Global Information Assurance Certification Paper

Hunt Teaming

GIAC (GPEN) Gold Certification

Author: Joff Thyer, jsthyer@gmail.com
Advisor: Dominicus Adriyanto
Accepted: November 25, 2014

Abstract

In today's world of persistent threats, and multiple compromise attack vectors, defenders are losing the battle. Mobile platforms, and cloud computing have broken the traditional defense model. Data assets are becoming increasingly spread and harder to maintain control of. Traditional defense models are breaking down. In the penetration testing profession, we are all too proud of the idea that we can gain access to environments and "win" in our engagements. But we should be honest with ourselves, its just not that difficult anymore. Sure we need sophisticated techniques, and depth of knowledge but our human targets continually succumb to our methods. So why don't we offer a new paradigm? Let's assume that enterprise X is already compromised and try to help them find out what the telltale signs are, how bad it might be, and help them assess the appropriate response. Let's arm our friends with new techniques, and become hunters of compromise activity within an environment.

Acknowledgements

The content of this paper is derived from my research, and my associations with many very talented people. Specifically I would like to thank John Strand, and all of the talented employees of Black Hills Information Security. I acknowledge the work of Seth Misenar and Eric Conrad in continuous security monitoring. There is no "I" in this industry but the collective "we" to make this digital world a more secure place.

Joff Thyer, jsthyer@gmail.com

Introduction

Over the past several years, the information technology industry has dramatically shifted from a desktop workstation centric, corporate owned computing asset model to a model of performing business processing tasks from anywhere with any capable device. This is evident through the dramatic increase in tablet, and smartphone use by organizational employees, and demand of employees to be able to use their own devices to manage daily business tasks. (Gartner, 2014)

While some organizations are resisting the change and retaining all critical business processing tasks on corporate managed endpoints, a great many others are adopting a hybrid model of enabling mobile devices to access a restricted set of business data. In many cases, a model of enabling email access natively on mobile devices, and using a form of virtual desktop technology to access business data remotely on mobile devices has emerged. (Tom's IT Pro, 2014)

Software vendors driving the mobile device revolution have well understood that the traditional organizational domain boundaries are blurring, and have positioned themselves as cloud solution providers to take advantage of this new computing paradigm, and to extend the services model for less capable smaller entities.

The traditional model of organizational information security defense is based on a hard shell exterior, perimeter based, approach whereby the boundary between the Internet and the organization is heavily fortified through the use of firewall, and intrusion detection/prevention technologies. This is usually complemented by desktop workstation protections such as host based intrusion detection, application whitelisting, and anti-virus solutions. The traditional model of information security defense is premised on the idea that the defenses will *prevent* system compromise a significant percentage of the time leaving organization staff to deal with the few systems that are compromised and need appropriate incident response and remediation processes applied.

In the context of today's threat environment, the outdated prevention-based model is dramatically failing. Compromise has become the norm, often discovered by third

party notification by which time is pervasive, widespread and data having already been ex-filtrated. (Schneier on Security, 2013)

It is well understood by information security professionals that an initial compromise of an environment is not something difficult to achieve. Many penetration testing professionals will use a social engineering based ruse, such as a targeted phishing based email with URL content or file attachment, that delivers malware to a system to achieve a command and control (C2) channel. Content is now designed to evade email delivery filters, and to not trip antivirus solutions. Some antivirus vendors have already admitted that their technology is failing. (Arstechnica, 2014)

Initial compromise only delivers a beachhead within a computing environment. The real work for an attacker, or professional penetration tester, begins after the beachhead is established. This work is that of learning the environment, spreading access across to other systems, becoming administrative, and exfiltration of data. This is work that takes much longer to achieve, sometimes measured in days, weeks, or even months.

If we assume that initial compromise and establishment of a beachhead of reasonably trivial, occurring in a short period of time, and we have a context of a traditional perimeter defense model which might not even have detected this activity, then the traditional model is not correctly focused.

As security defenders, and security solution providers, we should re-focus on the much longer window of time that is presented post the initial compromise situation. We should focus our efforts on examining the indicators of established C2 channels, indicators of escalation, and spreading (pivoting) access within an environment.

This idea is known as a continuous security monitoring model, and begins with a very important assumption of already being compromised and working from there. The activities to search for indicators of compromise within an environment are known as "Hunt Teaming". (Securosis Research, 2013)

Joff Thyer, jsthyer@gmail.com

1. Hunt Teaming

Hunt Teaming is the process of executing techniques within a computing environment to locate enterprise network indicators of compromise. When performing hunt teaming activities, ideally the team needs to:

- Identify command and control (C2) channels
  - Identify unusual domain name system (DNS) queries
  - Identify patterns of C2 traffic
  - Identify unusual HTTP agent strings
- Identify intruder pivoting activity
- Identify evidence of persistence, and unusual registry keys
- Identify software installations on systems that deviate from baseline
- Identity unusual code signing certificates

For the purposes of this paper, we are going to assume that the content is a mid-sized enterprise network environment of Windows 7 domain joined endpoints. We will also assume that while mobile devices are likely in use, the mobile device primary access mechanism to business processing data is via some form of remote access using a virtual desktop technology.

