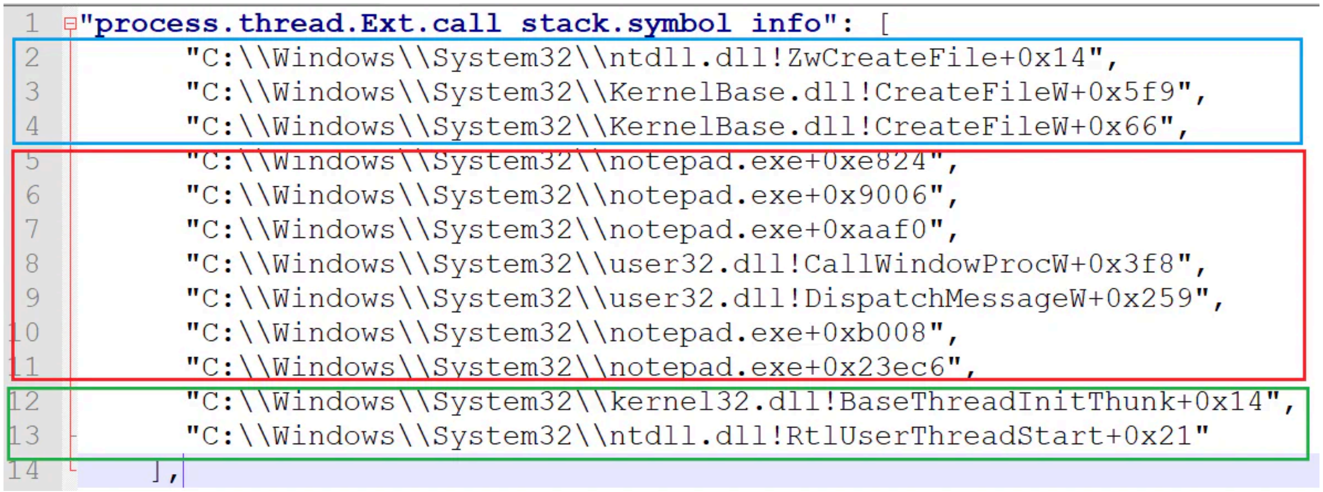
The following screenshot depicts the call stack for this execution chain:



```
        "C:\\Windows\\System32\\ntdll.dll!NtCreateUserProcess+0x14",
        "C:\\Windows\\System32\\KernelBase.dll!CreateProcessInternalW+0xfe3",
        "C:\\Windows\\System32\\KernelBase.dll!CreateProcessW+0x66",
        "C:\\Windows\\System32\\kernel32.dll!CreateProcessW+0x54",
        "C:\\Windows\\System32\\windows.storage.dll!SHTestTokenMembership+0x38d",
        "C:\\Windows\\System32\\windows.storage.dll!SHGetFolderPathEx+0x83ac",
        "C:\\Windows\\System32\\windows.storage.dll!SHCreateShellItemArrayFromIDLists+0x84c",
        "C:\\Windows\\System32\\windows.storage.dll!SHCreateShellItemArrayFromIDLists+0x673",
        "C:\\Windows\\System32\\windows.storage.dll!SHCreateShellItemArrayFromIDLists+0xcdd",
        "C:\\Windows\\System32\\windows.storage.dll!DllMain+0x175d0",
        "C:\\Windows\\System32\\windows.storage.dll!SHCreateShellItemArrayFromIDLists+0xe8f",
        "C:\\Windows\\System32\\windows.storage.dll!SHGetFolderPathEx+0x89d7",
        "C:\\Windows\\System32\\windows.storage.dll!PathCleanupSpec+0x2add",
        "C:\\Windows\\System32\\windows.storage.dll!PathCleanupSpec+0x29f5",
        "C:\\Windows\\System32\\shell32.dll!SHGetFolderLocation+0x9fb2",
        "C:\\Windows\\System32\\shell32.dll!SHGetFolderLocation+0x9e6a",
        "C:\\Windows\\System32\\shell32.dll!CommandLineToArgvW+0x3b4c",
        "C:\\Windows\\System32\\shell32.dll!CommandLineToArgvW+0x39cd",
        "C:\\Windows\\System32\\shell32.dll!Shell_NotifyIconA+0x69a5",
        "C:\\Windows\\System32\\shell32.dll!SHShowManageLibraryUI+0x25f49",
        "C:\\Windows\\System32\\SHCore.dll!SHCreateStreamOnFileW+0x1309",
        "C:\\Windows\\System32\\kernel32.dll!BaseThreadInitThunk+0x14",
        "C:\\Windows\\System32\\ntdll.dll!RtlUserThreadStart+0x21"
    ],
```

Here is an example of file creation using **notepad.exe** where we can see a similar pattern:

- The blue part lists the last user mode intermediary Windows APIs before forwarding the create file operation to kernel mode drivers for effective execution
- The red section includes functions from **user32.dll** and **notepad.exe**, which indicate that this file operation was likely initiated via GUI
- The green part represents the initial thread initialization

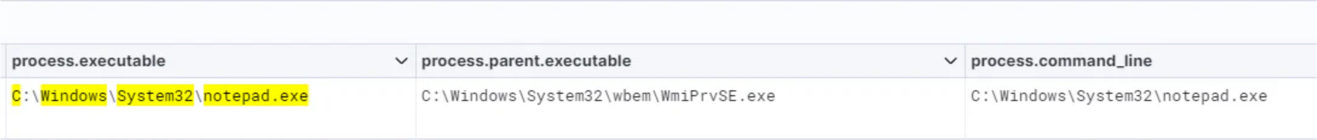


```
"process.thread.Ext.call stack.symbol info": [
        "C:\\Windows\\System32\\ntdll.dll!ZwCreateFile+0x14",
        "C:\\Windows\\System32\\KernelBase.dll!CreateFileW+0x5f9",
        "C:\\Windows\\System32\\KernelBase.dll!CreateFileW+0x66",
        "C:\\Windows\\System32\\notepad.exe+0xe824",
        "C:\\Windows\\System32\\notepad.exe+0x9006",
        "C:\\Windows\\System32\\notepad.exe+0xaaf0",
        "C:\\Windows\\System32\\user32.dll!CallWindowProcW+0x3f8",
        "C:\\Windows\\System32\\user32.dll!DispatchMessageW+0x259",
        "C:\\Windows\\System32\\notepad.exe+0xb008",
        "C:\\Windows\\System32\\notepad.exe+0x23ec6",
        "C:\\Windows\\System32\\kernel32.dll!BaseThreadInitThunk+0x14",
        "C:\\Windows\\System32\\ntdll.dll!RtlUserThreadStart+0x21"
    ],
```

Events Explainability

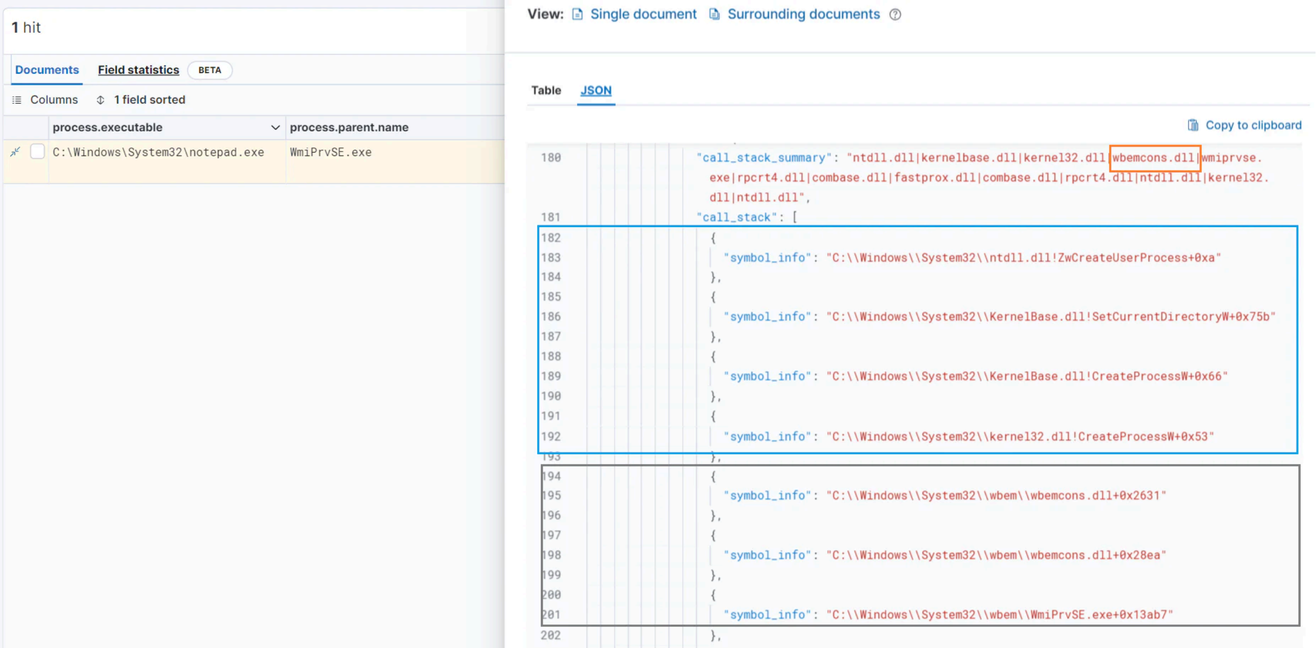
Apart from using call stacks for finding known bad, like unbacked memory regions with RWX permissions that may be the remnants of prior code injection. Call stacks provide very low-level visibility that often reveals greater insights than logs can otherwise provide.

