Hunting Evil with YARA (Windows Edition)

In this section, we'll explore using YARA on Windows systems for identifying threats both on disk and in memory.

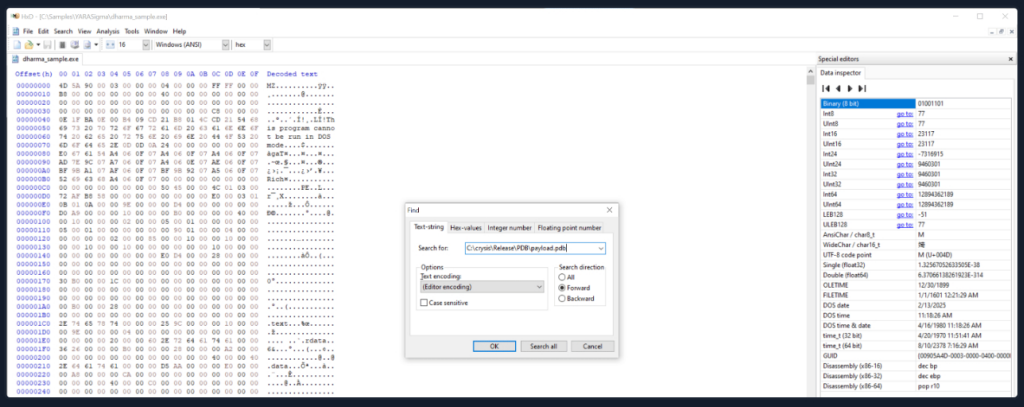
Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, let's RDP into the Target IP using the provided credentials. The vast majority of the actions/commands covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

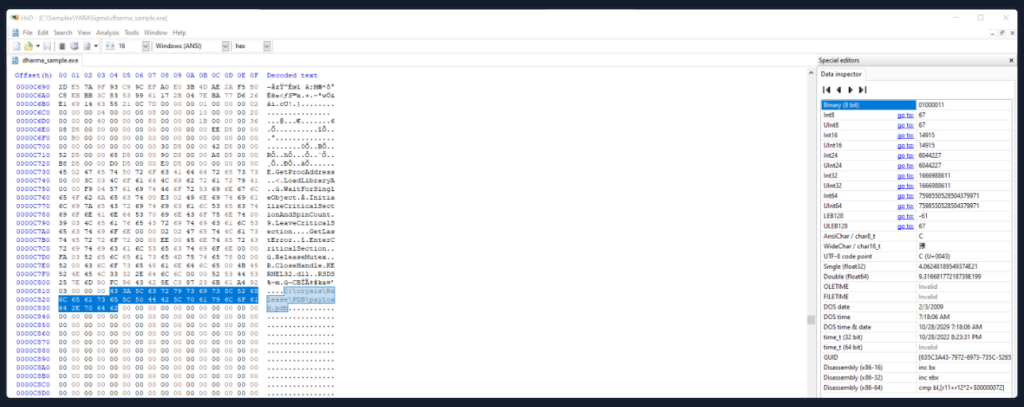
Hunting for Malicious Executables on Disk with YARA

As we saw in the previous section, YARA is a potent weapon in the arsenal of cybersecurity professionals for detecting and hunting malicious executables on a disk. With custom YARA rules or established ones at our disposal, we can pinpoint suspicious or potentially malicious files based on distinct patterns, traits, or behaviors.

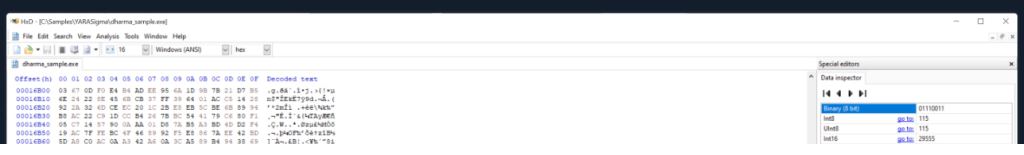
We will be using a sample that we analyzed previously named `dharma_sample.exe` residing in the `C:\Samples\YARASigma` directory of this section's target.

We'll first examine the malware sample inside a hex editor (`HxD`, located at `C:\Program Files\HxD`) to identify the previously discovered string `C:\crysis\Release\PDB\payload.pdb`.





If we scroll almost to the bottom, we will notice yet another seemingly unique `sssssbsss` string.



My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

Going forward, we will craft a rule grounded in these patterns and then utilize the YARA utility to scour the filesystem for similar executables.

Note: In a Linux machine the `hexdump` utility could have been used to identify the aforementioned hex bytes as follows.

