https://htmlpreview.github.io/?https://github.com/MatthewDemaske/blogbackup/blob/master/netshell.html

Archived: 2026-04-05 22:48:30 UTC

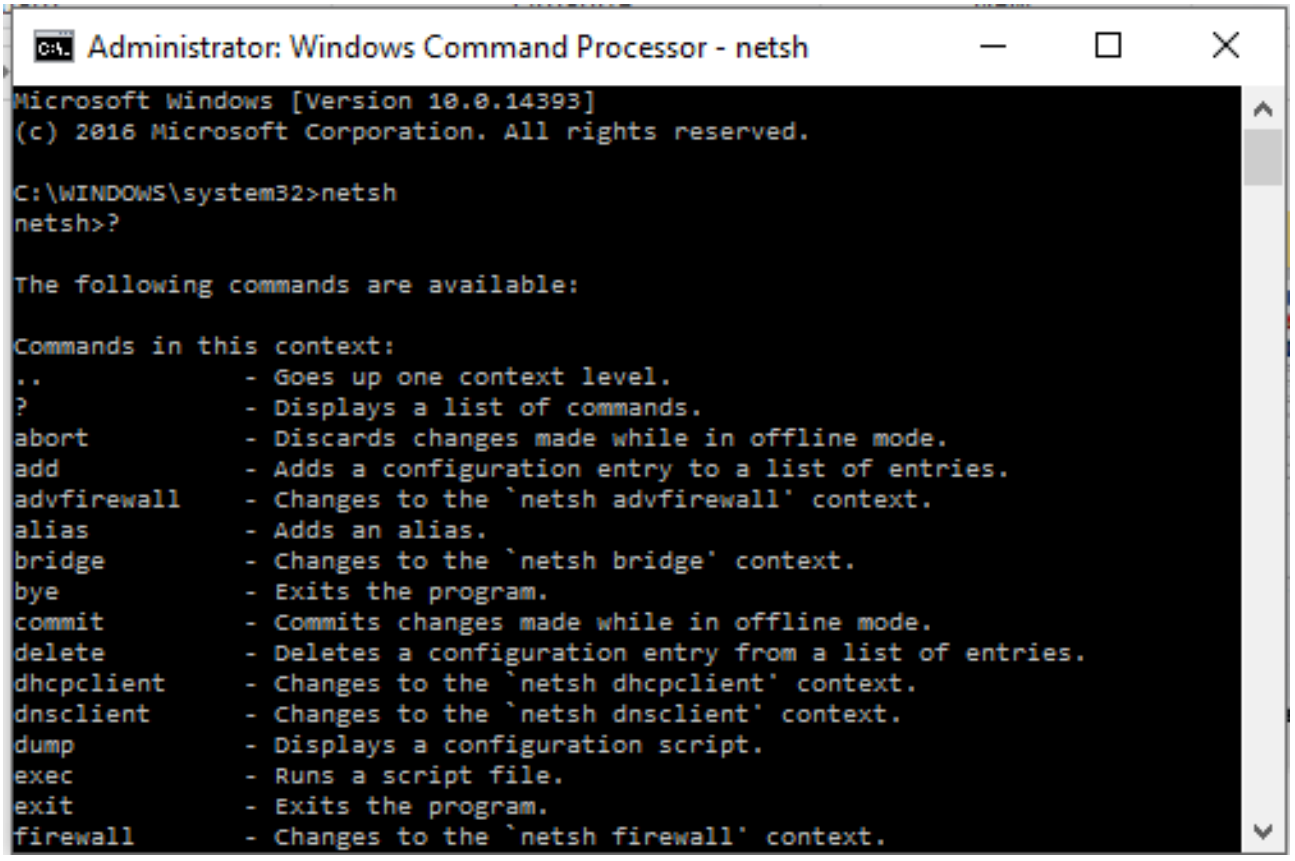
By Matthew Demaske, Director of Threat Research, Adapt Forward Cyber Security

I'm always looking for ways an adversary can execute something on a system via "trusted" methods. One great example is Powershell. It's beloved by sysadmins and hackers alike. AV won't care and Virustotal says it's squeaky clean. I'm not going to go into all the various avenues of attack via Powershell because I'll be here all night. Just know that anything that's available to your users/staff is available to an attacker. After all, once someone gets into your network, what separates them from a legitimate user? Nothing. Any tool that will give you information about a system(s) is fair game. Ipconfig may seem like a harmless command, but it can give an attacker useful information. Same goes for a ton of other commands.

