Challenge: Maranhao Lab

Platform: CyberDefenders

Category: Endpoint Forensics

Difficulty: Easy

Tools Used: FTK Imager, EvtxECmd, Timeline Explorer, DB Browser for SQLite, MFTECmd, VirusTotal

Summary: This lab involves investigating a host compromised by the Maranhao Stealer, which historically targets gamers. The victim downloaded a seemingly legitimate game mod launcher that contained a ZIP archive with a trojanised installer. Execution of this installer silently deployed a dropper binary, which then staged a secondary payload called updater.exe. The secondary payload established persistence via a Run key in the registry. The malware performed extensive reconnaissance using WMI commands, terminated browser processes, collected credentials from Edge, and communicated with C2 servers.

Read here for a full report on the Maranhao Stealer explored in this lab.

Scenario: A gaming enthusiast in a known organization has downloaded what they believed to be a free mod launcher for a popular survival game. The file which downloaded contained a ZIP archive with an installer that looked like a standard game setup package.

Eager to try it, the gamer downloaded the file and executed the installer. Unbeknownst to him, the program silently dropped hidden files into a directory. One of these files was configured to persist through registry keys, ensuring it would relaunch every time the system started.

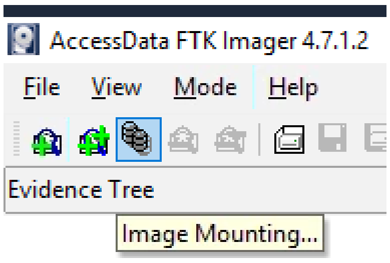
Within a short time, unusual activity triggered alerts on the Security Operations Center's (SOC) in GOAT Company's monitoring dashboard. The gamer's machine was observed making outbound requests to a malicious domain and a suspicious external IP address. Endpoint logs also showed evidence of process injection, suggesting credential theft. The Security Operations Center (SOC) quickly isolated the machine and saved a full disk image for your analysis.

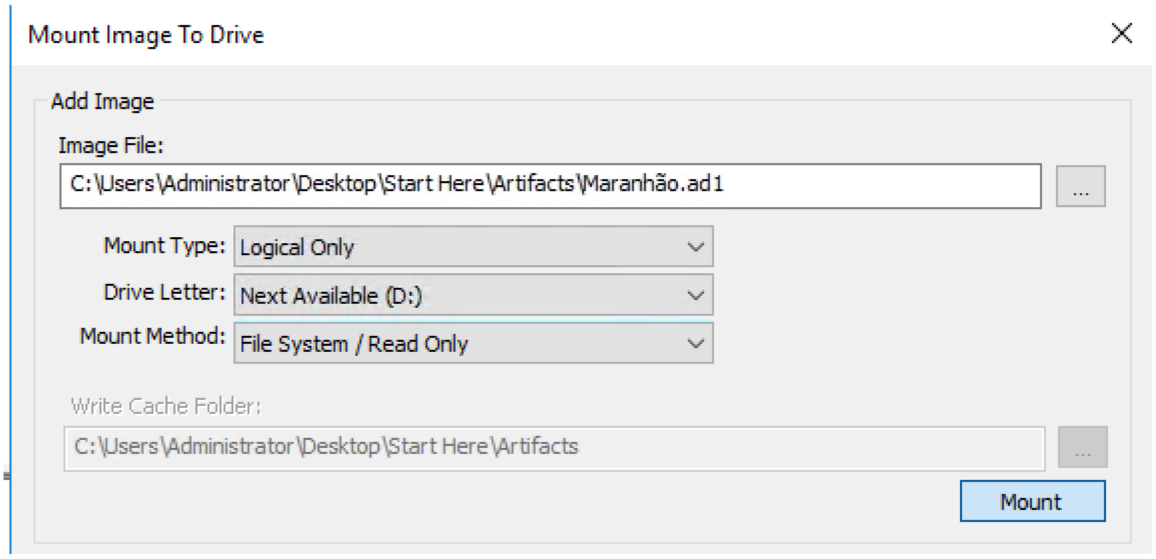
Initial Access

Analysts identified an external object that acted as the patient-zero delivery mechanism. Which remote resource URL initiated the chain of compromise by providing the archive disguised as a legitimate game utility?

