Threat Actor Targets Manufacturing Industry With Malware

Published: 2024-12-05 · Archived: 2026-04-05 13:13:46 UTC

Threat Actor Targets the Manufacturing industry with Lumma Stealer and Amadey Bot

Threat Actor Targets the Manufacturing industry with Lumma Stealer and Amadey Bot

Cyble analyzes a malicious campaign targeting the manufacturing industry, using process injections to deliver Lumma Stealer and Amadey bot.

Key takeaways

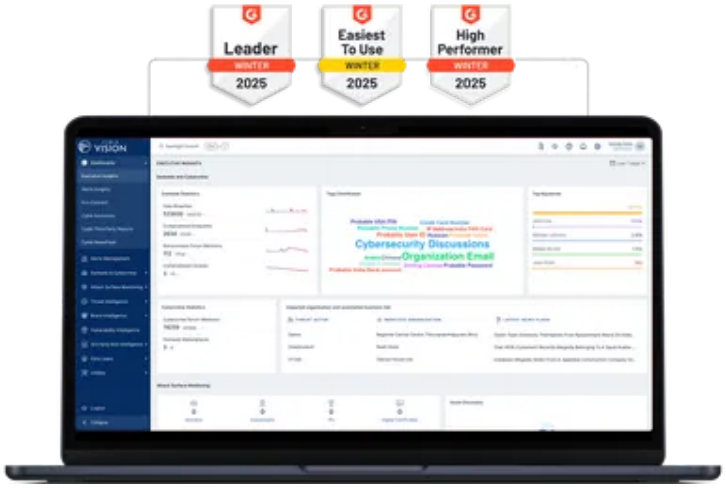
- Cyble Research and Intelligence Labs (CRIL) identified a malicious campaign targeting the manufacturing industry, leveraging a deceptive LNK file disguised as a PDF file.
- This campaign leverages multiple Living-off-the-Land Binaries (LOLBins), such as ssh.exe, powershell.exe, and mshta.exe, to bypass traditional security mechanisms and remotely execute the next-stage payload.
- The Threat Actor (TA) used Google Accelerated Mobile Pages (AMP) URL along with a shortened URL to evade detection by traditional URL scanners.
- The attack heavily relies on file injection techniques, where the TAs execute malicious payloads directly in memory to bypass conventional security mechanisms.
- The attack chain leverages DLL sideloading and IDATLoader to deploy the Lumma stealer and Amadey bot, enabling the attacker to gain control and exfiltrate sensitive information from the victim's machine.

Overview

CRIL recently identified a multi-stage cyberattack campaign originating from an LNK file. The initial infection vector remains unknown; however, the attack likely begins with a spear-phishing email, prompting the recipient to click on a link that leads to an LNK shortcut file disguised as a PDF document. The file is hosted on a remote WebDAV share at

"*hxxp://download-695-18112-001-webdav-logicaldoc[.]cdn-serveri4732-ns.shop/Downloads/18112.2022/Instruction_695-18121-002_Rev.PDF.lnk*".

Upon searching for the file name "*695-18121-002_Rev*" on Google, we discovered a technical engineering drawing for a component. Additionally, we observed similar samples using the name "*Instruction_18112*," which led us to another technical document detailing the installation of a chair. The malicious LNK file hosted on the URL impersonates *LogicalDOC*, a cloud-based document management system commonly used in Manufacturing and Engineering firms. Based on the targeting and nature of these attacks, we suspect that the campaign is likely targeting the manufacturing industry.



Once executed, the LNK file triggers a command to launch *ssh.exe*, which subsequently runs a PowerShell command. This PowerShell command fetches and executes an additional malicious payload from a remote server using *mshta.exe*.

The remote server is accessed via a URL that abuses Google's Accelerated Mobile Pages (AMP) framework, combined with a shortened URL that redirects to a location hosting malicious PowerShell code.

The PowerShell code then triggers another malicious script hosted on Pastebin, controlled by the TA. This script contains an encoded PowerShell command that downloads a ZIP archive to the Temp directory, extracts its contents, and executes a legitimate executable. The executable, in turn, sideloads a malicious DLL file.

In this sophisticated campaign, the TA uses multiple stages of code injection to deploy the Lumma stealer, which then downloads the Amadey Bot onto the victim's system. The figure below shows the infection chain.

