A quick analysis of the latest Shadow Brokers dump

By Nettitude Labs

Published: 2017-04-17 · Archived: 2026-04-05 14:19:04 UTC

Just in time for Easter, the Shadow Brokers released the latest installment of an NSA data dump, which contained an almost overwhelming amount of content – including, amongst other things, a number of Windows exploits. We thought we'd run some quick analysis on various elements of said content.

Before we get started

We're going to largely avoid the obvious elements of the dump because there's already been a lot of very helpful analysis of those elements. However, before we get to that, here's what you need to know:

- Patch! The majority of the high impact Microsoft vulnerabilities have recently been addressed in the MS17-010 patch.
- Disable SMBv1.
- Remove all Windows XP and 2003 machines from your network. These contain vulnerabilities that will not be patched.

The following table (raw data available at https://pastebin.com/5gkb6HLJ and courtesy of @etlow) contains some of the more pertinent information.


Shadow Brokers Exploit Table

Shadow Brokers Exploit Table

We can also recommend the following script by Luke Jennings, which is designed to sweep a network to find Windows systems compromised with the dumps DOUBLEPULSAR implant: https://github.com/countercept/doublepulsar-detection-script
With that out of the way…

Metadata, or a lack of

Throughout the Equation Group leak via the Shadow Brokers, there are a number of different languages being used. One interesting element is how it appears that there was originally a preference for Perl, that was then replaced with Python – we think that this mirrors how the offensive security industry has evolved, too.
As the age of the dump is pinned at some point in 2013, we would have expected to see a little bit of PowerShell; this was really starting to come into favor around that time. Now, this post isn't about dropping a new l33t PowerShell technique gained from the dump, but rather looking at what the capability was at the point in time.
Staying with the timing of the dump for a minute, we are reminded of the following series of Tweets from Edward Snowden back in August last year, when the ShadowBrokers [6] first dropped.

We know we run the risk of taking these out of context, and it is entirely possible that his mind has been changed since, however we find the following piece of information interesting. According to the time line from the Guardian [5], the first release of the material he took was on the 5th June 2013. It's probable that other dumps have since has contradicted this and the view of when the hacker/s were kicked off has been able to be narrowed, but we am unaware of this (so please if you know different answers on a postcard).
Examining of the tools *makedmgd.exe*, part of a toolkit DAMAGEDGOODS that is used within in a PowerShell delivery framework ZIPO we see the following. One of the first things that we noticed is that yeah hmmm the build date is baked into the exe. Also some different implants not within the dump are there *"distantuncle"* and *"finkdiffernt";* some of the coders definitely have a certain sense of humor.

Using Sysinternals excellent sigcheck.exe [7] we could view the publisher, version and build date in order to correlate. Yes, it is one of the many ways to list a binarys metadata, but some of its other superb features are that, as the name implies, it will verify the signature if the binary has been signed using Authenticode and it is also able to send the binary straight to VirusTotal and look at all files within a directory tree recursively. Running sigcheck,

unsurprisingly we get the following information or, some would say, a lack of.

Any trace of publisher or company which, to be fair, will be set in Visual Studio (or your toolchain of choice have either been stripped or not set). The Link date is there, which correlates to the build date, which is also five weeks after Snowden's material was first dropped. It is entirely possible to mess with and edit these dates, of course, before releasing the dump. We do find it strange to go the level of stripping all other information but hard coding a build date, particularly in a tool that will be released to a workstation. The directory structure that this is in implies it may have been copied in rather than part of a release, as it was new and may not have been sanitised properly (although there is a real danger of reading too much into it).