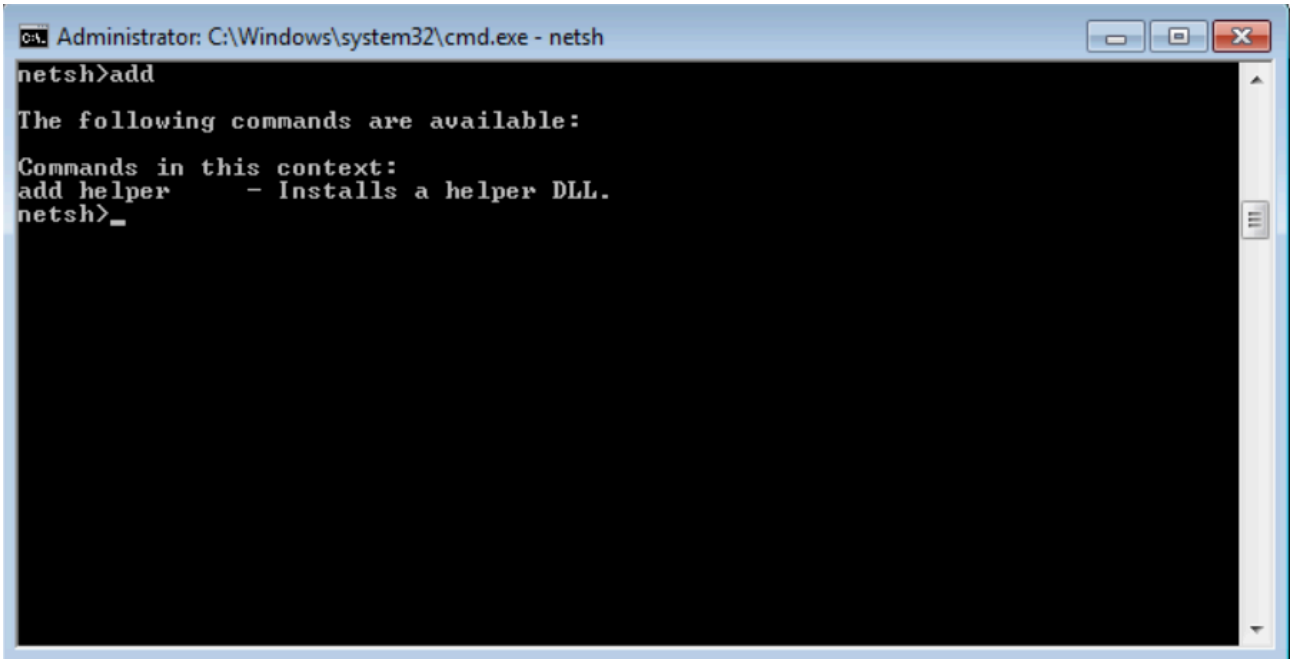
Check out this big list of native commands regularly used in recorded cyber attacks: http://blog.jpcert.or.jp/2016/01/windows-commands-abused-by-attackers.html. Built in native Windows tools are some of the best ways to pwn a network while avoiding detection.

The discouraging thing is that most of these commands occur thousands upon thousands of times legitimately on your network. Simply throwing ipconfig.exe into a blacklist for your SIEM to alert on will make people very angry at you. These aren't traditional indicators of compromise, but with added context, they absolutely can be. This is why I'm a fan of hiring real human people to hunt, instead of buying a box or a feed subscription. But, that's a rant for another post.