Hunting Evil with YARA (Windows Edition)

```
remnux@remnux:~$ hexdump dharma_sample.exe -C | grep crysis -n3
3140-0000c7e0  52 00 43 6c 6f 73 65 48  61 6e 64 6c 65 00 4b 45  |R.CloseHandle.KE|
3141-0000c7f0  52 4e 45 4c 33 32 2e 64  6c 6c 00 00 52 53 44 53  |RNEL32.dll..RSDS|
3142-0000c800  25 7e 6d 90 fc 96 43 42  8e c3 87 23 6b 61 a4 92  |%~m...CB...#ka..|
3143:0000c810  03 00 00 00 43 3a 5c 63  72 79 73 69 73 5c 52 65  |....C:\crysis\Re|
3144-0000c820  6c 65 61 73 65 5c 50 44  42 5c 70 61 79 6c 6f 61  |lease\PDB\payloa|
3145-0000c830  64 2e 70 64 62 00 00 00  00 00 00 00 00 00 00 00  |d.pdb...........|
3146-0000c840  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Hunting Evil with YARA (Windows Edition)

```
remnux@remnux:~$ hexdump dharma_sample.exe -C | grep sssssbsss -n3
5738-00016be0  3d 00 00 00 26 00 00 00  73 73 73 64 00 00 00 00  |=...&...sssd....|
5739-00016bf0  26 61 6c 6c 3d 00 00 00  73 64 00 00 2d 00 61 00  |&all=...sd..-.a.|
5740-00016c00  00 00 00 00 73 00 73 00  62 00 73 00 73 00 00 00  |....s.s.b.s.s...|
5741:00016c10  73 73 73 73 73 62 73 73  73 00 00 00 73 73 73 73  |sssssbsss...ssss|
5742-00016c20  73 62 73 00 22 00 00 00  22 00 00 00 5c 00 00 00  |sbs."...".....\...|
5743-00016c30  5c 00 00 00 5c 00 00 00  5c 00 00 00 5c 00 00 00  |\...\...\...\...|
5744-00016c40  22 00 00 00 20 00 22 00  00 00 00 00 5c 00 00 00  |"... ."".....\...|
```

Let's incorporate all identified hex bytes into a rule, enhancing our ability to detect this string across any disk-based executable.

Code: yara

```
rule ransomware_dharma {

    meta:
        author = "Madhukar Raina"
        version = "1.0"
        description = "Simple rule to detect strings from Dharma ransomware"
        reference = "https://www.virustotal.com/gui/file/bff6a1000a86f8edf3673d576786ec75b80bed0c45

    strings:
        $string_pdb = {  433A5C63727973697369735C52656C656173655C5044425C7061796C6F61642E706462 }
        $string_ssss = { 73 73 73 73 73 62 73 73 73 }

        condition: all of them
}
```

This rule (`dharma_ransomware.yar`) can be found inside the `C:\Rules\yara` directory of this section's target.

Initiating the YARA executable with this rule, let's observe if it highlights other analogous samples on the disk.

Hunting Evil with YARA (Windows Edition)

```
PS C:\Users\htb-student> yara64.exe -s C:\Rules\yara\dharma_ransomware.yar C:\Samples\YARASigma\ -r
ransomware_dharma C:\Samples\YARASigma\\dharma_sample.exe
0xc814:$string_pdb: 43 3A 5C 63 72 79 73 69 73 5C 52 65 6C 65 61 73 65 5C 50 44 42 5C 70 61 79 6C 6
0x16c10:$string_ssss: 73 73 73 73 73 62 73 73 73
```