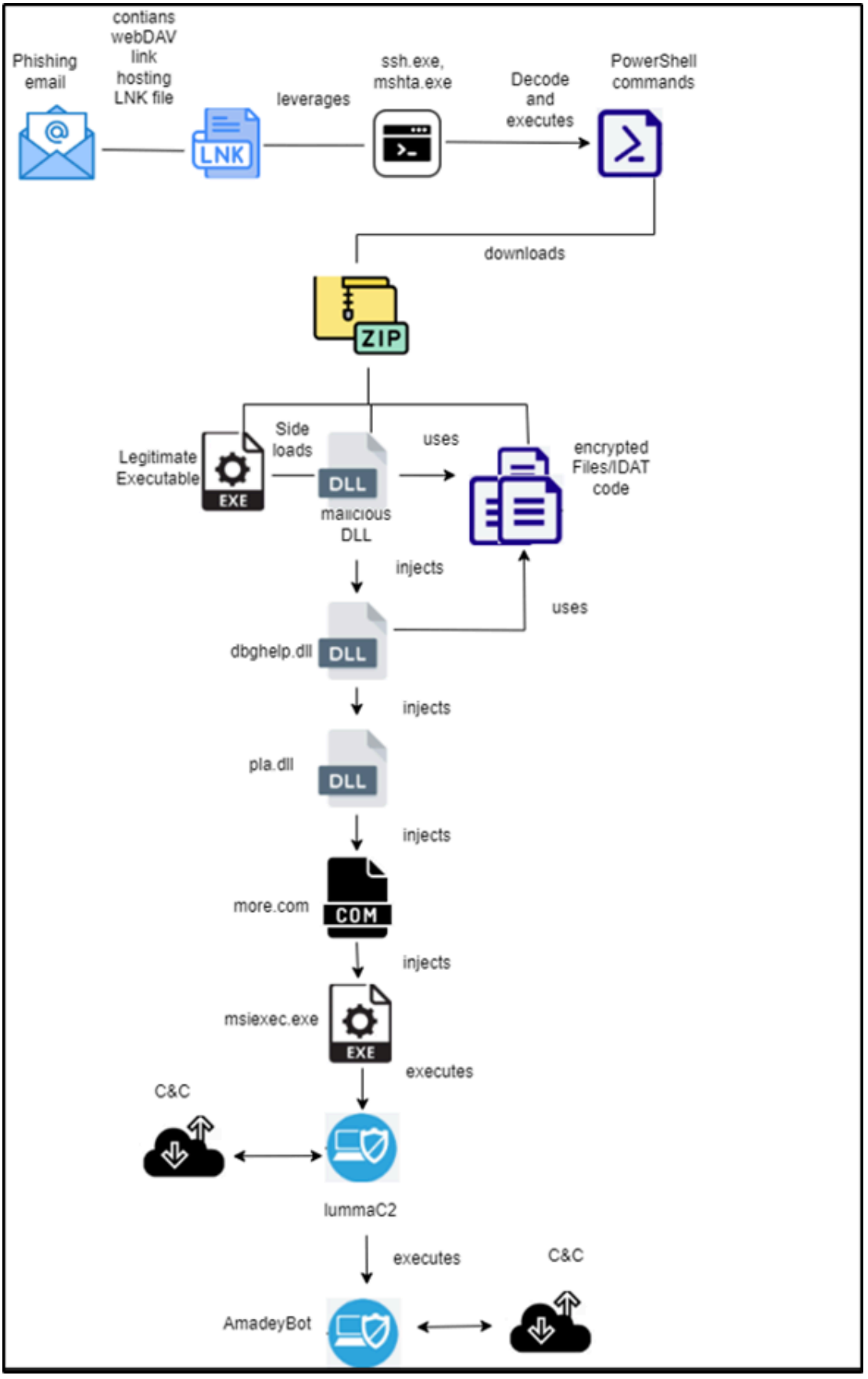


Figure 1 – Infection chain

Technical Analysis

Threat Actors are increasingly exploiting LNK files as their initial vector for malware distribution due to their flexibility in executing various commands. In this campaign, they specifically leveraged the Windows SSH client (*C:\Windows\System32\OpenSSH\ssh.exe*) as an alternative target in the LNK file's "Target" field. This approach reduces the likelihood of detection compared to using *cmd.exe* or *powershell.exe* as the target. The image below shows the LNK command.