TLDR: Export the user’s Edge browsing history file and use DB Browser for SQLite to view the downloads table.

Within this lab, we are provided a .ad1 image, therefore, I am going to start by mounting said image using FTK Imager:




Mount Image To Drive

Add Image

Image File: C:\Users\Administrator\Desktop\Start Here\Artifacts\Maranhão.ad1

Mount Type: Logical Only

Drive Letter: Next Available (D:)

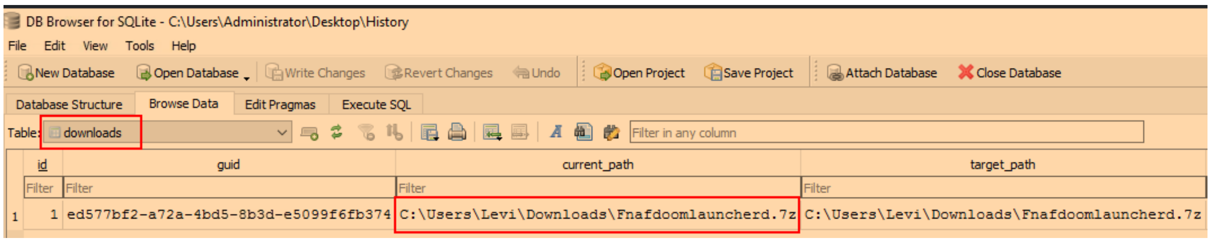
Mount Method: File System / Read Only

Write Cache Folder: C:\Users\Administrator\Desktop\Start Here\Artifacts

Now that we have mounted the image, we should start our investigation by looking at browsing history artifacts. After going through Levis' AppData/Local directory, I discovered that they used edge. Edge browsing artifacts are located at:

- `C:\Users\<username>\AppData\Local\Microsoft\Edge\User Data\Default`

In a file called "History". We can use a tool called DB Browser for SQLite to open this database file, and view the downloads table:

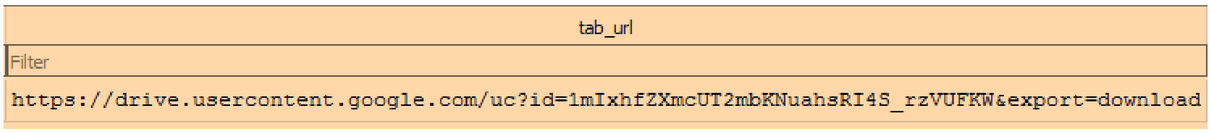

DB Browser for SQLite - C:\Users\Administrator\Desktop\History

File Edit View Tools Help
New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database
Database Structure Browse Data Edit Pragmas Execute SQL
Filter in any column

Table: downloads

	id	guid	current_path	target_path
	Filter	Filter	Filter	Filter
1	1	ed577bf2-a72a-4bd5-8b3d-e5099f6fb374	C:\Users\Levi\Downloads\Fnafdoomlauncherd.7z	C:\Users\Levi\Downloads\Fnafdoomlauncherd.7z

Within this table, we can see that the user downloaded a file called "Fnafdoomlauncherd.7z" from drive[.]usercontent[.]google[.]com:



tab_url
Filter
https://drive.usercontent.google.com/uc?id=1mIxhfZXmcUT2mbKNuahsRI4S_rzVUFKW&export=download

Answer:
https://drive.usercontent.google.com/uc?id=1mIxhfZXmcUT2mbKNuahsRI4S_rzVUFKW&export=download

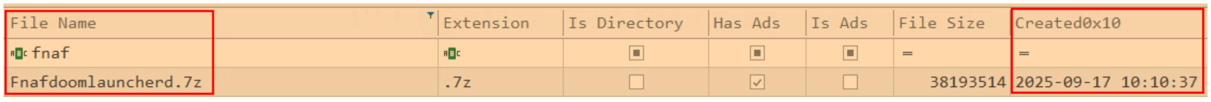
In reconstructing the timeline of compromise, which precise timestamp correlates to the adversary's delivery vector entering the victim environment as a ZIP file?

