Introduction

In this lab, a .vmem file was provided on which the command line program volatility was to be used to identify malware on the disk depicted in the file as well as the wider circumstances as presented by the artifacts of evidence revealed by volatility.

Objective

The goal of this lab is to utilize volatility and become familiar with subcommands that can be used to extrapolate more specific information from the file. Along with this, reasoning and research is needed to further investigate the identified IOCs in the file to draw out the full story of the intrusion we find traces of in the .vmem file. This can be done by following the steps of initial incident response, being initial assessment, initial evaluation, initial indications, and initial steps.

Report

Before we can dive into the content of the file of the vmem file the step of initial assessment of what is available for investigation and in what state it is in must be followed. An initial assessment follows an analysis of the initial reporting of the incident, along with the symptoms provided. While out of the scope of this lab, the fact that an image is provided to us rather than the original media, tells us that the incident may be already contained, as one would have to enter the machine or interact with it to receive the image, but in incident response nothing can be assumed, so the initial assessment of the incident is that a computer has been compromised with malware, and that off-line analysis must be performed to ensure safety any the containment of the code.

The next step is initial evaluation, where the questions of what happened start to get asked, and here it is important to document the systems involved so the size and the scope of the attack can be ascertained. Relating to this need to baseline the system that was compromised, volatility requires a profile to analyze the image according to the OS of the machine the vmem is a disk image of. This is because each OS and profile has certain specifics such as file system structure and computer organization/architecture, which is integral for volatility as it needs the context of the structure of the computer it is investigating that it may organize and classify the entirety of the virtual memory file according to the actual host system (Volatilityfoundation, *Volatility usage*). With this we can run the preliminary command “volatility -f ‘~/KobayashiMaru 1.vmem’ kdbgscan” which specifies to volatility the file to run kernel debugging block scan plugin. What this does is that raw memory is scanned for KDBG blocks which help identify computer architecture related to the OS. As seen in figure 1, we find that the profile WinXPSP3x86 is identified as a suggested profile, while KDBG is instantiated with the kernel as WinXPSP2x86. To find which profile to use we can also use --imageinfo which also gives a suggested profile which uses KDBG aspects along with other clues to give a suggestion as to what profile to use, in which it returns the profile WinXPSP2x86(*Using kdbgscan to identify correct OS profile* 2020). What this string means is that the OS of the computer is (most likely) a Windows XP system using Service Pack 2 with a 32-bit architecture (Fisher, *What is a service pack?* 2023). Even though we see that service pack 3 is also suggested, we can use the older profile listed in imageinfo for the rest of our investigation as the memory is aligned with the service pack 2 changes. Before we use volatility to extract artifacts, the size of the file can tell us a bit more about the system the vmem depicts, which also relates to the scope and severity of the

malicious programs. Because the file size is 524288 kb or 524 megabytes, this tells us that the vmem only depicts the primary data of the computer, that being the RAM of the machine, so mainly processes from the last session are investigable, meaning we must now look to the incident indications step of incident response, where looking at the processes is the best place to find IOCs and start framing the narrative of the attack.