1.1. Command and Control (C2) Channel Identification

Successful compromise of an organizational workstation is typically going to occur through social engineering, or inadvertent web browsing compromise. (The human OS: Overdue for a social engineering patch, 2014) In the social engineering category, electronic mail delivery of an HTML formatted email that contains targeting and compelling information is quite typical. An HTML URL/link will commonly be

employed that takes the user to some form of Java applet, or browser compromise malware. Alternatively, a richer document format such as PDF, or Microsoft Office macro based malware will be employed.

After successful initial workstation compromise, it is very common for an outbound command and control (C2) channel to be established. When establishing this channel, an attacker will typically use an outbound TCP port that is permitted through perimeter defenses. It is not unusual to see outbound C2 channels using TCP 443 over HTTPS, or other TCP ports such as 22, 25, 80, and 143 for example. While some C2 channels may be encrypted, there still remains a tremendously amount of C2 activity that is unencrypted, with some channels even using the Internet Relay Chat (IRC) protocol.

1.1.1. Examining DNS Server Cached Data

Organizations typically will use an internal DNS server that often is configured in a split DNS topology. Split DNS is where an internal DNS server is used to service the internal network clients, but is configured to forward any unresolved requests to an external caching server. Only the internal DNS will hold authoritative copies of the internal network zones. The internal/intranet DNS infrastructure will cache external records for the appropriate time to live (TTL) received after first resolution.

A DNS cache can be examined by dumping the data out of the DNS server. In the case of a BIND9 Linux based server, this is achieved using the "rndc dumpdb –cache" command. In the case of a Windows domain name server, "dnscmd" can be used. In Windows server 2008-R2 and beyond, a Powershell 4.0 cmdlet named "Show-DnsServerCache" is available.

After extracting DNS cache data, one method that can be used is to look for any suspicious Internet domains that are contained within the cache to detect evidence of potential botnet or C2 channel activity. One text-formatted list of suspicious domains is available at https://isc.sans.edu/suspicious_domains.html.

In addition to seeking out suspicious domains, it is possible to independently perform DNS lookups on the extracted cache data (for external Internet resources only), and validate that the results in the cache match the real-time resolution being performed. Any mismatches would be a potential indicator of DNS cache poisoning.

1.1.2. Detecting DNS Covert Channels

Another method of creating a C2 channel is to use DNS to tunnel the command and control data out of the secured enterprise network. Examples of DNS tunneling tools include *OzymanDNS, Dns2tcp, Iodine, Heyoka, Dnscat, NSTX, and Dnscapy.* (DNS Tunneling, InfoSec Institute) DNS can also used as a mechanism to ex-filtrate data from a network, and are commonly used to transfer files from an organization. (Psichron)

When a DNS covert channel is established, there will be specific characteristics of the DNS traffic that are not typical of normal DNS traffic. For example, the size of the response packet may be unusually large, or uncommon DNS record types will be used (such as a TXT record), or the number of NXDomain responses might be unusually high. It is also fairly common in many implementations for DNS covert channels to have fairly unusual query names such as "dnscat.29.a.b.c.d.domain.tld" for example. (Detecting DNS Tunneling, 2014)

In order to perform this sort of analysis, we either are required to have live network data access in the form of a network tap and analyze network data passing by, or we can also enable DNS logging on internal DNS servers and analyze log data.

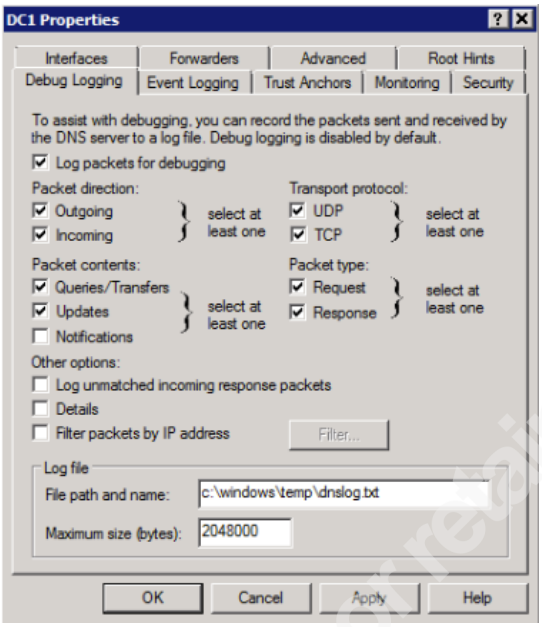

Figure 1: Enabling DNS logging on Windows domain controller

Other methods that might be employed include the frequency of DNS traffic detected on a per client basis. For example, if a sizable sample of endpoints in an environment send an average DNS query rate of ten queries per minute, however one workstation is suddenly performing 60 queries per minute, a statistical standard deviation of a core network traffic capture of DNS traffic would clearly show an outlier.