```
ransomware_dharma C:\Samples\YARASigma\\check_updates.exe
0xc814:$string_pdb: 43 3A 5C 63 72 79 73 69 73 5C 52 65 6C 65 61 73 65 5C 50 44 42 5C 70 61 79 6C 6
0x16c10:$string_ssss: 73 73 73 73 73 62 73 73 73
ransomware_dharma C:\Samples\YARASigma\\microsoft.com
0xc814:$string_pdb: 43 3A 5C 63 72 79 73 69 73 5C 52 65 6C 65 61 73 65 5C 50 44 42 5C 70 61 79 6C 6
0x16c10:$string_ssss: 73 73 73 73 73 62 73 73 73
ransomware_dharma C:\Samples\YARASigma\\KB5027505.exe
0xc814:$string_pdb: 43 3A 5C 63 72 79 73 69 73 5C 52 65 6C 65 61 73 65 5C 50 44 42 5C 70 61 79 6C 6
0x16c10:$string_ssss: 73 73 73 73 73 62 73 73 73
ransomware_dharma C:\Samples\YARASigma\\pdf_reader.exe
0xc814:$string_pdb: 43 3A 5C 63 72 79 73 69 73 5C 52 65 6C 65 61 73 65 5C 50 44 42 5C 70 61 79 6C 6
0x16c10:$string_ssss: 73 73 73 73 73 62 73 73 73
```

Command Breakdown:

- `yara64.exe`: Refers to the YARA64 executable, which is the YARA scanner specifically designed for 64-bit systems.
- `-s C:\Rules\yara\dharma_ransomware.yar`: Specifies the YARA rules file to be used for scanning. In this case, the rules file named `dharma_ransomware.yar` located in the `C:\Rules\yara` directory is provided.
- `C:\Samples\YARASigma`: Specifies the path or directory to be scanned by YARA. In this case, the directory being scanned is `C:\Samples\YARASigma`.
- `-r`: Indicates that the scanning operation should be performed recursively, meaning YARA will scan files within subdirectories of the specified directory as well.
- `2>nul`: Redirects the error output (stream `2`) to a null device, effectively hiding any error messages that might occur during the scanning process.

As we can see, the `pdf_reader.exe`, `microsoft.com`, `check_updates.exe`, and `KB5027505.exe` files are detected by this rule (in addition to `dharma_sample.exe` of course).

Now, let's pivot, applying YARA rules to live processes.

Hunting for Evil Within Running Processes with YARA

To ascertain if malware lurks in ongoing processes, we'll unleash the YARA scanner on the system's active processes. Let's demonstrate using a YARA rule that targets Metasploit's meterpreter shellcode, believed to be lurking in a running process.

YARA Rule Source: https://github.com/cuckoosandbox/community/blob/master/data/yara/shellcode/metasploit.yar

Code: yara

```
rule meterpreter_reverse_tcp_shellcode {
    meta:
        author = "FDD @ Cuckoo sandbox"
        description = "Rule for metasploit's  meterpreter reverse tcp raw shellcode"

    strings:
        $s1 = { fce8 8?00 0000 60 }     // shellcode prologe in metasploit
        $s2 = { 648b ??30 }             // mov edx, fs:[???+0x30]
        $s3 = { 4c77 2607 }             // kernel32 checksum
        $s4 = "ws2_"                    // ws2_32.dll
        $s5 = { 2980 6b00 }             // WSAStartUp checksum
        $s6 = { ea0f dfe0 }             // WSASocket checksum
        $s7 = { 99a5 7461 }             // connect checksum

    condition:
        5 of them
}
```

We will be using a sample that we analyzed previously named `htb_sample_shell.exe` residing in the `C:\Samples\YARASigma` directory of this section's targe

`htb_sample_shell.exe` injects Metasploit's meterpreter shellcode into the `cmdkey.exe` process. Let's activate it,

ensuring successful injection.

Note: Make sure you launch PowerShell as an administrator.

Hunting Evil with YARA (Windows Edition)

```
PS C:\Samples\YARASigma> .\htb_sample_shell.exe

<-- Hack the box sample for yara signatures -->

[+] Parent process with PID 7972 is created : C:\Samples\YARASigma\htb_sample_shell.exe
[+] Child process with PID 9084 is created : C:\Windows\System32\cmdkey.exe
[+] Shellcode is written at address 000002686B1C0000 in remote process C:\Windows\System32\cmdkey.e
[+] Remote thread to execute the shellcode is started with thread ID 368

Press enter key to terminate...
```

With the injection executed, let's scan every active system process as follows, through another PowerShell terminal (`Run as administrator`).

Hunting Evil with YARA (Windows Edition)