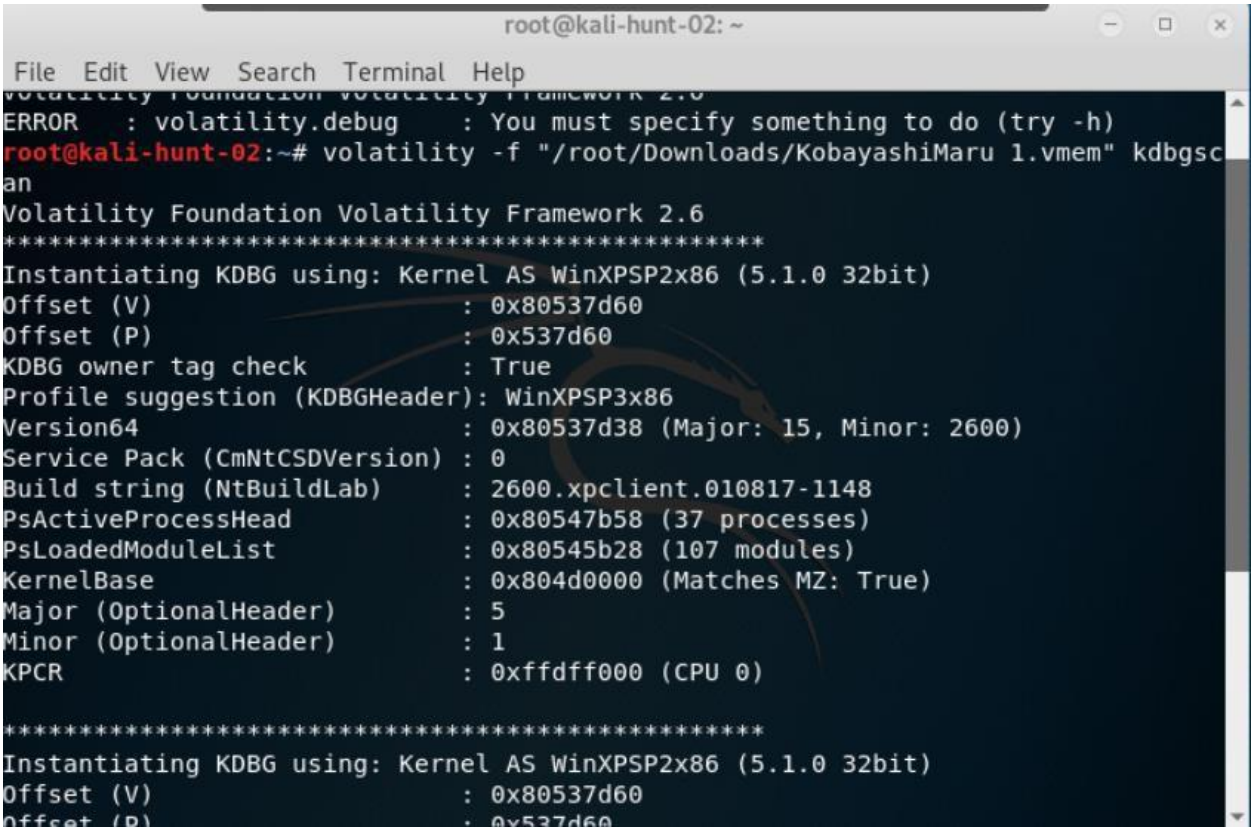


Figure 1 – Kdbgscan result

By using the command “volatility -f ‘~/KobayashiMaru 1.vmem’ --profile WinXPSP2x86 pslist”, as seen in figure 2 a display of the saved running processes starting from the system boot up with pid 4. Tracing the parent pid to child we come across 7 suspicious processes with different reasons as to why. The first one we spot is hxdef100.exe with a pid of 1416 as we see this process start two others and unlike the other processes started by services.exe, this one process is unaccounted for in the list of standard bootup executables (Mendiratta, *Windows boot process step by step* 2024). For this process to be so close to the starting processes tells us that this process is likely a rootkit residing inside the kernel of the OS, explaining the unaccounted-for process. We can therefore conclude that the two processes started by hxdef100.exe being cryptcat.exe and bircd.exe with pids of 1472 and 1480 respectively are suspicious and are flagged for further inquiry. Next is iroffer.exe, which is suspicious for a few reasons, with the primary reason being, as shown in figure 3, that the parent ID is not found in the chain of processes starting from the System process. The other primary reason is that we see 3 of the processes with the process before being the parent of the next. This is suspicious as we also see that the middle process is the only process with any threads or handles and that there is no end time, and with the third iroffer, the end time is before the start time, making these processes

suspicious. The last suspicious processes are poisonivy, nc (netcat), and win4vnc, as these processes were started by a hidden, unlisted process and by the names alone, we can assume that these processes relate to connections by the computer from and to the outside of the network. All these processes are suspicious, but to get a further idea of them, we must first analyze how they relate to the user of the machine, and whether the accounts used were of administrator status or the basic user account. This is important as documenting account usage is integral to incident response, so finding any abnormalities in this area is paramount.