TLDR: Parse the MFT using MFTECmd, filter the File Name column for the archive identified previously. You can find the time it landed on disk by looking at the Created timestamp.

To find the precise time when this ZIP file landed on disk, we can use a tool called MFTECmd to parse the MFT file. The Master File Table ($MFT) is a database that tracks all object (file and folder) changes on an NTFS filesystem. Each object has its own record in the $MFT, containing metadata about that file or folder.

- `.\MFTECmd.exe -f "`$MFT" --csv . --csvf mft_out.csv`

To view the output produced by MFTECmd, we can use Timeline Explorer. If you filter for the filename in the "File Name" column, we can find when the archive was created:



File Name	Extension	Is Directory	Has Ads	Is Ads	File Size	Created0x10
fnaf		▣	▣	▣	=	=
Fnafdoomlauncherd.7z	.7z	☐	✓	☐	38193514	2025-09-17 10:10:37

Answer: 2025-09-17 10:10

The ZIP archive's decompression exposed a loader binary that masqueraded as a legitimate launcher. What was the executable responsible for initializing this staged intrusion?

TLDR: Filter for file creation (Event ID 11) events in the Sysmon logs. Focus on files created by an image commonly used to extracting archived files.

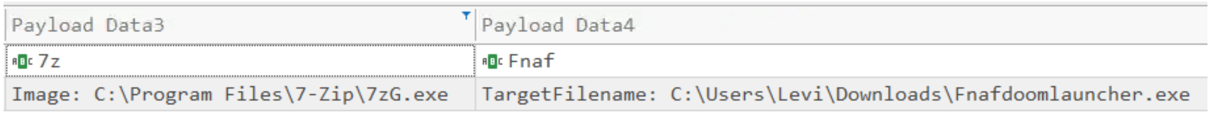
Fortunately for us, this host had Sysmon enabled. We can parse the Sysmon logs located at:

- `%SYSMROOT%\winevt\Logs`

Using EvtxECmd:

- `.\EvtxECmd.exe -f "Microsoft-Windows-Sysmon%4Operational.evtx" --csv . --csvf sysmon_out.csv`

If you filter for Event ID 11 (file creation), we can see that at 2025-09-17 10:12:10 7zG.exe (GUI executable for 7-zip) created a file called "Fnafdoomlauncher.exe". This suggests that the user extracted the archive using 7-zip and within it contained Fnafdoomlauncher.exe:



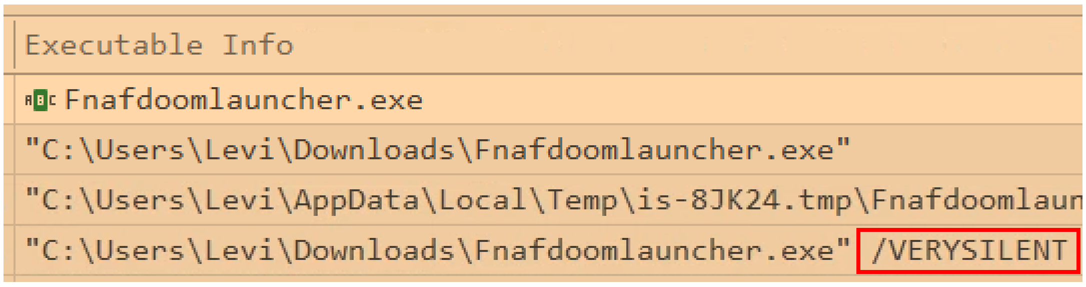
Payload Data3	Payload Data4
7z	Fnaf
Image: C:\Program Files\7-Zip\7zG.exe	TargetFilename: C:\Users\Levi\Downloads\Fnafdoomlauncher.exe

Answer: Fnafdoomlauncher.exe

Execution

Adversaries often alter installer behavior to remain invisible during deployment. Which installer flag was leveraged to suppress user-facing prompts during execution of the trojanized setup?