As an example, while hunting for suspicious process executions started by **WmiPrvSe.exe** via WMI, you find this instance of **notepad.exe**:



process.executable	process.parent.executable	process.command_line
C:\Windows\System32\notepad.exe	C:\Windows\System32\wbem\WmiPrvSE.exe	C:\Windows\System32\notepad.exe

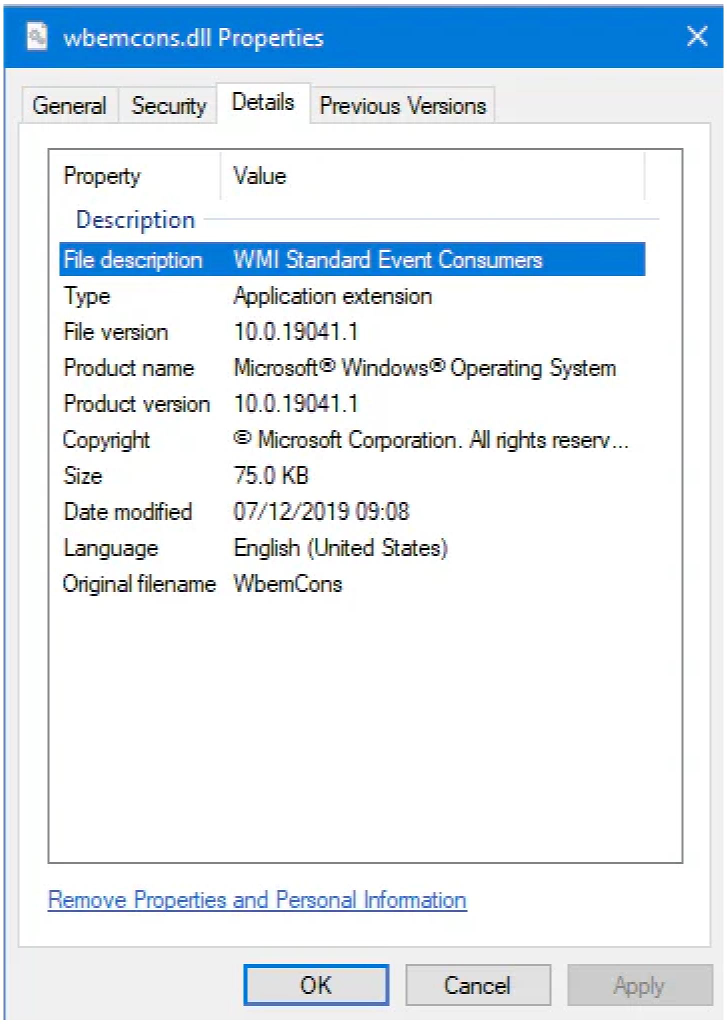
Reviewing the standard event log fields, you may expect that it was started using the Win32_Process class using the **wmic.exe process call create notepad.exe** syntax. However, the event details describe a series of modules and functions:



process.executable	process.parent.name
C:\Windows\System32\notepad.exe	WmiPrvSE.exe

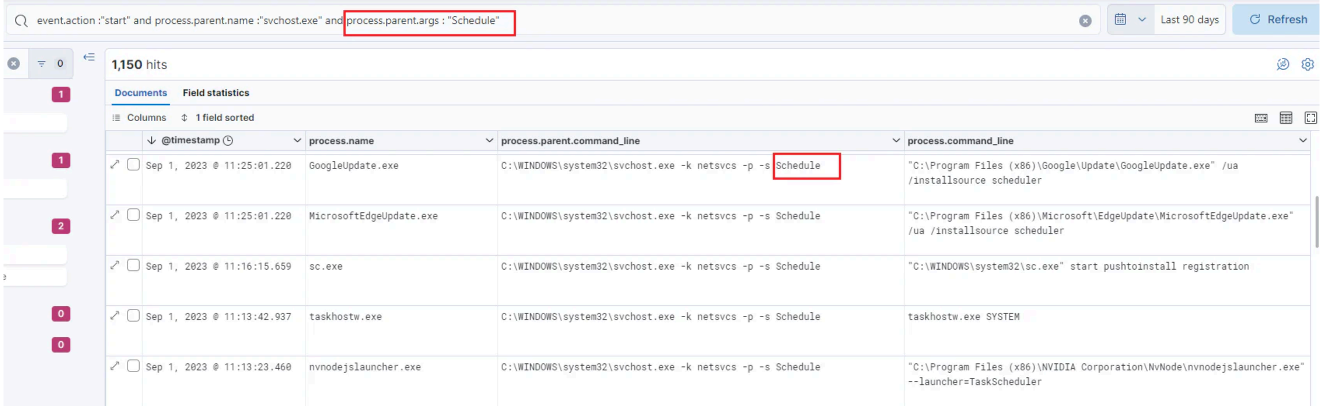
```
"call_stack_summary": "ntdll.dll|kernelbase.dll|kernel32.dll|wbemcons.dll|wmiprvse.
  exe|rpcrt4.dll|combase.dll|fastprox.dll|combase.dll|rpcrt4.dll|ntdll.dll|kernel32.
  dll|ntdll.dll",
"call_stack": [
  {
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!ZwCreateUserProcess+0xa"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\KernelBase.dll!SetCurrentDirectoryW+0x75b"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\KernelBase.dll!CreateProcessW+0x66"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\kernel32.dll!CreateProcessW+0x53"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\wbem\\wbemcons.dll+0x2631"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\wbem\\wbemcons.dll+0x28ea"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\wbem\\WmiPrvSE.exe+0x13ab7"
  },
```

The blue section depicts the standard intermediary **CreateProcess** Windows APIs, while the red section highlights better information in that we can see that the DLL before the first call to **CreateProcessW** is **wbemcons.dll** and when inspecting its properties we can see that it's related to WMI Event Consumers. We can conclude that this **notepad.exe** instance is likely related to a WMI Event Subscription. This will require specific incident response steps to mitigate the WMI persistence mechanism.



Property	Value
Description	
File description	WMI Standard Event Consumers
Type	Application extension
File version	10.0.19041.1
Product name	Microsoft® Windows® Operating System
Product version	10.0.19041.1
Copyright	© Microsoft Corporation. All rights reserv...
Size	75.0 KB
Date modified	07/12/2019 09:08
Language	English (United States)
Original filename	WbemCons

Another great example is Windows scheduled tasks. When executed, they are spawned as children of the Schedule service, which runs within a **svchost.exe** host process. Modern Windows 11 machines may have 50 or more **svchost.exe** processes running. Fortunately, the Schedule service has a specific process argument **-s Schedule** which differentiates it:

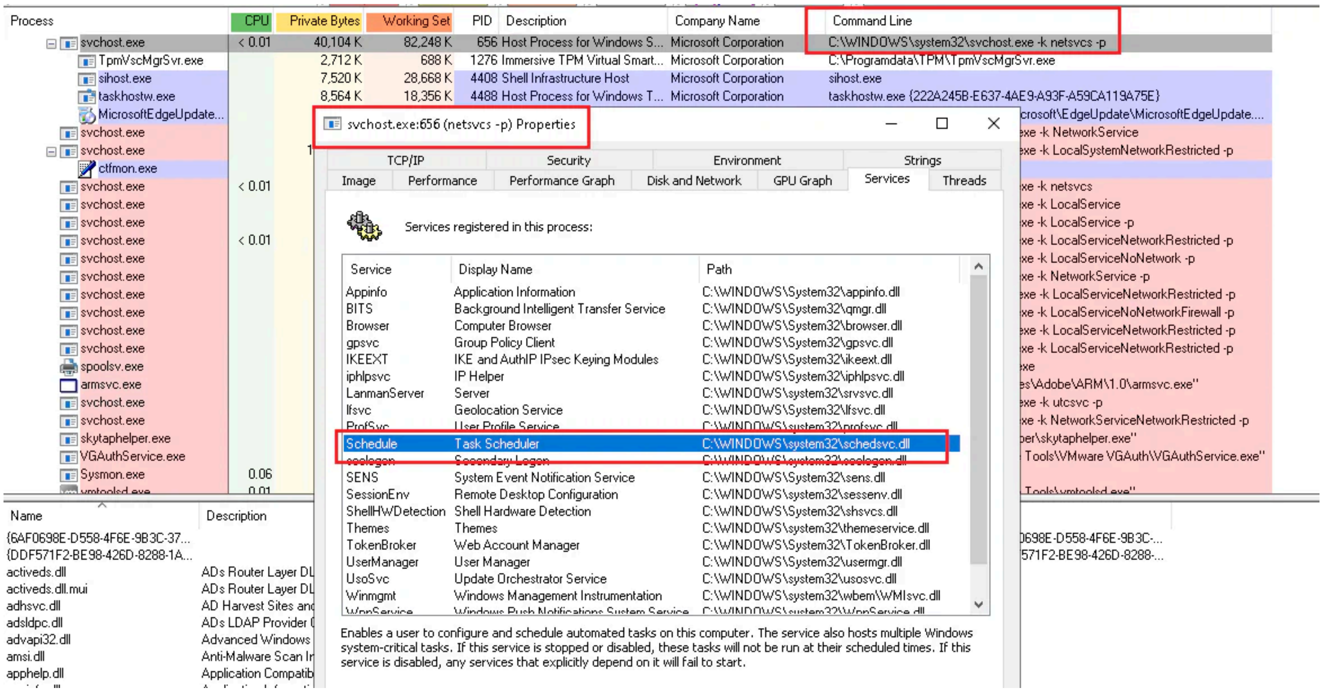


```
event.action :"start" and process.parent.name :"svchost.exe" and process.parent.args : "Schedule"
```

1,150 hits

@timestamp	process.name	process.parent.command_line	process.command_line
Sep 1, 2023 @ 11:25:01.220	GoogleUpdate.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	"C:\Program Files (x86)\Google\Update\GoogleUpdate.exe" /ua /installsource scheduler
Sep 1, 2023 @ 11:25:01.220	MicrosoftEdgeUpdate.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	"C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe" /ua /installsource scheduler
Sep 1, 2023 @ 11:16:15.659	sc.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	"C:\WINDOWS\system32\sc.exe" start pushtoinstall registration
Sep 1, 2023 @ 11:13:42.937	taskhostw.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	taskhostw.exe SYSTEM
Sep 1, 2023 @ 11:13:23.460	nvnodejslauncher.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	"C:\Program Files (x86)\NVIDIA Corporation\NvNode\nvnodejslauncher.exe" --launcher=TaskScheduler

In older Windows versions, the Scheduled Tasks service is a member of the Network Service group and executed as a component of the **netsvcs** shared **svchost.exe** instance. Not all children of this process are necessarily scheduled tasks in these older versions:



Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Command Line
svchost.exe	< 0.01	40,104 K	82,248 K	656	Host Process for Windows S...	Microsoft Corporation	C:\WINDOWS\system32\svchost.exe -k netsvcs -p
TpmVscMgrSvr.exe		2,712 K	688 K	1276	Immersive TPM Virtual Smart...	Microsoft Corporation	C:\Programdata\TPM\TpmVscMgrSvr.exe
sihost.exe		7,520 K	28,668 K	4408	Shell Infrastructure Host	Microsoft Corporation	sihost.exe
taskhostw.exe		8,564 K	18,356 K	4488	Host Process for Windows T...	Microsoft Corporation	taskhostw.exe {222A245B-E637-4AE9-A93F-A59CA119A75E}
MicrosoftEdgeUpdate...							crosoft\EdgeUpdate\MicrosoftEdgeUpdate....
svchost.exe							xe -k NetworkService
svchost.exe							xe -k LocalSystemNetworkRestricted -p
ctfmon.exe							
svchost.exe	< 0.01						xe -k netsvcs
svchost.exe							xe -k LocalService
svchost.exe							xe -k LocalService -p
svchost.exe	< 0.01						xe -k LocalServiceNetworkRestricted -p
svchost.exe							xe -k LocalServiceNoNetwork -p
svchost.exe							xe -k NetworkService -p
svchost.exe							xe -k LocalServiceNetworkRestricted -p
svchost.exe							xe -k LocalServiceNoNetworkFirewall -p
svchost.exe							xe -k LocalServiceNetworkRestricted -p
svchost.exe							xe -k LocalServiceNetworkRestricted -p
spoolsv.exe							xe
armsvc.exe							s\Adobe\ARM\1.0\armsvc.exe"
svchost.exe							xe -k utcsvc -p
svchost.exe							xe -k NetworkServiceNetworkRestricted -p
skytaphelper.exe							er\skytaphelper.exe"
VGAuthService.exe							Tools\VMware VGAuth\VGAuthService.exe"
Sysmon.exe	0.06						

svchost.exe:656 (netsvcs -p) Properties

Services registered in this process:

Service	Display Name	Path
Appinfo	Application Information	C:\WINDOWS\System32\appinfo.dll
BITS	Background Intelligent Transfer Service	C:\WINDOWS\System32\qmgr.dll
Browser	Computer Browser	C:\WINDOWS\System32\browser.dll
gpsvc	Group Policy Client	C:\WINDOWS\System32\gpsvc.dll
IKEEXT	IKE and AuthIP IPsec Keying Modules	C:\WINDOWS\System32\ikeext.dll
iphlpsvc	IP Helper	C:\WINDOWS\System32\iphlpsvc.dll
LanmanServer	Server	C:\WINDOWS\system32\srvsvc.dll
lfsvc	Geolocation Service	C:\WINDOWS\System32\lfsvc.dll
ProfSvc	User Profile Service	C:\WINDOWS\system32\profsvc.dll
Schedule	Task Scheduler	C:\WINDOWS\system32\schedsvc.dll
seclogon	Secondary Logon	C:\WINDOWS\system32\seclogon.dll
SENS	System Event Notification Service	C:\WINDOWS\System32\sens.dll
SessionEnv	Remote Desktop Configuration	C:\WINDOWS\system32\sessenv.dll
ShellHWDetection	Shell Hardware Detection	C:\WINDOWS\System32\shsvcs.dll
Themes	Themes	C:\WINDOWS\system32\themeservice.dll
TokenBroker	Web Account Manager	C:\WINDOWS\System32\TokenBroker.dll
UserManager	User Manager	C:\WINDOWS\System32\usermgr.dll
UsoSvc	Update Orchestrator Service	C:\WINDOWS\system32\usosvc.dll
Winmgmt	Windows Management Instrumentation	C:\WINDOWS\system32\wbem\WMIsvc.dll

Enables a user to configure and schedule automated tasks on this computer. The service also hosts multiple Windows system-critical tasks. If this service is stopped or disabled, these tasks will not be run at their scheduled times. If this service is disabled, any services that explicitly depend on it will fail to start.

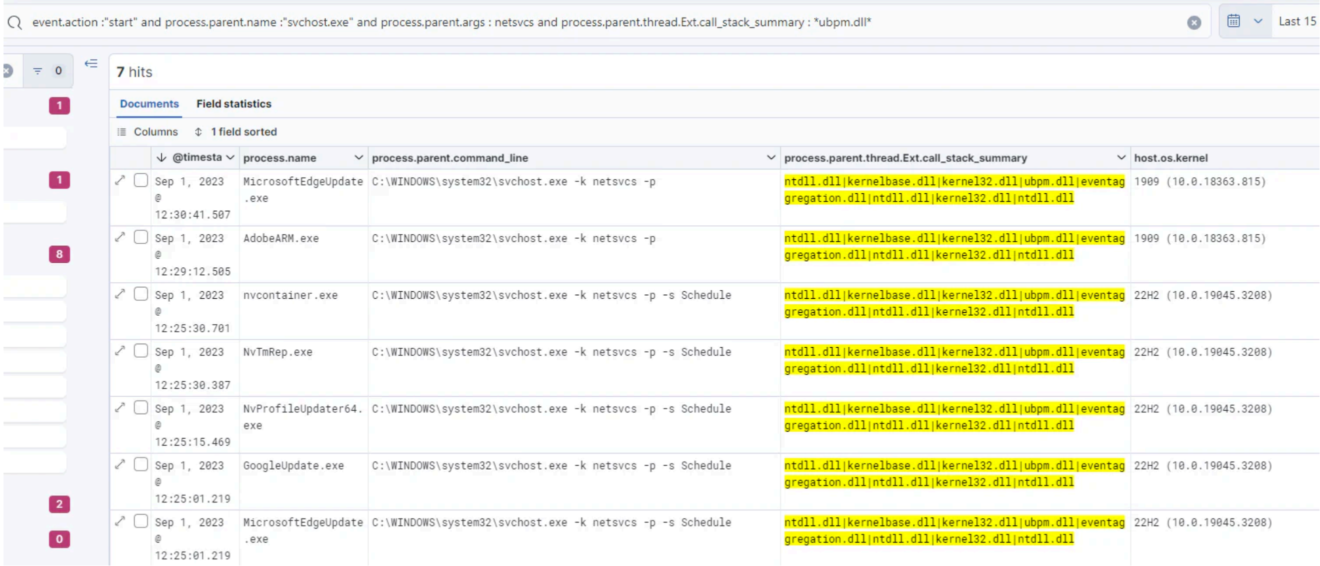
Name	Description
{6AF0698E-D558-4F6E-9B3C-37...	
{DDF571F2-BE98-426D-8288-1A...	
activeds.dll	ADs Router Layer DL
activeds.dll.mui	ADs Router Layer DL
adhsvc.dll	AD Harvest Sites and
adsldpc.dll	ADs LDAP Provider (
advapi32.dll	Advanced Windows
amsi.dll	Anti-Malware Scan Ir
apphelp.dll	Application Compatib

Inspecting the call stack on both versions, we can see the module that is adjacent to the **CreateProcess** call is the same **ubpm.dll** (Unified Background Process Manager DLL) executing the exported function **ubpm.dll!UbpmOpenTriggerConsumer**:

process.name	process.parent.command_line	process.parent.thread.Ext.call_stack_summary
taskhostw.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|eventaggregation.dll|ntdll.dll|kernel32.dll|ntdll.dll
taskhostw.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|ntdll.dll|kernel32.dll|ntdll.dll
taskhostw.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|ntdll.dll|kernel32.dll|ntdll.dll
taskhostw.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|ntdll.dll|kernel32.dll|ntdll.dll

Using the following KQL query, we can hunt for task executions on both versions:

```
event.action :"start" and
process.parent.name :"svchost.exe" and process.parent.args : netsvcs and
process.parent.thread.Ext.call_stack_summary : *ubpm.dll*
```



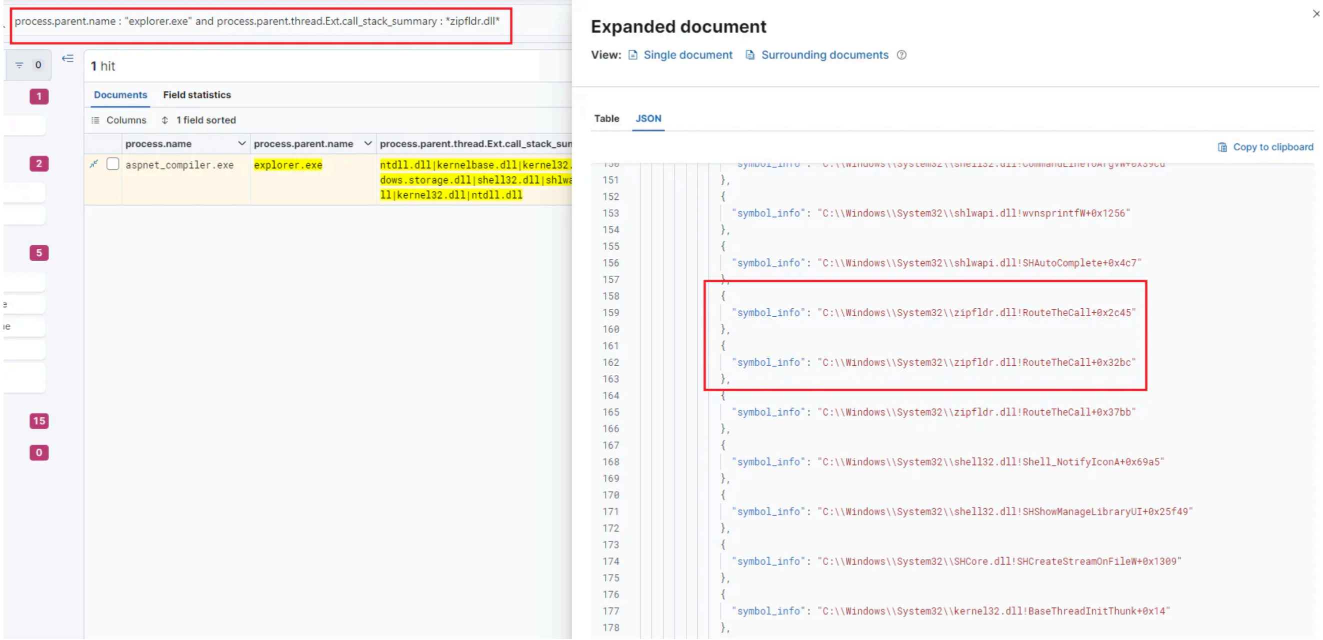
```
event.action :"start" and process.parent.name :"svchost.exe" and process.parent.args : netsvcs and process.parent.thread.Ext.call_stack_summary : *ubpm.dll*
```