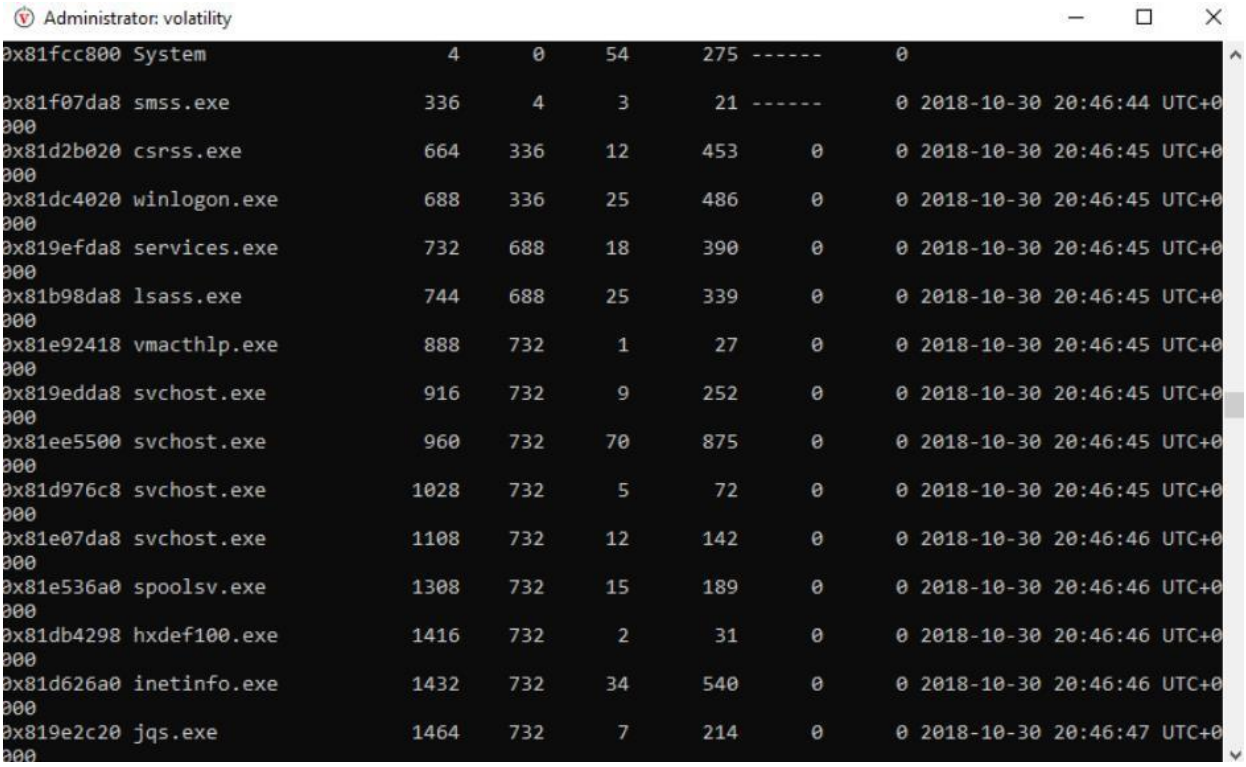


Figure 2 – Pslist result

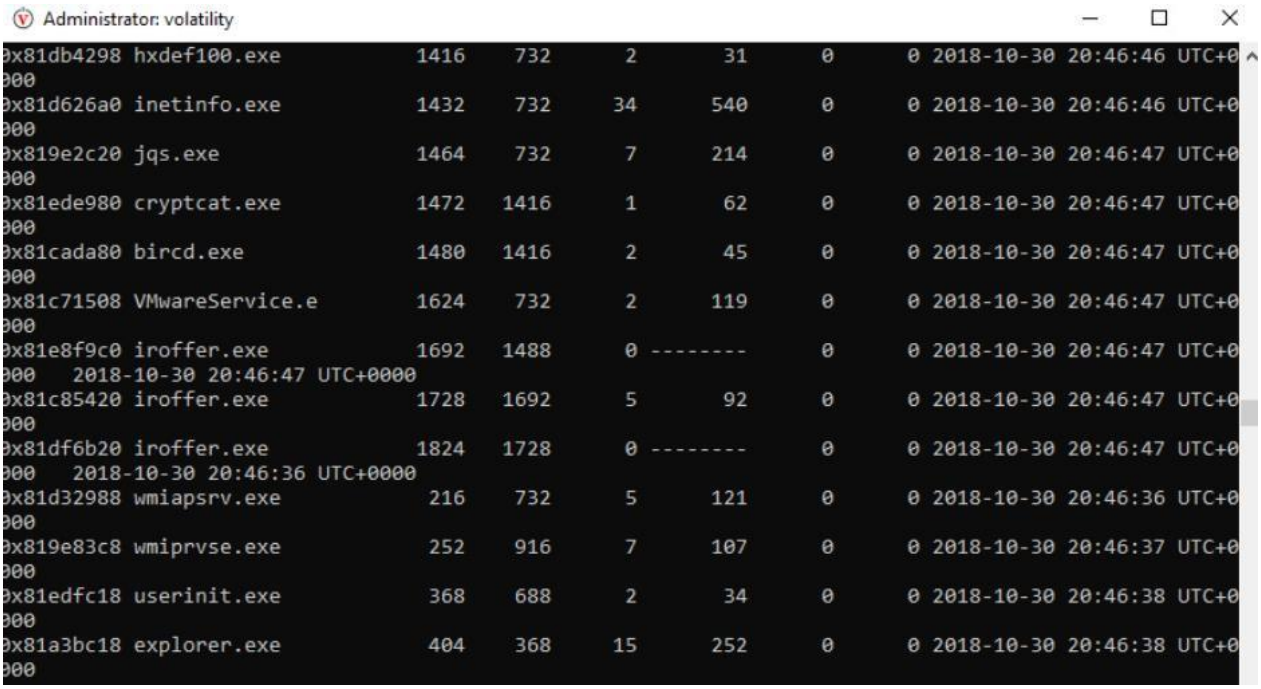


Figure 3 – Iroffer discrepancy

To find out the accounts of the computer, the command “volatility -f ‘~/KobayashiMaru 1.vmem’ --profile WinXPSP2x86 printkey -K "SAM\Domains\Account\Users\Names"” can be used, and as seen in figure 4, Daniel Faraday along with default user accounts are identified. This command manually traverses to the SAM registry where accounts and password hashes are stored, so to identify what accounts relate to the usage of the processes we see in pslist, the passwords of these accounts must be identified. Trace evidence must be used in this situation, as using the hashdump and lsadump commands to return any results for this image. While we see that the SAM file exists in our vmem file, volatilities commands cannot extract the password hashes. However, we can use the command strings -el ‘~/KobayashiMaru 1.vmem’ | grep -A5 -B5 "DefaultPassword", and with this command, as seen in figure _, we can manually look through trace evidence to find the cleartext representation of the user registry, revealing bond007 as the password. This information combined with the userinit.exe process found in pslist likely means that the attacker was able to access Daniel Faraday’s account and change the password to his own. This adds to the incident indications step in incident response, but this information can also help identify the scope of the breach on the machine.

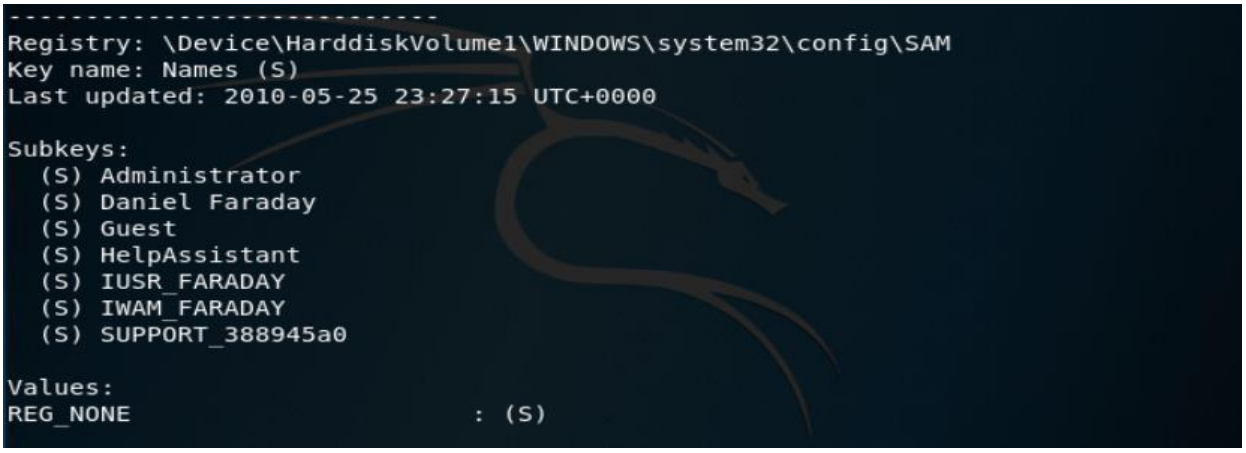
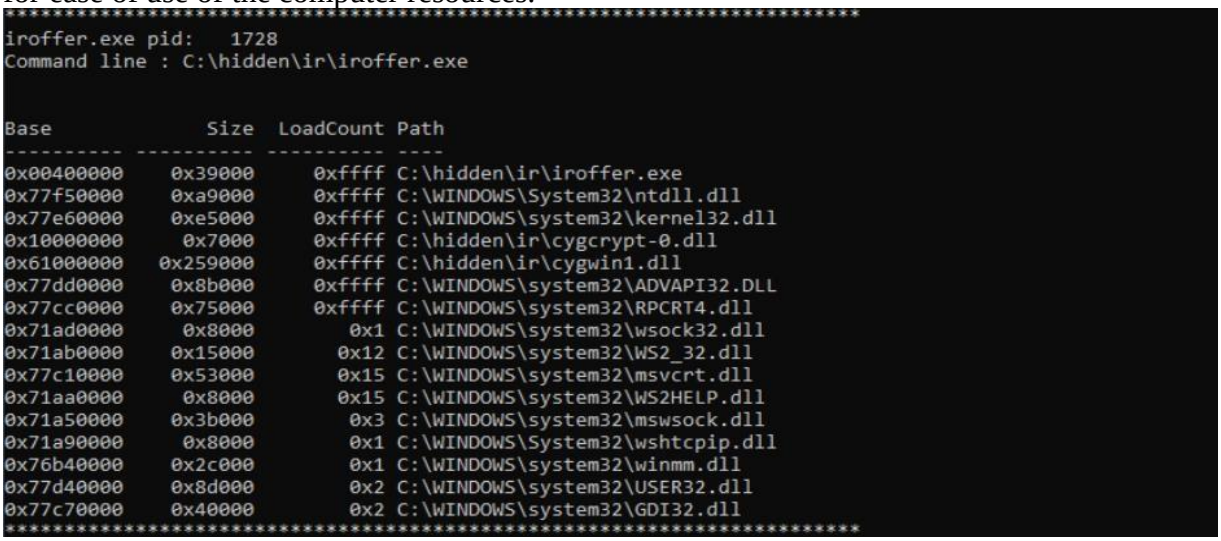


