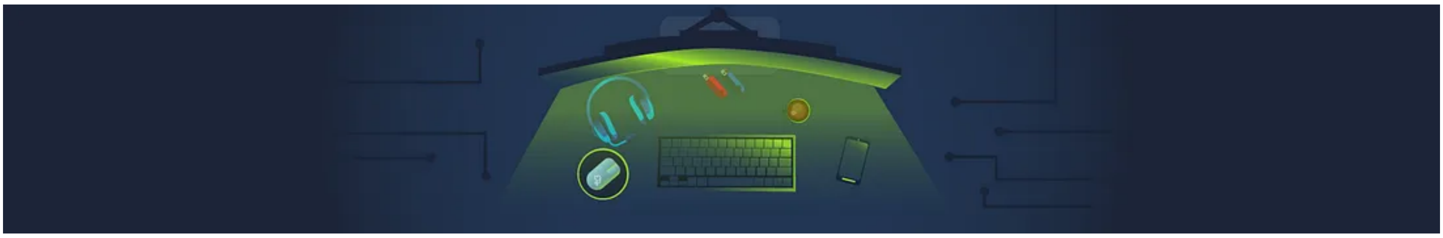


Volatility | TryHackMe — Walkthrough

jcm3 · Follow
16 min read · Mar 27, 2024

Listen Share More

Hey all, this is the forty-seventh installment in my walkthrough series on TryHackMe's SOC Level 1 path which covers the eighth room in this module on Digital Forensics and Incident Response, where we will come to understand what forensic artifacts are present in the Windows and Linux Operating Systems, how to collect them, and leverage them to investigate security incidents.

In this room, we will learn how to perform memory forensics with Volatility.

Link: https://tryhackme.com/room/volatility

Task 1: Introduction

Volatility is a free memory forensics tool developed and maintained by Volatility Foundation, commonly used by malware and SOC analysts within a blue team or as part of their detection and monitoring solutions. Volatility is written in Python and is made up of python plugins and modules designed as a plug-and-play way of analyzing memory dumps.

Volatility is available for Windows, Linux, and Mac OS and is written purely in Python.

This room uses memory dumps from THM rooms and memory samples from Volatility Foundation.

Before completing this room, we recommend completing the Core Windows Processes room.

If you plan on using your own machine or the AttackBox to run Volatility, download the files attached to this task. If you plan to use the provided machine, you can deploy it in Task 3.

Task 2: Volatility Overview

From the Volatility Foundation Wiki, “Volatility is the world’s most widely used framework for extracting digital artifacts from volatile memory (RAM) samples. The extraction techniques are performed completely independent of the system being investigated but offer visibility into the runtime state of the system. The framework is intended to introduce people to the techniques and complexities associated with extracting digital artifacts from volatile memory samples and provide a platform for further work into this exciting area of research.”

Volatility is built off of multiple plugins working together to obtain information from the memory dump. To begin analyzing a dump, you will first need to identify the image type; there are multiple ways of identifying this information that we will cover further in later tasks. Once you have your image type and other plugins sorted, you can then begin analyzing the dump by using various volatility plugins against it that will be covered in depth later in this room.

Since Volatility is entirely independent of the system under investigation, this allows complete segmentation but full insight into the runtime state of the system.

At the time of writing, there are two main repositories for Volatility; one built off of python 2 and another built off python 3. For this room, we recommend using the Volatility3 version build off of python 3.
https://github.com/volatilityfoundation/volatility3

Note: When reading blog posts and articles about Volatility, you may see volatility2 syntax mentioned or used, all syntax changed in volatility3, and within this room, we will be using the most recent version of the plugin syntax for Volatility.

Task 3: Installing Volatility

Since Volatility is written purely in Python, it makes the installation steps and requirements very easy and universal for Windows, Linux, and Mac. If you already attempted to use Python on Windows and Mac, it is suggested to begin on Linux; however, all operating systems will work the same.

If you're using TryHackMe's AttackBox, Volatility is already present on the box, and you can skip these steps and move on.

When downloading, you can make a choice to use the pre-packaged executable (.whl file) that will work the same and requires no dependencies (Windows Only), or you can decide to run it directly from Python.

To obtain a pre-packaged executable, simply download a zip file containing the application from their releases page.
https://github.com/volatilityfoundation/volatility3/releases/tag/v1.0.1

To begin running the project from source, you will need to first download the following dependencies: `Python 3.5.3 or later` and `Pefile 2017.8.1 or later`.
https://pypi.org/project/pefile/

You can also download these optional dependencies (recommended for this room): `yara-python 3.8.0 or later` https://github.com/VirusTotal/yara-python and `capstone 3.0.0 or later` https://www.capstone-engine.org/download.html.

Once you have your dependencies sorted, you can clone the repository from GitHub.

Command used: `git clone https://github.com/volatilityfoundation/volatility3.git`

You now have Volatility installed!

To test your installation run the `vol.py` file with the help parameter.

Command used: `python3 vol.py -h`

It is important to note that for any Linux or Mac memory files, you will need to download the symbol files from the Volatility GitHub.
https://github.com/volatilityfoundation/volatility3#symbol-tables

We have an Ubuntu machine with Volatility and Volatility 3 already present in the /opt directory, along with all the memory files you need throughout this room. The machine will start in a split-screen view. In case the VM is not visible, use the blue Show Split View button at the top-right of the page.

IP Address: `MACHINE_IP`

Username: `thmanalyst`

Password: `infected`

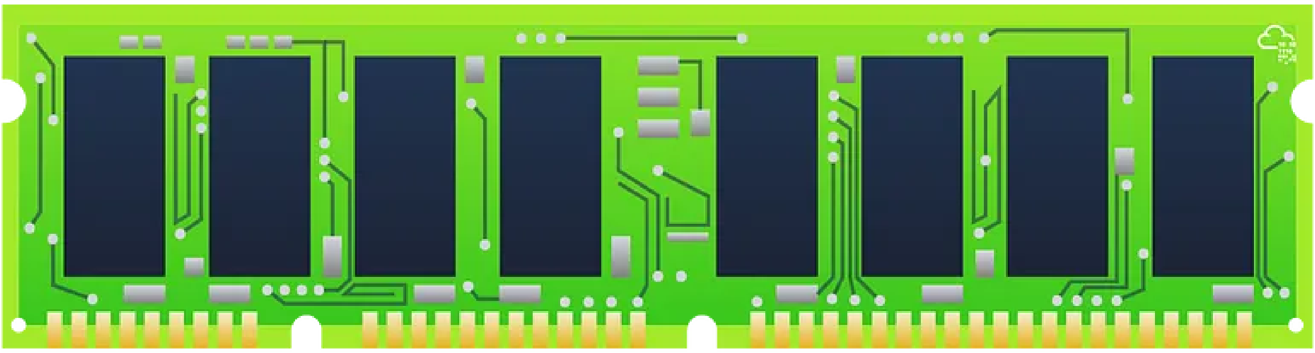
Task 4: Memory Extraction

Extracting a memory dump can be performed in numerous ways, varying based on the requirements of your investigation. Listed below are a few of the techniques and tools that can be used to extract a memory from a bare-metal machine.

- FTK Imager
- Redline
- DumpIt.exe
- win32dd.exe / win64dd.exe
- Memoryze
- FastDump

When using an extraction tool on a bare-metal host, it can usually take a considerable amount of time; take this into consideration during your investigation if time is a constraint.

Most of the tools mentioned above for memory extraction will output a .raw file with some exceptions like Redline that can use its own agent and session structure.



For virtual machines, gathering a memory file can easily be done by collecting the virtual memory file from the host machine's drive. This file can change depending on the hypervisor used; listed below are a few of the hypervisor virtual memory files you may encounter.

- VMWare — .vmem
- Hyper-V — .bin
- Parallels — .mem
- VirtualBox — .sav file **this is only a partial memory file*

Exercise caution whenever attempting to extract or move memory from both bare-metal and virtual machines.

Answer the questions below:

4.1 Read the above and understand the various methods of memory extraction.

> ***Answer:*** *No answer needed*

Task 5: Plugins Overview

Since converting to Python 3, the plugin structure for Volatility has changed quite drastically. In previous Volatility versions, you would need to identify a specific OS profile exact to the operating system and build version of the host, which could be hard to find or used with a plugin that could provide false positives. With Volatility3, profiles have been scrapped, and Volatility will automatically identify the host and build of the memory file.

The naming structure of plugins has also changed. In previous versions of Volatility, the naming convention has been simply the name of the plugin and was universal for all operating systems and profiles. Now with Volatility3, you need to specify the operating system prior to specifying the plugin to be used, for example, `windows.info` vs `linux.info`. This is because there are no longer profiles to distinguish between various operating systems for plugins as each operating system has drastically different memory structures and operations. Look below for options of operating system plugin syntax.

- .windows
- .linux
- .mac

There are several plugins available with Volatility as well as third-party plugins; we will only be covering a small portion of the plugins that Volatility has to offer.

To get familiar with the plugins available, utilize the help menu. As Volatility3 is currently in active development, there is still a short list of plugins compared to its python 2 counterpart; however, the current list still allows you to do all of your analysis as needed.

Answer the questions below:

5.1 Read the above and review the Volatility help menu.

> ***Answer:*** *No answer needed*

Task 6: Identifying Image Info and Profiles

By default, Volatility comes with all existing Windows profiles from Windows XP to Windows 10.

Image profiles can be hard to determine if you don't know exactly what version and build the machine you extracted a memory dump from was. In some cases, you may be given a memory file with no other context, and it is up to you to figure out where to go from there. In that case, Volatility has your back and comes with the `imageinfo` plugin. This plugin will take the provided memory dump and assign it a list of the best possible OS profiles. OS profiles have since been deprecated with Volatility3, so we will only need to worry about identifying the profile if using Volatility2; this makes life much easier for analyzing memory dumps.

Note: `imageinfo` is not always correct and can have varied results depending on the provided dump; use with caution and test multiple profiles from the provided list.

