Challenge: RepoReaper Lab

Platform: CyberDefenders

Category: Endpoint Forensics

Difficulty: Hard

Tools Used: FTK Imager, DB Browser for SQLite, DCode, Event Log Explorer, MFTECmd, Timeline Explorer, VirusTotal, Registry Explorer, PECmd

Summary: Initial access was achieved on July 11th, 2025, at 15:15 (UTC), after user DemonSlayer installed a zip archive called "Script2Scene-main.zip" from a GitHub repository. During the compilation process, a PowerShell script called "Clcw.ps1.ps1" was dropped on the host. Following this, another payload was deployed via a password protected zip archive. Within this archive was a malicious core Electron application. The malware achieved persistence via creating three scheduled tasks, and elevated privileges through abusing a UAC bypass technique using ComputerDefaults.exe. The malware performed registry modifications targeting the SystemRestore key to hinder recovery efforts and was also observed performing anti-analysis checks by testing whether Malwarebytes was installed on the host. Malware masquerading as NVIDIA Control Panel was also discovered on the host, communicating with an external IP over port 80 and 443. Furthermore, a malicious payload was injected into a legitimate system process called regasm.exe, which performed several thousand connections to two external IPs,

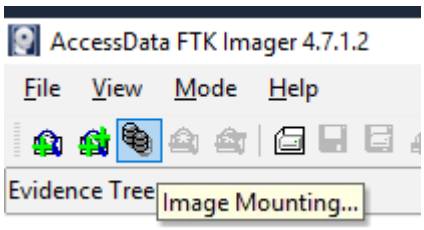
Scenario: In July 2025, an employee downloaded an open-source tool from GitHub to their workstation for internal use. Shortly after compiling the project, GOAT's SOC detected suspicious PowerShell and VBScript executions, system reconnaissance, and outbound connections to paste sites. Malware persistence was established via scheduled tasks, with browser data and credentials exfiltrated. Your task is to investigate the true infection chain, identify Threat Actor's techniques, and determine the scope of the compromise.

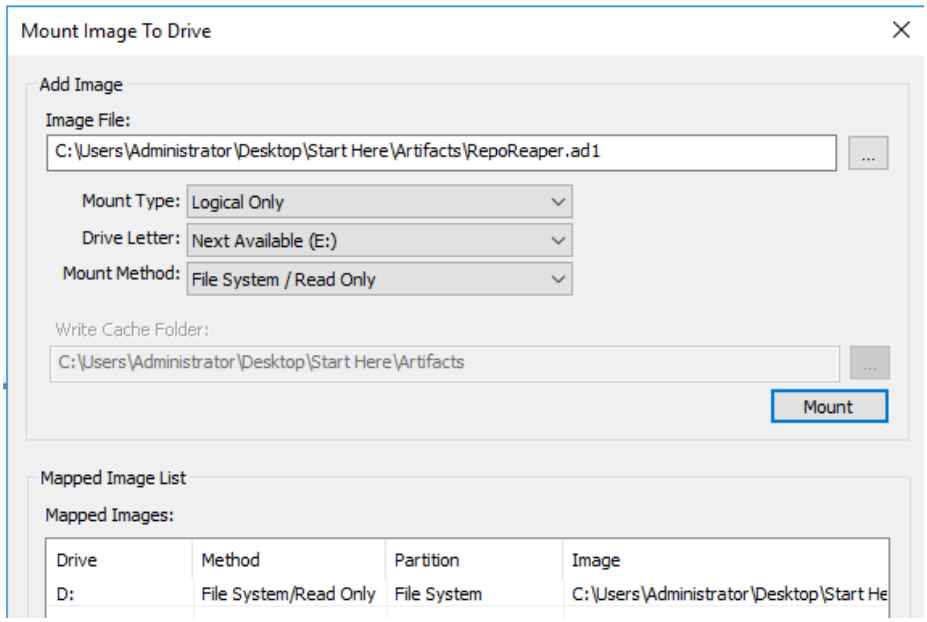
Initial Access

Understanding where the malicious file originated is crucial for identifying the phishing source or compromised server. What is the complete URL of the compromised repository from which the user downloaded the infected open-source tool?

TLDR: View the user's Edge browsing history using DB Browser for SQLite, focusing on the downloads table.