```
PS C:\Windows\system32> Get-Process | ForEach-Object { "Scanning with Yara for meterpreter shellcod
Scanning with Yara for meterpreter shellcode on PID 9000
Scanning with Yara for meterpreter shellcode on PID 9016
Scanning with Yara for meterpreter shellcode on PID 4940
Scanning with Yara for meterpreter shellcode on PID 5716
Scanning with Yara for meterpreter shellcode on PID 9084
meterpreter_reverse_tcp_shellcode 9084
Scanning with Yara for meterpreter shellcode on PID 7112
Scanning with Yara for meterpreter shellcode on PID 8400
Scanning with Yara for meterpreter shellcode on PID 9180
Scanning with Yara for meterpreter shellcode on PID 416
error scanning 416: can not attach to process (try running as root)
Scanning with Yara for meterpreter shellcode on PID 492
error scanning 492: can not attach to process (try running as root)
Scanning with Yara for meterpreter shellcode on PID 1824
error scanning 1824: can not attach to process (try running as root)
Scanning with Yara for meterpreter shellcode on PID 8268
Scanning with Yara for meterpreter shellcode on PID 3940
Scanning with Yara for meterpreter shellcode on PID 7960
Scanning with Yara for meterpreter shellcode on PID 988
Scanning with Yara for meterpreter shellcode on PID 6276
Scanning with Yara for meterpreter shellcode on PID 4228
Scanning with Yara for meterpreter shellcode on PID 772
Scanning with Yara for meterpreter shellcode on PID 780
Scanning with Yara for meterpreter shellcode on PID 1192
Scanning with Yara for meterpreter shellcode on PID 7972
meterpreter_reverse_tcp_shellcode 7972
Scanning with Yara for meterpreter shellcode on PID 0
error scanning 0: could not open file
Scanning with Yara for meterpreter shellcode on PID 6788
Scanning with Yara for meterpreter shellcode on PID 924
Scanning with Yara for meterpreter shellcode on PID 636
Scanning with Yara for meterpreter shellcode on PID 1780
error scanning 1780: can not attach to process (try running as root)
```

We're leveraging a concise PowerShell script. The `Get-Process` command fetches running processes, and with the help of the pipe symbol (`|`), this data funnels into the script block (`{...}`). Here, `ForEach-Object` dissects each process, prompting `yara64.exe` to apply our YARA rule on each process's memory.

Let's observe if the child process (`PID 9084`) gets flagged.

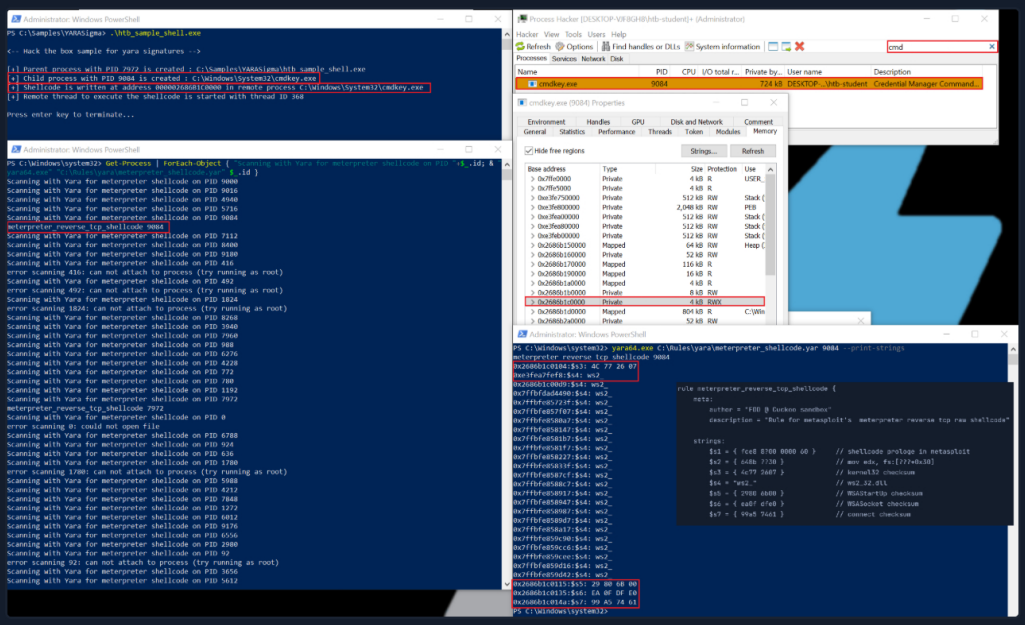
From the results, the meterpreter shellcode seems to have infiltrated a process with `PID 9084`. We can also guide the YARA scanner with a specific PID as follows.

Hunting Evil with YARA (Windows Edition)

```
PS C:\Windows\system32> yara64.exe C:\Rules\yara\meterpreter_shellcode.yar 9084 --print-strings
meterpreter_reverse_tcp_shellcode 9084
0x2686b1c0104:$s3: 4C 77 26 07
0xe3fea7fef8:$s4: ws2_
0x2686b1c00d9:$s4: ws2
```