Figure 4 – SAM file account registry

Most processes involve the use of dynamically linked libraries as these files contain code that the executables call on, so to further investigate how these processes are being used on the computer we can list the dlls to identify more suspicious artifacts by way of associating the earlier processes and physical commands that the dlls were called by. Using the volatility command dlllist, we see what dlls relate to what processes, and as we see in figure 5, volatility tells us the command used to start the process along with the associated dlls. From a neutral point of view, we see that a few dlls are suspicious even when separated from the context of association with the processes, as we find that some dlls are found under the C:/hidden directory, which is not standard on a windows machine (Stegner, *7 default windows files and folders you should never touch* 2021). Others are found under C:/intetpub, which is also abnormal, and looking at the process listed at the top of the section the dll is under, the abnormal processes identified earlier show up again in suspicion. The fact that the connection relating processes have their first dlls in this location shows that someone must have used ftp to place connection programs onto the computer. Looking at further associations, we find that in the hacker defender rootkit process the dlls used include user32 and kernel32, showing that this rootkit operated at the kernel level underneath the OS that was presented to the user, modifying what the OS did for the host (Pilici, *Kernel32.dll: What it is & how to fix errors* 2023). With cryptcat, we see the command "C:\hxdefrootkit\cryptcat.exe" -L -p 666 -e cmd.exe", which shows us that cryptcat was used to listen on a backdoor port at which point cmd.exe would be executed, i.e. the outside connector would be given admin command capability and privileges. Looking at the associated dlls, we find the same kernel related dlls as hxdef100, but we also find DNSAPI.dll and SAMLIB.dll, which also strengthens the idea that an attacker used this process to grant privilege maliciously to an incoming connection over the backdoor (*Microsoft Corporation, What is dnsapi.dll?*). Moving onto bircd.exe, we don't see as many dlls, but we see rpcrt4.dll, which is used for handling remote procedure calls, which informs us that this process related to the execution of code internally on the system moving data around rather than focusing on connections (*Rpcrt4.dll*). Looking at the last process before the related backdoor processes, we see that iroffer.exe is using Cygwin libraries, meaning that linux based commands are being used on top of the windows machine, which is also supported by the presence of ADVAPI.dll, which allows access control and other management tool to windows applications, which IRC is in this case(Microsoft, What is advapi32.dll?). While there could be many reasons for this, the fact that these dlls are located in the hidden directory shows they were moved or imported by the attacker,

and in conjunction with this effort and the information about bircd.exe, we can conclude that these IRC functions were the primary reason for the attack as the attacker is setting up utilities for ease of use of the computer resources.



```
************************************************************************
iroffer.exe pid:   1728
Command line : C:\hidden\ir\iroffer.exe


Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x00400000    0x39000     0xffff C:\hidden\ir\iroffer.exe
0x77f50000    0xa9000     0xffff C:\WINDOWS\System32\ntdll.dll
0x77e60000    0xe5000     0xffff C:\WINDOWS\system32\kernel32.dll
0x10000000     0x7000     0xffff C:\hidden\ir\cygcrypt-0.dll
0x61000000   0x259000     0xffff C:\hidden\ir\cygwin1.dll
0x77dd0000    0x8b000     0xffff C:\WINDOWS\system32\ADVAPI32.DLL
0x77cc0000    0x75000     0xffff C:\WINDOWS\system32\RPCRT4.dll
0x71ad0000     0x8000        0x1 C:\WINDOWS\system32\wsock32.dll
0x71ab0000    0x15000       0x12 C:\WINDOWS\system32\WS2_32.dll
0x77c10000    0x53000       0x15 C:\WINDOWS\system32\msvcrt.dll
0x71aa0000     0x8000       0x15 C:\WINDOWS\system32\WS2HELP.dll
0x71a50000    0x3b000        0x3 C:\WINDOWS\system32\mswsock.dll
0x71a90000     0x8000        0x1 C:\WINDOWS\System32\wshtcpip.dll
0x76b40000    0x2c000        0x1 C:\WINDOWS\system32\winmm.dll
0x77d40000    0x8d000        0x2 C:\WINDOWS\system32\USER32.dll
0x77c70000    0x40000        0x2 C:\WINDOWS\system32\GDI32.dll
************************************************************************
```

Figure 5 – Structure of dlllist command

Looking at the associated dlls of the last suspicious processes we identified, we see poisonivy share the same kernel related dlls as the rootkit hxdef100, along with ntdll.dll, which relates to low level kernel file I/O (Pilici, *Ntdll.dll: What it is & how to fix Ntdll.dll Errors* 2023). While serving a similar purpose to the first rootkit, poisionivy in this instance is used to manage and hide any outside connections. This is further proven by the existence of the command: C:\inetpub\ftproot\nc.exe -L -p 6666 -e cmd.exe and the shared dlls including ntdll in the nc.exe and win4vnc processes. Also, unlike poisonivy, netcat and win4vnc both are associated with NETAPI32.dll, which provides services to active applications, services, and network connections such as authentication and domain management, showing that while poisonivy dealt with the kernel in hiding and managing the connections, netcat and win4vnc were used in conjunction with each other actually make the connection from the outside, superseding authentication and notice with the help of poisonivy (Pilici, *Netapi32.dll: What it is & how to Fix Netapi32.dll errors* 2023). Before we investigate this final set of processes further, we see that, while not suspicious before, the cmd.exe process is being used maliciously, where, as seen in figure 6, the cmd is being used to run lock.bat. We also see that netcat was executed so that cmd.exe is running when the computer was entered so that the user was provided admin capabilities with the cmd console, showing that most of the usages of this tool captured in the virtual memory can be associated

with the rootkits and general malicious activities.