7 hits

@timesta	process.name	process.parent.command_line	process.parent.thread.Ext.call_stack_summary	host.os.kernel
Sep 1, 2023 @ 12:30:41.507	MicrosoftEdgeUpdate.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|eventaggregation.dll|ntdll.dll|kernel32.dll|ntdll.dll	1909 (10.0.18363.815)
Sep 1, 2023 @ 12:29:12.505	AdobeARM.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|eventaggregation.dll|ntdll.dll|kernel32.dll|ntdll.dll	1909 (10.0.18363.815)
Sep 1, 2023 @ 12:25:30.701	nvcontainer.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|eventaggregation.dll|ntdll.dll|kernel32.dll|ntdll.dll	22H2 (10.0.19045.3208)
Sep 1, 2023 @ 12:25:30.387	NvTmRep.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|eventaggregation.dll|ntdll.dll|kernel32.dll|ntdll.dll	22H2 (10.0.19045.3208)
Sep 1, 2023 @ 12:25:15.469	NvProfileUpdater64.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|eventaggregation.dll|ntdll.dll|kernel32.dll|ntdll.dll	22H2 (10.0.19045.3208)
Sep 1, 2023 @ 12:25:01.219	GoogleUpdate.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|eventaggregation.dll|ntdll.dll|kernel32.dll|ntdll.dll	22H2 (10.0.19045.3208)
Sep 1, 2023 @ 12:25:01.219	MicrosoftEdgeUpdate.exe	C:\WINDOWS\system32\svchost.exe -k netsvcs -p -s Schedule	ntdll.dll|kernelbase.dll|kernel32.dll|ubpm.dll|eventaggregation.dll|ntdll.dll|kernel32.dll|ntdll.dll	22H2 (10.0.19045.3208)

Another interesting example occurs when a user double-clicks a script file from a ZIP archive that was opened using Windows Explorer. Looking at the process tree, you will see that **explorer.exe** is the parent and the child is a script interpreter process like **wscript.exe** or **cmd.exe**.

This process tree can be confused with a user double-clicking a script file from any location on the file system, which is not very suspicious. But if we inspect the call stack we can see that the parent stack is pointing to **zipfld.dll** (Zipped Folders Shell Extension):



```
process.parent.name : "explorer.exe" and process.parent.thread.Ext.call_stack_summary : *zipfldr.dll*
```

1 hit

process.name	process.parent.name	process.parent.thread.Ext.call_stack_sun
aspnet_compiler.exe	explorer.exe	ntdll.dll|kernelbase.dll|kernel32. dows.storage.dll|shell32.dll|shlwa ll|kernel32.dll|ntdll.dll

```
              },
              {
                "symbol_info": "C:\\Windows\\System32\\shlwapi.dll!wvnsprintfW+0x1256"
              },
              {
                "symbol_info": "C:\\Windows\\System32\\shlwapi.dll!SHAutoComplete+0x4c7"
              },
              {
                "symbol_info": "C:\\Windows\\System32\\zipfldr.dll!RouteTheCall+0x2c45"
              },
              {
                "symbol_info": "C:\\Windows\\System32\\zipfldr.dll!RouteTheCall+0x32bc"
              },
              {
                "symbol_info": "C:\\Windows\\System32\\zipfldr.dll!RouteTheCall+0x37bb"
              },
              {
                "symbol_info": "C:\\Windows\\System32\\shell32.dll!Shell_NotifyIconA+0x69a5"
              },
              {
                "symbol_info": "C:\\Windows\\System32\\shell32.dll!SHShowManageLibraryUI+0x25f49"
              },
              {
                "symbol_info": "C:\\Windows\\System32\\SHCore.dll!SHCreateStreamOnFileW+0x1309"
              },
              {
                "symbol_info": "C:\\Windows\\System32\\kernel32.dll!BaseThreadInitThunk+0x14"
              },
```

Detection Examples

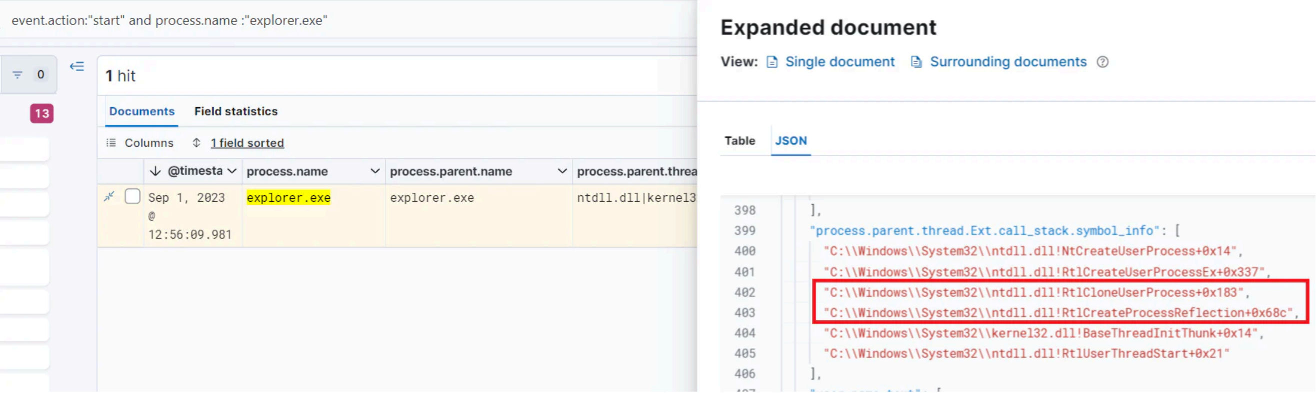
Now that we have a better idea of how to use the call stack to better interpret events, let's explore some advanced detection examples per event type.

Process

Suspicious Process Creation via Reflection

Dirty Vanity is a recent code-injection technique that abuses process forking to execute shellcode within a copy of an existing process. When a process is forked, the OS makes a copy of an existing process, including its address space and any inheritable handles therein.

When executed, Dirty Vanity will fork an instance of a targeted process (already running or a sacrificial one) and then inject into it. Using process creation notification callbacks won’t log forked processes because the forked process initial thread isn’t executed. But in the case of this injection technique, the forked process will be injected and a thread will be started, which triggers the process start event log with the following call stack:



```
event.action:"start" and process.name :"explorer.exe"
```

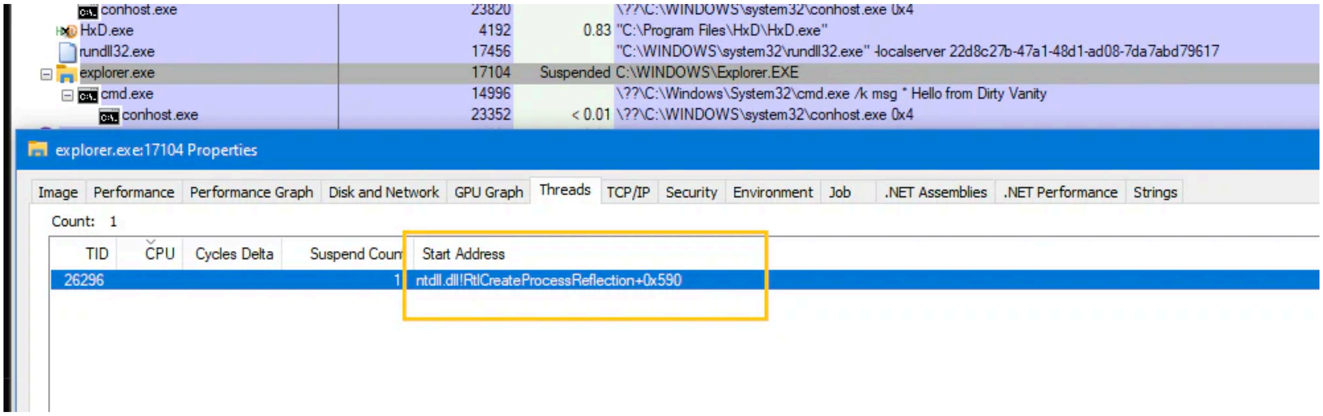
@timesta	process.name	process.parent.name	process.parent.threa
Sep 1, 2023 @ 12:56:09.981	explorer.exe	explorer.exe	ntdll.dll|kernel3

```
],
"process.parent.thread.Ext.call_stack.symbol_info": [
  "C:\\Windows\\System32\\ntdll.dll!NtCreateUserProcess+0x14",
  "C:\\Windows\\System32\\ntdll.dll!RtlCreateUserProcessEx+0x337",
  "C:\\Windows\\System32\\ntdll.dll!RtlCloneUserProcess+0x183",
  "C:\\Windows\\System32\\ntdll.dll!RtlCreateProcessReflection+0x68c",
  "C:\\Windows\\System32\\kernel32.dll!BaseThreadInitThunk+0x14",
  "C:\\Windows\\System32\\ntdll.dll!RtlUserThreadStart+0x21"
],
```

We can see the call to **RtlCreateProcessReflection** and **RtlCloneUserProcess** to fork the process. Now we know that this is a forked process, and the next question is “Is this common in normal conditions?” While diagnostically this behavior appears to be common and alone, it is not a strong signal of something malicious. Checking further to see if the forked processes perform any network connections, loads DLLs, or spawns child processes revealed to be less common and made for good detections:

```
// EQL detecting a forked process spawning a child process - very suspicious

process where event.action == "start" and

descendant of
   [process where event.action == "start" and
   _arraysearch(process.parent.thread.Ext.call_stack, $entry,
   $entry.symbol_info:
    ("*ntdll.dll!RtlCreateProcessReflection*",
    "*ntdll.dll!RtlCloneUserProcess*"))] and

not (process.executable :
      ("?:\\WINDOWS\\SysWOW64\\WerFault.exe",
      "?:\\WINDOWS\\system32\\WerFault.exe") and
     process.parent.thread.Ext.call_stack_summary :
      "*faultrep.dll|wersvc.dl*")

// EQL detecting a forked process loading a network DLL
//  or performs a network connection - very suspicious

sequence by process.entity_id with maxspan=1m
 [process where event.action == "start" and
  _arraysearch(process.parent.thread.Ext.call_stack,
  $entry, $entry.symbol_info:
    ("*ntdll.dll!RtlCreateProcessReflection*",
    "*ntdll.dll!RtlCloneUserProcess*"))]
 [any where
  (
   event.category : ("network", "dns") or
   (event.category == "library" and
    dll.name : ("ws2_32.dll", "winhttp.dll", "wininet.dll"))
  )]
```