If you filter for process creation events (Event ID 1) associated with “Fnafdoomlauncher.exe”, we can see that it was executed with the flag “/VERYSILENT”:



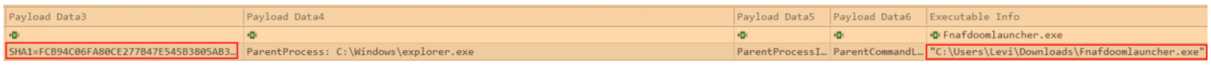
```
Executable Info
Fnafdoomlauncher.exe
"C:\Users\Levi\Downloads\Fnafdoomlauncher.exe"
"C:\Users\Levi\AppData\Local\Temp\is-8JK24.tmp\Fnafdoomlaun
"C:\Users\Levi\Downloads\Fnafdoomlauncher.exe" /VERYSILENT
```

The /VERYSILENT flag is a command-line argument that is used to perform silent, unattended installations. This makes it an effective technique for malware to operate and install additional components in the background.

Answer: /VERYSILENT

Forensic correlation across endpoints requires file-level fingerprinting. What SHA1 hash uniquely represents the dropper binary that initiated further payload deployment?

Sticking with process creation logs (Event ID 1), you can find the SHA1 hash for “Fnafdoomlauncher.exe” under the Payload Data 3 column:

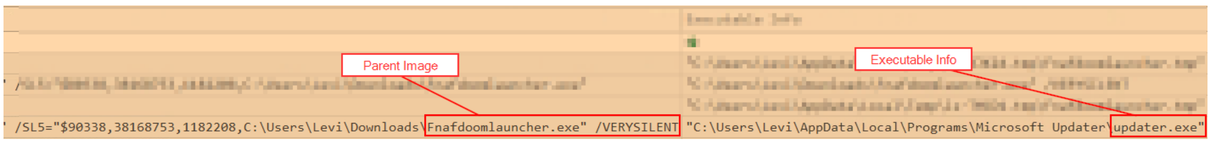


Payload Data3	Payload Data4	Payload Data5	Payload Data6	Executable Info
				Fnafdoomlauncher.exe
SHA1=FCB94C06FA80CE277B47E545B3805AB3…	ParentProcess: C:\Windows\explorer.exe	ParentProcessI…	ParentCommandL…	"C:\Users\Levi\Downloads\Fnafdoomlauncher.exe"

Answer: FCB94C06FA80CE277B47E545B3805AB38BB6ACF4

Post-installation, the secondary payload did not remain in temporary directories but was staged in a user-space program folder. Identify the exact directory path used for this execution pivot.

If you continue exploring process creation logs (Event ID 1), we can see that seconds after “Fnafdoomlauncher.exe” was executed, it was used to execute a binary called “updater.exe”:



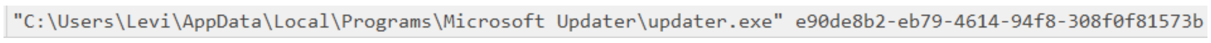
```
[redacted]
/SL5="$90338,38168753,1182208,C:\Users\Levi\Downloads\Fnafdoomlauncher.exe" /VERYSILENT "C:\Users\Levi\AppData\Local\Programs\Microsoft Updater\updater.exe"
```

Therefore, “Fnafdoomlauncher.exe” the first stage, deployed a secondary stage payload called “updater.exe”.

Answer: C:\Users\Levi\AppData\Local\Programs\Microsoft Updater\

During execution, the secondary component was invoked with a victim-tagging token for C2 identification. What globally unique string was provided as the argument?

The argument passed to updater.exe when it was executed is the token for C2 identification:



```
"C:\Users\Levi\AppData\Local\Programs\Microsoft Updater\updater.exe" e90de8b2-eb79-4614-94f8-308f0f81573b
```

Answer: e90de8b2-eb79-4614-94f8-308f0f81573b

Persistence

What was the complete file path of the binary embedded within the persistence mechanism to guarantee re-execution after reboot?

TLDR: Look for reg.exe in process creation logs.