To get back on track, I was researching ways an adversary could use the Windows Firewall command line tool called netsh(NetShell) when I saw something curious in the list of available commands: "add"

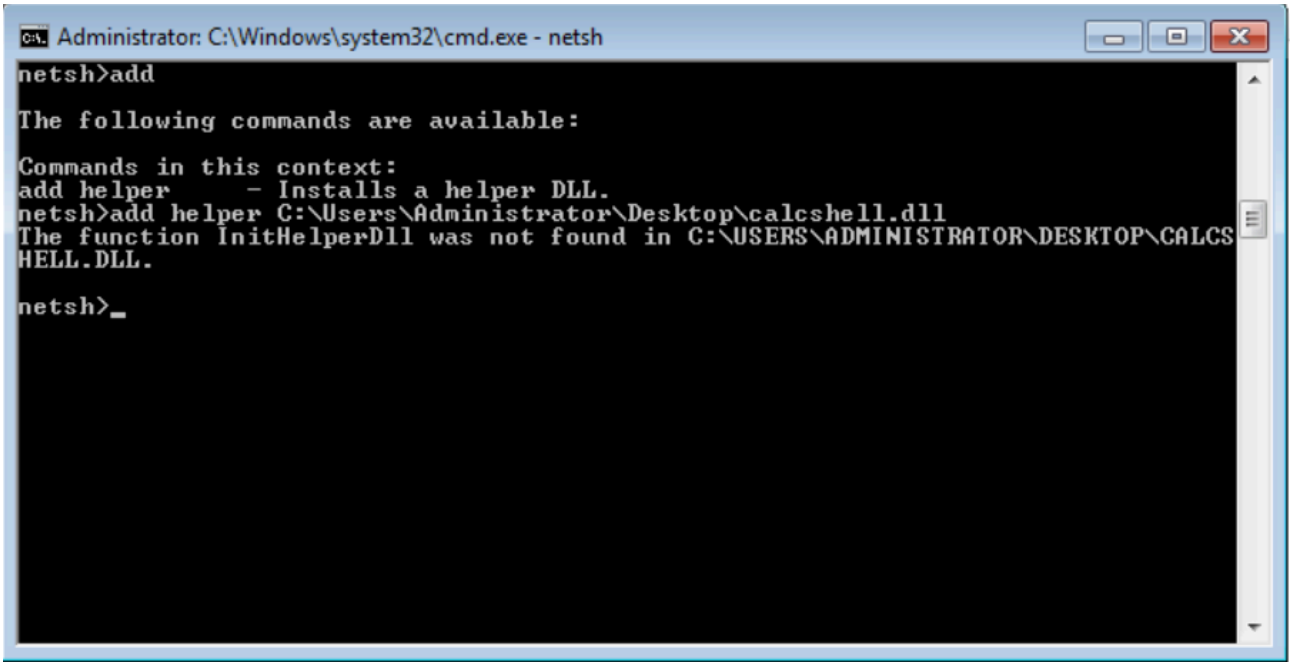


Add what?



Installs a DLL? Que!?

I found a POC DLL I use for stuff that just pops calc and figured why not. There’s no way it’s going to just run this, right?



Dang. What is InitHelperDLL? To Google we go. According to Microsoft

The InitHelperDll function is called by NetShell to perform an initial loading of a helper.