If we are still looking to get information about what the host is running from the memory dump, we can use the following three plugins `windows.info` `linux.info` `mac.info`. This plugin will provide information about the host from the memory dump.

Syntax: `python3 vol.py -f <file> windows.info`

Answer the questions below:

6.1 Read the above and move on to listing processes and connections.

> ***Answer:*** *No answer needed*

6.2 To practice any commands in this room you can utilize either of the memory files present in the /Scenarios/Investigations/ directory or downloaded from Task 1.

> ***Answer:*** *No answer needed*

Task 7: Listing Processes and Connections

Five different plugins within Volatility allow you to dump processes and network connections, each with varying techniques used. In this task, we will be discussing each and its pros and cons when it comes to evasion techniques used by adversaries.

The most basic way of listing processes is using `pslist`; this plugin will get the list of processes from the doubly-linked list that keeps track of processes in memory, equivalent to the process list in task manager. The output from this plugin will include all current processes and terminated processes with their exit times.

Syntax: `python3 vol.py -f <file> windows.pslist`

Some malware, typically rootkits, will, in an attempt to hide their processes, unlink itself from the list. By unlinking themselves from the list you will no longer see their processes when using `pslist`. To combat this evasion technique, we can use `psscan`;this technique of listing processes will locate processes by finding data structures that match `_EPROCESS`. While this technique can help with evasion countermeasures, it can also cause false positives.

Syntax: `python3 vol.py -f <file> windows.psscan`

The third process plugin, `pstree`, does not offer any other kind of special techniques to help identify evasion like the last two plugins; however, this plugin will list all processes based on their parent process ID, using the same methods as `pslist`. This can be useful for an analyst to get a full story of the processes and what may have been occurring at the time of extraction.

Syntax: `python3 vol.py -f <file> windows.pstree`

Now that we know how to identify processes, we also need to have a way to identify the network connections present at the time of extraction on the host machine. `netstat` will attempt to identify all memory structures with a network connection.

Syntax: `python3 vol.py -f <file> windows.netstat`

This command in the current state of volatility3 can be very unstable, particularly around old Windows builds. To combat this, you can utilize other tools like bulk_extractor to extract a PCAP file from the memory file. In some cases, this is preferred in network connections that you cannot identify from Volatility alone. https://tools.kali.org/forensics/bulk-extractor

The last plugin we will cover is `dlllist`. This plugin will list all DLLs associated with processes at the time of extraction. This can be especially useful once you have done further analysis and can filter output to a specific DLL that might be an indicator for a specific type of malware you believe to be present on the system.

Syntax: `python3 vol.py -f <file> windows.dlllist`

Answer the questions below:

7.1 Read the above and practice dumping processes and connections on the provided memory file.

> ***Answer:*** *No answer needed*

Task 8: Volatility Hunting and Detection Capabilities

Volatility offers a plethora of plugins that can be used to aid in your hunting and detection capabilities when hunting for malware or other anomalies within a system's memory.

It is recommended that you have a basic understanding of how evasion techniques and various malware techniques are employed by adversaries, as well as how to hunt and detect them before going through this section.

The first plugin we will be talking about that is one of the most useful when hunting for code injection is `malfind` . This plugin will attempt to identify injected processes and their PIDs along with the offset address and a Hex, Ascii, and Disassembly view of the infected area. The plugin works by scanning the heap and identifying processes that have the executable bit set `RWE or RX` and/or no memory-mapped file on disk (file-less malware).

Based on what `malfind` identifies, the injected area will change. An MZ header is an indicator of a Windows executable file. The injected area could also be directed towards shellcode which requires further analysis.

Syntax: `python3 vol.py -f <file> windows.malfind`

Volatility also offers the capability to compare the memory file against YARA rules. `yarascan` will search for strings, patterns, and compound rules against a rule set. You can either use a YARA file as an argument or list rules within the command line.

Syntax: `python3 vol.py -f <file> windows.yarascan`

There are other plugins that can be considered part of Volatility's hunting and detection capabilities; however, we will be covering them in the next task.

Answer the questions below:

8.1 Read the above and practice your hunting capabilities with Volatility.

> ***Answer:*** *No answer needed*

Task 9: Advanced Memory Forensics

Advanced Memory Forensics can become confusing when you begin talking about system objects and how malware interacts directly with the system, especially if you do not have prior experience hunting some of the techniques used such as hooking and driver manipulation. When dealing with an advanced adversary, you may encounter malware, most of the time rootkits that will employ very nasty evasion measures that will require you as an analyst to dive into the drivers, mutexes, and hooked functions. A number of modules can help us in this journey to further uncover malware hiding within memory.

The first evasion technique we will be hunting is hooking; there are five methods of hooking employed by adversaries, outlined below:

- SSDT Hooks
- IRP Hooks
- IAT Hooks
- EAT Hooks
- Inline Hooks

We will only be focusing on hunting SSDT hooking as this one of the most common techniques when dealing with malware evasion and the easiest plugin to use with the base volatility plugins.

The `ssdt` plugin will search for hooking and output its results. Hooking can be used by legitimate applications, so it is up to you as the analyst to identify what is evil. As a brief overview of what SSDT hooking is: `SSDT` stands for *System Service Descriptor Table;* the Windows kernel uses this table to look up system functions. An adversary can hook into this table and modify pointers to point to a location the rootkit controls.

There can be hundreds of table entries that `ssdt` will dump; you will then have to analyze the output further or compare against a baseline. A suggestion is to use this plugin after investigating the initial compromise and working off it as part of your lead investigation.

Syntax: `python3 vol.py -f <file> windows.ssdt`

Adversaries will also use malicious driver files as part of their evasion. Volatility offers two plugins to list drivers.

The `modules` plugin will dump a list of loaded kernel modules; this can be useful in identifying active malware. However, if a malicious file is idly waiting or hidden, this plugin may miss it.

This plugin is best used once you have further investigated and found potential indicators to use as input for searching and filtering.

Syntax: `python3 vol.py -f <file> windows.modules`

The `driverscan` plugin will scan for drivers present on the system at the time of extraction. This plugin can help to identify driver files in the kernel that the `modules` plugin might have missed or were hidden.

As with the last plugin, it is again recommended to have a prior investigation before moving on to this plugin. It is also recommended to look through the `modules` plugin before `driverscan`.

Syntax: `python3 vol.py -f <file> windows.driverscan`

In most cases, `driverscan` will come up with no output; however, if you do not find anything with the `modules` plugin, it can be useful to attempt using this plugin.

There are also other plugins listed below that can be helpful when attempting to hunt for advanced malware in memory.

- `modscan`
- `driverirp`
- `callbacks`

- `idt`
- `apihooks`
- `moddump`
- `handles`

Note: Some of these are only present on Volatility2 or are part of third-party plugins. To get the most out of Volatility, you may need to move to some third-party or custom plugins.

Answer the questions below:

9.1 Read the above and practice looking through the SSDT and drivers.

> ***Answer:*** *No answer needed*

9.2 Research the other plugins provided; some of them will be used within the practical investigations.

> ***Answer:*** *No answer needed*

Task 10: Practical Investigations

Case 001 — BOB! THIS ISN'T A HORSE!

Your SOC has informed you that they have gathered a memory dump from a quarantined endpoint thought to have been compromised by a banking trojan masquerading as an Adobe document. Your job is to use your knowledge of threat intelligence and reverse engineering to perform memory forensics on the infected host.

You have been informed of a suspicious IP in connection to the file that could be helpful. `41.168.5.140`

The memory file is located in `/Scenarios/Investigations/Investigation-1.vmem`

Case 002 — That Kind of Hurt my Feelings

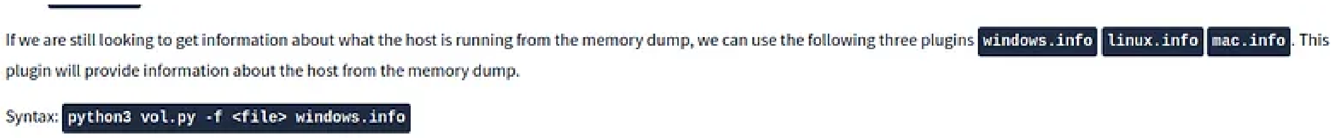
You have been informed that your corporation has been hit with a chain of ransomware that has been hitting corporations internationally. Your team has already retrieved the decryption key and recovered from the attack. Still, your job is to perform post-incident analysis and identify what actors were at play and what occurred on your systems. You have been provided with a raw memory dump from your team to begin your analysis.

The memory file is located in `/Scenarios/Investigations/Investigation-2.raw`

Answer the questions below:

10.1 What is the build version of the host machine in Case 001?

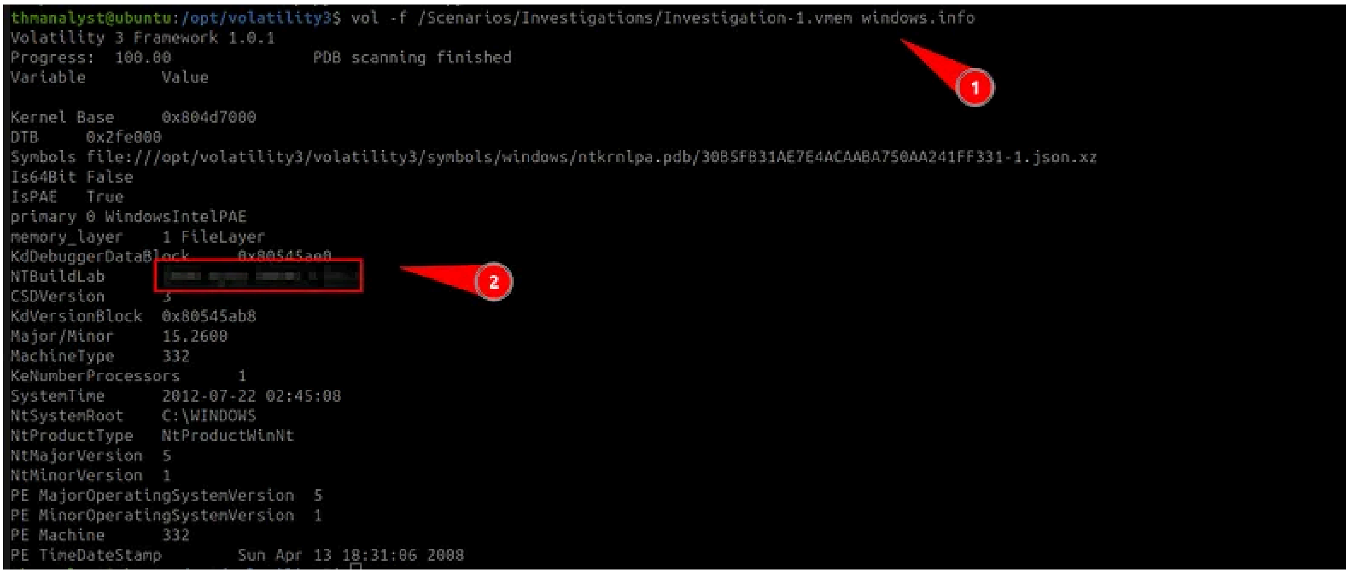
Per task 6, we can find this by running windows.info:



If we are still looking to get information about what the host is running from the memory dump, we can use the following three plugins `windows.info` `linux.info` `mac.info`. This plugin will provide information about the host from the memory dump.