If you filter the Parent CommandLine column for "updater.exe", we can see all process creation events associated with the secondary payload. Here we can see it executing reg.exe to create a run key called "updater":

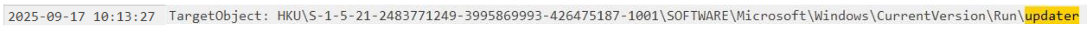
```
C:\Windows\system32\reg.exe ADD HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v
updater /t REG_SZ /d "\"C:\Users\Levi\AppData\Local\Programs\Microsoft
Updater\updater.exe\"" /f
```

A Run key is used to make a program run when a user logs on, it is a common persistence mechanism used by threat actors.

Answer: C:\Users\Levi\AppData\Local\Programs\Microsoft Updater\updater.exe

Temporal analysis of registry modifications showed the exact moment persistence was locked in. What is the date and time this key entry was created?

Filtering for Event ID 13 (Registry Value Set), we can see when the Run key was created:



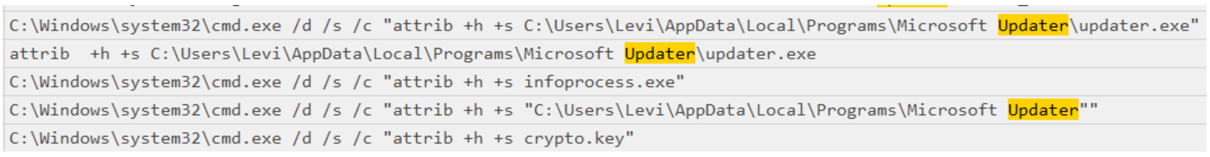
```
2025-09-17 10:13:27  TargetObject: HKU\S-1-5-21-2483771249-3995869993-426475187-1001\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\updater
```

Answer: 2025-09-17 10:13

Defence Evasion

Post-installation, the adversary concealed its artifacts at the file-system level. Which native Windows utility and attribute combination was used to render both files and directories hidden and system-protected?

Filtering for commands executed by updater.exe, we can see it used the attrib command to add the +h (hidden) and +s (system) attribute to multiple files and directories:



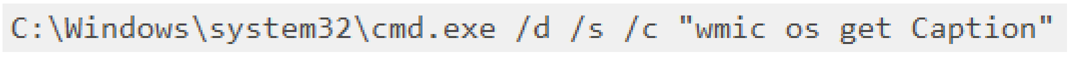
```
C:\Windows\system32\cmd.exe /d /s /c "attrib +h +s C:\Users\Levi\AppData\Local\Programs\Microsoft Updater\updater.exe"
attrib  +h +s C:\Users\Levi\AppData\Local\Programs\Microsoft Updater\updater.exe
C:\Windows\system32\cmd.exe /d /s /c "attrib +h +s infoprocess.exe"
C:\Windows\system32\cmd.exe /d /s /c "attrib +h +s "C:\Users\Levi\AppData\Local\Programs\Microsoft Updater""
C:\Windows\system32\cmd.exe /d /s /c "attrib +h +s crypto.key"
```

Answer: attrib +h +s

Discovery

Investigators observed the malware pulling system-level metadata that revealed the installed edition of Windows (e.g., "Microsoft Windows 10 Pro"). This information could later be used by the attacker to determine compatibility with payload execution. Which exact query facilitated this operating system enumeration?

Updater.exe was later observed using WMIC to perform discovery on the host. Windows Management Instrumentation Command-Line (WMIC) is a command-line tool used to interact with WMI to query system information:

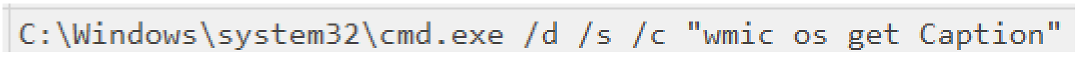


```
C:\Windows\system32\cmd.exe /d /s /c "wmic os get Caption"
```

This command is used to retrieve operating system information, specifically, the name of the Windows OS.

Answer: wmic os get Caption

To assess whether the compromised system had sufficient processing resources or was running in a sandbox with emulated hardware, the malware issued a command to extract the processor's vendor and model string. What specific query enabled this reconnaissance?