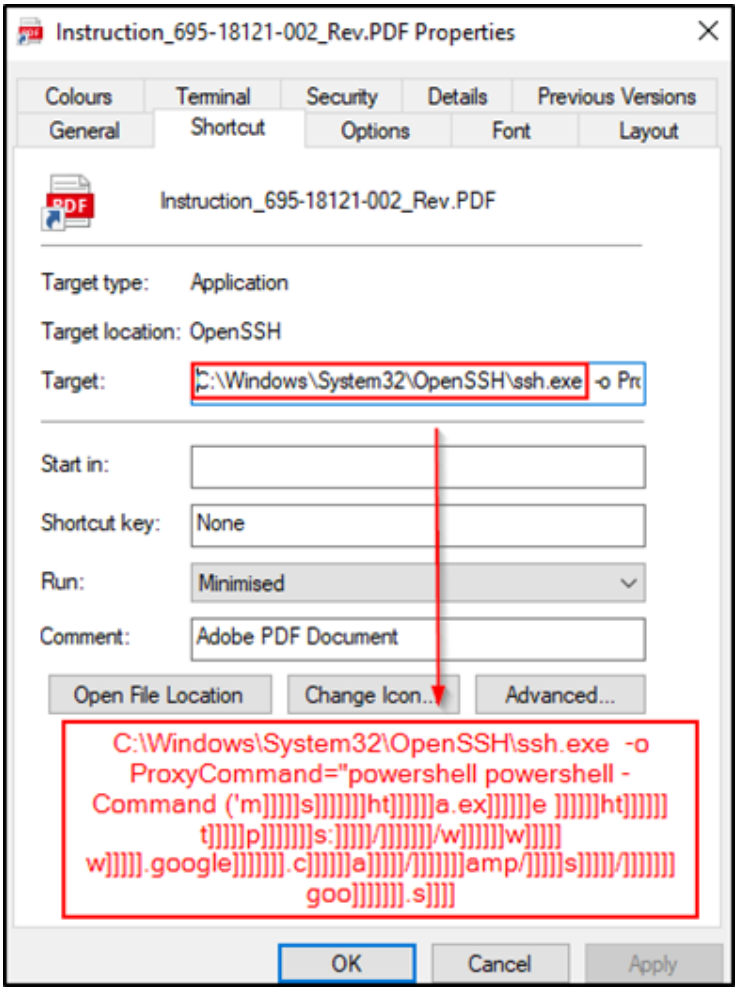


Figure 2 – LNK using SSH as a target

When a user opens the disguised LNK file, it triggers "*ssh.exe*" to run a PowerShell command through the ProxyCommand option in *ssh.exe*. The embedded PowerShell command contains obfuscated content, as shown in the image above. The de-obfuscated code attempts to execute PowerShell content hosted at the AMP URL "*hxxps://www.google[.]ca/amp/s/goo.su/IwPQJP*" using *mshta.exe*. In this case, the hosted content contains AES-encrypted data, as shown in the image below.

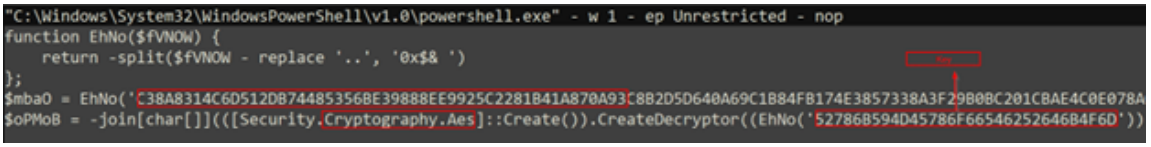


Figure 3 – AES-encrypted content hosted in AMP URL

Upon decryption, the data reveals Base64-encoded content, which is displayed in the image below.

Figure 4 – Base64-encoded content

The decoded Base64 content reveals an obfuscated PowerShell command, as shown in the image below.

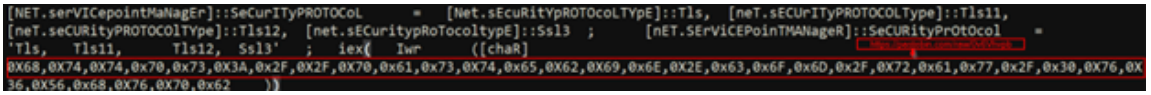


Figure 5 – Obfuscated PowerShell command

This PowerShell command manipulates security protocols and performs the following actions:

- First, it configures various security protocols, including TLS 1.0, TLS 1.1, TLS 1.2, and SSL 3.0, using the .NET *ServicePointManager* class.
- Then, it initiates a web request using Invoke-WebRequest (iwr) to fetch a payload from the URL *hxxps://Pastebin[.]com/raw/0v6Vhvpb*, which is then immediately executed using Invoke-Expression (iex).

The image below shows the retrieved payload from the Pastebin URL.

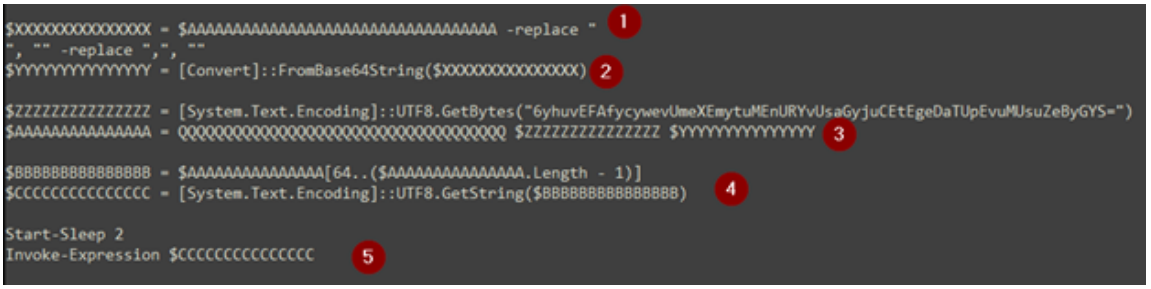


Figure 6 – Partial PowerShell script fetched from the Pastebin URL

The retrieved content from the Pastebin link consists of a PowerShell script that performs several actions:

1. The script begins by sanitizing the content fetched from Pastebin, removing newline characters (“n”) and commas (,).
2. The cleaned string is then decoded from Base64 into binary data.
3. Using a hardcoded decryption key, the script decrypts the binary data.
4. Once decrypted, the script extracts a portion of the data starting from the 64th byte to the end, which is the actual code to execute. This code is then converted into a readable PowerShell command using UTF-8 encoding.
5. Before executing the decoded command, a 2-second delay is introduced with Start-Sleep. Finally, the decoded PowerShell command is executed in memory using Invoke-Expression.