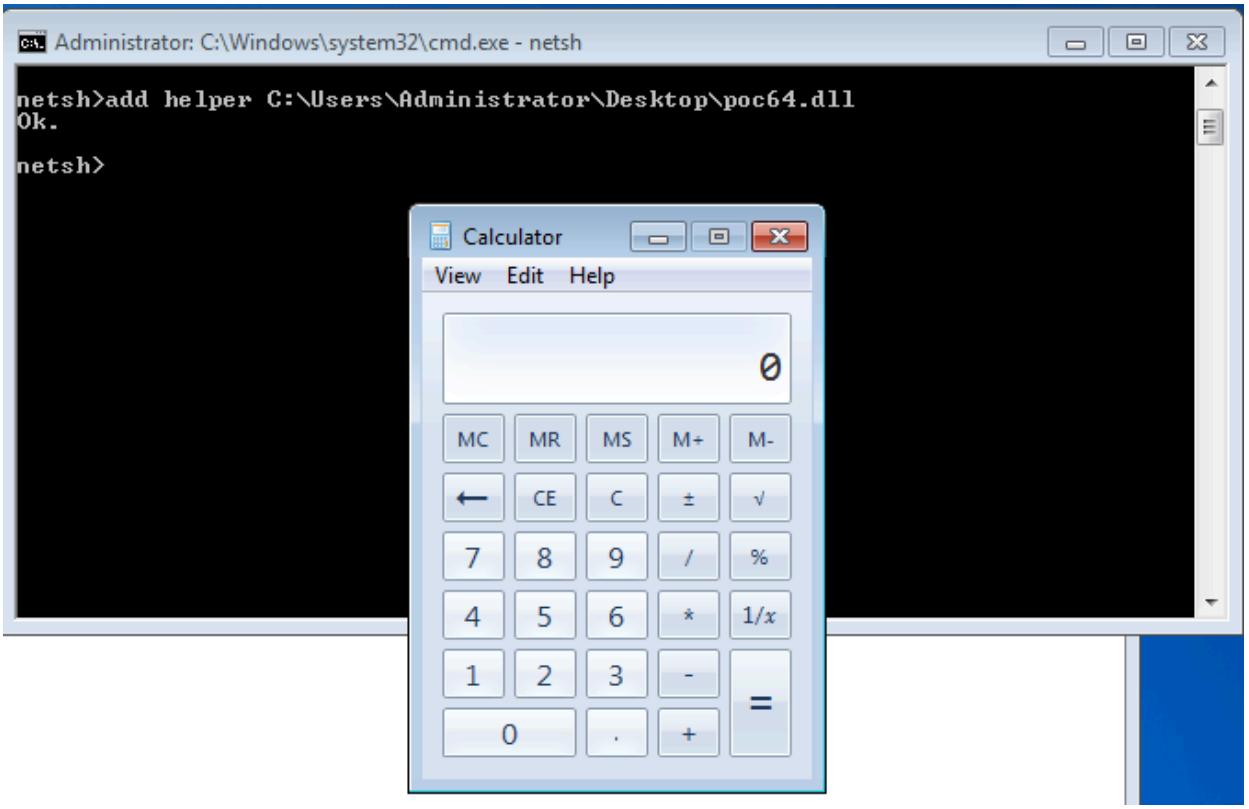
–https://msdn.microsoft.com/en-us/library/windows/desktop/ms708327(v=vs.85).aspx

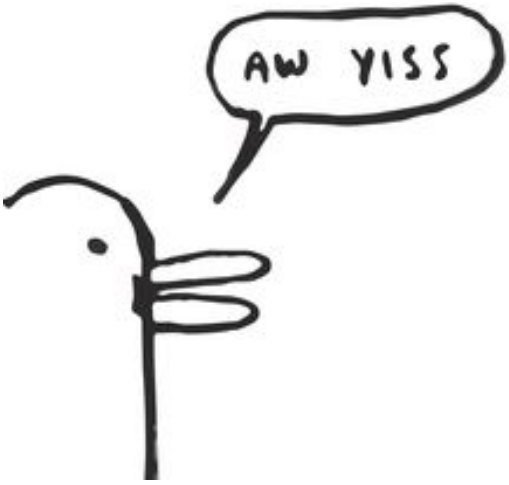
Ok, a required export. What's a helper?

NetShell helpers are DLL files that provide the functionality of a context. Additional helpers extend the functionality of NetShell by providing administrative scripting for networking tasks. Helpers generally provide configuration support, monitoring support, or both, for networking services, utilities, or protocols.