Aside from frequency on overall DNS record queries, and responses, the DNS logging data gives us the ability to total up individual query types, response types, and response codes. After totaling this data, a mean and standard deviation can be calculated. If any of these totals are more than two standard deviations away from the mean, we can flag these records and display them as outliers for further examination.

1.1.3. Examining HTTP User-Agent Strings

In an environment where web proxy logs are available, or perhaps using a network packet trace utility such as “tcpdump”, or “ngrep”, it is possible to look at all HTTP protocol user-agent strings being used in web requests. Since there are only a

limited number of browser technologies, the most common user-agent strings should be identifiable and have a high transaction frequency count. Other agent strings such as those generated from Anti-Virus engines, and gaming products may also be visible with a relatively high frequency. (Identifying Malware via User-Agent Headers, 2010)

If we gather a relatively long period (at least one full day) of data capture, or logs, we can count the HTTP user-agents by frequency and possibly source address. Then we can sort the frequencies and look for the most infrequent requests that will likely be highly unusual user-agent strings. Assuming that we retain the internal source address that the request is transmitted from, the unusual infrequent requests can be traced back to specific devices for further analysis.

1.2. Identifying Pivoting Activity

A very typical sequence of events, after a beachhead is established, is for an attacker to seek administrative escalation on a local workstation and then examine opportunities for expanding access. There are numerous methods for attempting to find a locally administrative credential which range from examining group policy preferences (GPP) data, examining unattended installation data, keystroke logging, extracting desktop application credentials, and dictionary, or brute force attacks. A domain level administrative credential is even more useful for escalation; if this data is discoverable through one of the common escalation methods.

Additionally, it is quite common in many organizations for a locally administrative credential to exist on many Windows endpoints, often designed for recovery in the event that a workstation become disjoined from the domain environment. It is also relatively common for this locally administrative credential to be the same username, and password across multiple client endpoints. It should be noted here that changes to Microsoft Windows in the form of KB2871997 (May 2014) were implemented to prevent remote logon of any local credential that does not have relative identifier (RID) of 500.

Assuming that either a domain level or locally administrative credential has been obtained, it is very common to test logon against neighboring workstation authentications

using an SMB logon method, and then use either PSEXEC or "*wmic process call create*" as a method to achieve a login session on a remote workstation.

To provide some context surrounding figures 2 through 10 depicted below, a beachhead workstation with IP address of 10.10.1.198 has been compromised and in turn being used to examine the surrounding sub-network for remote login opportunities. It is assumed that a locally administrative credential has already been discovered and that the encrypted NT hash is being used for attempted login.

Depending on the audit policies deployed in the Windows environment, the PSEXEC scenario can produce a significant amount of information in event logs. In the example entries below, audit policies were enabled for logon, process tracking, object access, privilege use, and system events on a Windows 7 system. Admittedly, a stock audit policy under Windows 7 typically logs only the account logon event however this is still sufficient to detect pivoting activity. Notably, in August 2014, Microsoft Sysinternals (Mark Russinovich) released a process and file event-logging tool called sysmon which logs full process creation details along with an MD5, or SHA hash.

```
msf auxiliary(smb_login) > set SMBUser
SMBUser => Administrator
msf auxiliary(smb_login) > set SMBPass
SMBPass => aad3b435b51404eeaad3b435b51404ee:8a7b0739afe5ec216e20568bdf85ba62
msf auxiliary(smb_login) > set RHOSTS
RHOSTS => 10.10.1.224/27
msf auxiliary(smb_login) > set THREADS
THREADS => 20
msf auxiliary(smb_login) > run

[+] 10.10.1.236:445 - SUCCESSFUL LOGIN (Windows 7 Ultimate 7601 Service Pack 1)
Administrator : aad3b435b51404eeaad3b435b51404ee:8a7b0739afe5ec216e20568bdf85ba6
2 [STATUS_SUCCESS]
[*] Scanned 19 of 32 hosts (059% complete)
[*] Scanned 20 of 32 hosts (062% complete)
[*] Scanned 32 of 32 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(smb_login) >
```