The image below shows the decrypted PowerShell code extracted using the above steps.

Figure 7 – Decrypted PowerShell code

The newly introduced script represents the final stage in delivering malicious files to the system. The script operates as follows:

1. The script first verifies the system's internet connectivity by sending HTTP requests to two distinct domains: *360.net* and *baidu.com*. These requests ensure the system is online before proceeding with further actions.
2. Once the victim's system is connected to the internet, the script downloads a malicious CPL file named *naailq0.cpl* from the remote URL *hxxps://berb.fitnessclub-filmfanatics.com/naailq0.cpl*.
3. The downloaded CPL file is saved as a ZIP file within the Temp directory. This ZIP file is then copied to a newly created folder under the *LocalAppData* folder. The folder name is dynamically generated using a GUID (Globally Unique Identifier).
4. After extraction, the script scans the folder for any executable files (EXEs). Any EXE files found within the extracted contents are then executed.
5. The script includes a commented-out line that, if activated, would delete the extracted files and folder after execution, potentially covering its tracks.

The image below shows the contents of the downloaded ZIP file. The ZIP file also contains encrypted files, which will be decrypted and loaded in the subsequent stages of infection.

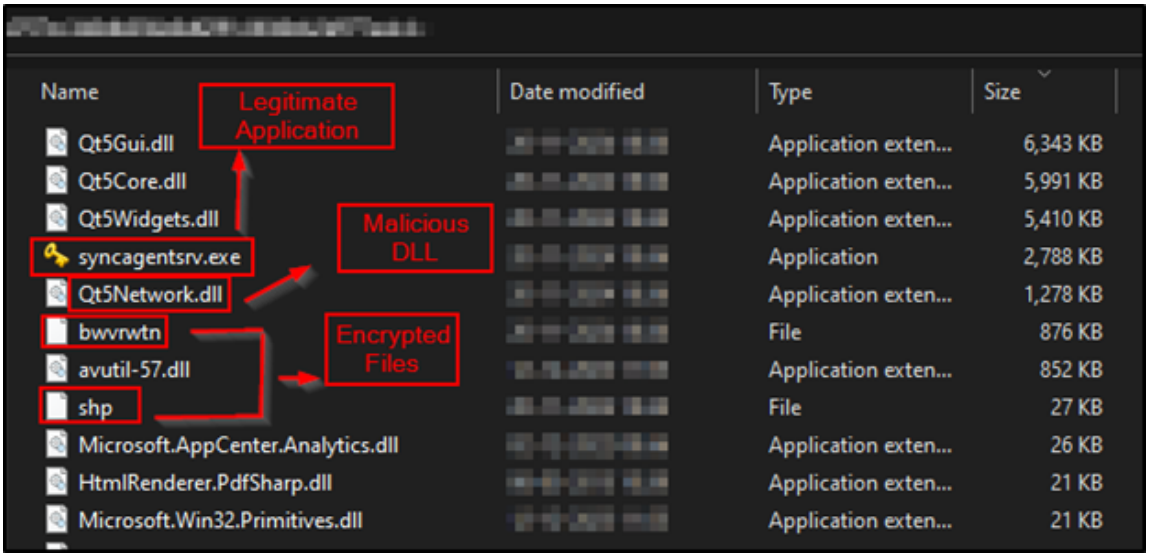


Figure 8 – Extracted files in the archive

In this case, the script executes "*syncagentsrv.exe*", which performs DLL sideloading by loading the malicious "*Qt5Network.dll*" upon execution. The malicious DLL then reads an encrypted file named "*shp*" from the same directory, decrypts its contents, and reveals strings such as *LoadLibraryA*, *VirtualProtect*, and *dbghelp.dll*, as shown in the figure below.