Syntax: `python3 vol.py -f <file> windows.info`

```
vol -f /Scenarios/Investigations/Investigation-1.vmem windows.info
```

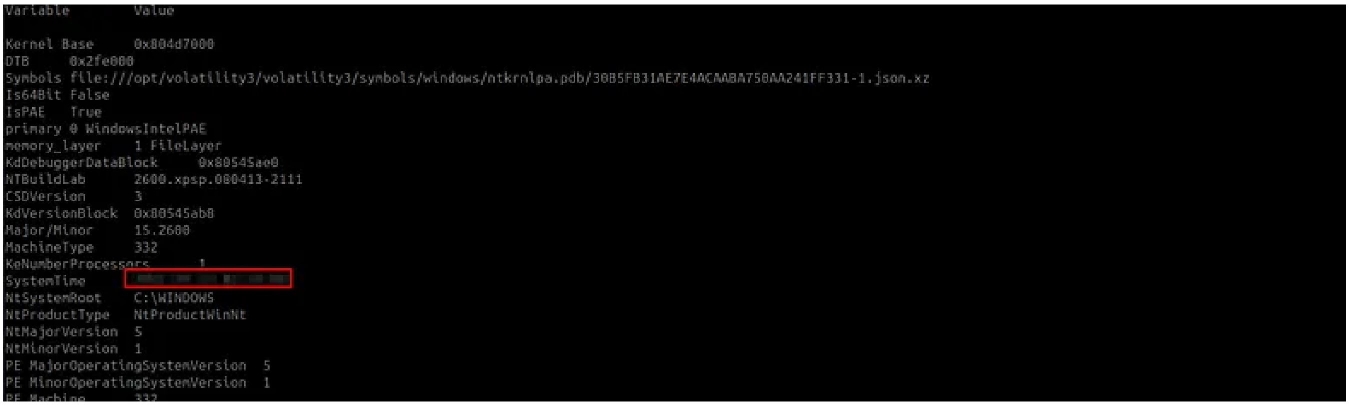


```
thmanalyst@ubuntu:/opt/volatility3$ vol -f /Scenarios/Investigations/Investigation-1.vmem windows.info
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
Variable        Value

Kernel Base     0x804d7000
DTB     0x2fe000
Symbols file:///opt/volatility3/volatility3/symbols/windows/ntkrnlpa.pdb/30B5FB31AE7E4ACAABA750AA241FF331-1.json.xz
Is64Bit False
IsPAE   True
primary 0 WindowsIntelPAE
memory_layer    1 FileLayer
KdDebuggerDataBlock     0x80545ae0
NTBuildLab      [redacted]
CSDVersion      3
KdVersionBlock  0x80545ab8
Major/Minor     15.2600
MachineType     332
KeNumberProcessors      1
SystemTime      2012-07-22 02:45:08
NtSystemRoot    C:\WINDOWS
NtProductType   NtProductWinNt
NtMajorVersion  5
NtMinorVersion  1
PE MajorOperatingSystemVersion  5
PE MinorOperatingSystemVersion  1
PE Machine      332
PE TimeDateStamp        Sun Apr 13 18:31:06 2008
```

> ***Answer:*** *2600.xpsp.080413–2111*

10.2 At what time was the memory file acquired in Case 001?

Same location as the previous question:



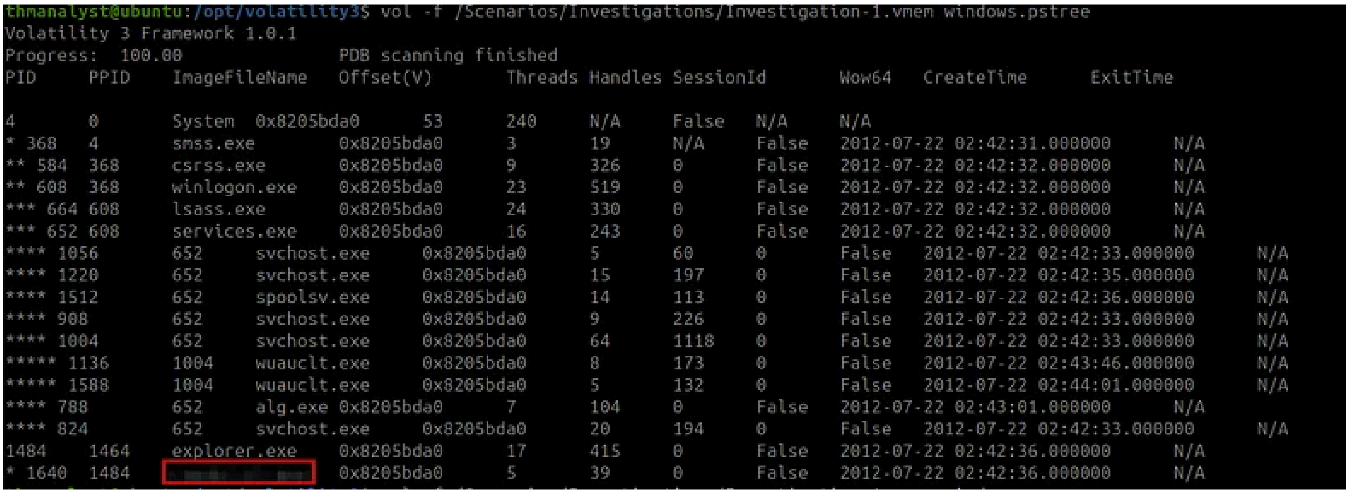
```
Variable        Value

Kernel Base     0x804d7000
DTB     0x2fe000
Symbols file:///opt/volatility3/volatility3/symbols/windows/ntkrnlpa.pdb/30B5FB31AE7E4ACAABA750AA241FF331-1.json.xz
Is64Bit False
IsPAE   True
primary 0 WindowsIntelPAE
memory_layer    1 FileLayer
KdDebuggerDataBlock     0x80545ae0
NTBuildLab      2600.xpsp.080413-2111
CSDVersion      3
KdVersionBlock  0x80545ab8
Major/Minor     15.2600
MachineType     332
KeNumberProcessors      1
SystemTime      [redacted]
NtSystemRoot    C:\WINDOWS
NtProductType   NtProductWinNt
NtMajorVersion  5
NtMinorVersion  1
PE MajorOperatingSystemVersion  5
PE MinorOperatingSystemVersion  1
PE Machine      332
```

> ***Answer:*** *2012–07–22 02:45:08*

10.3 What process can be considered suspicious in Case 001?
Note: Certain special characters may not be visible on the provided VM. When doing a copy-and-paste, it will still copy all characters.

Using pstree command, you see one service that is non-essential in the list:

```
vol -f /Scenarios/Investigations/Investigation-1.vmem windows.pstree
```

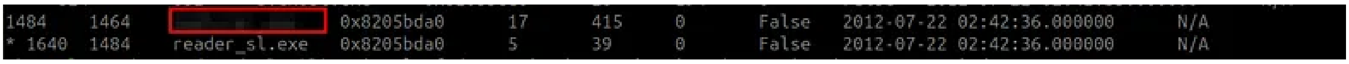


```
thmanalyst@ubuntu:/opt/volatility3$ vol -f /Scenarios/Investigations/Investigation-1.vmem windows.pstree
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime

4       0       System  0x8205bda0      53      240     N/A     False   N/A     N/A
* 368   4       smss.exe        0x8205bda0      3       19      N/A     False   2012-07-22 02:42:31.000000      N/A
** 584  368     csrss.exe       0x8205bda0      9       326     0       False   2012-07-22 02:42:32.000000      N/A
** 608  368     winlogon.exe    0x8205bda0      23      519     0       False   2012-07-22 02:42:32.000000      N/A
*** 664 608     lsass.exe       0x8205bda0      24      330     0       False   2012-07-22 02:42:32.000000      N/A
*** 652 608     services.exe    0x8205bda0      16      243     0       False   2012-07-22 02:42:32.000000      N/A
**** 1056       652     svchost.exe     0x8205bda0      5       60      0       False   2012-07-22 02:42:33.000000      N/A
**** 1220       652     svchost.exe     0x8205bda0      15      197     0       False   2012-07-22 02:42:35.000000      N/A
**** 1512       652     spoolsv.exe     0x8205bda0      14      113     0       False   2012-07-22 02:42:36.000000      N/A
**** 908        652     svchost.exe     0x8205bda0      9       226     0       False   2012-07-22 02:42:33.000000      N/A
**** 1004       652     svchost.exe     0x8205bda0      64      1118    0       False   2012-07-22 02:42:33.000000      N/A
***** 1136      1004    wuauclt.exe     0x8205bda0      8       173     0       False   2012-07-22 02:43:46.000000      N/A
***** 1588      1004    wuauclt.exe     0x8205bda0      5       132     0       False   2012-07-22 02:44:01.000000      N/A
**** 788        652     alg.exe 0x8205bda0      7       104     0       False   2012-07-22 02:43:01.000000      N/A
**** 824        652     svchost.exe     0x8205bda0      20      194     0       False   2012-07-22 02:42:33.000000      N/A
1484    1464    explorer.exe    0x8205bda0      17      415     0       False   2012-07-22 02:42:36.000000      N/A
* 1640  1484    [redacted]      0x8205bda0      5       39      0       False   2012-07-22 02:42:36.000000      N/A
```