```
0x7ffbfdad4490:$s4: ws2_
0x7ffbfe85723f:$s4: ws2_
0x7ffbfe857f07:$s4: ws2_
0x7ffbfe8580a7:$s4: ws2_
0x7ffbfe858147:$s4: ws2_
0x7ffbfe8581b7:$s4: ws2_
0x7ffbfe8581f7:$s4: ws2_
0x7ffbfe858227:$s4: ws2_
0x7ffbfe85833f:$s4: ws2_
0x7ffbfe8587cf:$s4: ws2_
0x7ffbfe8588c7:$s4: ws2_
0x7ffbfe858917:$s4: ws2_
0x7ffbfe858947:$s4: ws2_
0x7ffbfe858987:$s4: ws2_
0x7ffbfe8589d7:$s4: ws2_
0x7ffbfe858a17:$s4: ws2_
0x7ffbfe859c90:$s4: ws2_
0x7ffbfe859cc6:$s4: ws2_
0x7ffbfe859cee:$s4: ws2_
0x7ffbfe859d16:$s4: ws2_
0x7ffbfe859d42:$s4: ws2_
0x2686b1c0115:$s5: 29 80 6B 00
0x2686b1c0135:$s6: EA 0F DF E0
0x2686b1c014a:$s7: 99 A5 74 61
```

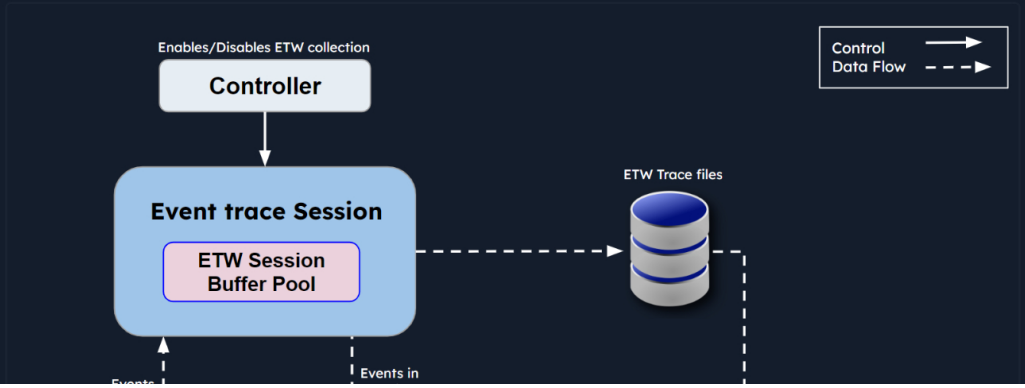
The screenshot below shows an overview of what we discussed above.

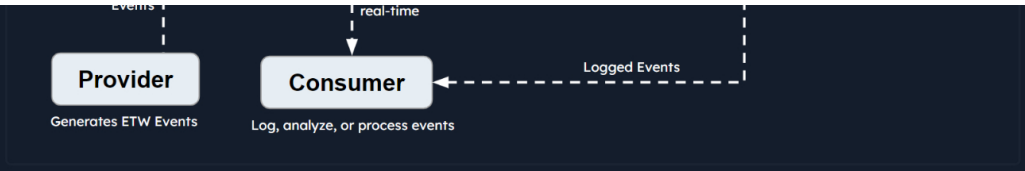


Hunting for Evil Within ETW Data with YARA

In the module titled `Windows Event Logs & Finding Evil` we explored `ETW` and introduced SilkETW. In this section, we'll circle back to ETW data, highlighting how YARA can be used to filter or tag certain events.

A quick recap first. According to Microsoft, `Event Tracing For Windows (ETW)` is a general-purpose, high-speed tracing facility provided by the operating system. Using a buffering and logging mechanism implemented in the kernel, ETW provides a tracing mechanism for events raised by both user-mode applications and kernel-mode device drivers.





- Controllers: Controllers possess functionalities that encompass initiating and terminating trace sessions. They also have the capability to enable or disable providers within a specific trace.
- Providers: Providers are crucial, as they generate events and channel them to the designated ETW sessions.
- Consumers: Consumers are the subscribers to specific events. They tap into these events and then receive them for in-depth processing or analysis.

Useful Providers

- Microsoft-Windows-Kernel-Process: This ETW provider is instrumental in monitoring process-related activity within the Windows kernel. It can aid in detecting unusual process behaviors such as process injection, process hollowing, and other tactics commonly used by malware and advanced persistent threats (APTs).
- Microsoft-Windows-Kernel-File: As the name suggests, this provider focuses on file-related operations. It can be employed for detection scenarios involving unauthorized file access, changes to critical system files, or suspicious file operations indicative of exfiltration or ransomware activity.
- Microsoft-Windows-Kernel-Network: This ETW provider offers visibility into network-related activity at the kernel level. It's especially useful in detecting network-based attacks such as data exfiltration, unauthorized network connections, and potential signs of command and control (C2) communication.
- Microsoft-Windows-SMBClient/SMBServer: These providers monitor Server Message Block (SMB) client and server activity, providing insights into file sharing and network communication. They can be used to detect unusual SMB traffic patterns, potentially indicating lateral movement or data exfiltration.
- Microsoft-Windows-DotNETRuntime: This provider focuses on .NET runtime events, making it ideal for identifying anomalies in .NET application execution, potential exploitation of .NET vulnerabilities, or malicious .NET assembly loading.
- OpenSSH: Monitoring the OpenSSH ETW provider can provide important insights into Secure Shell (SSH) connection attempts, successful and failed authentications, and potential brute force attacks.
- Microsoft-Windows-VPN-Client: This provider enables tracking of Virtual Private Network (VPN) client events. It can be useful for identifying unauthorized or suspicious VPN connections.
- Microsoft-Windows-PowerShell: This ETW provider tracks PowerShell execution and command activity, making it invaluable for detecting suspicious PowerShell usage, script block logging, and potential misuse or exploitation.
- Microsoft-Windows-Kernel-Registry: This provider monitors registry operations, making it useful for detection scenarios related to changes in registry keys, often associated with persistence mechanisms, malware installation, or system configuration changes.
- Microsoft-Windows-CodeIntegrity: This provider monitors code and driver integrity checks, which can be key in identifying attempts to load unsigned or malicious drivers or code.
- Microsoft-Antimalware-Service: This ETW provider can be employed to detect potential issues with the antimalware service, including disabled services, configuration changes, or potential evasion techniques employed by malware.
- WinRM: Monitoring the Windows Remote Management (WinRM) provider can reveal unauthorized or suspicious remote management activity, often indicative of lateral movement or remote command execution.

- `Microsoft-Windows-TerminalServices-LocalSessionManager`: This provider tracks local Terminal Services sessions, making it useful for detecting unauthorized or suspicious remote desktop activity.
- `Microsoft-Windows-Security-Mitigations`: This provider keeps tabs on the effectiveness and operations of security mitigations in place. It's essential for identifying potential bypass attempts of these security controls.
- `Microsoft-Windows-DNS-Client`: This ETW provider gives visibility into DNS client activity, which is crucial for detecting DNS-based attacks, including DNS tunneling or unusual DNS requests that may indicate C2 communication.
- `Microsoft-Antimalware-Protection`: This provider monitors the operations of antimalware protection mechanisms. It can be used to detect any issues with these mechanisms, such as disabled protection features, configuration changes, or signs of evasion techniques employed by malicious actors.

YARA Rule Scanning on ETW (Using SilkETW)

SilkETW is an open-source tool to work with Event Tracing for Windows (ETW) data. SilkETW provides enhanced visibility and analysis of Windows events for security monitoring, threat hunting, and incident response purposes. The best part of SilkETW is that it also has an option to integrate YARA rules. It includes YARA functionality to filter or tag event data.

SilkETW resides in the `C:\Tools\SilkETW\v8\SilkETW` directory of this section's target.

Hunting Evil with YARA (Windows Edition)