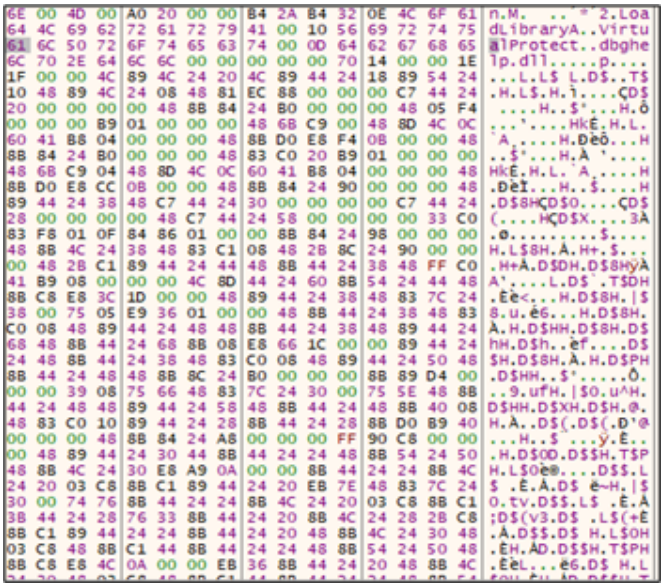


Figure 9 – Decrypted content

After decryption, the malicious DLL extracts the string "*dbghelp.dll*" from the decrypted content and utilizes it to load the DLL via the *LoadLibraryA* API. The "*dbghelp.dll*" is a Microsoft Windows library designed for debugging and managing symbol information. After loading the DLL, the malicious code employs the *VirtualProtect* API to modify the memory region permissions of "*dbghelp.dll*" to *PAGE_EXECUTE_READWRITE*, as illustrated below.

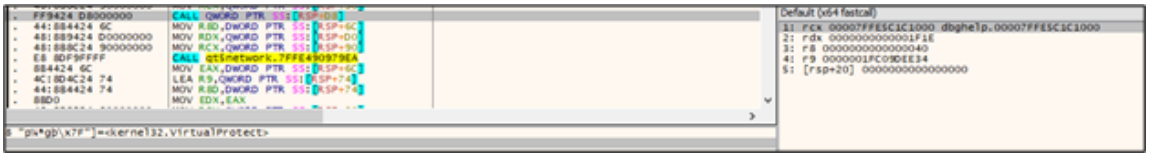


Figure 10 – Modifying permission of dbghelp.dll

It then overwrites the contents of "*dbghelp.dll*" with the decrypted data and subsequently modifies the memory protection of the overwritten region to *PAGE_EXECUTE_READ*, as depicted below.

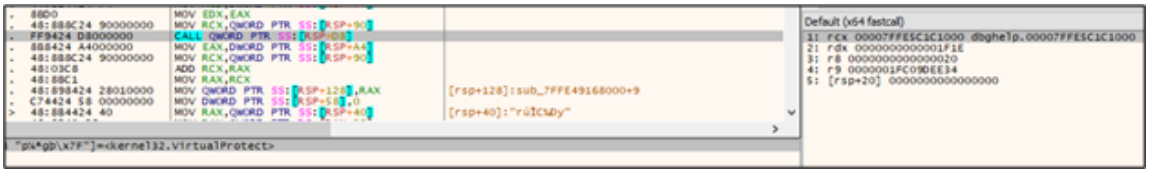


Figure 11 – Modifying the permissions of dbghelp.dll

After modifying the memory protection, the malicious code begins executing the injected content within "*dbghelp.dll*". The injected code then proceeds to read another file named "*bwvrwtn*", located in the same directory. The file "*bwvrwtn*" is an encrypted IDAT file containing multiple encrypted chunks, each prefixed with the string "IDAT," as illustrated below.

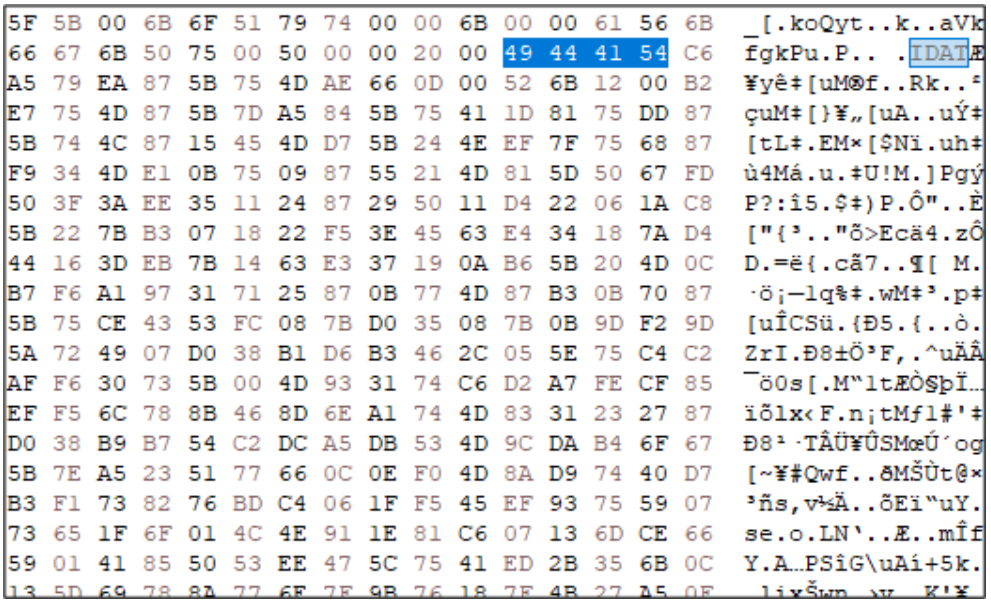


Figure 12 – IDAT marker

The DLL now searches the strings IDAT, takes four bytes following IDAT, and performs a comparison with C6 A5 79 EA. If the comparison is successful, the DLL proceeds to copy all the data following IDAT into memory, decrypts it using the XOR key, and then decompresses the decrypted content using the RTLDecompressBuffer API, as shown below.

