Table of Contents

Intro

The “Red & Blue” Ping Pong talk was fairly fast paced and covered a lot of ground regarding modern Windows attacks and defenses. We wanted to leave the attendees and anyone else interested in the topic a reference guide so that the demos could be replicated and so that the concepts in the talk could be easily applied in your respective organizations. This reference paper will also contain some links and other material that we didn’t get a chance to cover during the talk. This work isn’t new, but rather an aggregation of the current security research in this area.

Components of the Detection Tool

Get-InjectedThread

This tool is written by Jared Atkinson and is available here:
https://gist.github.com/jaredcatkinson/23905d34537ce4b5b1818c3e6405c1d2

It’s designed to detect malicious memory injection. Malware will often inject itself into running threads in attempts to avoid detection. The industry buzzword for this type of attack is “fileless” malware.

These links offer a good overview of DLL injection:

http://blog.opensecurityresearch.com/2013/01/windows-dll-injection-basics.html

https://github.com/stephenfewer/ReflectiveDLLInjection

The talk by Jared Atkinson and Joe Desimone is available here:

https://www.youtube.com/watch?v=EVBCoV8lpWc

While you probably cannot run this script on every system in your environment, we thought it would make a powerful addition to alerts, a Sysmon CreateRemoteThread event, for example, would make a good candidate.

Parent Child Relationships

This part of the script looked at what processes spawned cmd.exe. These types of process relationships can be interesting to examine in your environment. For example, if msword.exe spawned powershell.exe, that would definitely be an event worth looking into. In the script example, WMI queries were used to get this information. However, you could also apply the same type of logic to Sysmon Event ID 1 or Windows Event ID 4688.

Command Line Arguments

Recently there has been a flurry of research done by Casey Smith and others that attempts to abuse Microsoft binaries by using them to execute attackers' code. An example of this can be found here:

http://subt0x10.blogspot.ca/2017/04/bypassing-application-whitelisting.html

There are countless other examples of these types of attacks, a great resource can be found here:

https://github.com/api0cradle/UltimateAppLockerByPassList

Although they are labelled "App Locker Bypasses" they are also methods for attacker code execution. With tools like Sysmon and Event ID 4688, it is now feasible to monitor the usage of these "dual-purpose" binaries.

PE File Signatures

Certain Portable Executable files are "signed" by a trusted source. During malware triage it is common for analysts to examine the digital signature of an executable file to help determine its legitimacy. Attackers are well aware of this and certain techniques apply "steal" a digital signature for a trusted file and apply it to an untrusted one. This process has also been automated. During the demo, we ran a PowerShell scriptlet on all executables ran on the system. The demo showed that a certain file failed the signature check. Newer research on this topic by Matt Graeber has demonstrated how to bypass this check. This research can be found here:

https://specterops.io/assets/resources/SpecterOps_Subverting_Trust_in_Windows.pdf

PowerShell Logging

The script ran through all process executions and looked for those executed by PowerShell, it then looked for the –encodedcommand flag and attempted to decode the malicious string used. PowerShell is an extremely powerful language used regularly by attackers. The most comprehensive way to detect these kinds of attacks is by enabling the powerful logging features available in PowerShell.

More information regarding this can be found here:

https://www.fireeye.com/blog/threat-research/2016/02/greater_visibilityt.html

https://blogs.msdn.microsoft.com/powershell/2015/06/09/powershell-the-blue-team/

It is important to remember that PowerShell transcript logs will log the **decoded** version of all encoded commands. It will also log the output of the command.

Credential Access

This portion of the demo demonstrated an – in my opinion – incredibly powerful feature of Sysmon; the ability to log process access. As a reference, the Sysmon rule used for this is:

```
<ProcessAccess onmatch="include">
<TargetImage condition="is">C:\Windows\System32\lsass.exe</TargetImage>
```

```
        <TargetImage condition="is">C:\Windows\System32\winlogon.exe</TargetImage>
</ProcessAccess>
<ProcessAccess onmatch="exclude">
        <SourceImage condition="is">C:\Windows\system32\MRT.exe</SourceImage>
        <SourceImage condition="is">C:\Windows\system32\wbem\wmiprvse.exe</SourceImage>
        <SourceImage condition="is">c:\windows\system32\svchost.exe</SourceImage>
        <SourceImage condition="is">C:\Program Files\Windows Defender\MsMpEng.exe</SourceImage>
</ProcessAccess>
```

Although Sysmon rules are a little bit tricky, there will be some resources at the end of the document to help manage them. This rule in particular tells Sysmon to log access to lsass & winlogon.exe, excluding the noisy process below.