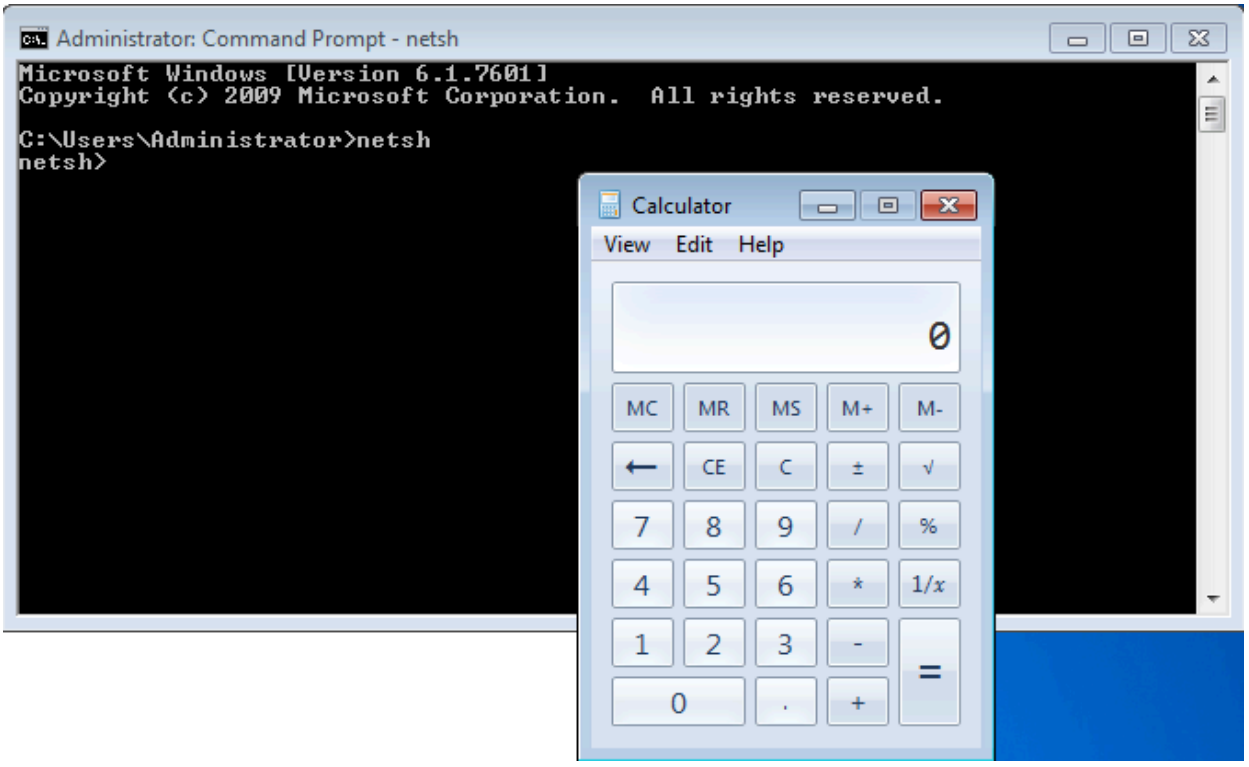
–https://msdn.microsoft.com/en-us/library/windows/desktop/ms708347(v=vs.85).aspx

At this point, I reach out to Casey Smith, who is really good at finding obscure ways of executing code in Windows. He's written extensively on the subject @ https://subt0x10.blogspot.com. I ask him if he's ever heard of this technique and he says he hasn't. A few minutes later and he's got a working POC going.





So where do we go from here? Well, I wanted to reverse what I had just done via the “delete helper <PATH>” command. So I opened another prompt to delete the entry and…



Whoa, it executed again. It's persistent. So, I went back to the Net Helper reference section and found this.

Helpers are DLL files that implement a NetShell context and zero or more of its subcontexts, and are registered with Windows through the system registry.