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

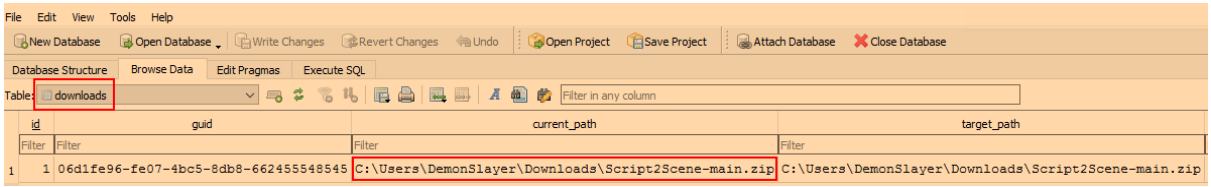


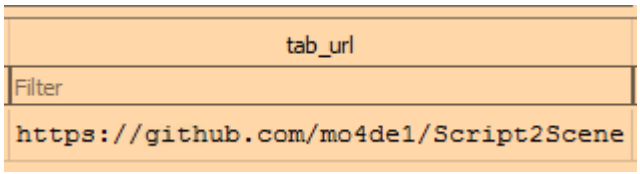


Given the nature of the question, we likely need to view the user's browsing history. After exploring the file system of the mounted image, I located the Edge browsing history for the user DemonSlayer:

- `%SYSTEMROOT%\DemonSlayer\AppData\Local\Microsoft\Edge\User Data\Default`

The history data is stored in a file called "History"; we can use DB Browser for SQLite to view this database. The table we are most concerned with within this database is the downloads table, as it stores all downloads made via the browser. Here we can find that the user installed a zip archive called "Script2Scene-main.zip" from hxxps[://]github[.]com/mo4de1/Script2Scene:

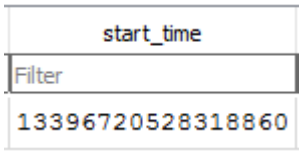


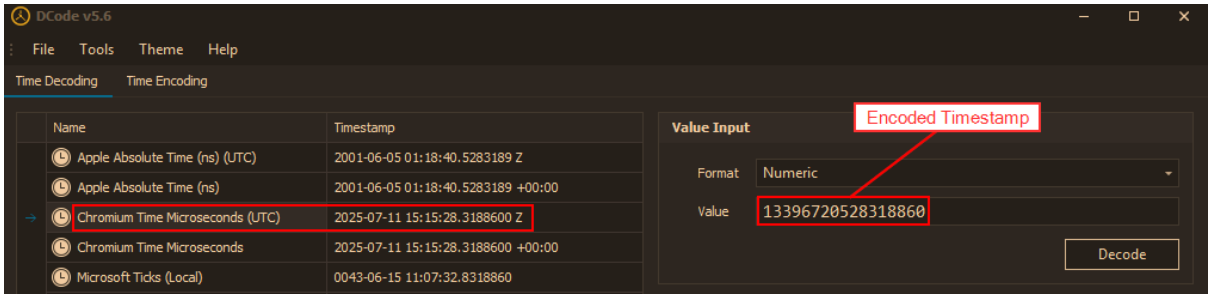


Answer: https://github.com/mo4de1/Script2Scene

Pinpointing the exact download time helps establish the beginning of the attack chain. When did the user download the malicious tool?

Within the download record discovered earlier, there is a field called “start_time” which is encoded in chromium time. To decode this timestamp, we can use a tool called DCode:





Answer: 2025-07-11 15:15

Execution

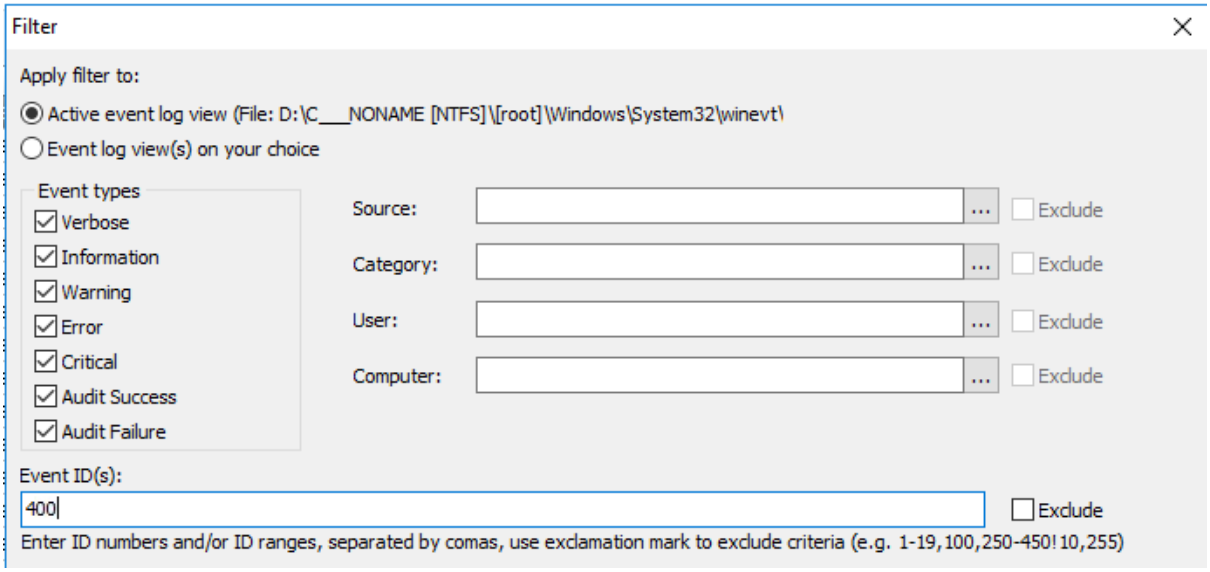
Identifying the initial malicious script provides insight into the attack chain and helps in recognizing similar threats. What is the filename of the PowerShell script that was dropped during the project compilation process?