```
PS C:\Tools\SilkETW\v8\SilkETW> .\SilkETW.exe -h

                [v0.8 - Ruben Boonen => @FuzzySec]


 >--ʷʷ--> Args? <--ʷʷ--<

-h  (--help)          This help menu
-s  (--silk)          Trivia about Silk
-t  (--type)          Specify if we are using a Kernel or User collector
-kk (--kernelkeyword) Valid keywords: Process, Thread, ImageLoad, ProcessCounters, ContextSwitch,
                      DeferedProcedureCalls, Interrupt, SystemCall, DiskIO, DiskFileIO, DiskIOInit,
                      Dispatcher, Memory, MemoryHardFaults, VirtualAlloc, VAMap, NetworkTCPIP, Regi
                      AdvancedLocalProcedureCalls, SplitIO, Handle, Driver, OS, Profile, Default,
                      ThreadTime, FileIO, FileIOInit, Verbose, All, IOQueue, ThreadPriority,
                      ReferenceSet, PMCProfile, NonContainer
-uk (--userkeyword)   Define a mask of valid keywords, eg 0x2038 -> JitKeyword|InteropKeyword|
                      LoaderKeyword|NGenKeyword
-pn (--providername)  User ETW provider name, eg "Microsoft-Windows-DotNETRuntime" or its
                      corresponding GUID eg "e13c0d23-ccbc-4e12-931b-d9cc2eee27e4"
-l  (--level)         Logging level: Always, Critical, Error, Warning, Informational, Verbose
-ot (--outputtype)    Output type: POST to "URL", write to "file" or write to "eventlog"
-p  (--path)          Full output file path or URL. Event logs are automatically written to
                      "Applications and Services Logs\SilkETW-Log"
-f  (--filter)        Filter types: None, EventName, ProcessID, ProcessName, Opcode
-fv (--filtervalue)   Filter type capture value, eg "svchost" for ProcessName
-y  (--yara)          Full path to folder containing Yara rules
-yo (--yaraoptions)   Either record "All" events or only "Matches"