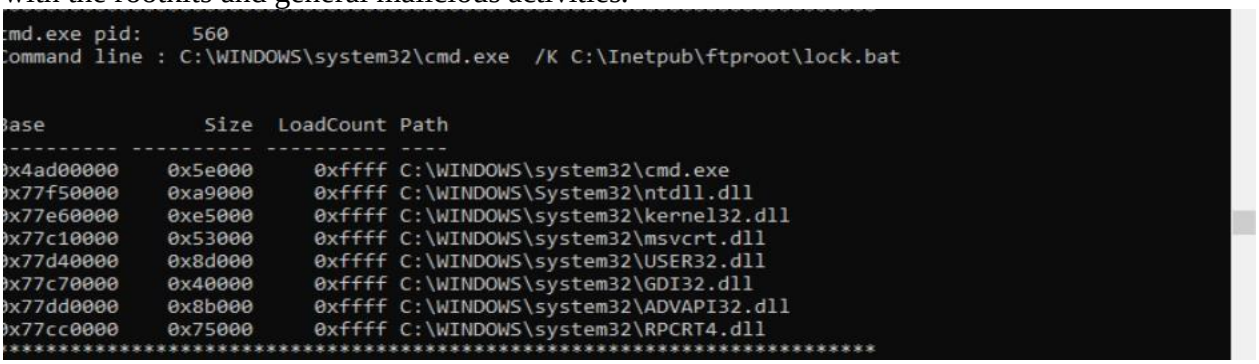


Figure 6 – Cmd command for lock.bat

We can use the volatility function malfind to investigate and malicious code found withing the files relating to the suspicious processes, and doing this, as seen in figure 7, returns assembly code for nc.exe, poisonivy.exe, and win4vnc.exe, all the processes relating to outside connections. Examining this code shows us a common Position-Independent Code trick used in shellcode or exploits, where the CALL and POP EAX are used to get the current instruction pointer (Bendersky, *Position independent code (PIC) in shared libraries on x64*). Using procdump and then using an md5 hash, we can then use virus total to see if the website will flag any of the hashes as malicious. While all the assembly codes are the same, the hashes are separate, so as seen in figure 8, we get different virus total reports. Examining these reports tells us different things about each process, and the hash that generated the most alert was poison ivy, where virustotal labeled it as a trojan. More specifically, in the report many security vendors labeled it as a backdoor program that opened connections to an outside attacker. This relates to the nc.exe report, where netcat was attributed to be a hacking tool, and more specifically, remote admin connection. Lastly, despite a lower community score, we see win4vnc has a low reputation, and that it is also a network tool. With this we can tie these processes together by the reports generated, where poisonivy helped open and manage the backdoor, nc.exe was used to connect to the host, and win4vnc allowed a graphical interface for this remote connection, and with all the same malware on each process and with nc.exe and win4vnc being in the inetpub folder, we can conclude that the attacker was using this chain of processes to hide backdoor remote connections.

```
C:\Users\admin\Desktop>volatility -f "C:\Users\Administrator\Desktop\KobayashiMaru 1.vmem" --profile W
inXPSP2x86 malfind -p 480
Volatility Foundation Volatility Framework 2.6
Process: poisonivy.exe Pid: 480 Address: 0x7ffa0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 5, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x7ffa0000  e8 00 00 00 00 58 2d be 5d 40 00 c3 5f 2e 2d 3d   .....X-.]@.._._.-=
0x7ffa0010  5b 48 61 63 6b 65 72 20 44 65 66 65 6e 64 65 72   [Hacker.Defender
0x7ffa0020  5d 3d 2d 2e 5f 00 00 00 00 00 00 00 00 04 00 00   ]=-._...........
0x7ffa0030  00 6b 65 72 6e 65 6c 33 32 2e 64 6c 6c 00 53 65   .kernel32.dll.Se

0x7ffa0000 e800000000       CALL 0x7ffa0005
0x7ffa0005 58               POP EAX
0x7ffa0006 2dbe5d4000       SUB EAX, 0x405dbe
0x7ffa000b c3               RET
0x7ffa000c 5f               POP EDI
0x7ffa000d 2e2d3d5b4861     SUB EAX, 0x61485b3d
0x7ffa0013 636b65           ARPL [EBX+0x65], BP
0x7ffa0016 7220             JB 0x7ffa0038
0x7ffa0018 44               INC ESP
0x7ffa0019 6566656e         OUTS DX, BYTE [GS:ESI]
0x7ffa001d 6465725d         JB 0x7ffa007e
0x7ffa0021 3d2d2e5f00       CMP EAX, 0x5f2e2d
0x7ffa0026 0000             ADD [EAX], AL
0x7ffa0028 0000             ADD [EAX], AL
0x7ffa002a 0000             ADD [EAX], AL
0x7ffa002c 000400           ADD [EAX+EAX], AL
0x7ffa002f 0000             ADD [EAX], AL
0x7ffa0031 6b65726e         IMUL ESP, [EBP+0x72], 0x6e
0x7ffa0035 656c             INS BYTE [ES:EDI], DX
0x7ffa0037 3332             XOR ESI, [EDX]
0x7ffa0039 2e646c           INS BYTE [ES:EDI], DX
0x7ffa003c 6c               INS BYTE [ES:EDI], DX
0x7ffa003d 005365           ADD [EBX+0x65], DL
```