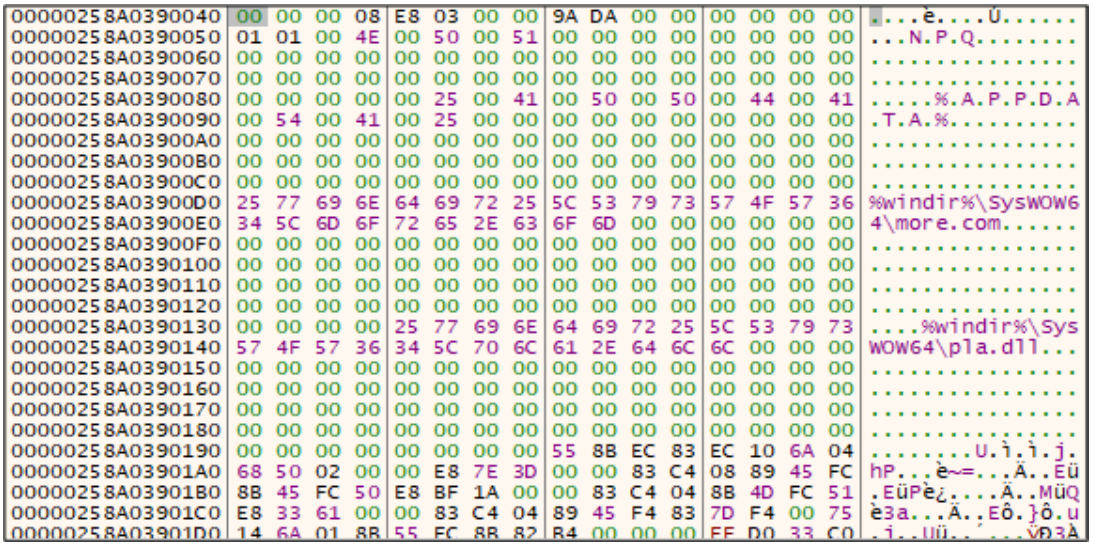


Figure 13 – Decompressed data

It then loads a legitimate "*pla.dll*" from the %syswow64% directory using the *LoadLibraryW* API. After loading, it changes the memory permissions of "*pla.dll*" to *PAGE_EXECUTE_READWRITE*, copies the decrypted content into its memory, changes the permissions to *PAGE_EXECUTE_READ*, and finally executes the injected code in the "*pla.dll*" as shown below.