 >--ʷʷ--> Usage? <--ʷʷ--<

# Use a VirtualAlloc Kernel collector, POST results to Elasticsearch
SilkETW.exe -t kernel -kk VirtualAlloc -ot url -p https://some.elk:9200/valloc/_doc/

# Use a Process Kernel collector, filter on PID
SilkETW.exe -t kernel -kk Process -ot url -p https://some.elk:9200/kproc/_doc/ -f ProcessID -fv 112

# Use a .Net User collector, specify mask, filter on EventName, write to file
SilkETW.exe -t user -pn Microsoft-Windows-DotNETRuntime -uk 0x2038 -ot file -p C:\Some\Path\out.jso

# Use a DNS User collector, specify log level, write to file
SilkETW.exe -t user -pn Microsoft-Windows-DNS-Client -l Always -ot file -p C:\Some\Path\out.json
```

```
# Use an LDAP User collector, perform Yara matching, POST matches to Elasticsearch
SilkETW.exe -t user -pn Microsoft-Windows-Ldap-Client -ot url -p https://some.elk:9200/ldap/_doc/ -

# Specify "Microsoft-Windows-COM-Perf" by its GUID, write results to the event log
SilkETW.exe -t user -pn b8d6861b-d20f-4eec-bbae-87e0dd80602b -ot eventlog
```

The help menu provides many examples of how we can use the tool. Let's experiment with some of the YARA scanning options on a few ETW providers.

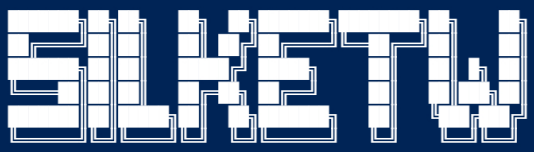
Example 1: YARA Rule Scanning on Microsoft-Windows-PowerShell ETW Data

The command below executes the SilkETW tool with specific options to perform event tracing and analysis on PowerShell-related events in Windows.

Note: Make sure you launch PowerShell as an administrator.

Hunting Evil with YARA (Windows Edition)

```
PS C:\Tools\SilkETW\v8\SilkETW> .\SilkETW.exe -t user -pn Microsoft-Windows-PowerShell -ot file -p
```



```
                [v0.8 - Ruben Boonen => @FuzzySec]

[+] Collector parameter validation success..
[>] Starting trace collector (Ctrl-c to stop)..
[?] Events captured: 0
```

Command Breakdown:

- `-t user`: Specifies the event tracing mode. In this case, it is set to "user," indicating that the tool will trace user-mode events (events generated by user applications).
- `-pn Microsoft-Windows-PowerShell`: Specifies the name of the provider or event log that you want to trace. In this command, it targets events from the "Microsoft-Windows-PowerShell" provider, which is responsible for generating events related to PowerShell activity.
- `-ot file`: Specifies the output format for the collected event data. In this case, it is set to "file," meaning that the tool will save the event data to a file.
- `-p ./etw_ps_logs.json`: Specifies the output file path and filename. The tool will save the collected event data in JSON format to a file named "etw_ps_logs.json" in the current directory.
- `-l verbose`: Sets the logging level to "verbose." This option enables more detailed logging information during the event tracing and analysis process.
- `-y C:\Rules\yara`: Enables YARA scanning and specifies a path containing YARA rules. This option indicates that the tool will perform YARA scanning on the collected event data.
- `-yo Matches`: Specifies the YARA output option. In this case, it is set to "Matches," meaning that the tool will display YARA matches found during the scanning process.

Inside the `C:\Rules\yara` directory of this section's target there is a YARA rules file named `etw_powershell_hello.yar` that looks for certain strings in PowerShell script blocks.

Code: yara