Figure 2: SMB scanning for a Windows system that accepts our credentials

```
msf exploit(psexec_psh) > set SMBUser
SMBUser => Administrator
msf exploit(psexec_psh) > set SMBPass
SMBPass => aad3b435b51404eeaad3b435b51404ee:8a7b0739afe5ec216e20568bdf85ba62
msf exploit(psexec_psh) > set PAYLOAD
PAYLOAD => windows/x64/meterpreter/reverse_tcp
msf exploit(psexec_psh) > run

[*] 10.10.1.236:445 - Executing the payload...
[*] 10.10.1.236:445 - Binding to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_
np:10.10.1.236[\svcctl] ...
[*] 10.10.1.236:445 - Bound to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np
:10.10.1.236[\svcctl] ...
[*] 10.10.1.236:445 - Obtaining a service manager handle...
[*] 10.10.1.236:445 - Creating the service...
[*] 10.10.1.236:445 - Starting the service...
[*] 10.10.1.236:445 - Removing the service...
[*] 10.10.1.236:445 - Closing service handle...
msf exploit(psexec_psh) >
[*] Encoded stage with x86/countdown
[*] Sending encoded stage (972818 bytes) to [redacted]
[*] Meterpreter session 10 opened ([redacted]:443 -> [redacted]:49298) a
t 2014-08-29 15:19:50 -0400
```

Figure 3: Using Metasploit "psexec_psh" to remotely login and start a service

Figure 2 shows how a penetration tester or attacker might use the Metasploit "smb_login" module to scan for neighboring systems to attack. Figure 3 shows at attacker setting up the "psexec" method with a Powershell based payload to attempt remote system login, and the delivery of shellcode starting as a service. This shellcode will create a TCP connection from the exploited workstation back to the attacker controller server.