> ***Answer:*** *reader_sl.exe*

10.4 What is the parent process of the suspicious process in Case 001?

This is easy given that's what we used to answer the previous question:



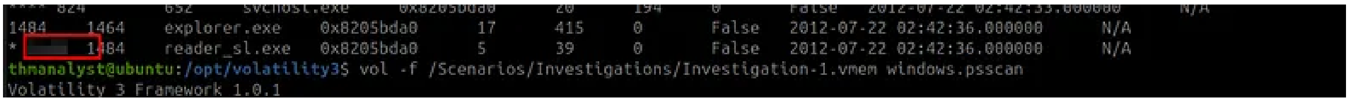
```
1484    1464    [redacted]    0x8205bda0    17    415    0    False    2012-07-22 02:42:36.000000    N/A
* 1640  1484    reader_sl.exe  0x8205bda0    5     39     0    False    2012-07-22 02:42:36.000000    N/A
```

> ***Answer:*** *explorer.exe*

10.5 What is the PID of the suspicious process in Case 001?

Same command as the previous two questions:

```
vol -f /Scenarios/Investigations/Investigation-1.vmem windows.pstree
```

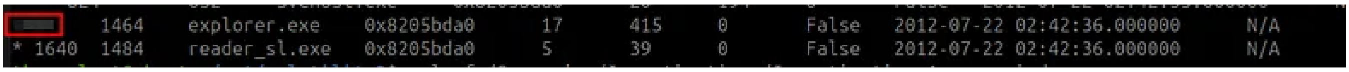


```
**** 824    652     svchost.exe    0x8205bda0    20    194    0    False    2012-07-22 02:42:33.000000    N/A
1484    1464    explorer.exe    0x8205bda0    17    415    0    False    2012-07-22 02:42:36.000000    N/A
* [redacted] 1484    reader_sl.exe    0x8205bda0    5    39    0    False    2012-07-22 02:42:36.000000    N/A
thmanalyst@ubuntu:/opt/volatility3$ vol -f /Scenarios/Investigations/Investigation-1.vmem windows.psscan
Volatility 3 Framework 1.0.1
```

> ***Answer:*** *1640*

10.6 What is the parent process PID in Case 001?

Same command again.



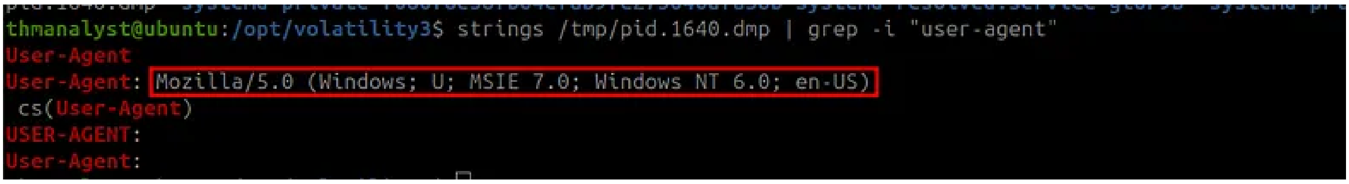
```
[redacted]    1464    explorer.exe    0x8205bda0    17    415    0    False    2012-07-22 02:42:36.000000    N/A
* 1640  1484    reader_sl.exe   0x8205bda0    5     39     0    False    2012-07-22 02:42:36.000000    N/A
```

> ***Answer:*** *1484*

10.7 What user-agent was employed by the adversary in Case 001?

I had to use the hint here, not sure I would have been able to figure it out otherwise, bashed my head against it for a while:

```
vol -f /Scenarios/Investigations/Investigation-1.vmem -o /tmp windows.memmap.Me
strings /tmp/pid.1640.dmp | grep -i "user-agent"
```

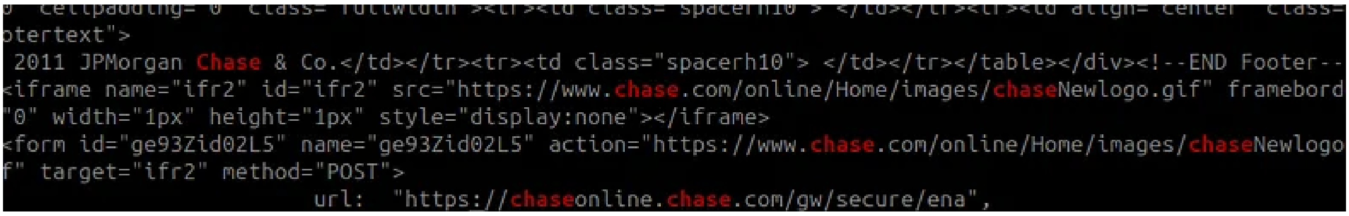


```
thmanalyst@ubuntu:/opt/volatility3$ strings /tmp/pid.1640.dmp | grep -i "user-agent"
User-Agent
User-Agent: Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)
 cs(User-Agent)
USER-AGENT:
User-Agent:
```

> ***Answer:*** *Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)*

10.8 Was Chase Bank one of the suspicious bank domains found in Case 001? (Y/N)

Searching for strings with chase reveals many hits:



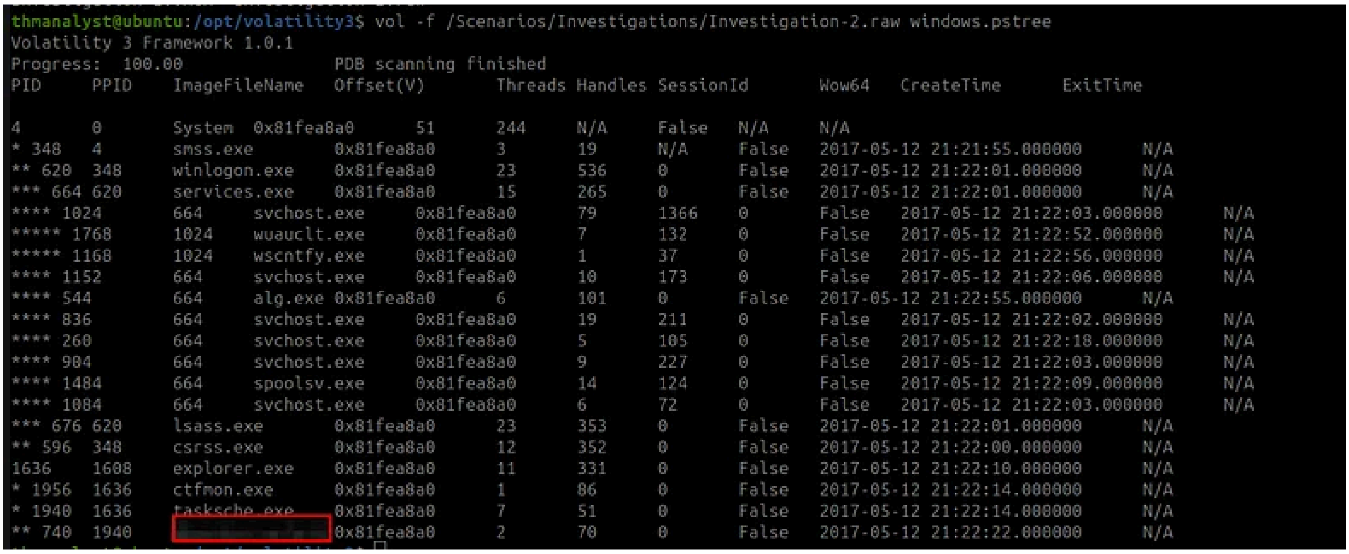
```
0" cellpadding="0" class="fullwidth"><tr><td class="spacerh10"> </td></tr><tr><td align="center" class=
otertext">
 2011 JPMorgan Chase & Co.</td></tr><tr><td class="spacerh10"> </td></tr></table></div><!--END Footer--
<iframe name="ifr2" id="ifr2" src="https://www.chase.com/online/Home/images/chaseNewlogo.gif" framebord
"0" width="1px" height="1px" style="display:none"></iframe>
<form id="ge93Zid02L5" name="ge93Zid02L5" action="https://www.chase.com/online/Home/images/chaseNewlogo
f" target="ifr2" method="POST">
                        url:  "https://chaseonline.chase.com/gw/secure/ena",
```

> ***Answer:*** *y*

10.9 What suspicious process is running at PID 740 in Case 002?

Run pstree on the second file:

```
vol -f /Scenarios/Investigations/Investigation-2.raw windows.pstree
```