Figure 7 - Malfind assembly code in poisonivy

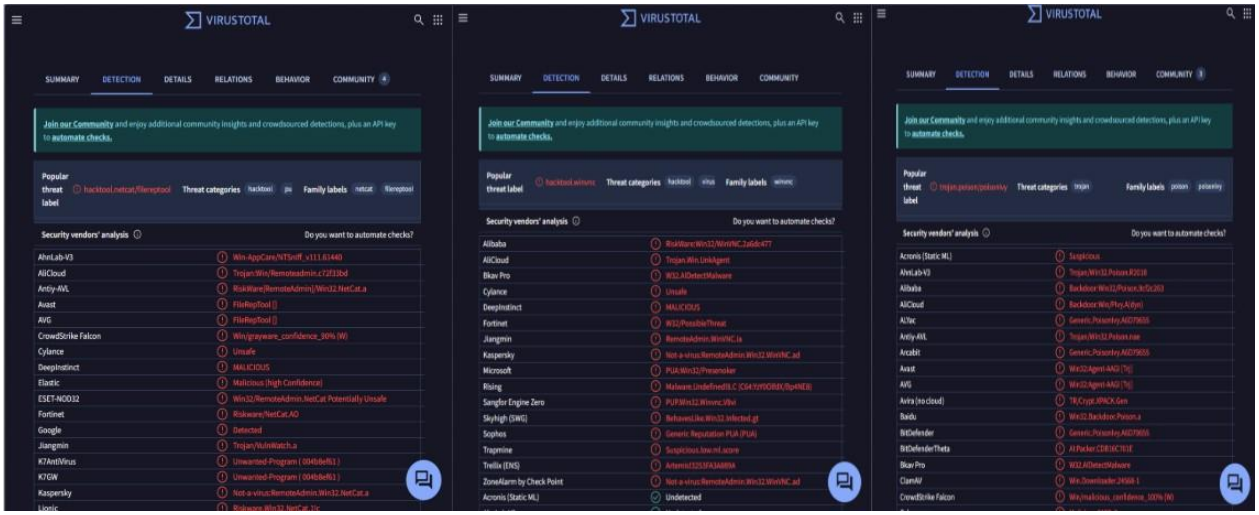


Figure 8 – Differing virustotal reports

To tell the full story of the attack, we must revisit the IRC functions mentioned earlier as it has been revealed that with the setup of Cygwin the attacker is investing time into these functions to be performed locally on the machine. All the other suspicious processes were run to facilitate entry into the machine and the hiding of the nefarious activities, so to investigate the goal of the attacker the contents of the disk that relate to IRC must be examined. We can use the

subcommand yarascan to search for keywords across the image file, and with this we extract artifacts that relate to the malicious use of IRC on the machine. As seen in figure 9, when looking up Cygwin and IRC, we come across the host name of a local IRC server that was started on the compromised machine. Allen626! is the name of this host and following this fact we see that the machine is named mybotDCC as an IRC bot, where outside users can communicate over IRC (port 6667) to connect to the hosted server. We can also locate files and programs that were distributed and available over this network, such as pwdump and JohnTheRipper, as seen in figure 10. From this we can conclude that the compromised machine was used for the distribution of these cracked programs, and all the other imported rootkits and tools Allen used were to support the hiding and connections the IRC net needed.

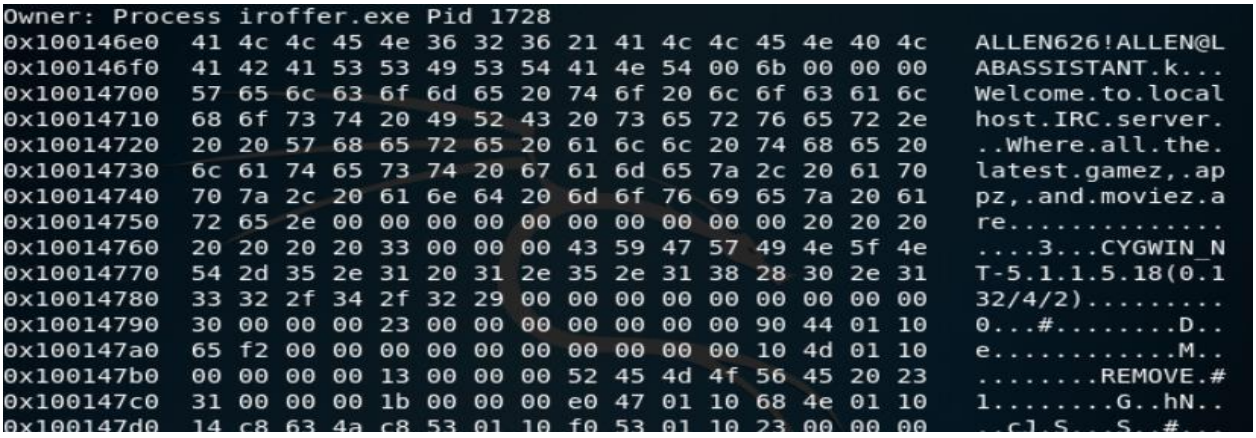


Figure 9 – Allen626! username usage

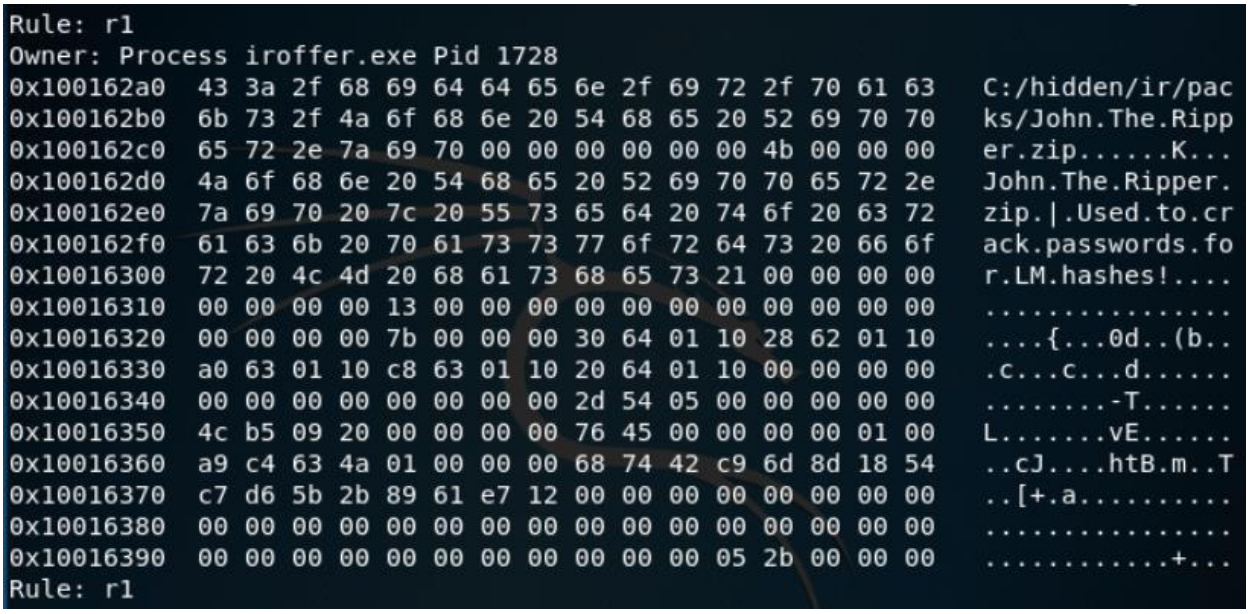


Figure 10 – Cracked software presence

Another tool we can use to investigate Allen's use of the machine is by running the subcommand connscan. As seen in figure 11, we see that there are two active connections that relate to the local network, and by the designation of 127.0.0.1, we understand that these are

loopback connections, and by using the pids and port numbers, we find that the first connection relates to the iroffer process and the second to bircd. With this we find that the computer is using a loopback connection over IRC ports that outside connections can dial into these running connections and receive the cracked software mentioned earlier. The third connection shows a remote IP address of 192.168.5.98 with a connection to the poisonivy process. With this we can identify this IP address as belonging to Allen's point of connection to the machine. These connections can further be used to identify how to contain the incident and how to prosecute the intruder and is the most important mark of identification of the intruder that would warrant more research after containment of the incident.