Here’s an example of forking **explore.exe** and executing shellcode that spawns **cmd.exe** from the forked **explorer.exe** instance:



conhost.exe	23820		\??\C:\WINDOWS\system32\conhost.exe 0x4
HxD.exe	4192	0.83	"C:\Program Files\HxD\HxD.exe"
rundll32.exe	17456		"C:\WINDOWS\system32\rundll32.exe" -localserver 22d8c27b-47a1-48d1-ad08-7da7abd79617
explorer.exe	17104	Suspended	C:\WINDOWS\Explorer.EXE
cmd.exe	14996		\??\C:\Windows\System32\cmd.exe /k msg * Hello from Dirty Vanity
conhost.exe	23352	< 0.01	\??\C:\WINDOWS\system32\conhost.exe 0x4

Count: 1

TID	CPU	Cycles Delta	Suspend Coun	Start Address
26296			1	ntdll.dll!RtlCreateProcessReflection+0x590

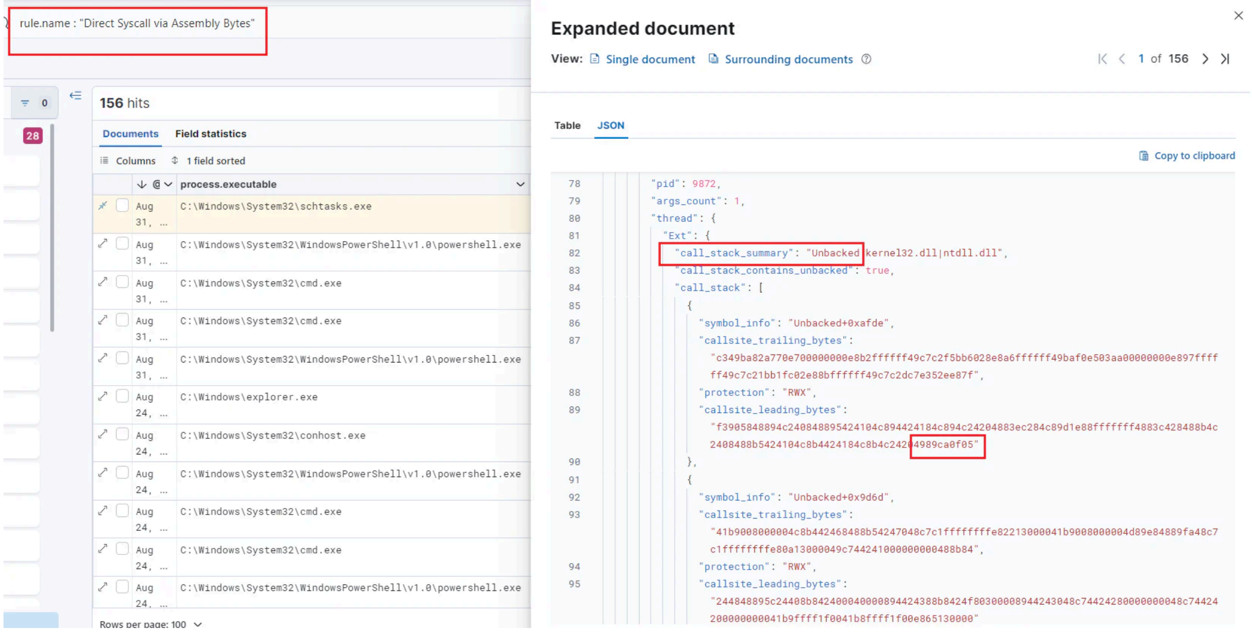
rule.name	process.command_line	process.parent.executable	process.parent.thread.Ext.call_...
Suspicious Process Creation via Reflection	\??\C:\Windows\System32\cmd.exe /k msg * Hello from Dirty Vanity	C:\Windows\explorer.exe	ntdll.dll|Unbacked
Suspicious Process Creation via Reflection	\??\C:\Windows\System32\cmd.exe /k msg * Hello from Dirty Vanity	C:\Windows\explorer.exe	ntdll.dll|Unbacked
Suspicious Process Creation via Reflection	\??\C:\Windows\System32\cmd.exe /k msg * Hello from Dirty Vanity	C:\Windows\System32\RtkAudUService64.exe	ntdll.dll|Unbacked
Suspicious Process Creation via Reflection	\??\C:\Windows\System32\cmd.exe /k msg * Hello from Dirty Vanity	C:\Windows\System32\RtkAudUService64.exe	ntdll.dll|Unbacked
Suspicious Process Creation via Reflection	\??\C:\Windows\System32\cmd.exe /k msg * Hello from Dirty Vanity	C:\Windows\explorer.exe	ntdll.dll|Unbacked
Suspicious Process Creation via Reflection	\??\C:\Windows\System32\cmd.exe /k msg * Hello from Dirty Vanity	C:\Windows\explorer.exe	ntdll.dll|Unbacked

Direct Syscall via Assembly Bytes

The second and final example for process events is process creation via direct syscall. This directly uses the syscall instruction instead of calling the **NtCreateProcess** API. Adversaries may use this method to avoid security products that are reliant on usermode API hooking (which Elastic Defend is not):

```
process where event.action : "start" and

// EQL detecting a call stack not ending with ntdll.dll
not process.parent.thread.Ext.call_stack_summary : "ntdll.dll*" and

/* last call in the call stack contains bytes that execute a syscall
 manually using assembly <mov r10,rcx, mov eax,ssn, syscall> */

_arraysearch(process.parent.thread.Ext.call_stack, $entry,
 ($entry.callsite_leading_bytes : ("*4c8bd1b8??????000f05",
 "*4989cab8??????000f05", "*4c8bd10f05", "*4989ca0f05")))
```

This example matches when the final memory region in the call stack is unbacked and contains assembly bytes that end with the syscall instruction (**0F05**):



rule.name : "Direct Syscall via Assembly Bytes"

156 hits

	process.executable
Aug 31, ...	C:\Windows\System32\schtasks.exe
Aug 31, ...	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Aug 31, ...	C:\Windows\System32\cmd.exe
Aug 31, ...	C:\Windows\System32\cmd.exe
Aug 31, ...	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Aug 24, ...	C:\Windows\explorer.exe
Aug 24, ...	C:\Windows\System32\conhost.exe
Aug 24, ...	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Aug 24, ...	C:\Windows\System32\cmd.exe
Aug 24, ...	C:\Windows\System32\cmd.exe
Aug 24, ...	C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe

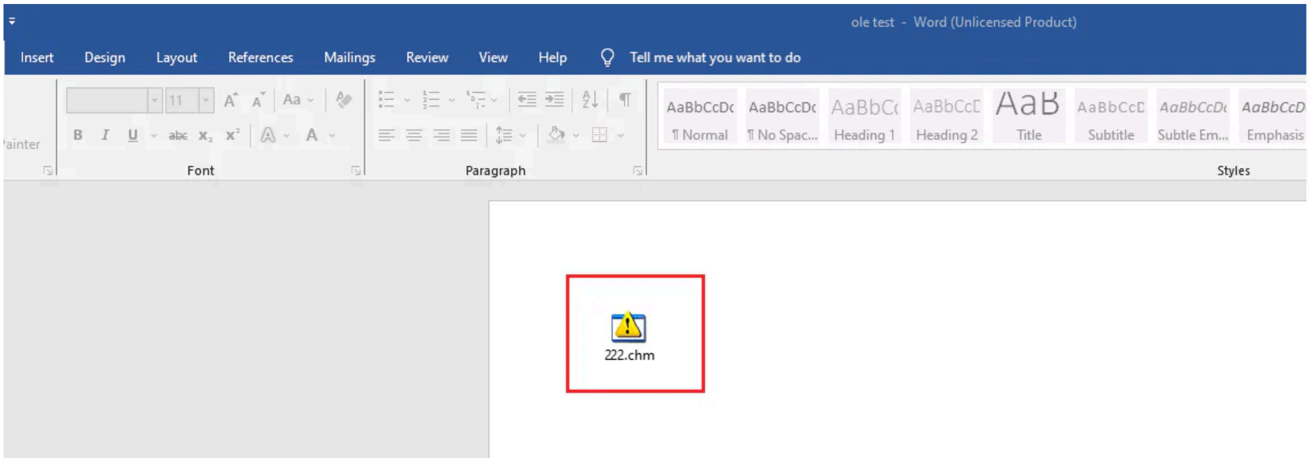
Expanded document

```
"pid": 9872,
"args_count": 1,
"thread": {
  "Ext": {
    "call_stack_summary": "Unbacked|kernel32.dll|ntdll.dll",
    "call_stack_contains_unbacked": true,
    "call_stack": [
      {
        "symbol_info": "Unbacked+0xafde",
        "callsite_trailing_bytes":
          "c349ba82a770e700000000e8b2ffffff49c7c2f5bb6028e8a6ffffff49baf0e503aa00000000e897ffff
          ff49c7c21bb1fc02e88bffffff49c7c2dc7e352ee87f",
        "protection": "RWX",
        "callsite_leading_bytes":
          "f3905848894c240848895424104c894424184c894c24204883ec284c89d1e88fffffff4883c428488b4c
          2408488b5424104c8b4424184c8b4c24204989ca0f05"
      },
      {
        "symbol_info": "Unbacked+0x9d6d",
        "callsite_trailing_bytes":
          "41b9008000004c8b442468488b54247048c7c1ffffffffe82213000041b9008000004d89e84889fa48c7
          c1ffffffffe80a13000049c744241000000000488b84",
        "protection": "RWX",
        "callsite_leading_bytes":
          "244848895c24408b842400040000894424388b8424f80300008944243048c744242800000000048c74424
          200000000041b9ffff1f0041b8ffff1f00e865130000"
```

File

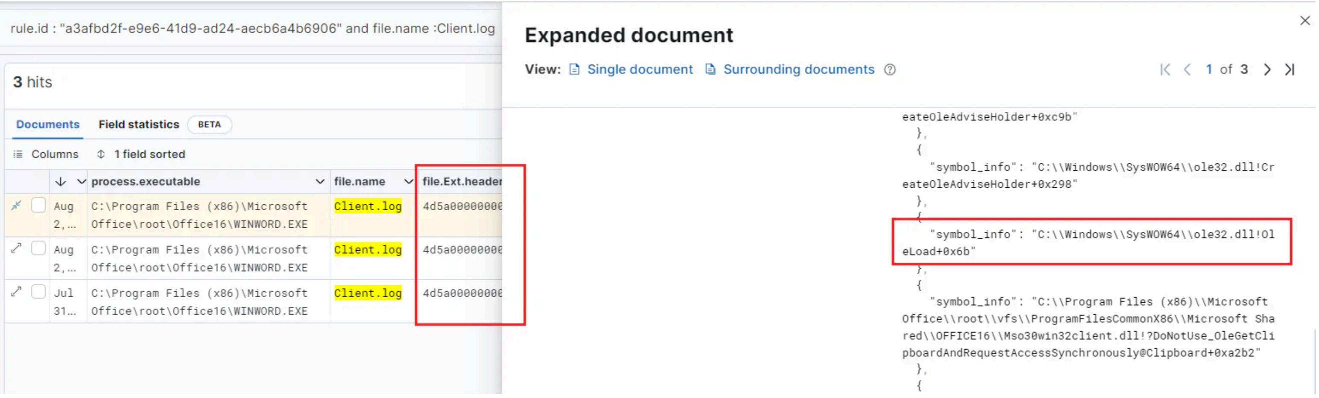
Suspicious Microsoft Office Embedded Object