By enabling this rule, we are able to log when a process accesses sensitive processes containing credentials. We all know about Mimikatz, yet few defenses exist for PowerShell & Jscript versions of it.

Unmanaged PowerShell

As a response to the rising tide of PowerShell attacks, many thought that disabling powershell.exe from their systems would help mitigate risk. Unfortunately, security is not so simple. In the words of Lee Homes "PowerShell.exe itself is just a simple native application that hosts the CLR, and the –Version switch tells PowerShell which version of the PowerShell assemblies to load." In other words, the engine that runs PowerShell can be called by any application capable of leveraging the .NET framework.

We can write a Sysmon rule to detect this:

```
<ImageLoad onmatch="include">
        <ImageLoaded condition="contains">System.Management.Automation.ni.dll</ImageLoaded>
        <ImageLoaded condition="contains">System.Management.Automation.dll</ImageLoaded>
</ImageLoad>
<ImageLoad onmatch="exclude">
        <Image condition="is">C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell_ise.exe</Image>
        <Image condition="is">C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe</Image>
        <Image condition="is">C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe</Image>
</ImageLoad>
```

This rule looks for the two .dll file required for the PowerShell engine to run not loaded by powershell.exe or the powershell ISE.

More information regarding this topic can be found here:

http://www.leeholmes.com/blog/2017/03/17/detecting-and-preventing-powershell-downgrade-attacks/

https://github.com/leechristensen/UnmanagedPowerShell

Registry Persistence

Sysmon is a great tool to detect basic registry persistence. Although Windows offers SACL auditing on registry objects, the filtering capabilities of the Sysmon configuration file make Sysmon a more attractive option for those that want to reduce noise. The following configuration would trigger an event when a Run Key is added to the Windows registry:

```
<RegistryEvent onmatch="include">
        <TargetObject condition="contains">\CurrentVersion\Run</TargetObject>
</RegistryEvent>
```

The SwiftOnSecurity Sysmon configuration comes pre-loaded with an extremely robust set of registry rules. I would encourage you to test the rules however, as some may not fire as intended in your specific environment.

WMI Persistence

A more advanced and stealthy persistence technique involves the use of WMI filters and consumers. The following resources give a very good explanation of the technique:

https://www.fireeye.com/blog/threat-research/2016/08/wmi_vs_wmi_monitor.html

This is a relatively new Sysmon rule, and the configuration syntax used to catch WMI persistence is simply:

```
<WmiEvent onmatch="exclude"></WmiEvent>
```

I have tried this rule in a production and it did not produce a lot of noise, however, please note that this rule will capture every Sysmon event so please test in a development environment prior to deployment.

Additional resources regarding WMI can be found here:

https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf

https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/wp-windows-management-instrumentation.pdf

https://www.darkoperator.com/blog/2017/10/15/sysinternals-sysmon-610-tracking-of-permanent-wmi-events

https://www.darkoperator.com/blog/2017/10/14/basics-of-tracking-wmi-activity

Network Connectivity of Processes

Capturing Layer 7 application traffic has always proved to be somewhat of a challenge. Many organizations ingest firewall logs into a centralized SIEM device, however, it is often difficult to build a picture of what happened due to NAT's, non static IPs and the like. Sysmon gives you the ability to view network connectivity at the process level. During the demo this functionality was utilized to demonstrate beaconing behavior as Powershell was "calling home" quite often.

The following Sysmon configuration was used:

```
<NetworkConnect onmatch="include">
        <DestinationPort>80</DestinationPort>
        <DestinationPort>443</DestinationPort>
        <DestinationPort>4444</DestinationPort>
        <DestinationPort>4445</DestinationPort>
</NetworkConnect>
<NetworkConnect onmatch="exclude">
        <Image condition="image">OneDrive.exe</Image> <!--Microsoft:OneDrive-->
        <Image condition="image">Spotify.exe</Image> <!--Spotify-->
        <Image condition="end with">AppData\Roaming\Dropbox\bin\Dropbox.exe</Image> <!--Dropbox-->
        <Image condition="is">C:\Windows\System32\svchost.exe</Image>
        <!--SECTION: Microsoft-->
        <Image condition="image">OneDriveStandaloneUpdater.exe</Image> <!--Microsoft:OneDrive-->
        <DestinationHostname condition="end with">microsoft.com</DestinationHostname> <!--Microsoft:Update delivery-->
        <DestinationHostname condition="end with">microsoft.com.akadns.net</DestinationHostname>
```

```
        <DestinationHostname condition="end with">microsoft.com.nsatc.net</DestinationHostname>
</NetworkConnect>
```