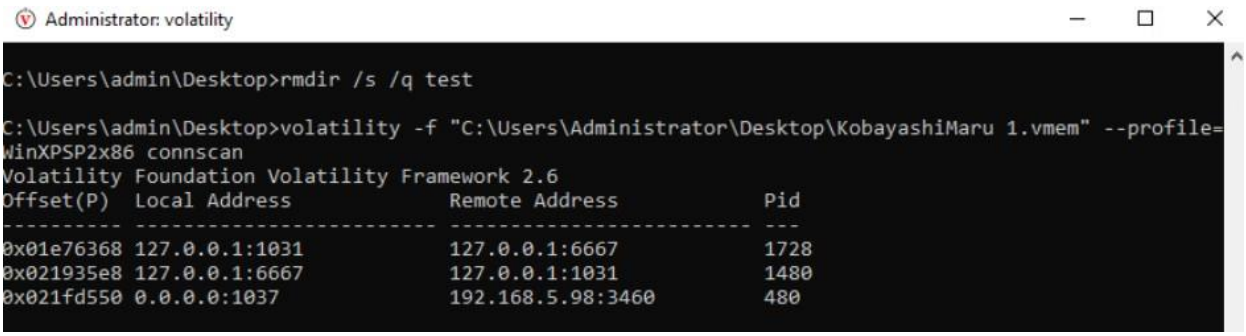


Figure 11 – Connscan result

To summarize the artifacts found we can classify the information into the questions they answer, being who, what, when, where, why, and how. Many of the artifacts can be associated together, or rather, one set of information can create a piece of contributing evidence, for example the process netcat is associated with the executable nc.exe and was started by the command C:\inetpub\ftproot\nc.exe -L -p 6666 -e cmd.exe along with being associated to ntdll.dll and NETAPI32.dll, a kernel level file I/O library and network service library respectively. Associations like these with known dlls help provide a picture of what each process in doing on the machine, along with what areas of the computer were subject to intrusion, honing the investigation to find the root cause. It is also important to note the usage of the tools used, as documentation is paramount when preserving the chain of custody in an investigation. The investigation was an off-line analysis using a target media in the format of a disk image file, and the native OS was found by imageinfo being Windows XP. It is important to have the disk image reflect the original media as much as possible, so using volatility puts us in a read only state, where artifacts can be extracted with the confidence that it reflects the original media. As seen in figure 12, we can classify the artifacts into what questions they address, and with this we can extrapolate a conclusion based on the order of events. Another tool we can use is an incidence response checklist, where this document could be used to inform others of the incident and what was performed on the image. Classification of artifacts is important to have so that a general sense of relevant information can be used in the later stages of incident response.

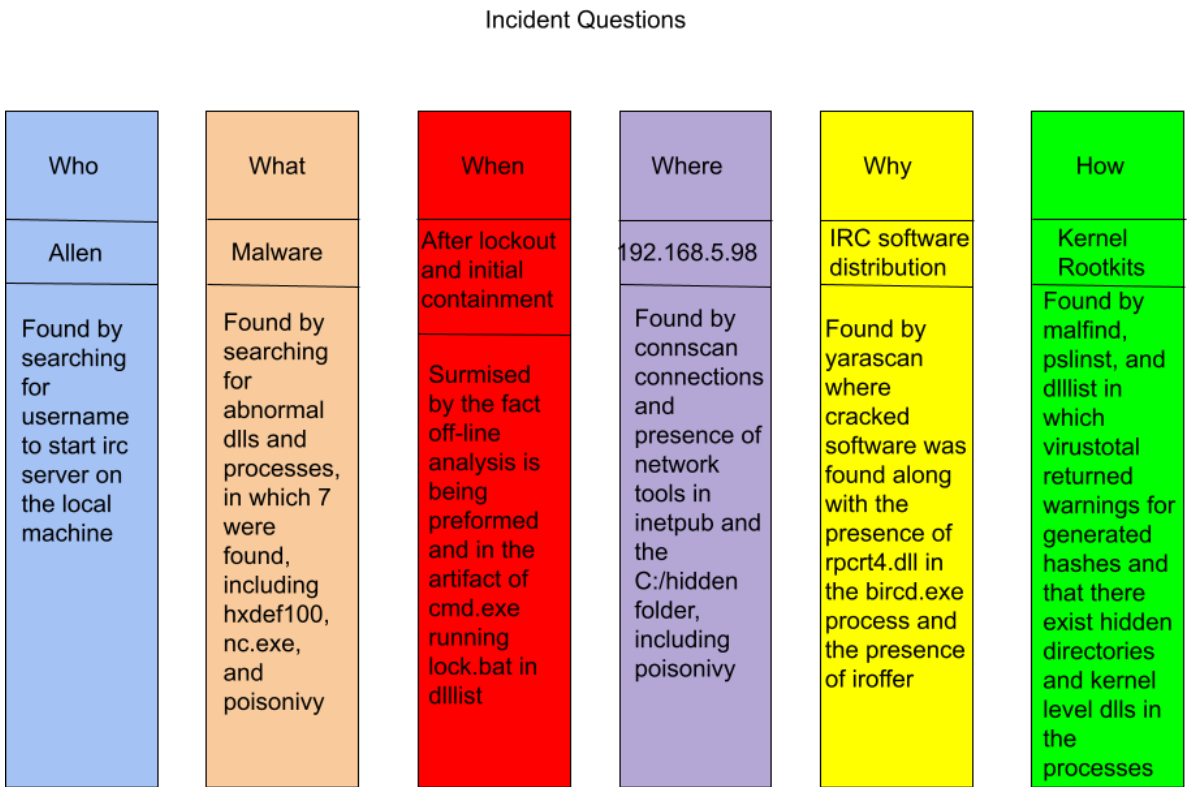


Figure 12 – Artifact classification

Lastly, we can summarize the steps taken into the early stages of incident response as mentioned at the beginning of the report, being initial assessment, initial evaluation, and initial indications. As seen in figure 13, the order of the questions answered follows a natural progression, and at each step all actions and artifacts would be documented so that any law official or investigator may come to understand the findings and actions performed on the evidence. After the initial response step of incident response is the formulate strategy step, where using the documentation gathered by the generated reports, a method of responding to the incident on the original media can be formed. After this is a more thorough examination of the incident that coincides with containment and eradication. All throughout this process documentation is made, bringing the incident to the final step of resolution, reporting, and lessons learned.