First steps into PowerShell

As stated above, we would have expected to see a reasonable amount of PowerShell considering the year, but actually there is very little. The only real example that we have found is a tool called ZiPo which can be found within the dump at /Resources/Ops/Tools/ZiPo. It contains the following tools

- decryptor_downloader.base
- makedmgd.exe
- ZIPO.py
- ps_base.txt
- powershellify.py

In order to run this tool we call ZIPO.py, which first asks you to select a "project" directory then presents a menu asking if we want to:

1. Upload / Create Execute an Egg
2. Upload/ Create PowerShell script
3. Create Compressed script to be run manually

Now Egg is a term that is used quite heavily throughout the dump and we're not entirely sure what it means at this point in time. Pretty sure it is an Equation Group term.

Choosing PowerShell script we are then asking for the location of it, what the IP address and port of the “redirector” which we assume is a proxy and then the local IP address and proxy. This is so that the script can spin up a HTTPd listener to serve up the files that have been created.
In order to test, we created a very simple PowerShell script containing:

```
[System.Reflection.Assembly]::LoadWithPartialName("System.Windows.Forms")
[System.Windows.Forms.MessageBox]::Show("Hey mate, do you wanna run some powershell?", "you know you
```

It has generated a public/private key pair, created an index.html & index.htm, provided us with a script to run on the target and also started up a HTTPd so that we could download the payloads on the target. That's not too bad for a couple of commands.

Looking at the command to run its pretty standard PowerShell from the time, in fact we find it really interesting there is absolutely no attempt at obfuscating anything here. They are encrypting the payload and building a chain to download/decrypt etc, but no effort is made at hiding what the command is doing or where it is obtaining the script from (of course we would be very interested to see what they are doing now).
So what is contained within the two index files? Well, index.html is base64 PowerShell script, which is why it was executed as an encodedCommand; decoding you get the output below. It encrypts a known "questionable" password value using RSA, another WebClient is created which has the encrypted value set as a cookie. The index.html is then downloaded and decrypted using the key, which is a SHA1 hash of the "questionable value". The payload is then executed and on the server the two files are then deleted. This is a lot of effort to hide the final payload and once again absolutely no effort to obfuscate any of the script.

This is how it looks when it is run:

DAMAGEDGOODS

The next thing that we did was to just create a meterpreter payload; nothing special and wasn't going to get to connect back, but we felt that AV should still be able to pick it up.

Running Zipo again, we selected the third option. It asks you for a payload DLL and also the ordinal [8] that you want to fire. This is where DAMAGEDGOODS comes into play; makedmged.exe is the exe that appears to do some kind of shellcode encoding. In this case it takes the encoded binary with a script called ps_base.txt, then compresses/base64 encodes and then builds a decompression payload around it.

The script that is output at the end of this using the name you supplied is the decode/decompression/execute mentioned above and is shown below.

Decoding it you get the following, which is quite interesting; it's a PowerShell script that allocates memory, writes the shellcode into it, creates a thread and then executes the shellcode, all in memory. The shellcode in this case is going to be the meterpreter DLL that we originally used. Running it multiple times over the same DLL you get a different version. There appears to be some kind of prologue in the shellcode that doesn't change, but it is pretty short, running the script multiple times and then diffing with Scooter Software's excellent Beyond Compare you find that the only section that has changed is the shellcode except for:

This series of bytes which appears to be some kind of prologue probably a decoder for the rest of the shellcode. What does it do, how does it work? Well that, we're afraid, is for part 2 as we've spent too much time away from the family already this easter ;o)

```
0x68,0xc0,0x1e,0x00,0x00,0xe8,0x00,0x00,0x00,0x00,0x58,0x83,0xc0,0x0b,0x50,0xff,0xd0,0x83,0xc4,0x08,
```

This kinda looks familiar….

Now the great irony in a dump like this is finding code that appears to have come from GitHub but doesn't appear to have the same licence or any at all for that matter [1]. This script is built from file called *ps_base.txt.* This is primarily used to dynamically build a type that will eventually hold a function pointer to a native function. This is then used to store the fp's for native functions Win32 functions such as VirtualAlloc[2], GetProcessAddress[3] & GetModuleHandle[4] that can be used to perform some actions such as allocating memory and looking up the addresses of exports within DLL's. Further are shown in this screen shot:

Now the method to create the delegate's used in the above code is:

Programmers (ourselves included) can be utter sticklers for formatting, so it is conspicuous that there is such a big difference in formatting between the code in ps_base.txt vs decryptor_downloader.base. It's almost as if ps_base.txt has come from somewhere else.