The following rule logic identifies suspicious file extensions written by a Microsoft Office process from an embedded OLE stream, frequently used by malicious documents to drop payloads for initial access.



```
// EQL detecting file creation event with call stack indicating
// OleSaveToStream call to save or load the embedded OLE object

file where event.action != "deletion" and

process.name : ("winword.exe", "excel.exe", "powerpnt.exe") and

_arraysearch(process.thread.Ext.call_stack, $entry, $entry.symbol_info:
 ("*!OleSaveToStream*", "*!OleLoad*")) and
(
 file.extension : ("exe", "dll", "js", "vbs", "vbe", "jse", "url",
 "chm", "bat", "mht", "hta", "htm", "search-ms") or

 /* PE & HelpFile */
 file.Ext.header_bytes : ("4d5a*", "49545346*")
 )
```

Example of matches :



```
rule.id : "a3afbd2f-e9e6-41d9-ad24-aecb6a4b6906" and file.name :Client.log
```

3 hits

	process.executable	file.name	file.Ext.header
Aug 2,…	C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE	Client.log	4d5a0000000
Aug 2,…	C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE	Client.log	4d5a0000000
Jul 31…	C:\Program Files (x86)\Microsoft Office\root\Office16\WINWORD.EXE	Client.log	4d5a0000000

Expanded document

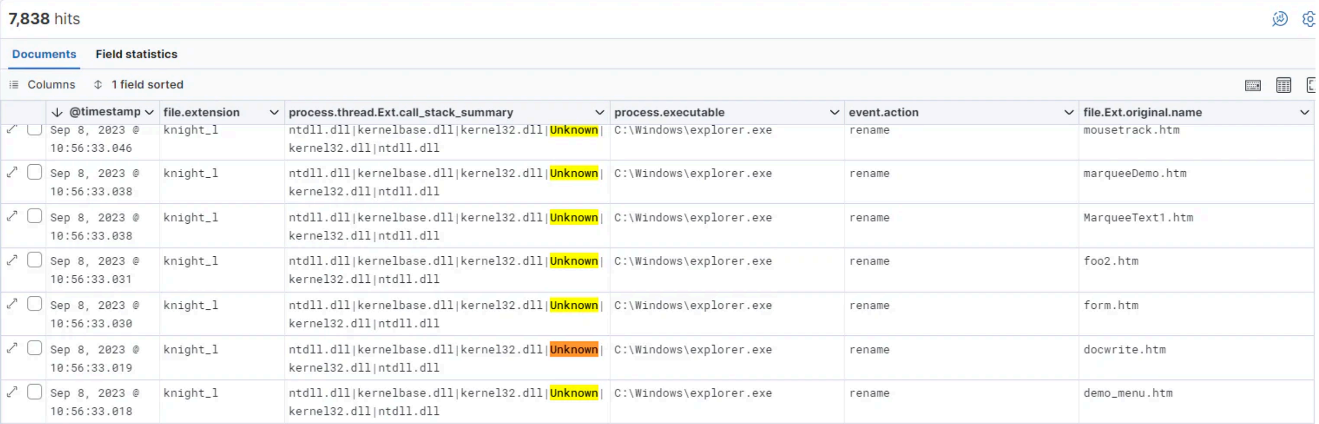
```
eateOleAdviseHolder+0xc9b"
  },
  {
    "symbol_info": "C:\\Windows\\SysWOW64\\ole32.dll!Cr
eateOleAdviseHolder+0x298"
  },
  {
    "symbol_info": "C:\\Windows\\SysWOW64\\ole32.dll!Ol
eLoad+0x6b"
  },
  {
    "symbol_info": "C:\\Program Files (x86)\\Microsoft
Office\\root\\vfs\\ProgramFilesCommonX86\\Microsoft Sha
red\\OFFICE16\\Mso30win32client.dll!?DoNotUse_OleGetCli
pboardAndRequestAccessSynchronously@Clipboard+0xa2b2"
  },
  {
```

Suspicious File Rename from Unbacked Memory

Certain ransomware may inject into signed processes before starting their encryption routine. File rename and modification events will appear to originate from a trusted process, potentially bypassing some heuristics that exclude signed processes as presumed false positives. The following KQL query looks for file rename of documents, from a signed binary and with a suspicious call stack:

```
file where event.action : "rename" and

process.code_signature.status : "trusted" and file.extension != null and

file.Ext.original.name : ("*.jpg", "*.bmp", "*.png", "*.pdf", "*.doc",
"*.docx", "*.xls", "*.xlsx", "*.ppt", "*.pptx") and

not file.extension : ("tmp", "~tmp", "diff", "gz", "download", "bak",
"bck", "lnk", "part", "save", "url", "jpg",  "bmp", "png", "pdf", "doc",
"docx", "xls", "xlsx", "ppt", "pptx") and

process.thread.Ext.call_stack_summary :
("ntdll.dll|kernelbase.dll|Unbacked",
 "ntdll.dll|kernelbase.dll|kernel32.dll|Unbacked",
 "ntdll.dll|kernelbase.dll|Unknown|kernel32.dll|ntdll.dll",
 "ntdll.dll|kernelbase.dll|Unknown|kernel32.dll|ntdll.dll",
 "ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll",
 "ntdll.dll|kernelbase.dll|kernel32.dll|mscorlib.ni.dll|Unbacked",
 "ntdll.dll|wow64.dll|wow64cpu.dll|wow64.dll|ntdll.dll|kernelbase.dll|
 Unbacked", "ntdll.dll|wow64.dll|wow64cpu.dll|wow64.dll|ntdll.dll|
 kernelbase.dll|Unbacked|kernel32.dll|ntdll.dll",
 "ntdll.dll|Unbacked", "Unbacked", "Unknown")
```

Here are some examples of matches where **explorer.exe** (Windows Explorer) is injected by the KNIGHT/CYCLOPS ransomware:

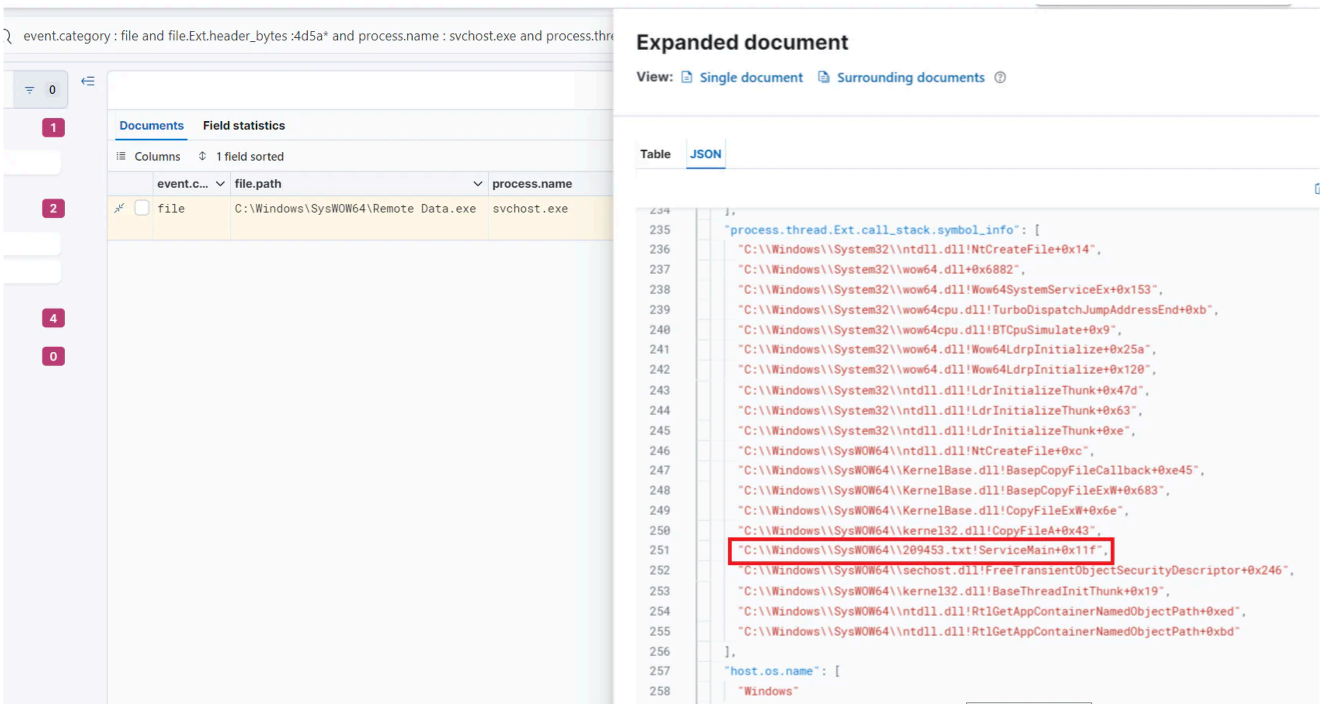


@timestamp	file.extension	process.thread.Ext.call_stack_summary	process.executable	event.action	file.Ext.original.name
Sep 8, 2023 @ 10:56:33.046	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename	mousetrack.htm
Sep 8, 2023 @ 10:56:33.038	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename	marqueeDemo.htm
Sep 8, 2023 @ 10:56:33.038	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename	MarqueeText1.htm
Sep 8, 2023 @ 10:56:33.031	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename	foo2.htm
Sep 8, 2023 @ 10:56:33.030	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename	form.htm
Sep 8, 2023 @ 10:56:33.019	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename	docwrite.htm
Sep 8, 2023 @ 10:56:33.018	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename	demo_menu.htm