-https://msdn.microsoft.com/en-us/library/windows/desktop/ms708320(v=vs.85).aspx

through the system registry

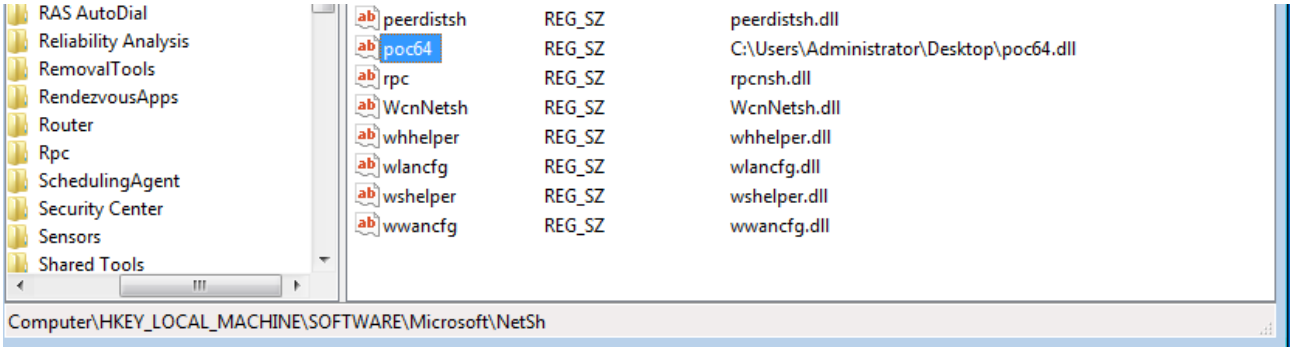
through the system registry

through the system registry

through the system registry

through the system registry

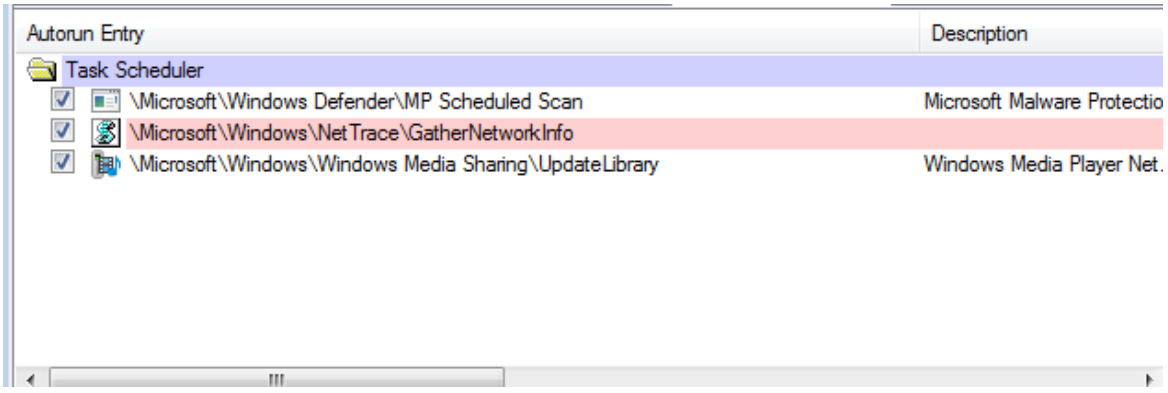
This just got better. Pulled up the registry and searched for my DLL.



The entry is made in the HKLM\SOFTWARE\Microsoft\Netsh key. All the other DLLs reside in the System folder, but it's not a requirement for your evil DLL. It'll run from anywhere. My advice would be to put it in a location where any user account can read from, like System or AppData. You do need admin rights for this by the way. Or at least rights that will let whatever context you're in write to HKLM.

The only caveat is that netsh.exe must be ran first for the dll to execute. Netsh doesn't automatically run on boot by default, but you could easily use a scheduled task for example. Or a start service. Or a Powershell profile. Or a RunOnce key. Or blah blah blah.