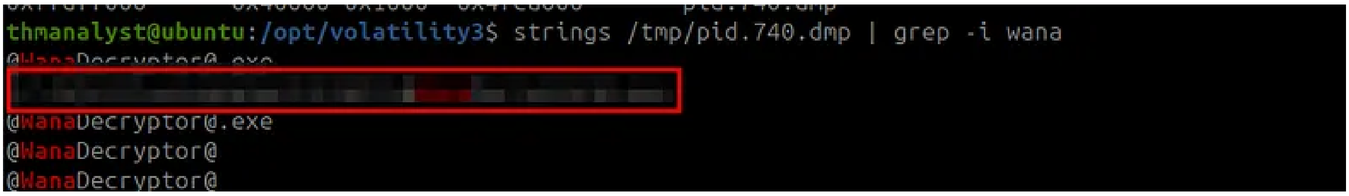
```
thmanalyst@ubuntu:/opt/volatility3$ vol -f /Scenarios/Investigations/Investigation-2.raw windows.pstree
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime

4       0       System  0x81fea8a0      51      244     N/A     False   N/A     N/A
* 348   4       smss.exe        0x81fea8a0      3       19      N/A     False   2017-05-12 21:21:55.000000      N/A
** 620  348     winlogon.exe    0x81fea8a0      23      536     0       False   2017-05-12 21:22:01.000000      N/A
*** 664 620     services.exe    0x81fea8a0      15      265     0       False   2017-05-12 21:22:01.000000      N/A
**** 1024       664     svchost.exe     0x81fea8a0      79      1366    0       False   2017-05-12 21:22:03.000000      N/A
***** 1768      1024    wuauclt.exe     0x81fea8a0      7       132     0       False   2017-05-12 21:22:52.000000      N/A
***** 1168      1024    wscntfy.exe     0x81fea8a0      1       37      0       False   2017-05-12 21:22:56.000000      N/A
**** 1152       664     svchost.exe     0x81fea8a0      10      173     0       False   2017-05-12 21:22:06.000000      N/A
**** 544        664     alg.exe 0x81fea8a0      6       101     0       False   2017-05-12 21:22:55.000000      N/A
**** 836        664     svchost.exe     0x81fea8a0      19      211     0       False   2017-05-12 21:22:02.000000      N/A
**** 260        664     svchost.exe     0x81fea8a0      5       105     0       False   2017-05-12 21:22:18.000000      N/A
**** 904        664     svchost.exe     0x81fea8a0      9       227     0       False   2017-05-12 21:22:03.000000      N/A
**** 1484       664     spoolsv.exe     0x81fea8a0      14      124     0       False   2017-05-12 21:22:09.000000      N/A
**** 1084       664     svchost.exe     0x81fea8a0      6       72      0       False   2017-05-12 21:22:03.000000      N/A
*** 676 620     lsass.exe       0x81fea8a0      23      353     0       False   2017-05-12 21:22:01.000000      N/A
** 596  348     csrss.exe       0x81fea8a0      12      352     0       False   2017-05-12 21:22:00.000000      N/A
1636    1608    explorer.exe    0x81fea8a0      11      331     0       False   2017-05-12 21:22:10.000000      N/A
* 1956  1636    ctfmon.exe      0x81fea8a0      1       86      0       False   2017-05-12 21:22:14.000000      N/A
* 1940  1636    tasksche.exe    0x81fea8a0      7       51      0       False   2017-05-12 21:22:14.000000      N/A
** 740  1940    [redacted]0x81fea8a0      2       70      0       False   2017-05-12 21:22:22.000000      N/A
```

> ***Answer:*** *@WanaDecryptor@*

10.10 What is the full path of the suspicious binary in PID 740 in Case 002?

I did a process dump on PID 740 as we did in question number 7 but on the second file:

```
vol -f /Scenarios/Investigations/Investigation-2.raw -o /tmp windows.memmap.Mem
strings /tmp/pid.740.dmp | grep -i wana
```

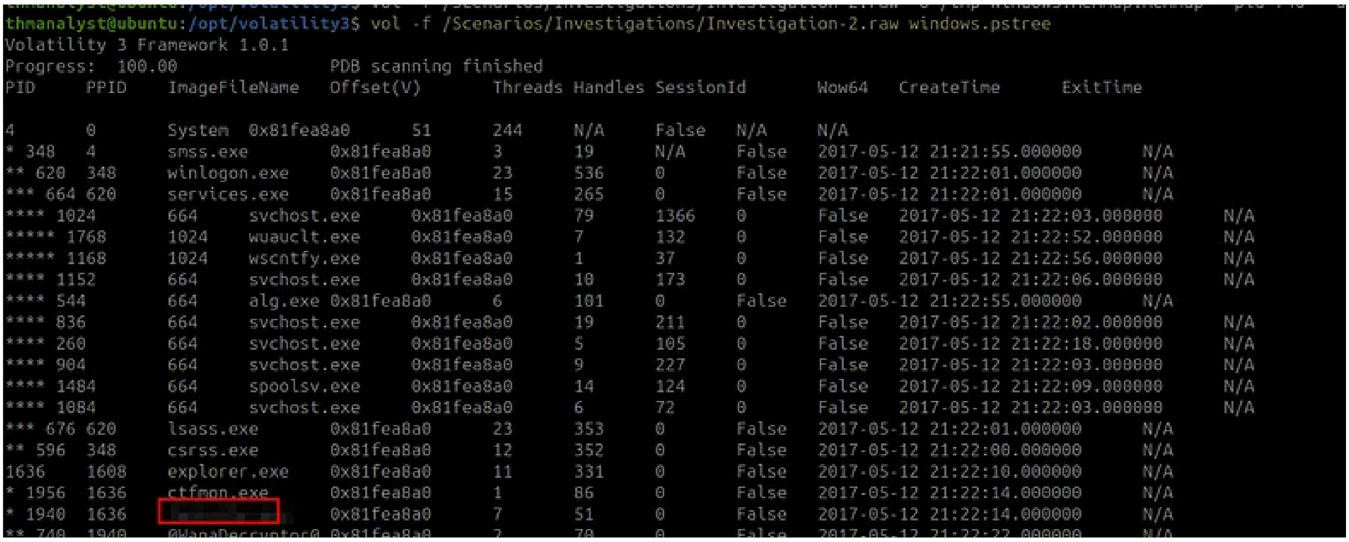


```
thmanalyst@ubuntu:/opt/volatility3$ strings /tmp/pid.740.dmp | grep -i wana
@WanaDecryptor@.exe
[redacted]
@WanaDecryptor@.exe
@WanaDecryptor@
@WanaDecryptor@
```

> ***Answer:*** *C:\Intel\ivecuqmanpnirkt615\@WanaDecryptor@.exe*

10.11 What is the parent process of PID 740 in Case 002?

Run a pstree on it:

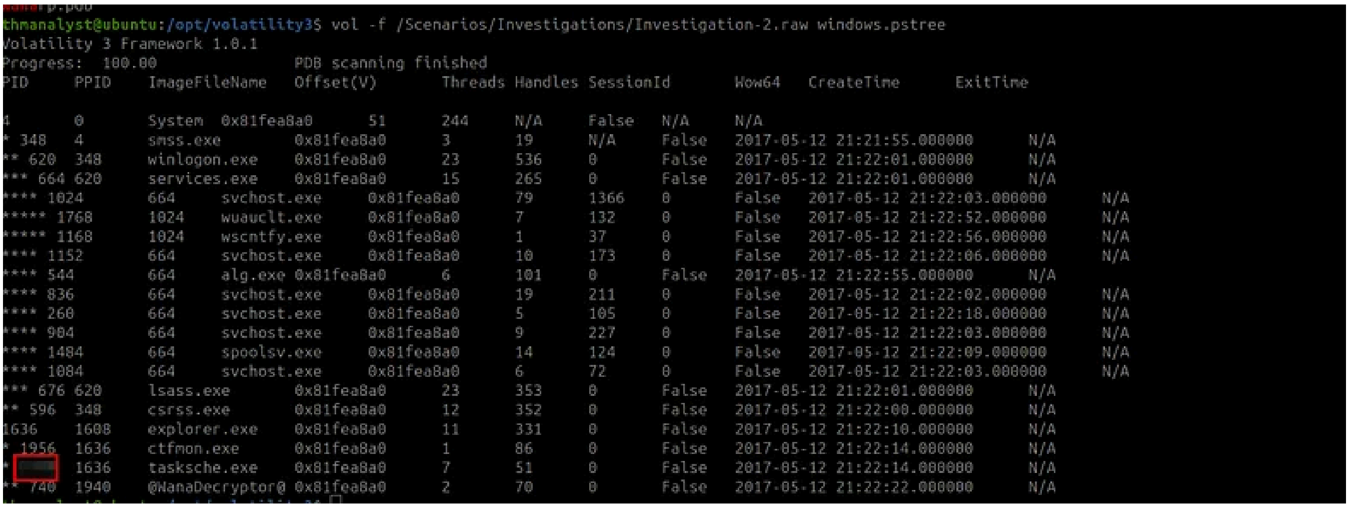


```
thmanalyst@ubuntu:/opt/volatility3$ vol -f /Scenarios/Investigations/Investigation-2.raw windows.pstree
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime

4       0       System  0x81fea8a0      51      244     N/A     False   N/A     N/A
* 348   4       smss.exe        0x81fea8a0      3       19      N/A     False   2017-05-12 21:21:55.000000      N/A
** 620  348     winlogon.exe    0x81fea8a0      23      536     0       False   2017-05-12 21:22:01.000000      N/A
*** 664 620     services.exe    0x81fea8a0      15      265     0       False   2017-05-12 21:22:01.000000      N/A
**** 1024       664     svchost.exe     0x81fea8a0      79      1366    0       False   2017-05-12 21:22:03.000000      N/A
***** 1768      1024    wuauclt.exe     0x81fea8a0      7       132     0       False   2017-05-12 21:22:52.000000      N/A
***** 1168      1024    wscntfy.exe     0x81fea8a0      1       37      0       False   2017-05-12 21:22:56.000000      N/A
**** 1152       664     svchost.exe     0x81fea8a0      10      173     0       False   2017-05-12 21:22:06.000000      N/A
**** 544        664     alg.exe 0x81fea8a0      6       101     0       False   2017-05-12 21:22:55.000000      N/A
**** 836        664     svchost.exe     0x81fea8a0      19      211     0       False   2017-05-12 21:22:02.000000      N/A
**** 260        664     svchost.exe     0x81fea8a0      5       105     0       False   2017-05-12 21:22:18.000000      N/A
**** 904        664     svchost.exe     0x81fea8a0      9       227     0       False   2017-05-12 21:22:03.000000      N/A
**** 1484       664     spoolsv.exe     0x81fea8a0      14      124     0       False   2017-05-12 21:22:09.000000      N/A
**** 1084       664     svchost.exe     0x81fea8a0      6       72      0       False   2017-05-12 21:22:03.000000      N/A
*** 676 620     lsass.exe       0x81fea8a0      23      353     0       False   2017-05-12 21:22:01.000000      N/A
** 596  348     csrss.exe       0x81fea8a0      12      352     0       False   2017-05-12 21:22:00.000000      N/A
1636    1608    explorer.exe    0x81fea8a0      11      331     0       False   2017-05-12 21:22:10.000000      N/A
* 1956  1636    ctfmon.exe      0x81fea8a0      1       86      0       False   2017-05-12 21:22:14.000000      N/A
* 1940  1636    [redacted]      0x81fea8a0      7       51      0       False   2017-05-12 21:22:14.000000      N/A
** 740  1940    @WanaDecryptor@ 0x81fea8a0      2       70      0       False   2017-05-12 21:22:22.000000      N/A
```