Executable File Dropped by an Unsigned Service DLL

Certain types of malware maintain their presence by disguising themselves as Windows service DLLs. To be recognized and managed by the Service Control Manager, a service DLL must export a function named **ServiceMain**. The KQL query below helps identify instances where an executable file is created, and the call stack includes the **ServiceMain** function.

```
event.category : file and
 file.Ext.header_bytes :4d5a* and process.name : svchost.exe and
 process.thread.Ext.call_stack.symbol_info :*!ServiceMain*
```



event.category : file and file.Ext.header_bytes :4d5a* and process.name : svchost.exe and process.thre

event.c...	file.path	process.name
file	C:\Windows\SysWOW64\Remote Data.exe	svchost.exe

```
  ],
  "process.thread.Ext.call_stack.symbol_info": [
    "C:\\Windows\\System32\\ntdll.dll!NtCreateFile+0x14",
    "C:\\Windows\\System32\\wow64.dll+0x6882",
    "C:\\Windows\\System32\\wow64.dll!Wow64SystemServiceEx+0x153",
    "C:\\Windows\\System32\\wow64cpu.dll!TurboDispatchJumpAddressEnd+0xb",
    "C:\\Windows\\System32\\wow64cpu.dll!BTCpuSimulate+0x9",
    "C:\\Windows\\System32\\wow64.dll!Wow64LdrpInitialize+0x25a",
    "C:\\Windows\\System32\\wow64.dll!Wow64LdrpInitialize+0x120",
    "C:\\Windows\\System32\\ntdll.dll!LdrInitializeThunk+0x47d",
    "C:\\Windows\\System32\\ntdll.dll!LdrInitializeThunk+0x63",
    "C:\\Windows\\System32\\ntdll.dll!LdrInitializeThunk+0xe",
    "C:\\Windows\\SysWOW64\\ntdll.dll!NtCreateFile+0xc",
    "C:\\Windows\\SysWOW64\\KernelBase.dll!BasepCopyFileCallback+0xe45",
    "C:\\Windows\\SysWOW64\\KernelBase.dll!BasepCopyFileExW+0x683",
    "C:\\Windows\\SysWOW64\\KernelBase.dll!CopyFileExW+0x6e",
    "C:\\Windows\\SysWOW64\\kernel32.dll!CopyFileA+0x43",
    "C:\\Windows\\SysWOW64\\209453.txt!ServiceMain+0x11f",
    "C:\\Windows\\SysWOW64\\sechost.dll!FreeTransientObjectSecurityDescriptor+0x246",
    "C:\\Windows\\SysWOW64\\kernel32.dll!BaseThreadInitThunk+0x19",
    "C:\\Windows\\SysWOW64\\ntdll.dll!RtlGetAppContainerNamedObjectPath+0xed",
    "C:\\Windows\\SysWOW64\\ntdll.dll!RtlGetAppContainerNamedObjectPath+0xbd"
  ],
  "host.os.name": [
    "Windows"
```

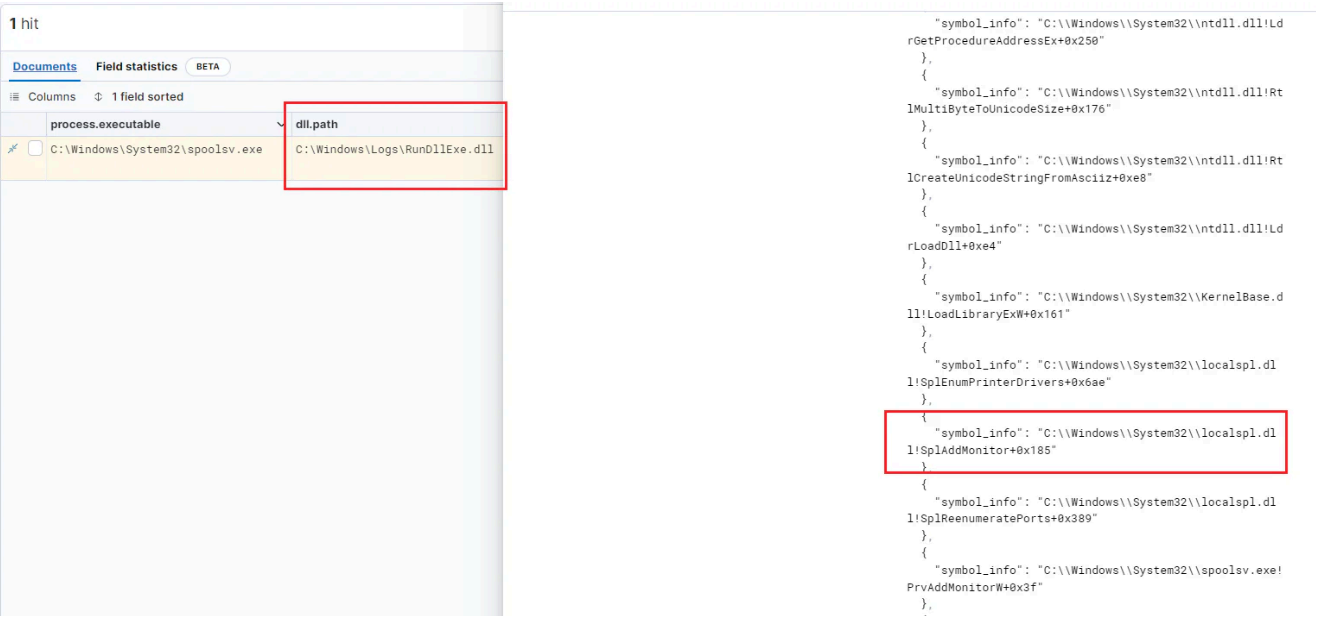
Library

Unsigned Print Monitor Driver Loaded

The following EQL query identifies the loading of an unsigned library by the print spooler service where the call stack indicates the load is coming from **SplAddMonitor**. Adversaries may use port monitors to run an adversary-supplied DLL during system boot for persistence or privilege escalation.

```
library where
process.executable : ("?:\\Windows\\System32\\spoolsv.exe",
"?:\\Windows\\SysWOW64\\spoolsv.exe") and not dll.code_signature.status :
"trusted" and _arraysearch(process.thread.Ext.call_stack, $entry,
$entry.symbol_info: "*localspl.dll!SplAddMonitor*")
```

Example of match:



process.executable	dll.path
C:\Windows\System32\spoolsv.exe	C:\Windows\Logs\RunDllExe.dll

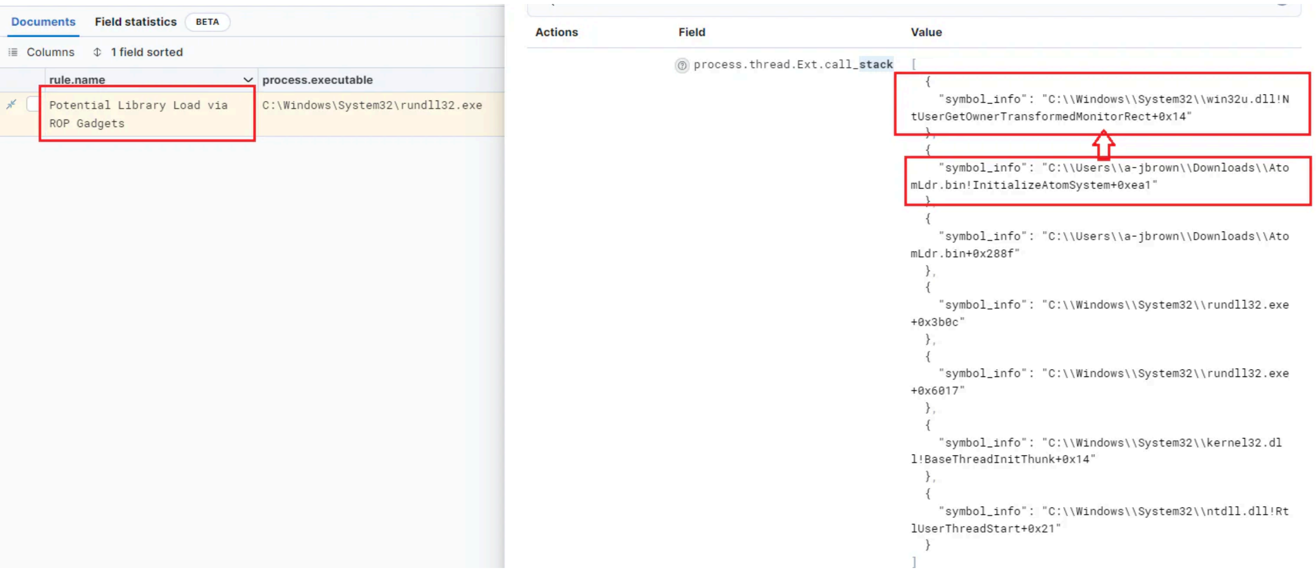
```
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!Ld
rGetProcedureAddressEx+0x250"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!Rt
lMultiByteToUnicodeSize+0x176"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!Rt
lCreateUnicodeStringFromAsciiz+0xe8"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!Ld
rLoadDll+0xe4"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\KernelBase.d
ll!LoadLibraryExW+0x161"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\localspl.dl
l!SplEnumPrinterDrivers+0x6ae"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\localspl.dl
l!SplAddMonitor+0x185"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\localspl.dl
l!SplReenumeratePorts+0x389"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\spoolsv.exe!
PrvAddMonitorW+0x3f"
  },
```

Potential Library Load via ROP Gadgets

This EQL rule identifies the loading of a library from unusual **win32u** or **ntdll** offsets. This may indicate an attempt to bypass API monitoring using Return Oriented Programming (ROP) assembly gadgets to execute a syscall instruction from a trusted module.

```
library where
// adversaries try to use ROP gadgets from ntdll.dll or win32u.dll
// to construct a normal-looking call stack

process.thread.Ext.call_stack_summary : ("ntdll.dll|*", "win32u.dll|*") and

// excluding normal Library Load APIs - LdrLoadDll and NtMapViewOfSection
not _arraysearch(process.thread.Ext.call_stack, $entry,
 $entry.symbol_info: ("*ntdll.dll!Ldr*",
 "*KernelBase.dll!LoadLibrary*", "*ntdll.dll!*MapViewOfSection*"))
```

This example matches when AtomLdr loads a DLL using ROP gadgets from **win32u.dll** instead of using **ntdll**’s load library APIs (**LdrLoadDll** and **NtMapViewOfSection**).



rule.name	process.executable
Potential Library Load via ROP Gadgets	C:\Windows\System32\rundll32.exe