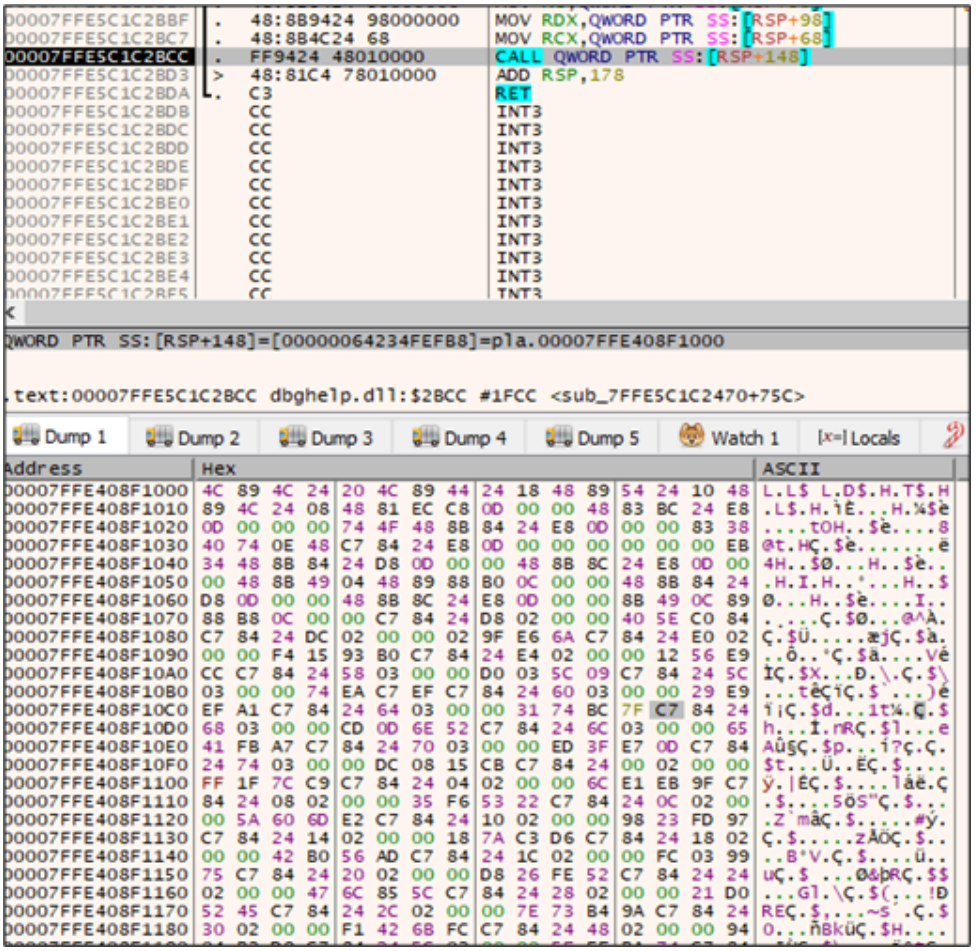


Figure 14 – Executing the injected code

The code within "*pla.dll*" proceeds to inject malicious code into "*more.com*" and then executes it. The malicious code in "*more.com*" is responsible for deploying the final payload by injecting it into a newly created process, "msiexec.exe." The injected payload is Lumma Stealer – which is capable of stealing sensitive information from the victim's machine. The figure below shows the memory string of "*msiexec.exe*" containing Lumma Stealer's C2 details.

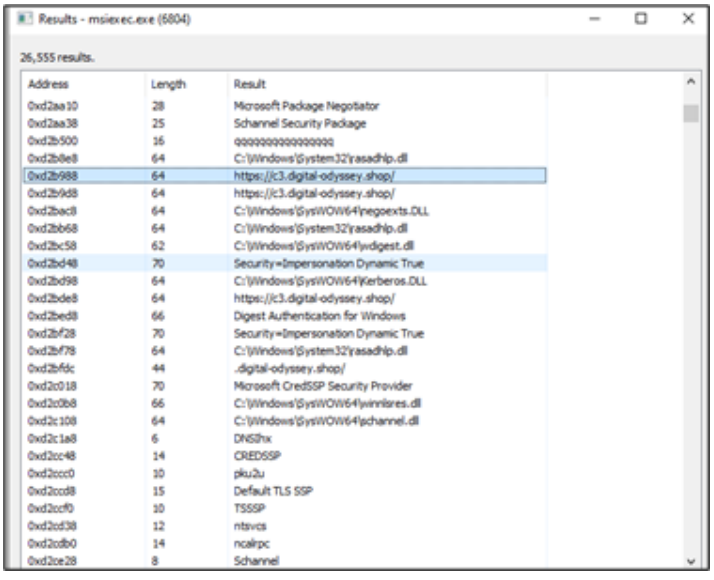


Figure 15 – Msiexec Process memory strings

Amadey Bot

The TA behind this campaign also deploys the Amadey bot in the "*%temp%*" directory, employing the same technique of injecting code into "*more.com.*" This injected code further injects the final Amadey bot payload into "*explorer.exe*". To achieve persistence, the malware creates a Task Scheduler entry named "*NodeJS Web Framework.*" This task is configured to execute a copy of the Amadey bot stored in the %Appdata% directory, as illustrated below.

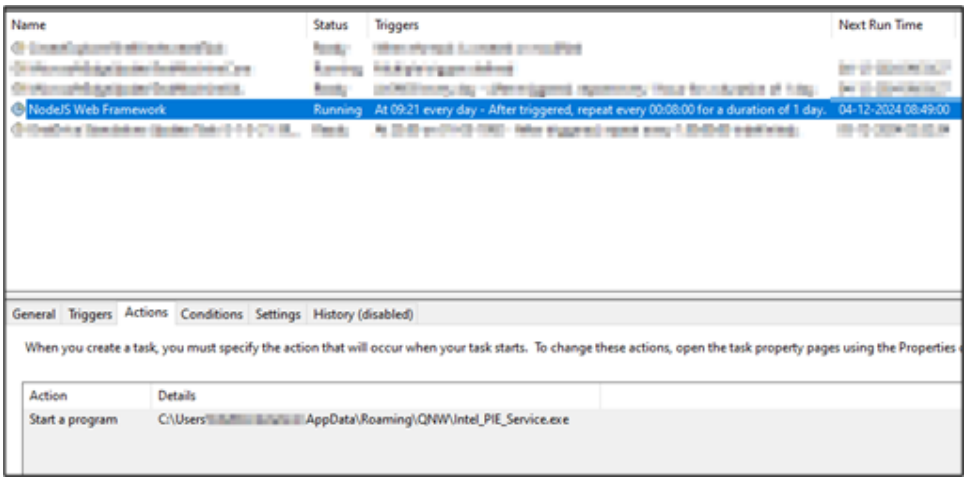


Figure 16 – Task Scheduler for Persistence

The figure below shows the execution flow of Lumma Stealer and Amadey bot.