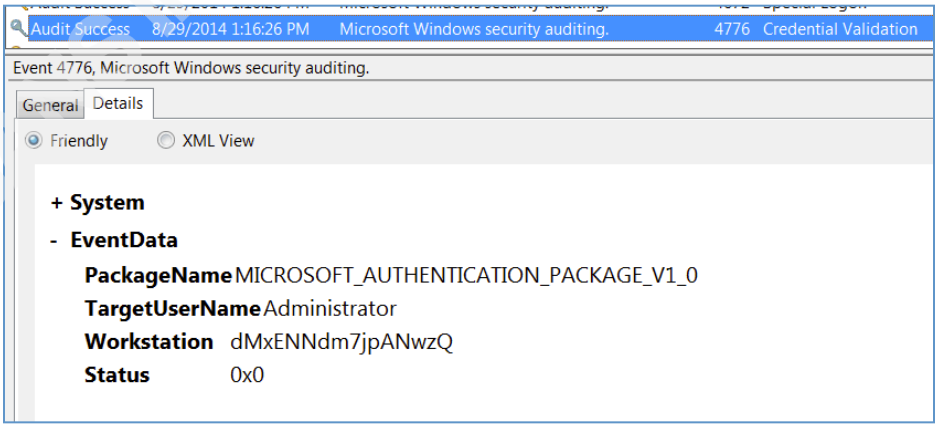

Figure 4: "Credential validation" event entry with unusual Workstation name

Joff Thyer, jsthyer@gmail.com

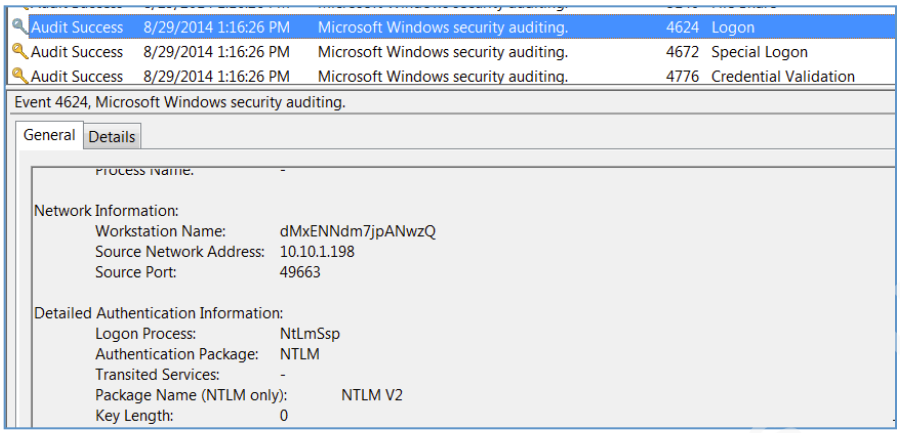

Figure 5: "Logon" event entry showing source IP address

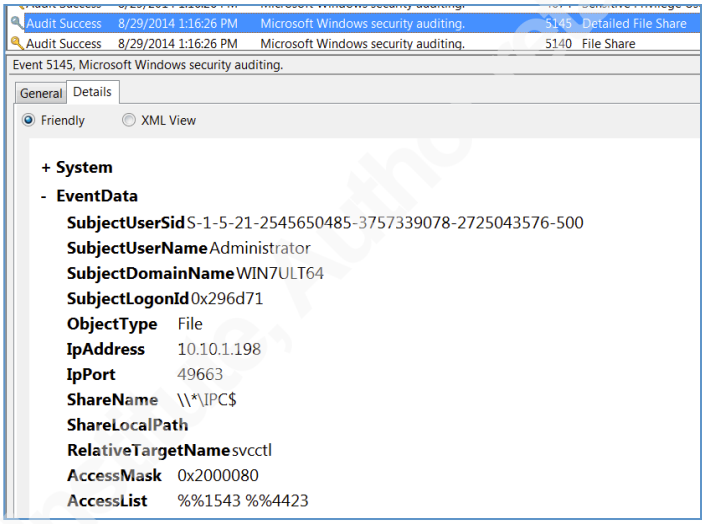

Figure 6: "File share" event showing target name of "svcctl"

In figures 4 through 9, we see that when a "psexec" method is used, the Windows event log on the remote system will show multiple different entries offering a lot of evidence of "pivoting" activity. This evidence includes events showing logon activity from a peer / neighboring workstation, the IPC$ file share access event, a service creation event, PowerShell process creation event, and logoff event.