Field	Value
process.thread.Ext.call_stack	see below

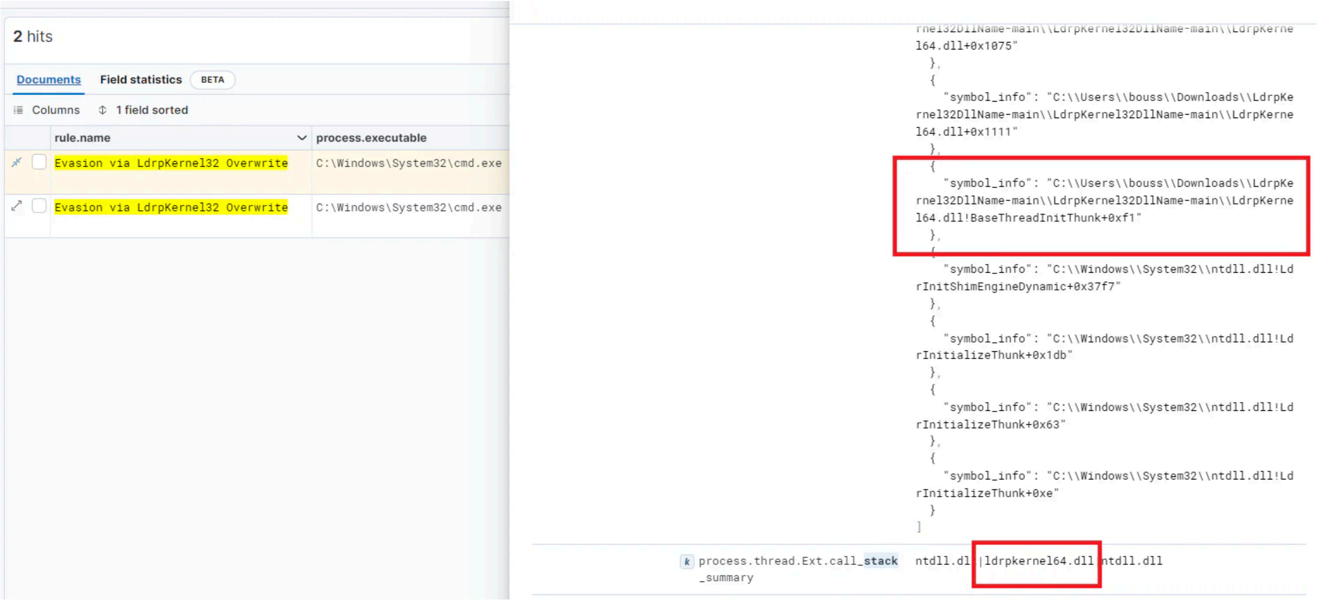
```
[
  {
    "symbol_info": "C:\\Windows\\System32\\win32u.dll!NtUserGetOwnerTransformedMonitorRect+0x14"
  },
  {
    "symbol_info": "C:\\Users\\a-jbrown\\Downloads\\AtomLdr.bin!InitializeAtomSystem+0xea1"
  },
  {
    "symbol_info": "C:\\Users\\a-jbrown\\Downloads\\AtomLdr.bin+0x288f"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\rundll32.exe+0x3b0c"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\rundll32.exe+0x6017"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\kernel32.dll!BaseThreadInitThunk+0x14"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!RtlUserThreadStart+0x21"
  }
]
```

Evasion via LdrpKernel32 Overwrite

The [LdrpKernel32(https://github.com/rbmm/LdrpKernel32DllName) evasion is an interesting technique to hijack the early execution of a process during the bootstrap phase by overwriting the bootstrap DLL name referenced in **ntdll.dll** memory– forcing the process to load a malicious DLL.

```
library where

// BaseThreadInitThunk must be exported by the rogue bootstrap DLL
 _arraysearch(process.thread.Ext.call_stack, $entry, $entry.symbol_info :
  "*!BaseThreadInitThunk*") and

// excluding kernel32 that exports normally exports BasethreadInitThunk
not _arraysearch(process.thread.Ext.call_stack, $entry, $entry.symbol_info
 ("?:\\Windows\\System32\\kernel32.dll!BaseThreadInitThunk*",
 "?:\\Windows\\SysWOW64\\kernel32.dll!BaseThreadInitThunk*",
 "?:\\Windows\\WinSxS\\*\\kernel32.dll!BaseThreadInitThunk*",
 "?:\\Windows\\WinSxS\\Temp\\PendingDeletes\\*!BaseThreadInitThunk*",
 "\\Device\\*\\Windows\\*\\kernel32.dll!BaseThreadInitThunk*"))
```

Example of match:



2 hits

	rule.name	process.executable
☐	Evasion via LdrpKernel32 Overwrite	C:\Windows\System32\cmd.exe
☐	Evasion via LdrpKernel32 Overwrite	C:\Windows\System32\cmd.exe

```
rnel32DllName-main\\LdrpKernel32DllName-main\\LdrpKerne
l64.dll+0x1075"
  },
  {
    "symbol_info": "C:\\Users\\bouss\\Downloads\\LdrpKe
rnel32DllName-main\\LdrpKernel32DllName-main\\LdrpKerne
l64.dll+0x1111"
  },
  {
    "symbol_info": "C:\\Users\\bouss\\Downloads\\LdrpKe
rnel32DllName-main\\LdrpKernel32DllName-main\\LdrpKerne
l64.dll!BaseThreadInitThunk+0xf1"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!Ld
rInitShimEngineDynamic+0x37f7"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!Ld
rInitializeThunk+0x1db"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!Ld
rInitializeThunk+0x63"
  },
  {
    "symbol_info": "C:\\Windows\\System32\\ntdll.dll!Ld
rInitializeThunk+0xe"
  }
]
```

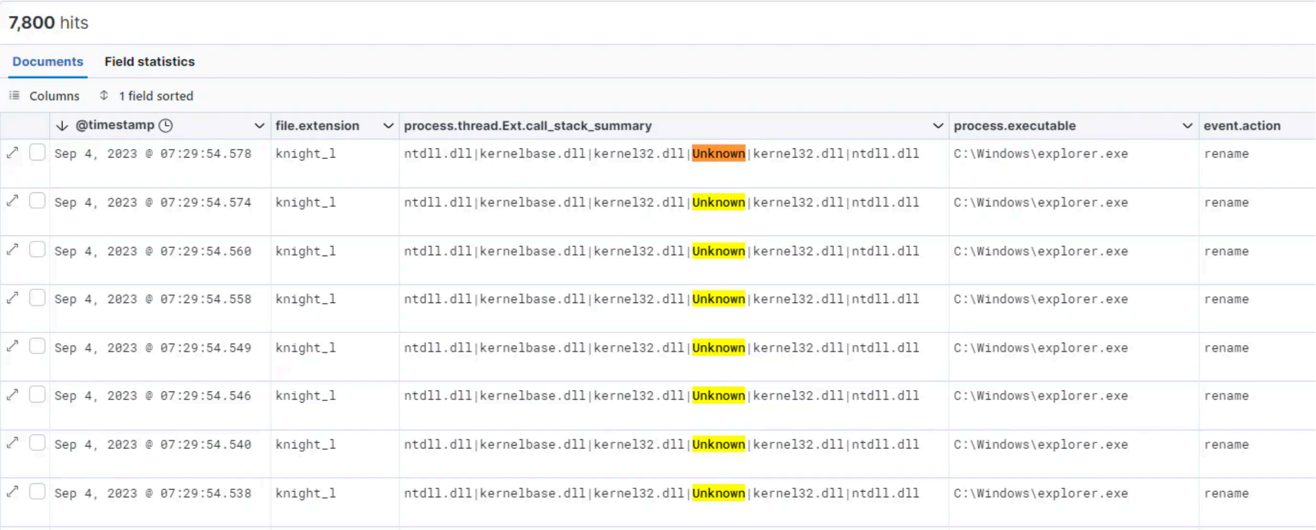
```
k process.thread.Ext.call_stack    ntdll.dll|ldrpkernel64.dll|ntdll.dll
  _summary
```

Suspicious Remote Registry Modification

Similar to the scheduled task example, the remote registry service is hosted in **svchost.exe**. We can use the call stack to detect registry modification by monitoring when the Remote Registry service points to an executable or script file. This may indicate an attempt to move laterally via remote configuration changes.

```
registry where

event.action == "modification" and

user.id : ("S-1-5-21*", "S-1-12-*") and

 process.name : "svchost.exe" and

// The regsvc.dll in call stack indicate that this is indeed the
// svchost.exe instance hosting the Remote registry service

process.thread.Ext.call_stack_summary : "*regsvc.dll|rpcrt4.dll*" and

 (
  // suspicious registry values
  registry.data.strings : ("*:\\*\\*", "*.exe*", "*.dll*", "*rundll32*",
  "*powershell*", "*http*", "* /c *", "*COMSPEC*", "\\\\*.*") or

  // suspicious keys like Services, Run key and COM
  registry.path :
         ("HKLM\\SYSTEM\\ControlSet*\\Services\\*\\ServiceDLL",
          "HKLM\\SYSTEM\\ControlSet*\\Services\\*\\ImagePath",
          "HKEY_USERS\\*Classes\\*\\InprocServer32\\",
          "HKEY_USERS\\*Classes\\*\\LocalServer32\\",
          "H*\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\\*") or

  // potential attempt to remotely disable a service
  (registry.value : "Start" and registry.data.strings : "4")
  )
```

This example matches when the Run key registry value is modified remotely via the Remote Registry service:



	@timestamp	file.extension	process.thread.Ext.call_stack_summary	process.executable	event.action
☐	Sep 4, 2023 @ 07:29:54.578	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename
☐	Sep 4, 2023 @ 07:29:54.574	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename
☐	Sep 4, 2023 @ 07:29:54.560	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename
☐	Sep 4, 2023 @ 07:29:54.558	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename
☐	Sep 4, 2023 @ 07:29:54.549	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename
☐	Sep 4, 2023 @ 07:29:54.546	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename
☐	Sep 4, 2023 @ 07:29:54.540	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename
☐	Sep 4, 2023 @ 07:29:54.538	knight_l	ntdll.dll|kernelbase.dll|kernel32.dll|Unknown|kernel32.dll|ntdll.dll	C:\Windows\explorer.exe	rename

Conclusion

As we’ve demonstrated, call stacks are not only useful for finding known bad patterns, but also for reducing ambiguity in standard EDR events, and easing behavior interpretation. The examples we've provided here represent just a minor portion of the potential detection possibilities achievable by applying enhanced enrichment to the same dataset.