This section is again straight out of the SwifOnSecurity Sysmon configuration, if you wish to replicate the data found in the report, the following PowerShell script will accomplish this:

```
$EventsID3 = Get-WinEvent -FilterHashtable @{logname="Microsoft-Windows-Sysmon/Operational";id=3} |
Get-WinEventData | select *
$IPConnectionData = $EventsID3 | Group-Object -Property e_Image,e_DestinationIP | sort -Property
count -Descending | Select-Object count, name
```

Please note Get-WinEventData is a separate module which can be found on GitHub

Lateral Movement

Does this sound like your environment: hundreds of laptops deployed, all using the same local administrator account that was set years ago? If so, you're probably not alone. Thankfully, Microsoft has made this problem solvable.

During the talk we showed a screenshot of an attacker running CrackMapExec to execute a command on multiple machines using the same local administrator password. By implementing the Group Policies in the screenshot we are preventing two things:

1) Local accounts cannot use a network (Type 3) logon. Therefore, even if all passwords are the same, an attacker is limited to using these credentials locally and not across the network
2) Domain administrators are not permitted to log into machines that are in a lower trust tier and are therefore most likely to get compromised

By doing this, we limit our risk of both an attacker spreading laterally across the environment as well as domain admin credentials getting compromised.

I wrote a short blog post illustrating the basics of these GPOs here:

https://haveyousecured.blogspot.ca/2017/09/making-lateral-movement-difficult-in.html

More in-depth explanations can be found here:

https://blogs.technet.microsoft.com/jepayne/author/jessica-payne-msft/

https://adsecurity.org/wp-content/uploads/2017/04/2017-BSidesCharm-DetectingtheElusive-ActiveDirectoryThreatHunting-Final.pdf

Device Guard

Device guard is an application whitelisting solution built into Windows 10 Enterprise and Server 2016. The demo showed a screenshot of a system running an untrusted application (putty) and a script designed to inject shellcode into a running process. The screenshot below showed the same applications running on a system with Device Guard running, both were blocked from executing. Although Device Guard is a little bit of a beast to implement the reward is certainly there. There is not an abundance of resources available for Device Guard implementations, however, Matt Graeber has authored a number of must read blog posts about deploying device guard:

http://www.exploit-monday.com/2016/09/introduction-to-windows-device-guard.html

http://www.exploit-monday.com/2016/10/code-integrity-policy-reference.html

http://www.exploit-monday.com/2016/11/code-integrity-policy-audit-methodology.html

http://www.exploit-monday.com/2016/12/updating-device-guard-code-integrity.html

Microsoft also provides some documentation regarding Device Guard:

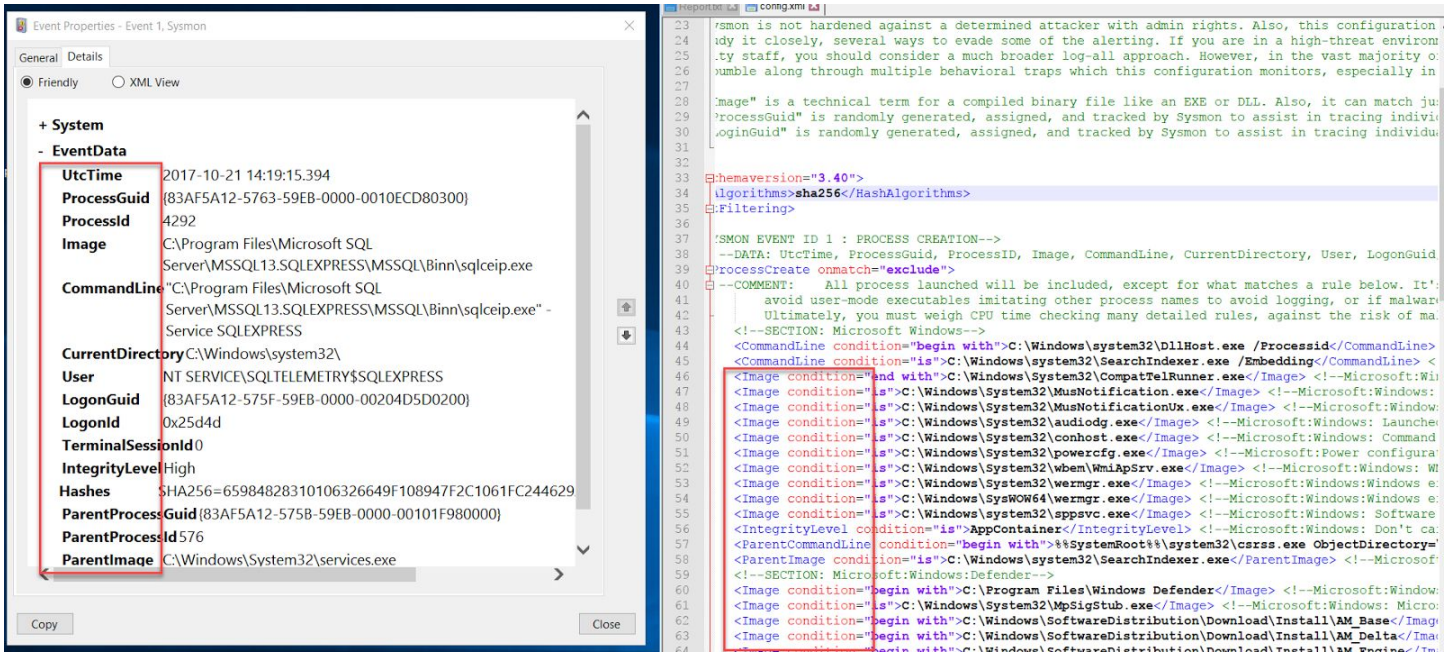
https://github.com/MicrosoftDocs/windows-itpro-docs/blob/master/windows/device-security/device-guard/deploy-code-integrity-policies-steps.md

Sysmon Configuration and Events

Sysmon events can be tricky to work with and manipulate. The best method I found to tweak Sysmon events is to start with what you would like Sysmon to alert on and go from there. Depending on the configuration and

your environment, the odds are that Sysmon would create noise that you would like to filter away, and not that Sysmon would not generate an event for a particular action.

The details tab of the event properties gives a clear view of what elements are available to filter on within the configuration:



Another extremely helpful resource for creating Sysmon configurations is a GUI interface made by Nader Shalabi, it is available here:

https://nosecurecode.blog/2017/08/12/sysmon-shell-release-1-1/

As of time of writing, this version does not support the newest version of Sysmon and new WMI event filter and consumer events. However, you can easily build a base configuration with the GUI, export it, change the Schema Version of the configuration file to 3.40 and add the relevant WMI filters.

Another resource for programmatically managing Sysmon configurations is by Carlos Perez and is available here:

https://www.darkoperator.com/blog/2017/2/17/posh-sysmon-powershell-module-for-creating-sysmon-configuration-files

Windows Event IDs

Recently, Windows logs have been sexy again with Microsoft introducing more detailed logging capabilities in newer (2012+) versions of its operating systems. As with so many things related to Windows and Active Directory, Sean Metcalf provides a wealth of resources. Sean's 2016 presentation at BSides Charm contains an invalueable listing of Windows Events to monitor in your environment

The slides can be found here:

https://adsecurity.org/wp-content/uploads/2017/04/2017-BSidesCharm-DetectingtheElusive-ActiveDirectoryThreatHunting-Final.pdf

And the video here:

https://www.youtube.com/watch?v=9Uo7V9OUaUw

More generally, ADSecurity.org and anything written by Jessica Payne at Microsoft should be considered required reading for those looking to secure Window environment.

Microsoft also provides a resource for a recommended audit policy configuration:

https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/audit-policy-recommendations

The Microsoft recommended baseline is a great start, but does not seem to include things like "Include command line in process creation events" or any type of PowerShell logging.