Joff Thyer, jsthyer@gmail.com

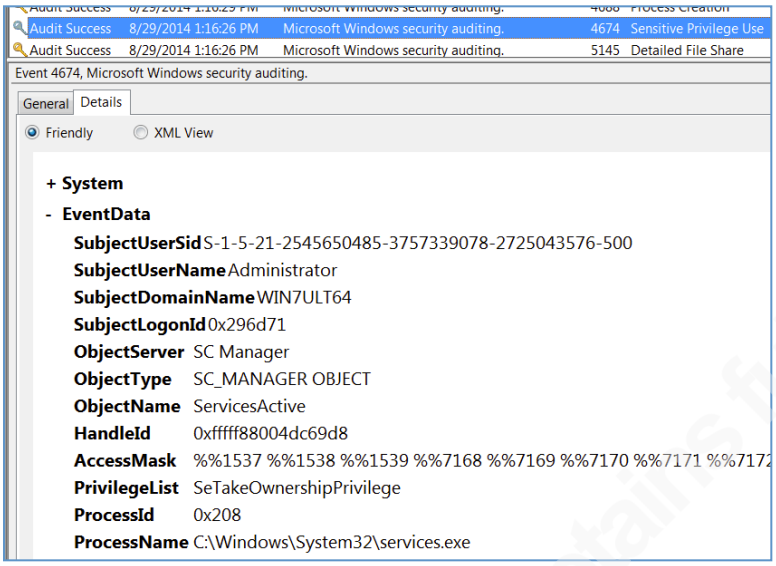

Figure 7: "Sensitive privilege use" event showing SC Manager access

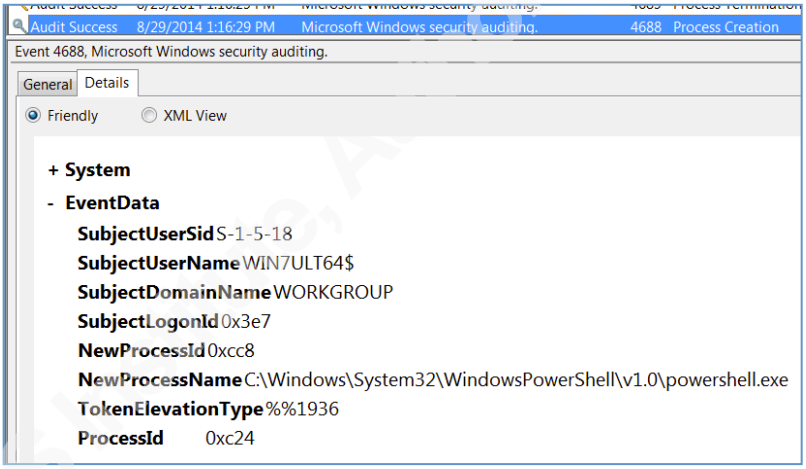

Figure 8: "Process creation" event showing PowerShell being executed

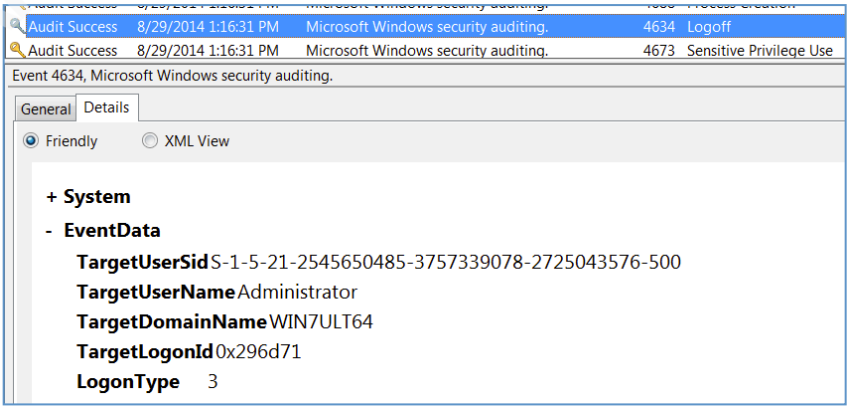

Figure 9: "Logoff" event showing Administrator username

Joff Thyer, jsthyer@gmail.com

```
Log Name:      Microsoft-Windows-Sysmon/Operational
Source:        Microsoft-Windows-Sysmon
Date:          8/29/2014 4:48:12 PM
Event ID:      1
Task Category: (1)
Level:         Information
Keywords:
User:          SYSTEM
Computer:      WIN7ULT64
Description:
Process Create:
UtcTime: 8/29/2014 8:48 PM
ProcessGuid: {00000000-e70c-5400-0000-0010e3f85100}
ProcessId: 4020
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
CommandLine: powershell.exe  -Command $si = New-Object System.Diagnostics.Proces
sStartInfo;$si.FileName = 'powershell.exe';$si.Arguments = ' -EncodedCommand JAB
zAHQAcgBlAGEAbQAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAASQBPAC4ATQBlAG0AbwByAHkAUwB
0AHIAZQBhAG0AKAAsACQAKABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBlADYANAB
TAHQAcgBpAG4AZwAoACcAZQBOAHEAbABWAG0AMQB2ADIAawBnAFEALwB0AHgASwAvAFEAOAA....
User: NT AUTHORITY\SYSTEM
LogonGuid: {00000000-a6dd-5400-0000-0020e7030000}
LogonId: 0x3e7
TerminalSessionId: 0
IntegrityLevel: System
HashType: SHA256
Hash: A8FDBA9DF15E41B6F5C69C79F66A26A9D48E174F9E7018A371600B866867DAB8
ParentProcessGuid: {00000000-e70c-5400-0000-00106bf85100}
ParentProcessId: 2012
ParentImage: C:\Windows\system32\cmd.exe
ParentCommandLine: C:\Windows\system32\cmd.exe /B /C start powershell.exe -Comma
nd $si = New-Object System.Diagnostics.ProcessStartInfo;$si.FileName = 'powershe
ll.exe';$si.Arguments = ' -EncodedCommand JABzAHQAcgBlAGEAbQAgAD0AIABOAGUAdwAtAE
8AYgBqAGUAYwB0ACAA.....
```

Figure 10: Sysinternals "Sysmon" server process creation event entry

The Microsoft SysInternals project named "Sysmon" takes process event logging a step further by showing more detailed process creation information which includes the full arguments of the process being created. Depicted in figure 10, we see the PowerShell process created with an encoded Base64 command. This data can be extracted and decoded to discover exactly what nature PowerShell script is being executed.

Based on the examination of this captured event information, there are numerous Windows event identifiers that stand out, and could be periodically examined for evidence of pivoting activity. These identifiers include:

- 4624: Logon success
- 4625: Logon failure
- 4776: Credential validation
- 7035: Service control manager (start service)

Joff Thyer, jsthyer@gmail.com

Using either Windows Management Instrumentation (WMI) or Windows Remote Management in PowerShell, in combination with LDAP information about a Windows domain, specific event identifiers of interest can be retrieved and examined. An example PowerShell script that can perform this task is as follows in figure 11:

```
$cred = Get-Credential
$events = Get-WinEvent -Credential $cred `
    -LogName "Security"
$events | Group-Object -Property Id -NoElement `
    | Sort-Object -property Count -descending
```

Figure 11: Example PowerShell script to retrieve security events

1.3. Identifying Evidence of Persistence

In most cases, Windows malware is going to use the typical mechanisms of installing a registry key to ensure that the software starts again upon system boot. Commonly the registry entries used are either in the local machine registry hive (HKEY Local Machine / HKLM), and/or in the user registry hive (HKEY Current User / HKCU). When looking for this sort of data, we should also account for the 64-bit variants of the registry keys. (Journey Into Incident Response: Tracking Down Persistence Mechanisms, 2013)

- \HKLM\Software\Microsoft\Windows\CurrentVersion\Run
- \HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce
- \HKCU\Software\Microsoft\Windows\CurrentVersion\Run
- \HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce

- \HKLM\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Run
- \HKLM\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\RunOnce
- \HKCU\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Run
- \HKCU\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\RunOnce

In much the same way as the path redirection feature for system DLL files (C:\Windows\SYSWOW64), when a 32-bit application requires a 32-bit version of a key it will query within the "Wow6432Node" key portion of the associated hive.