Default view of Autoruns won't catch it with any listed user account.



You would need to uncheck the "Hide Windows Entries" options to see it

"But, it's signed, and Virustotal didn't find anything!"

I know of one popular corporate VPN client program that regularly invokes netsh. I wonder how many of them do? Besides VPN clients, how many programs check or alter the windows firewall from the command line when they're installed or start? They use netsh to do it. Netsh is usually run under SYSTEM context, too. So, depending on the environment, you may not even need to force netsh to run with a traditional persistence mechanism. This is why recon is important before you go making noise you don't necessarily need to make.

Regarding the defensive side, if you're doing real-time hunting with a tool like Sysmon(which I HIGHLY HIGHLY recommend), you're going to want to look for any child processes of netsh.exe

```
Process Create:
UtcTime: 2016-09-22 04:53:20.994
ProcessGuid: {bd389c3e-63c0-57e3-0000-00107fc73800}
ProcessId: 3200
Image: C:\Windows\System32\calc.exe
CommandLine: "C:\Windows\System32\calc.exe"
CurrentDirectory: C:\Users\Victim2Win7\
User: VICTIMWIN7\Victim2Win7
LogonGuid: {bd389c3e-4192-57e3-0000-002023a11400}
LogonId: 0x14a123
TerminalSessionId: 2
IntegrityLevel: Medium
Hashes: SHA1=42D36EEB2140441B48287B7CD30B38105986D68F,MD5=10E4A1D2132CCB5C6759F038CDB6F3C9,SHA256
=C6A91CBA00BF87CDB064C49ADAAC82255CBEC6FDD48FD21F9B3B96ABF019916B,IMPHASH=CA7337BD1DFA93FD45FF30B369488A37
ParentProcessGuid: {bd389c3e-63c0-57e3-0000-0010f4b03800}
ParentProcessId: 3532
ParentImage: C:\Windows\System32\netsh.exe
ParentCommandLine: netsh
```

I have a client with a pretty sizable group of hosts and I searched going back 120 days looking for children of netsh.exe. There were zero among MILLIONS of netsh.exe processes started.

Other general tips/methods to stop or detect this attack:

-Obviously scan the HKLM\SOFTWARE\Microsoft\Netsh key for any new entries. Easy. You should have a dynamic list of possible persistence locations anyway in the registry anyway.

–Your team should be looking for registry changes made via CMD, powershell, and/or WMI. It may happen frequently, but the more time an analyst spends getting to know their territory, the easier it gets to spot things that look odd.

-DLL whitelisting. Microsoft's Applocker will let you configure policy rules on dll executions. This is why I'm a huge fan of organizations creating "gold images" of their operating systems. As a hunter, I know what the baseline is and searching for anomalies is easier. If I'm a system admin, gold images make whitelisting so much easier. I'll know exactly what to allow and what to block. Any changes need to be approved. Now, if you have no gold image, creating DLL whitelists can be a nightmare. If you start rolling out DLL rules, you can break a lot of important stuff. The good news is that you can create Applocker DLL rules that are audit only. The DLLs will still run, but there will be a Warning message written to the Applocker log. Suck those logs up into your SIEM and go hunting.

So, how important is this finding? I have no idea. Will it become the next heartbleed? Is it super NSA zero day complicated? Hardly. But, it's another avenue an adversary can use. Remember, defenders need to worry about numerous of ways an attacker can carry out their plan. Attackers only need to find one.

I doubt too many folks are monitoring the netsh key for changes or monitoring child processes of netsh.exe. But hey, maybe you will now.

Again, thanks to Casey Smith for the quick response and for the work on the POC.

I also want to give a shout out to @Adamb who hosts one of the best persistence/DFIR blogs out there. He wrote about the existence of net helper DLLs back in 2013: http://www.hexacorn.com/blog/2013/08/21/da-

lil-world-of-dll-exports-and-entry-points-part-3/

-Matt