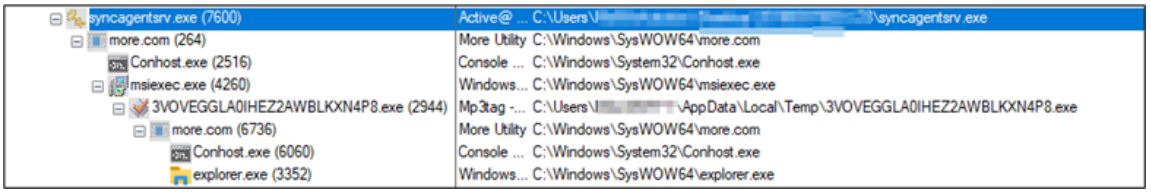


Figure 17 – Execution Flow

Conclusion

This multi-stage cyberattack campaign demonstrates the increasing sophistication and adaptability of threat actors. By leveraging various evasion techniques such as URL shortening and AMP URLs, the attackers successfully bypass traditional security mechanisms.

The use of legitimate system tools like ssh.exe and mshta.exe to execute malicious PowerShell commands further illustrates the complexity of the attack. The final payload, which involves the deployment of both Lumma stealer and Amadey bot, highlights the TA's intent to steal sensitive information and maintain persistent control over compromised systems.

Yara and Sigma rules to detect this campaign are available for download from the linked Github repository.

Recommendations

- The initial breach may occur via spam emails. Therefore, it's advisable to deploy strong email filtering systems to identify and prevent the dissemination of harmful attachments.
- Exercise caution when handling email attachments or links, particularly those from unknown senders. Verify the sender's identity, particularly if an email seems suspicious.

- Disable WebDAV if it is not required for business operations to minimize potential attack vectors.
- Consider disabling the execution of shortcut files (.lnk) originating from remote locations, such as WebDAV links, or implementing policies that require explicit user consent before executing such files.
- The campaign abused the legitimate ssh utility; hence, it is advised to monitor the activities conducted by the ssh utility and restrict access to limited users.
- Consider limiting the execution of scripting languages, such as PowerShell and mshta.exe, on user workstations and servers if they are not essential.
- Implement application whitelisting to ensure only approved and trusted applications and DLLs can be executed on the systems.
- Monitor AMP links using advanced URL filtering and threat intelligence feeds to detect suspicious activity.
- Set up network-level monitoring to detect unusual activities or data exfiltration by malware. Block suspicious activities to prevent potential breaches.

MITRE ATT&CK® Techniques

Tactic	Technique	Procedure
Initial Access (TA0001)	Phishing (T1566)	The LNK file may be delivered through phishing or spam emails
Execution (TA0002)	User Execution: Malicious Link (T1204.001) Command and Scripting Interpreter: PowerShell (T1059.001)	Execution begins when a user executes the LNK file. The LNK file executes PowerShell commands.
Defence Evasion (TA0005)	Masquerading: Masquerade File Type (T1036.008)	Uses LNK files with altered icons to disguise as legitimate
Defense Evasion (TA0005)	System Binary Proxy Execution: Mshta (T1218.005)	Abuse mshta.exe to proxy execution of malicious files.
Defense Evasion (TA0005)	Obfuscated Files or Information (T1027)	Scripts include packed or encrypted data.
Defense Evasion (TA0005)	System Binary Proxy Execution: Msiexec (T1218.007)	msiexec.exe used for proxy execution of malicious payloads
Privilege Escalation (TA0004)	DLL Side-Loading (T1574.002)	Malicious DLL Side loaded.
Privilege Escalation (TA0004)	Process Injection (T1055)	Injects malicious content into explorer.exe and other process.

Persistence (TA0002)	Scheduled Task/Job (T1053.005)	Adds task schedular entry for persistence.
C&C (TA0011)	Application Layer Protocol (T1071)	Malware communicates to the C&C server.
Exfiltration (TA0010)	Automated Exfiltration (T1020)	Data is exfiltrated after collection

Indicators Of Compromise

Indicators	**Indicator Type**	**Description**
5b6dc2ecb0f7f2e1ed759199822cb56f5b7bd993f3ef3dab0744c6746c952e36	SHA-256	Instruction_695-18121-002_Rev.PDF.lnk
8ed1af83cf70b363658165a339f45ae22d92c51841b06c568049d3636a04a2a8	SHA-256	Malicious PowerShell Script downloaded from Pastebin(0v6Vhvpb)
7b8958ed2fc491b8e43ffb239cdd757ec3d0db038a6d6291c0fd6eb2d977adc4	SHA-256	Zip file disguised as .cpl
dc36a3d95d9a476d773b961b15b188aa3aae0e0a875bca8857fca18c691ec250	SHA-256	Malicious DLL (Sideloaded)
hxxps://www.google[.]ca/amp/s/goo.su/IwPQJP hxxps://pastebin[.]com/raw/0v6Vhvpb hxxps://berb.fitnessclub-filmfanatics[.]com/naailq0.cpl	URL	remote servers
hxxp://download-695-18112-001-webdav-logicaldoc[.]cdn-serveri4732-ns.shop/Downloads/18112.2022/	URL	WebDAV server link hosting malicious LNK file

References

https://www.rapid7.com/blog/post/2023/08/31/fake-update-utilizes-new-idat-loader-to-execute-stealc-and-Lumma-infostealers

https://www.rapid7.com/blog/post/2024/03/28/stories-from-the-soc-part-1-idat-loader-to-bruteratel

---