Using PowerShell within an environment, we can again query all systems on a domain wide basis to fetch all of the registry run keys and place them in a central

location. Our goal should be to sort the run keys and count frequencies across the domain. We are looking for low frequency count, atypical run keys belonging to machines we can perform deeper forensic examination upon. An example PowerShell script for fetching a run key is listed below. For brevity, this example script is written for only a single run key.

```
# Registry constants
$HKLM = 2147483650
$reg_run = "Software\Microsoft\Windows\CurrentVersion\Run"

# Get a Credential
$cred = Get-Credential

# gather Targets
$Targets = @()
$adsisearcher = [adsisearcher]'objectclass=computer'
$adsisearcher.FindAll() | % {$Targets += $($_.properties.name)}

foreach($Target in $Targets)
{
    Write-Output "[+] Registry key [$reg_run] from [$Target]"
    $registry = Get-WmiObject StdRegProv -Namespace Root/Default `
      -Credential $cred -ComputerName $Target -List
    $enum = $registry.EnumValues($HKLM, $reg_run)
    ForEach ($key in $enum.sNames) {
      $value = ($registry.GetStringValue($HKLM, $reg_run, $key)).sValue
      Write-Output "  [+] $reg_run : $key = $value"
    }
  }
}
```

Figure 12: PowerShell example to obtain registry "run" key

1.4. Identifying Deviations from Software Baseline

Using a method that is very similar to the registry key information presented above, we can iterate across our domain and examine the Win32_Product WMI class to get a sense of what the installed base of software across an environment looks like. Assuming a well-managed environment with appropriate security policies, and no locally administrative users, it is safe to assume a degree of uniformity for all operating endpoints. Where our endpoints deviate from this uniformity indicates an area that should be flagged for further examination.

In consideration of this, we should seek to obtain frequency counts of unique software installation information and maintain the source workstation information from where each item of information was obtained. As an additional measure, we should consider examining the "C:\Program Files", and "C:\Program Files (X86)" directory listings

across the environment. Below is a sample script to fetch the appropriate information on a domain wide basis. In this example, we are using the Win32_Product WMI class rather than the registry key, however a registry key enumeration method could also be employed.

```
# Get a Credential
$cred = Get-Credential

# gather Targets
$Targets = @()
$adsisearcher = [adsisearcher]'objectclass=computer'
$adsisearcher.FindAll() | % {$Targets += $($_.properties.name)}

foreach($Target in $Targets)
{
    Write-Output "[+] Getting list of software from [$Target]"
    $software = Get-WmiObject Win32_Product -Credential $cred -ComputerName $Target
    $software | format-list -property Name, Version
  }
}
```

Figure 13: PowerShell example to obtain installed software list

Once this information has been retrieved, our goal should be to answer some questions for ourselves, such as:

- Are there unusual or unexpected software packages installed?
- Are the software publisher names consistent?
- Are there unexpected software versions installed?
- Are there unexpected install locations, or unexpected vendor strings present?

1.5. Identify Unusual Code Signing Certificates

There are a number of past examples that exist whereby malware has accidentally been code signed, fraudulent code signing companies created, or there has been theft of code signing private keys, and certificates. (HP accidentally signed malware, will revoke certificate, 2014)

Through PowerShell, we can leverage access to the entire certificate store on individual domain joined endpoints. This enables us to examine machine, user / personal certificates which includes the root certificate store, intermediate certificate stores, and code signing information.

Using a combination of scripting to retrieve the certificate information, and a statistical post processing approach, we should create totals on the number of code signing certificates. Sub-totals should be created on publisher information, publisher URL information, and timestamps (certificate lifetime). Questions to be answered include the following:

- Are there endpoints with statistical variance in the overall total and type of code signing certificates?
- Is the lifetime signing flag set? Is the timestamp valid?
- Can the certificate be verified / validated?
- What is the certificate revocation status?
    - A combination of certificate revocation list checking, and online certificate status protocol (OCSP) should be employed to answer this question.