Quick Wins

Disable NetBIOS over TCP/IP

Although not mentioned in the presentation, NetBIOS over TCP/IP is an extremely dangerous configuration item to have enabled on your network. A good explanation of this attack can be found here:

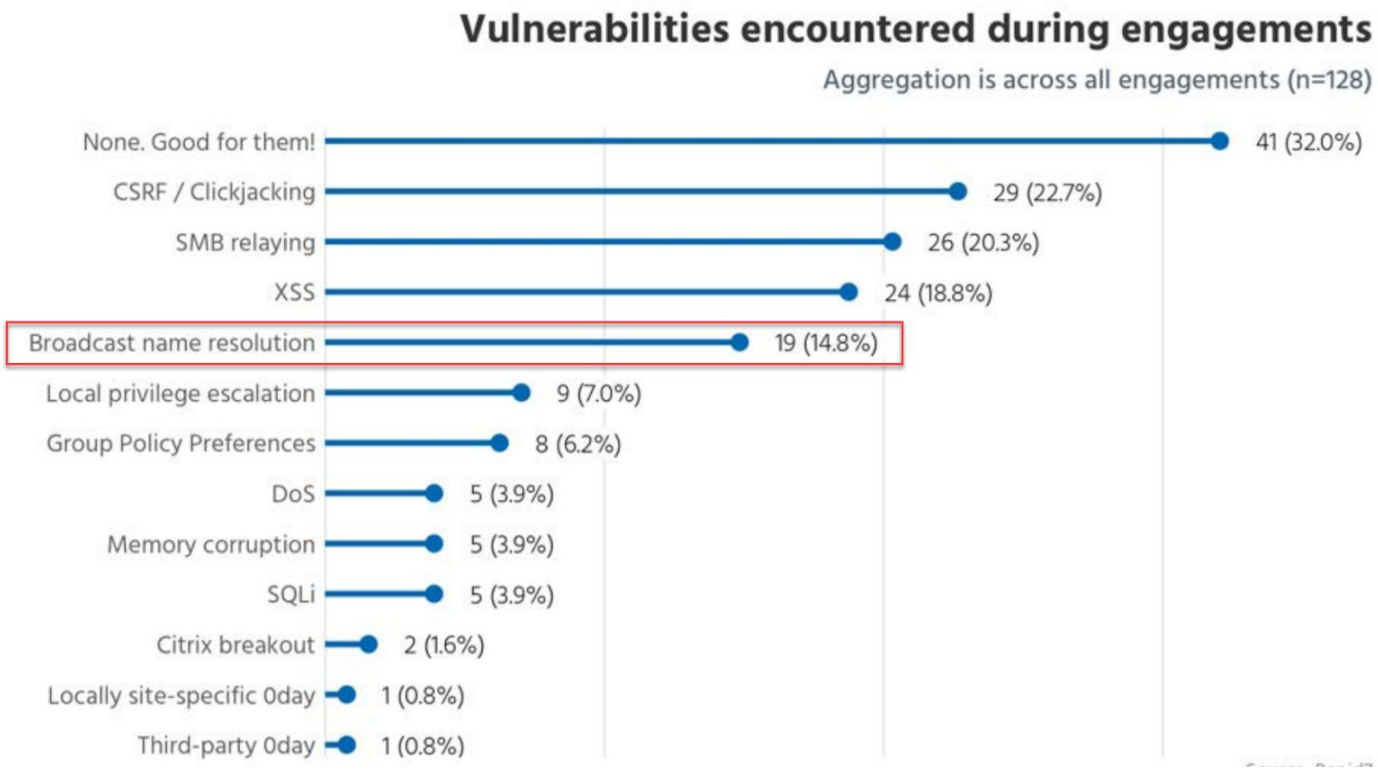
https://pentest.blog/what-is-llmnr-wpad-and-how-to-abuse-them-during-pentest/

In short, this attack allows an attacker on your network to capture NTLM or LM (hopefully not) hashes on your network for offline cracking. Given that users tend to have weak passwords, this attack is extremely dangerous. You might be thinking to yourself that NetBIOS over TCP/IP is an ancient protocol, but looking at some data, this is still a very common attack vector.

The following Report from Rapid 7 presents some data from their pen tests, I have found it very useful in prioritizing defenses, as solid data like this is difficult to come by:

https://www.rapid7.com/globalassets/_pdfs/whitepaperguide/rapid7-research-report-under-the-hoodie.pdf

Take a look at the following chart from the Rapid 7 Report:



Broadcast name resolution attacks accounted for almost 15 percent of the vulnerabilities discovered by Rapid 7 during their assessments.