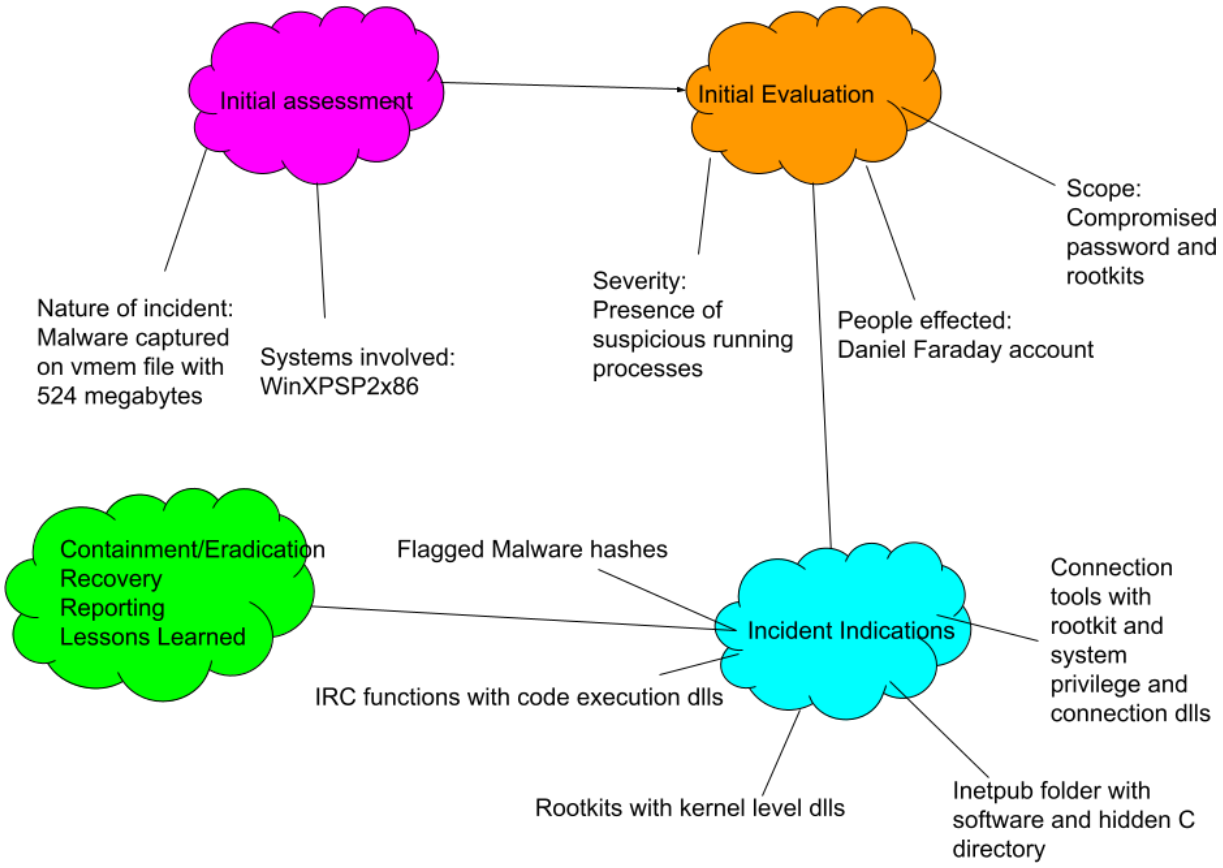


Figure 13 – Incident response stage progression

Conclusion

In this lab, the tool volatility was used to investigate processes and the contents of a virtual memory file, but perhaps the most useful tools volatility provides is malfind and dlllist, where the deeper story can be extracted rather simply. Using the steps of incident response, a progression of questions can be asked that lead to an understanding of an incident, and at the end of the process, a thorough report can be made on the findings, much like the efforts taken in this lab report.

Bibliography

Bendersky, E. (n.d.). *Position independent code (PIC) in shared libraries on x64*. Eli Benderskys website ATOM. https://eli.thegreenplace.net/2011/11/11/position-independent-code-pic-in-shared-libraries-on-x64

Fisher, T. (2023, February 10). *What is a service pack?*. Lifewire. https://www.lifewire.com/what-is-a-service-pack-2626010

Mendiratta, S. (2024, September 26). *Windows boot process step by step*. Automate Infra. https://automateinfra.com/2021/11/09/windows-boot-process-step-by-step/

Microsoft Corporation. dnsapi.dll Windows process - What is it? (n.d.). https://www.file.net/process/dnsapi.dll.html

Microsoft. (n.d.). *Microsoft*. advapi32.dll Windows process - What is it? https://www.file.net/process/advapi32.dll.html

Pilici, S. (2023a, June 28). *Kernel32.dll: What it is & how to fix errors*. MalwareTips Blog. https://malwaretips.com/blogs/kernel32-dll-what-it-is-how-to-fix-errors/

Pilici, S. (2023a, June 28). *Netapi32.dll: What it is & how to fix Netapi32.dll errors.* MalwareTips Blog. https://malwaretips.com/blogs/netapi32-dll-what-it-is-how-to-fix-netapi32-errors/

Pilici, S. (2023b, June 28). *Ntdll.dll: What it is & how to fix Ntdll.dll Errors*. MalwareTips Blog. https://malwaretips.com/blogs/ntdll-dll-what-it-is-how-to-fix-errors/

Rpcrt4.dll. What is rpcrt4.dll? (n.d.). https://www.processlibrary.com/en/directory/files/rpcrt4/23580/

Stegner, B. (2021, September 22). *7 default windows files and folders you should never touch*. MUO. https://www.makeuseof.com/tag/default-windows-files-folders/

Using kdbgscan to identify correct OS profile. Forensic Focus. (2020, June 13). https://www.forensicfocus.com/forums/general/using-kdbgscan-to-identify-correct-os-profile/

Volatilityfoundation. (n.d.). *Volatility usage.* GitHub. https://github.com/volatilityfoundation/volatility/wiki/Volatility-Usage