Well funnily enough it bears more than just a striking resemblance to some code from Powersploit[1]; screenshots from GitHub are below. Surprisingly not too much effort has been made to change the method names.

And also…

The commit date for this code is…

And as stated above we have a built date of July 2013; does this mean we will find StackOverflow answer code within the dump at some point?

But anyway back to makedmg.exe running it we get this list of other implants that are not in this dump; obviously

still a lot out there.

DOUBLEPULSAR

From analysis we did on some implant configuration files, Darkpulsar appears to create a service called ‘dapu’ It also seems that when it upgrades itself it drops the new file using the following path:
‘c:\windows\system32\sipauth32.tsp’.
We also had a look at tdip.sys driver.
(sha256: A5EC4D102D802ADA7C5083AF53FD9D3C9B5AA83BE9DE58DBB4FAC7876FAF6D29)

We found same magic DWORDs as those mentioned by Kaspersky Labs in the following link: https://securelist.com/blog/incidents/75812/the-equation-giveaway/ which contains information from a previous 'ShadowBrokers' dump.
The following code snippet is taken from tdip.sys:

```
text:000130A0                push    31h
.text:000130A2                lea     eax, [ecx+4]
.text:000130A5                movsd
.text:000130A6                mov     dword ptr [ecx], 0B7E15163h <--------------
.text:000130AC                pop     edx
.text:000130AD
.text:000130AD loc_130AD:                             ; CODE XREF:
sub_13084+38
.text:000130AD                mov     esi, [eax-4]
.text:000130B0                sub     esi, 61C88647h <------------
.text:000130B6                mov     [eax], esi
.text:000130B8                add     eax, 4
.text:000130BB                dec     edx
.text:000130BC                jnz     short loc_130AD
```

This driver was most probably used to capture network traffic and it also accepts IOCTLs from userland. There is probably a relation between this driver and "TrafficCapture_Target.dll" module that we found inside the recent ShadowBrokers dump, which we noticed that it is able to communicate with a kernel driver via IOCTLs.

Conclusion

Keeping in mind that this is a subset of the techniques that the Equation Group had in 2013, we still find it pretty interesting that just like the rest of the world they were starting to wake up to the potential of offensive PowerShell. The lack of any obfuscation i.e attempt to hide any of the decryption/download code was another surprise too considering how much "effort" has gone into encrypting the payload over the network at that point. The source of some of the code is intriguing, too.
But back to the initial thoughts, we probably can be sure that this code was from 2013. Is it possible that Ed's assertion the "hacker squatting lost access in June" may be flawed and they had access until at least the first couple of weeks in July. Assuming SB and no one else has tampered with the metadata within DAMAGEDGOODS, then yes.

[1] https://github.com/PowerShellMafia/PowerSploit/blob/a233…
[2] https://msdn.microsoft.com/en-us/library/windows/desktop/aa366890(v=vs.85).aspx
[3] https://msdn.microsoft.com/en-us/library/windows/desktop/ms683212(v=vs.85).aspx
[4] https://msdn.microsoft.com/en-us/library/windows/desktop/ms683199(v=vs.85).aspx
[5] https://www.theguardian.com/world/2013/jun/23/edward-snowden-nsa-files-timeline
[6] https://twitter.com/snowden/status/765515087062982656?lang=en
[7] https://technet.microsoft.com/en-gb/sysinternals/bb897441.aspx
[8] https://msdn.microsoft.com/en-us/library/e7tsx612.aspx
[9] https://www.scootersoftware.com/

Source: https://labs.nettitude.com/blog/a-quick-analysis-of-the-latest-shadow-brokers-dump/