From the PowerShell perspective, retrieving the certificate store is a little more challenging. The reason is that we can use administrative access to pass our credentials to the remote workstation to enumerate the store, but we require some form of credential to write the results back to another central location. We could take the approach of offering up an open file share to write the results back to, or write results to individual stations and then gather the data in a second pass back to the central location.

In the prototyping scripts, we choose to take the latter approach to ease implementation complexity. It is also useful to leverage the PowerShell encoded command feature to pass the script text across WMI for remote execution. Additionally, our WMI call listed below sets the token impersonation level to allow the remotely invoked objects to use the source credentials.

```
$command = ("
    try { mkdir '//localhost/admin$/Temp/certstorage' } catch {}
    Get-ChildItem -Path cert: -Recurse -CodeSigningCert |
      Where-Object { `$_.GetType().Name -eq 'X509Certificate2' } | %{
            [System.IO.file]::WriteAllBytes(
            '//localhost/admin$/temp/certstorage/' + `$_.Thumbprint + '.cer',
            (`$_.Export('CERT', 'secret')))}
")
    $bytes = [System.Text.Encoding]::Unicode.GetBytes($command)
    $encoded = [Convert]::ToBase64String($bytes)

$wmic = Invoke-WmiMethod -Credential $cred `
      -ComputerName $Target `
      -Class Win32_Process `
      -Impersonate 4 `
      -Name Create `
      -ArgumentList "powershell -Exec bypass -EncodedCommand $encoded"

```

Figure 14: Example PowerShell script to retrieve remote certificate store

Joff Thyer, jsthyer@gmail.com

2. Conclusion

It is important to understand that all of the methods presented above are focused on collecting meta-data about an environment to primarily search for anomalies. The approach to searching for macro-level indicators of compromise is not focused directly on finding malware, but rather on looking at the high level behavior of an environment for anomalous behavior. Using statistical analysis for each sub-component of analysis, a security analyst can then correlate results and use the emerging correlations about individual endpoints to focus down to individual endpoints.

The environmental data used to search for macro-level indicators of compromise can not only be systemic logging, and event information but can also be extended to a deeper level by examining sources of full packet capture on critical network paths. Examining packet capture data can improve our results by giving a real-time heuristic view of anomalies, which can be correlated with the non real-time logging data sources.

Ultimately the methods presented offer a complement to traditional penetration testing resulting not only in clear demonstration of vulnerability, but accompanying detailed evidence of compromise.

Joff Thyer, jsthyer@gmail.com

References

Gartner: Top 10 Strategic IT Trends For 2015 - Forbes. (October 2014). Retrieved from http://www.forbes.com/sites/peterhigh/2014/10/07/gartner-top-10-strategic-it-trends-for-2015/

Schneier on Security: More on Feudal Security. (June 2013). Retrieved from https://www.schneier.com/blog/archives/2013/06/more_on_feudal.html

Desktop as a Service vs. VDI - Pros and Cons - Tom's IT Pro. (August 2014). Retrieved from http://www.tomsitpro.com/articles/vdi-vs-desktop-as-a-service,2-791.html

Antivirus pioneer Symantec declares AV “dead” and “doomed to failure” | Ars Technica. (May 2014). Retrieved from http://arstechnica.com/security/2014/05/antivurus-pioneer-symantec-declares-av-dead-and-doomed-to-failure/

Securosis Research |. (September 2013). Retrieved from https://securosis.com/research/publication/continuous-security-monitoring

The human OS: Overdue for a social engineering patch | CSO Online. (October 2014). Retrieved from http://www.csoonline.com/article/2824563/social-engineering/the-human-os-overdue-for-a-social-engineering-patch.html

Joff Thyer, jsthyer@gmail.com

pwnag3: What Did Microsoft Just Break with KB2871997 and KB2928120. (May 2014). Retrieved from http://www.pwnag3.com/2014/05/what-did-microsoft-just-break-with.html

Detecting DNS Tunneling – SANS Institute. (February 2013). Retrieved from http://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152

DNS Tunnelling - InfoSec Institute. (March 2014). Retrieved from http://resources.infosecinstitute.com/dns-tunnelling/

Identifying Malware via User-Agent Headers – Intrepidus Group - Insight. (June 2010). Retrieved from https://intrepidusgroup.com/insight/2010/06/identifying-malware-via-user-agent-headers/

Journey Into Incident Response: Tracking Down Persistence Mechanisms. (March 2013). Retrieved from http://journeyintoir.blogspot.com/2013/03/tracking-down-persistence-mechanisms.html

HP accidentally signed malware, will revoke certificate | Ars Technica. (October 2014). Retrieved from http://arstechnica.com/security/2014/10/hp-accidentally-signed-malware-will-revoke-certificate/

Joff Thyer, jsthyer@gmail.com