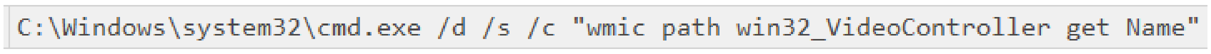


```
C:\Windows\system32\cmd.exe /d /s /c "wmic os get Caption"
```

This command retrieves the name/model of the CPU.

Answer: wmic cpu get Name

As part of its environment fingerprinting, the malware attempted to identify graphics hardware to help distinguish between a physical workstation and a low-resource virtual machine. Which query would return the video controller model?

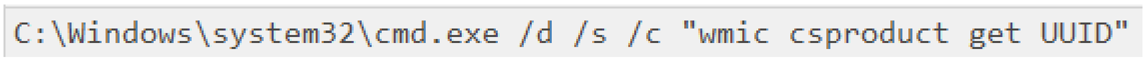


```
C:\Windows\system32\cmd.exe /d /s /c "wmic path win32_VideoController get Name"
```

This command retrieves the name/model of the graphics card.

Answer: wmic path win32_VideoController get Name

The malware generated a unique victim identifier that would remain stable across reboots and reinstalls by retrieving a machine's hardware UUID. Which WMI command was responsible for collecting this globally unique identifier?

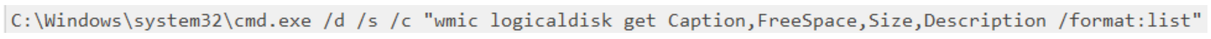


```
C:\Windows\system32\cmd.exe /d /s /c "wmic csproduct get UUID"
```

This command retrieves the unique hardware identifier assigned to the computer's motherboard.

Answer: wmic csproduct get UUID

During host triage, analysts identified a query that enumerated logical drives along with their free space and size. This could help an attacker determine whether the host was worth further exploitation (e.g., data exfiltration feasibility). Which WMI command produced this disk inventory?

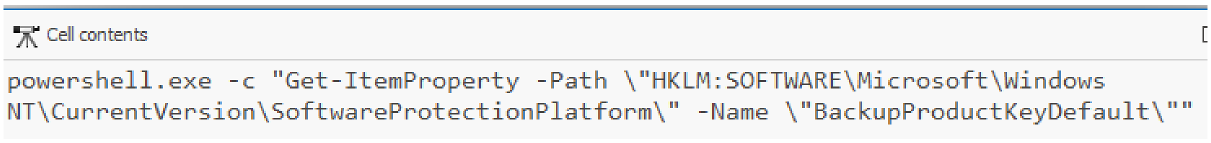


```
C:\Windows\system32\cmd.exe /d /s /c "wmic logicaldisk get Caption,FreeSpace,Size,Description /format:list"
```

This command queries information about all logical disks (drives) on the system.

Answer: wmic logicaldisk get Caption,FreeSpace,Size,Description /format:list

Unlike transient licensing tokens stored in tokens.dat, the malware pursued a static registry artifact used as a backup for Windows activation. Identify the precise registry entry (hive, key path, and value) that serves as a fallback product key reference.



```
powershell.exe -c "Get-ItemProperty -Path \"HKLM:SOFTWARE\Microsoft\Windows
NT\CurrentVersion\SoftwareProtectionPlatform\" -Name \"BackupProductKeyDefault\""
```

Answer: HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SoftwareProtectionPlatform\BackupProductKeyDefault

Credential Access

Attackers often terminate browsers before attempting to steal session data, cookies, or inject a malicious browser extension. What is the command that was used to forcibly terminate all browser processes?

Continuing to explore commands executed by updater.exe, we can see the taskkill command be used to terminate the msedge.exe process:

```
C:\Windows\system32\cmd.exe /d /s /c "taskkill /F /IM msedge.exe"
```

Answer: taskkill /F /IM msedge.exe

Collection

After injection, the malware established an interprocess channel for credential theft. What named pipe was created to ferry stolen browser data?

Event ID 17 (Pipe created) logs each time a named pipe is created. Named pipes are interprocess communication (IPC) method in Windows similar to sockets/TCP. Here we can find a suspicious named pipe:

```
PipeName: \ChromeDecryptIPC_e7e223c5-50d5-40ae-8513-64c9962789c2
```

After doing some research, you can determine that stolen data is transmitted back to the calling process over this named pipe.