> ***Answer:*** *tasksche.exe*

10.12 What is the suspicious parent process PID connected to the decryptor in Case 002?

It’s in the same screen shot:



```
thmanalyst@ubuntu:/opt/volatility3$ vol -f /Scenarios/Investigations/Investigation-2.raw windows.pstree
Volatility 3 Framework 1.0.1
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(V)       Threads Handles SessionId       Wow64   CreateTime      ExitTime

4       0       System  0x81fea8a0      51      244     N/A     False   N/A     N/A
* 348   4       smss.exe        0x81fea8a0      3       19      N/A     False   2017-05-12 21:21:55.000000      N/A
** 620  348     winlogon.exe    0x81fea8a0      23      536     0       False   2017-05-12 21:22:01.000000      N/A
*** 664 620     services.exe    0x81fea8a0      15      265     0       False   2017-05-12 21:22:01.000000      N/A
**** 1024       664     svchost.exe     0x81fea8a0      79      1366    0       False   2017-05-12 21:22:03.000000      N/A
***** 1768      1024    wuauclt.exe     0x81fea8a0      7       132     0       False   2017-05-12 21:22:52.000000      N/A
***** 1168      1024    wscntfy.exe     0x81fea8a0      1       37      0       False   2017-05-12 21:22:56.000000      N/A
**** 1152       664     svchost.exe     0x81fea8a0      10      173     0       False   2017-05-12 21:22:06.000000      N/A
**** 544        664     alg.exe 0x81fea8a0      6       101     0       False   2017-05-12 21:22:55.000000      N/A
**** 836        664     svchost.exe     0x81fea8a0      19      211     0       False   2017-05-12 21:22:02.000000      N/A
**** 260        664     svchost.exe     0x81fea8a0      5       105     0       False   2017-05-12 21:22:18.000000      N/A
**** 904        664     svchost.exe     0x81fea8a0      9       227     0       False   2017-05-12 21:22:03.000000      N/A
**** 1484       664     spoolsv.exe     0x81fea8a0      14      124     0       False   2017-05-12 21:22:09.000000      N/A
**** 1084       664     svchost.exe     0x81fea8a0      6       72      0       False   2017-05-12 21:22:03.000000      N/A
*** 676 620     lsass.exe       0x81fea8a0      23      353     0       False   2017-05-12 21:22:01.000000      N/A
** 596  348     csrss.exe       0x81fea8a0      12      352     0       False   2017-05-12 21:22:00.000000      N/A
1636    1608    explorer.exe    0x81fea8a0      11      331     0       False   2017-05-12 21:22:10.000000      N/A
* 1956  1636    ctfmon.exe      0x81fea8a0      1       86      0       False   2017-05-12 21:22:14.000000      N/A
* [redacted]    1636    tasksche.exe    0x81fea8a0      7       51      0       False   2017-05-12 21:22:14.000000      N/A
** 740  1940    @WanaDecryptor@ 0x81fea8a0      2       70      0       False   2017-05-12 21:22:22.000000      N/A
```

> ***Answer:*** *1940*

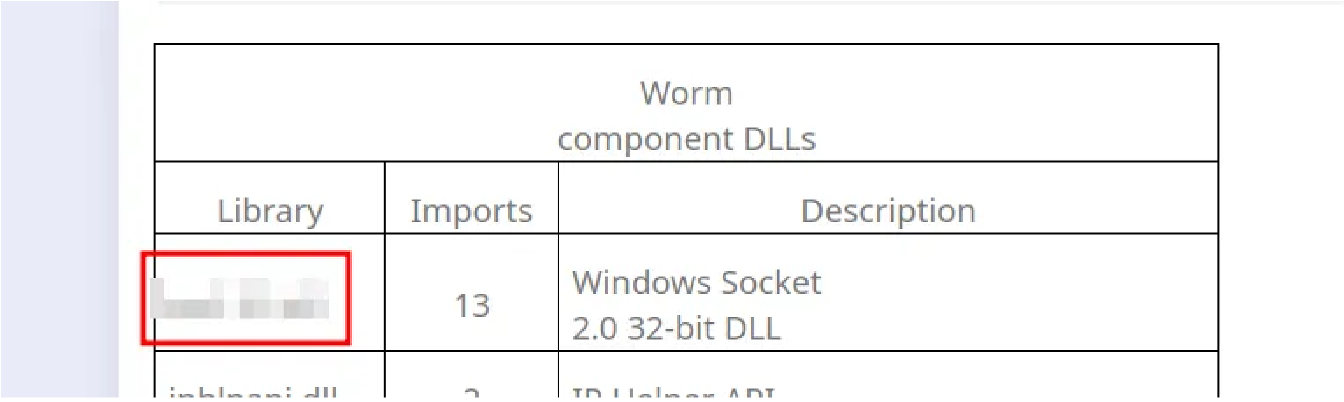
10.13 From our current information, what malware is present on the system in Case 002?

Does it really need an explanation…

> ***Answer:*** *WannaCry*

10.14 What DLL is loaded by the decryptor used for socket creation in Case 002?

I did some Google-Fu to find this and ended up locating the answer on this site.

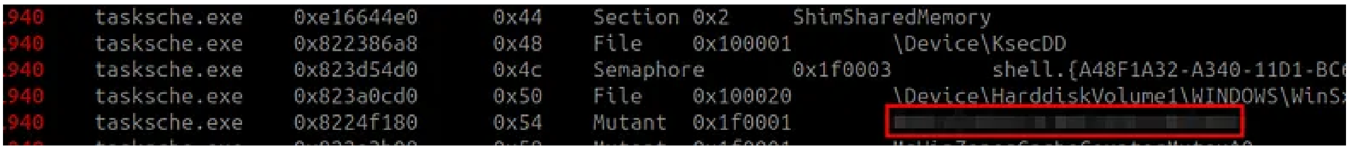


Worm component DLLs		
Library	Imports	Description
[redacted]	13	Windows Socket 2.0 32-bit DLL
iphlpapi.dll	2	IP Helper API

> ***Answer:*** *ws2_32.dll*

10.15 What mutex can be found that is a known indicator of the malware in question in Case 002?

We can find this using windows.handles per the hint:

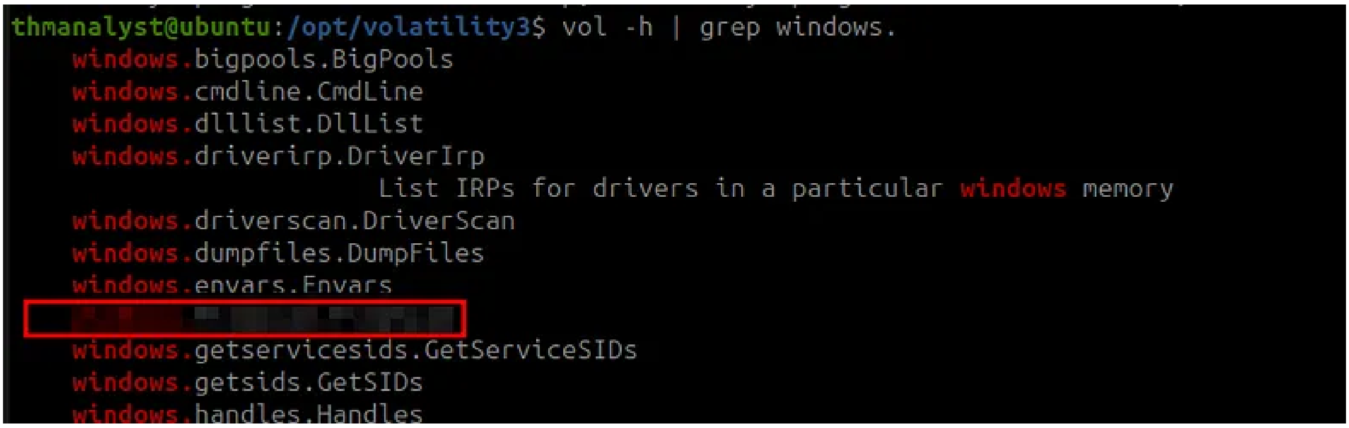


```
940   tasksche.exe   0xe16644e0   0x44   Section 0x2     ShimSharedMemory
940   tasksche.exe   0x822386a8   0x48   File    0x100001      \Device\KsecDD
940   tasksche.exe   0x823d54d0   0x4c   Semaphore      0x1f0003      shell.{A48F1A32-A340-11D1-BC6
940   tasksche.exe   0x823a0cd0   0x50   File    0x100020      \Device\HarddiskVolume1\WINDOWS\WinS
940   tasksche.exe   0x8224f180   0x54   Mutant  0x1f0001      [redacted]
```

> ***Answer:*** *MsWinZonesCacheCounterMutexA*

10.16 What plugin could be used to identify all files loaded from the malware working directory in Case 002?

For this I reviewed the help menu and grepped "windows." into it. The answer stands out:



```
thmanalyst@ubuntu:/opt/volatility3$ vol -h | grep windows.
    windows.bigpools.BigPools
    windows.cmdline.CmdLine
    windows.dlllist.DllList
    windows.driverirp.DriverIrp
                        List IRPs for drivers in a particular windows memory
    windows.driverscan.DriverScan
    windows.dumpfiles.DumpFiles
    windows.envars.Envars
    [redacted]
    windows.getservicesids.GetServiceSIDs
    windows.getsids.GetSIDs
    windows.handles.Handles
```

> ***Answer:*** *windows.filescan*

Task 11: Conclusion

We have only covered a very thin layer of memory forensics that can go much deeper when analyzing the Windows, Mac, and Linux architecture. If you're looking for a deep dive into memory forensics, I would suggest reading: The Art of Memory Forensics.