Similar research from Praetorian suggests the same trend:

https://p16.praetorian.com/downloads/report/How%20to%20Dramatically%20Improve%20Corporate%20IT%20Security%20Without%20Spending%20Millions%20-%20Praetorian.pdf

Further resources regarding these attacks:

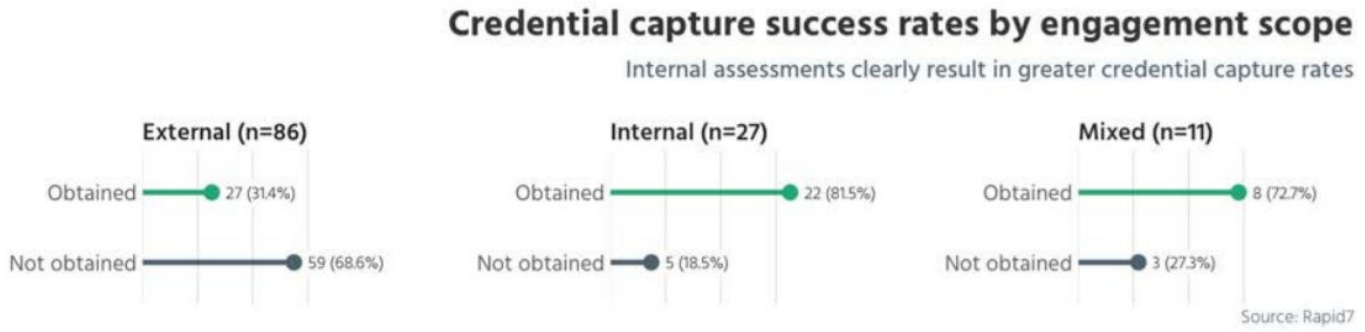
https://www.trustwave.com/Resources/SpiderLabs-Blog/Top-Five-Ways-SpiderLabs-Got-Domain-Admin-on-Your-Internal-Network/

https://www.trustwave.com/Resources/SpiderLabs-Blog/Responder-2-0---Owning-Windows-Networks-part-3/

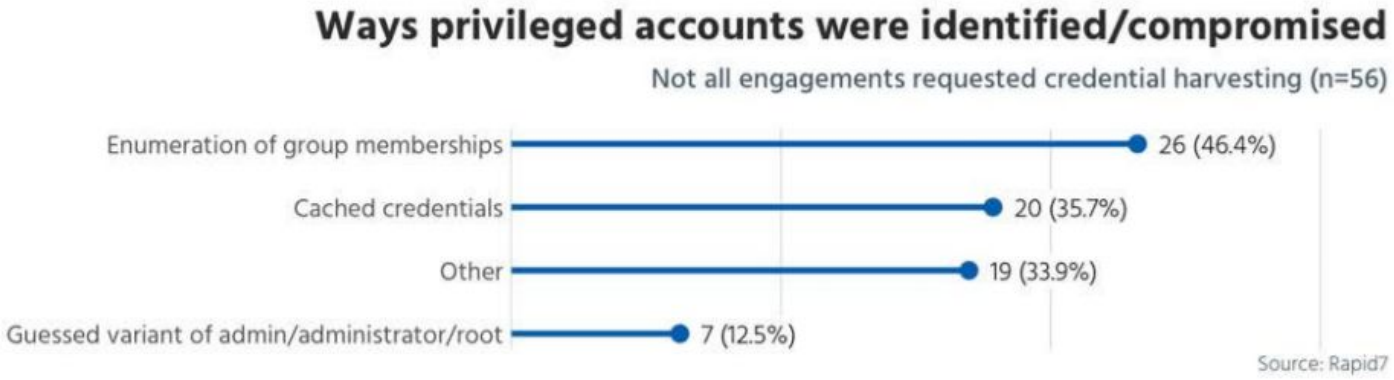
Defacto tool to perform these attacks, Responder by Laurent Gaffie

WDIGEST

Looking at the same Rapid 7 report as highlighted above, let's take a look at how often credentials were compromised during engagements:



In internal engagements, credentials were captured over eighty-percent of the time. Safe to say, therefore, that attackers are after credentials and that the likelihood of credential compromise is high. Let's dig a bit deeper and look at privileged accounts.



Interesting, cached credentials result in thirty-five percent of total credential compromise, if we add the guessed variants that means that close to fifty-percent of credential compromise could have been very easily mitigated. As mentioned above again, the numbers from Praetorian seem to suggest the same trend:

Praetorian has a very helpful and detailed article around disabling clear-text storage of credentials, which can be found here:

https://p16.praetorian.com/blog/mitigating-mimikatz-wdigest-cleartext-credential-theft

Taking a step back, it's pretty clear that just by disabling NetBIOS over TCP/IP and WDIGEST, you mitigate a huge number of attacks.