```
rule powershell_hello_world_yara {
    strings:
        $s0 = "Write-Host" ascii wide nocase
        $s1 = "Hello" ascii wide nocase
        $s2 = "from" ascii wide nocase
        $s3 = "PowerShell" ascii wide nocase
```

```
    condition:
        3 of ($s*)
}
```

Let's now execute the following PowerShell command through another PowerShell terminal and see if it will get detected by SilkETW (where the abovementioned YARA rule has been loaded).

Hunting Evil with YARA (Windows Edition)

```
PS C:\Users\htb-student> Invoke-Command -ScriptBlock {Write-Host "Hello from PowerShell"}
```

We have a match!

Hunting Evil with YARA (Windows Edition)

```
PS C:\Tools\SilkETW\v8\SilkETW> .\SilkETW.exe -t user -pn Microsoft-Windows-PowerShell -ot file -p

                [v0.8 - Ruben Boonen => @FuzzySec]

[+] Collector parameter validation success..
[>] Starting trace collector (Ctrl-c to stop)..
[?] Events captured: 28
     -> Yara match: powershell_hello_world_yara
     -> Yara match: powershell_hello_world_yara
```

Example 2: YARA Rule Scanning on Microsoft-Windows-DNS-Client ETW Data

The command below executes the SilkETW tool with specific options to perform event tracing and analysis on DNS-related events in Windows.

Hunting Evil with YARA (Windows Edition)

```
PS C:\Tools\SilkETW\v8\SilkETW> .\SilkETW.exe -t user -pn Microsoft-Windows-DNS-Client -ot file -p

                [v0.8 - Ruben Boonen => @FuzzySec]

[+] Collector parameter validation success..
[>] Starting trace collector (Ctrl-c to stop)..
[?] Events captured: 0
```

Inside the `C:\Rules\yara` directory of this section's target there is a YARA rules file named `etw_dns_wannacry.yar` that looks for a hardcoded domain that exists in Wannacry ransomware samples in DNS events.

Code: yara

```
rule dns_wannacry_domain {
    strings:
        $s1 = "iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com" ascii wide nocase
    condition:
        $s1
}
```

Let's now execute the following command through another PowerShell terminal and see if it will get detected by SilkETW (where the abovementioned YARA rule has been loaded).

Hunting Evil with YARA (Windows Edition)

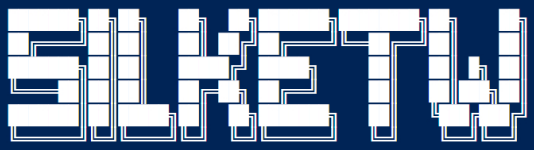
```
PS C:\Users\htb-student> ping iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com
Reply from 104.17.244.81: bytes=32 time=14ms TTL=56
Reply from 104.17.244.81: bytes=32 time=14ms TTL=56
Reply from 104.17.244.81: bytes=32 time=14ms TTL=56
Reply from 104.17.244.81: bytes=32 time=14ms TTL=56

Ping statistics for 104.17.244.81:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 14ms, Maximum = 14ms, Average = 14ms
```

We have a match!

Hunting Evil with YARA (Windows Edition)

```
PS C:\Tools\SilkETW\v8\SilkETW> .\SilkETW.exe -t user -pn Microsoft-Windows-DNS-Client -ot file -p
```



```
                [v0.8 - Ruben Boonen => @FuzzySec]

[+] Collector parameter validation success..
[>] Starting trace collector (Ctrl-c to stop)..
[?] Events captured: 60
     -> Yara match: dns_wannacry_domain
     -> Yara match: dns_wannacry_domain
```

VPN Servers

⚠ **Warning:** Each time you "Switch", your connection keys are regenerated and you must re-download your VPN connection file.

All VM instances associated with the old VPN Server will be terminated when switching to a new VPN server.

Existing PwnBox instances will automatically switch to the new VPN server.

US Academy 3 — Medium Load

PROTOCOL

UDP 1337 TCP 443

DOWNLOAD VPN CONNECTION FILE

Connect to Pwnbox

Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK — 161ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

Download VPN Connection File

Target(s): Click here to spawn the target system!

RDP to with user "htb-student" and password "HTB_@cademy_stdnt!"

+ 2 Study the "C:\Rules\yara\shell_detector.yar" YARA rule that aims to detect "C:\Samples\MalwareAnalysis\shell.exe" inside process memory. Then, specify the appropriate hex values inside the "$sandbox" variable to ensure that the "Sandbox detected" message will also be detected. Enter the correct hex values as your answer. Answer format: Remove any spaces

53616e64626f7820646574656374564

Submit

Previous Next

Mark Complete & Next