There are also a number of wikis and various community resources that can be used for more information about Volatility techniques found below.

- https://github.com/volatilityfoundation/volatility/wiki
- https://github.com/volatilityfoundation/volatility/wiki/Volatility-Documentation-Projec

- https://digital-forensics.sans.org/media/Poster-2015-Memory-Forensics.pdf
- https://eforensicsmag.com/finding-advanced-malware-using-volatility/

From this room, as you continue on the SOC Level 1 path, more rooms will contain memory forensics challenges.

This concludes the Volatility room on TryHackMe. Good room, I'm always a sucker for a robust tool with a CLI interface.

Thanks for joining me on this walkthrough and I'll see you in the next one where we dabble with endpoint monitoring, digital forensics and cyber response with **Velociraptor**.

Volatility Cybersecurity Tryhackme Tryhackme Walkthrough Soc Analyst

Follow

Written by jcm3

110 Followers · 9 Following

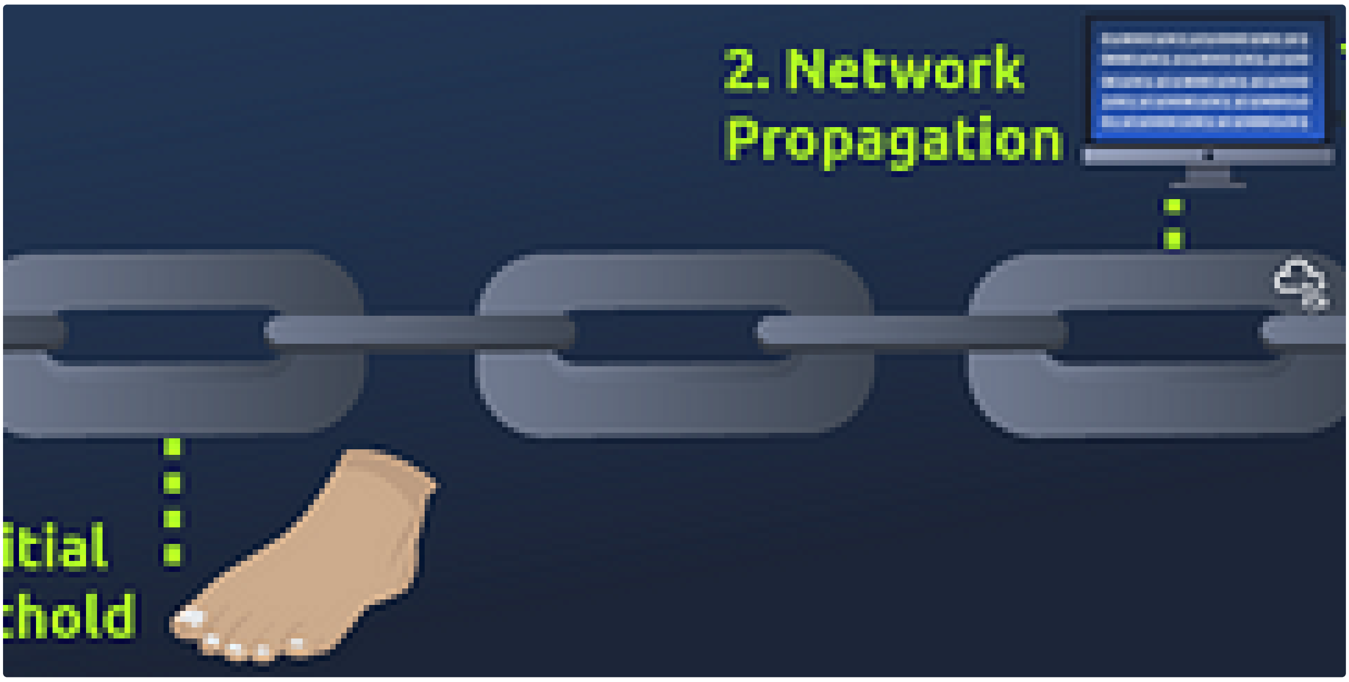
Proud dad, WGU cybersecurity grad, future MS:Cybersecurity & Information Assurance, aspiring cybersecurity professional, top 2% on TryHackMe.

No responses yet

What are your thoughts?

Respond

More from jcm3



jcm3

Unified Kill Chain | TryHackMe — Walkthrough

Hey all, this the is fourth installment in my walkthrough series covering TryHackMe's SOC Level 1 path and the fourth room in this module...

Feb 12, 2024 53

Open in app ↗

Medium Search

jcm3

Wireshark: Packet Operations | TryHackMe — Walkthrough

Hey all, this is the twenty-second installment in my walkthrough series on TryHackMe's SOC Level 1 path and the tenth room in this module...

Feb 29, 2024 69 1

jcm3

KAPE | TryHackMe — Walkthrough

Hey all, this is the forty-sixth installment in my walkthrough series on TryHackMe's SOC Level 1 path which covers the sixth room in this...

Mar 25, 2024 65 1

jcm3

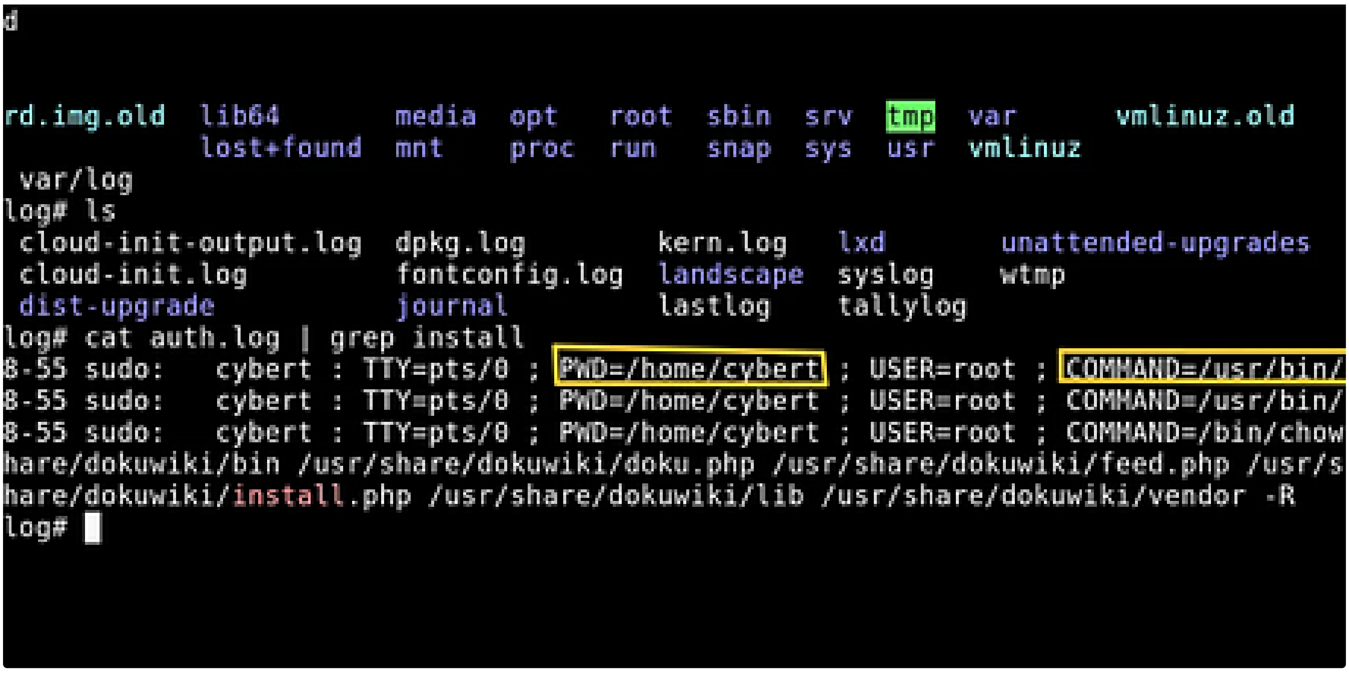
Sysinternals | TryHackMe — Walkthrough

Hey all, this is the twenty-seventh installment in my walkthrough series on TryHackMe's SOC Level 1 path which covers the third room in…