Answer: ChromeDecryptIPC_e7e223c5-50d5-40ae-8513-64c9962789c2

Command and Control

To enrich host discovery with geolocation data, the malware beaconed to an external resolver. Which service endpoint did it query?

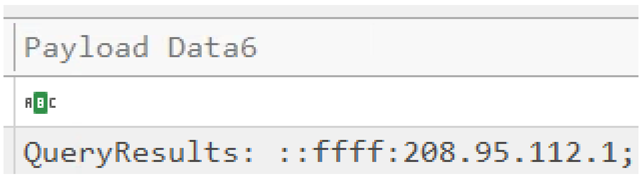
Filtering for Event ID 22 (DNS Event), we can see that updater.exe queries a domain called "ip-api.com":

```
Payload Data4: QueryName: ip-api.com  (Count: 1)
```

Answer: ip-api.com

Blocking by domain is insufficient; analysts confirmed the resolved address of the geolocation API. Which single IP must be blacklisted?

If you look at the query results for “ip-api.com”, we can find the returned IP address:



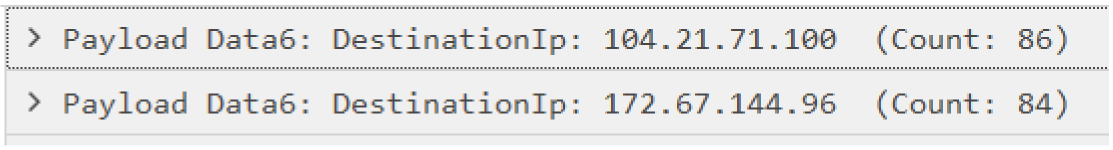
```
Payload Data6

QueryResults: ::ffff:208.95.112.1;
```

Answer: 208.95.112.1

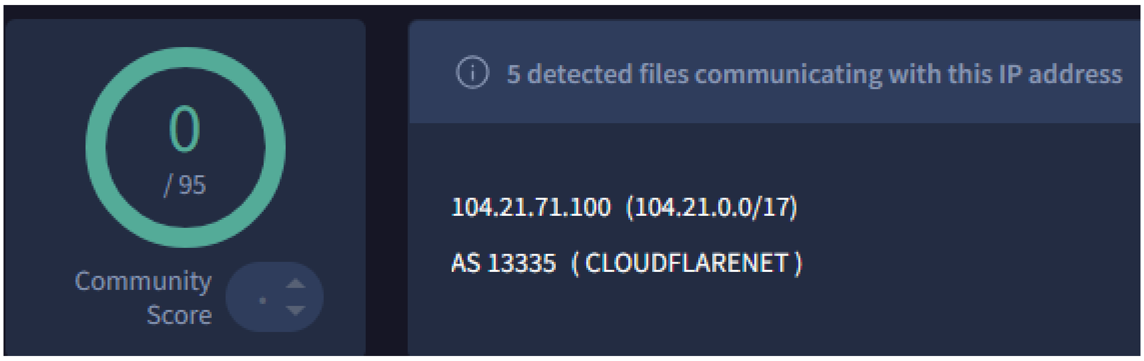
During network traffic analysis, the malware's outbound request did not resolve to a direct host but instead terminated at Cloudflare's edge network, a common tactic to conceal attacker infrastructure. Which two IP addresses were returned as part of this resolution?

If you investigate network creation events (Event ID 3) for updater.exe, we can see it communicated with 3 unique IP addresses, including the ip-api.com domain identified previously:



```
> Payload Data6: DestinationIp: 104.21.71.100  (Count: 86)
> Payload Data6: DestinationIp: 172.67.144.96  (Count: 84)
```

If you use something like VirusTotal, we can lookup these IP addresses and see that they are owned by Cloudflare:



5 detected files communicating with this IP address

104.21.71.100 (104.21.0.0/17)

AS 13335 (CLOUDFLARENET)

Answer: 104.21.71.100, 172.67.144.96