Mar 5, 2024 69

See all from jcm3

Recommended from Medium



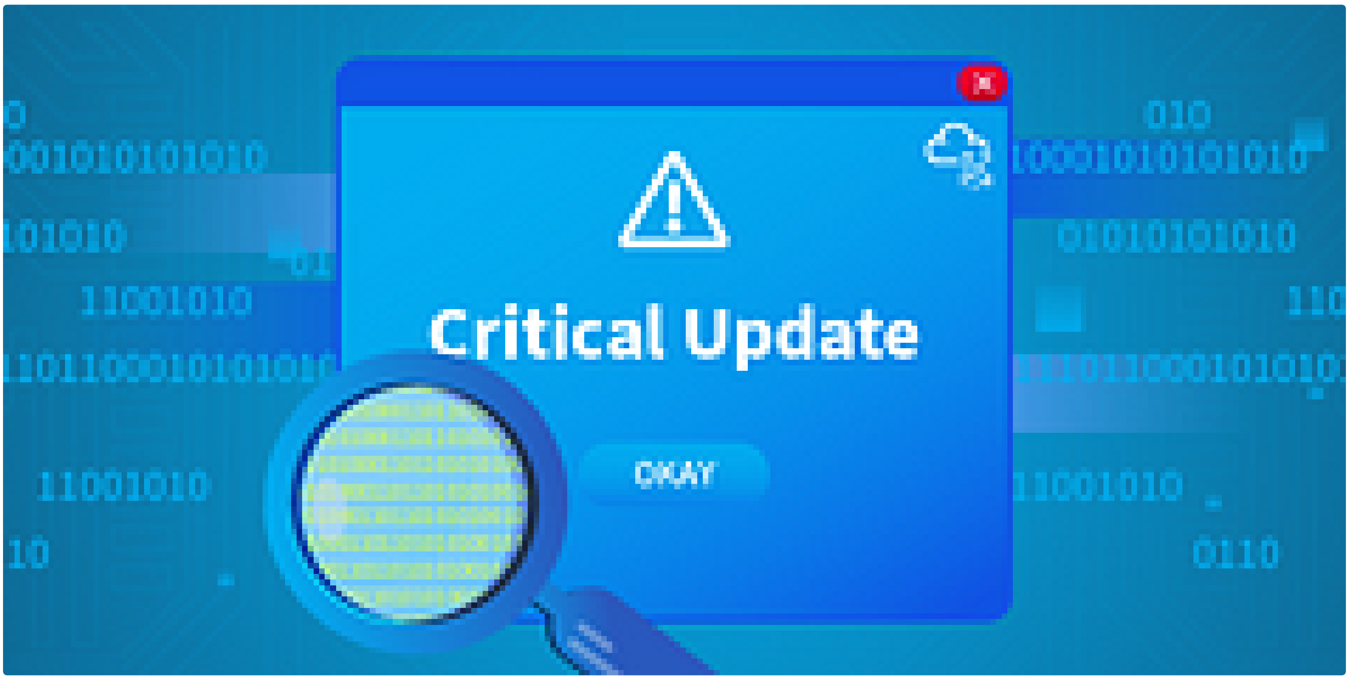
```
d


rd.img.old  lib64        media  opt   root  sbin  srv  tmp  var       vmlinuz.old
            lost+found   mnt    proc  run   snap  sys  usr  vmlinuz
 var/log
log# ls
 cloud-init-output.log  dpkg.log         kern.log   lxd       unattended-upgrades
 cloud-init.log         fontconfig.log   landscape  syslog    wtmp
 dist-upgrade           journal          lastlog    tallylog
log# cat auth.log | grep install
8-55 sudo:   cybert : TTY=pts/0 ; PWD=/home/cybert ; USER=root ; COMMAND=/usr/bin/
8-55 sudo:   cybert : TTY=pts/0 ; PWD=/home/cybert ; USER=root ; COMMAND=/usr/bin/
8-55 sudo:   cybert : TTY=pts/0 ; PWD=/home/cybert ; USER=root ; COMMAND=/bin/chow
hare/dokuwiki/bin /usr/share/dokuwiki/doku.php /usr/share/dokuwiki/feed.php /usr/s
hare/dokuwiki/install.php /usr/share/dokuwiki/lib /usr/share/dokuwiki/vendor -R
log# 
```

Dan Molina

Disgruntled CTF Walkthrough

This is a great CTF on TryHackMe that can be accessed through this link here: https://tryhackme.com/room/disgruntled

Oct 22, 2024



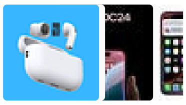
In T3CH by Axoloth

TryHackMe | Critical | WriteUp

Acquire the basic skills to analyze a memory dump in a practical scenario.

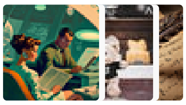
Jul 21, 2024 104

Lists



Tech & Tools

22 stories · 380 saves

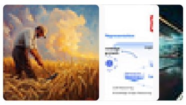


Medium's Huge List of Publications Accepting Submissions

377 stories · 4345 saves

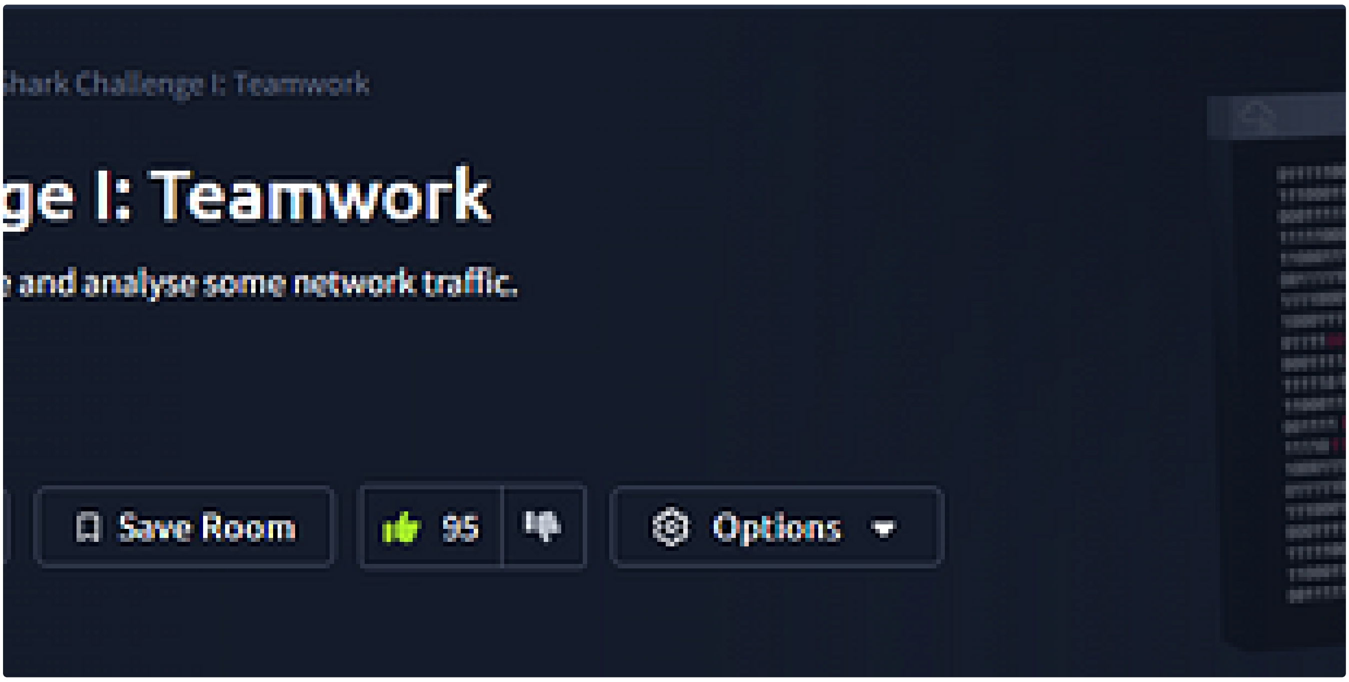
Staff picks

796 stories · 1561 saves



Natural Language Processing

1884 stories · 1529 saves



hark Challenge I: Teamwork

ge I: Teamwork

and analyse some network traffic.

Abhijeet Singh

TShark Challenge I: Teamwork | SOC Level 1 | TryHackMe Walkthrough

Task 1 - Introduction

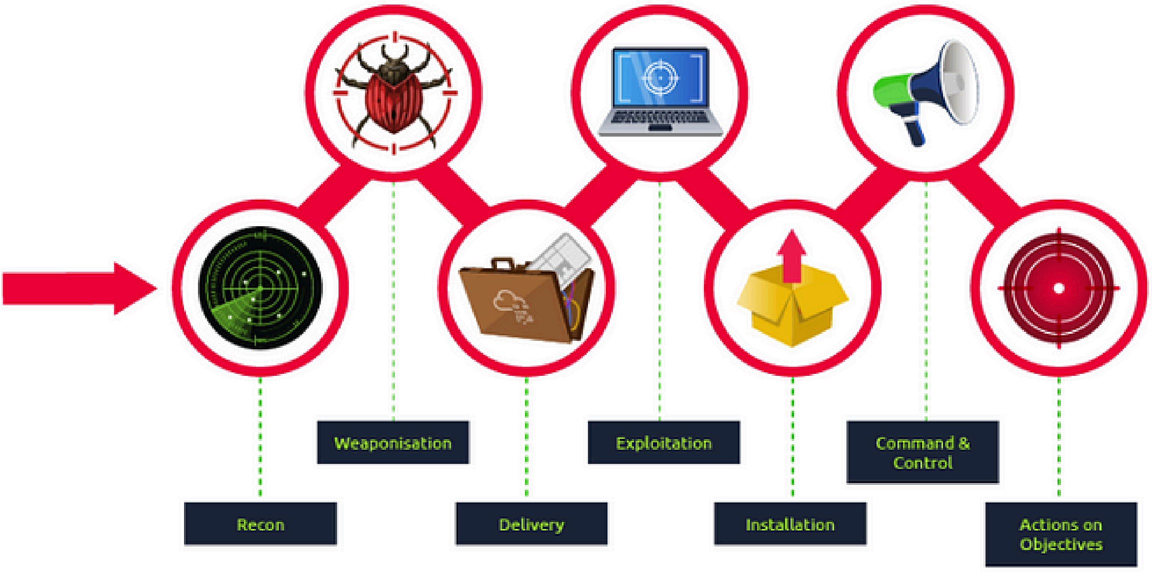
Nov 11, 2024

IritT

Windows Event Logs — Cyber Defense-Security Operations & Monitoring — TryHackMe Walkthrough

Introduction to Windows Event Logs and the tools to query them.

Oct 15, 2024



RosanaFSS

TryHackMe: Threat Hunting With YARA, detailed Write-up

This room focuses on using YARA for threat hunting.

Nov 26, 2024

Fritzadriano

Retracted — TryHackMe WriteUp

IInvestigate the case of the missing ransomware. After learning about Wazuh previously, today's task is a bit different.

Sep 4, 2024 50

See more recommendations