TLDR: Analyse PowerShell Event ID 400 logs looking for the execution of a PowerShell script.

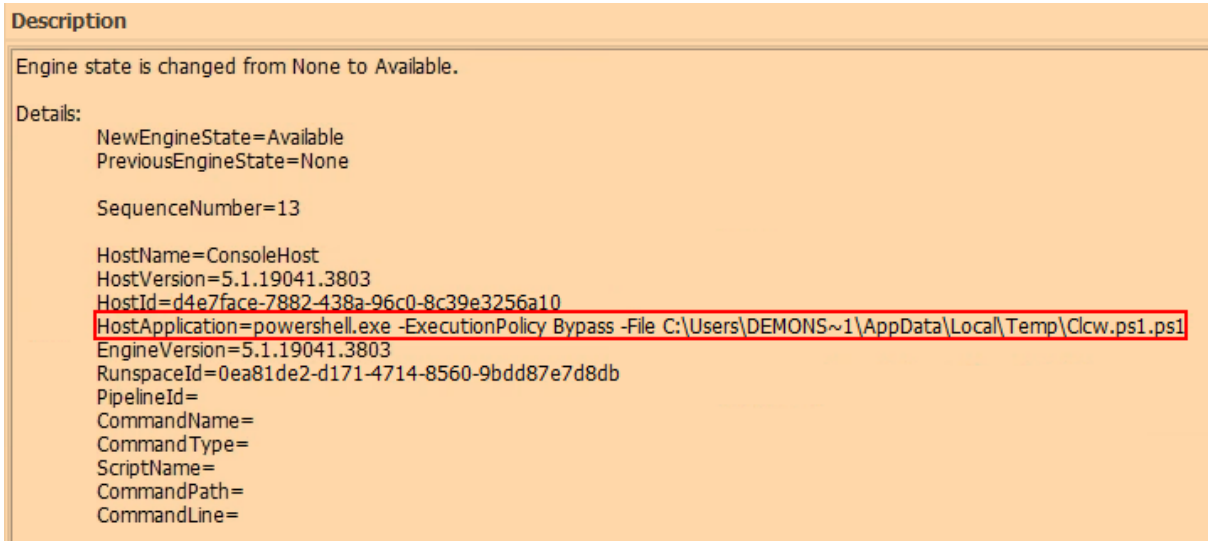
To identify the PowerShell script that was dropped during the compilation process, we can analyse the PowerShell event logs, focusing on Event ID 400 which is generated each time a PowerShell script or command begins. The PowerShell event logs are located at:

- `%SYSTEMROOT%\System32\winevt\Logs`

To view these logs, I am going to use a tool called Event Log Explorer, making sure to filter for Event ID 400:



After exploring these logs, we can see that a PowerShell script called "Clcw.ps1.ps1" was executed within the Temp directory:



Answer: Clcw.ps1.ps1

Understanding which file initiated the payload execution is critical, as it reveals how the attacker gained initial access. What is the filename of the password-protected archive that was downloaded for payload deployment?

TLDR: Look for file creation events in the MFT around the time that Clcw.ps1.ps1 was executed.

To find the filename of the password-protected archive, we can parse the MFT. 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. The $MFT is located in the root directory of the disk image. To parse the MFT, we can use a tool by Eric Zimmerman called MFTECmd:

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

To view the output, I recommend using Timeline Explorer. If you search for the PowerShell script discovered previously. We can see that it was created at 2025-07-11 15:16:23, therefore, I am going to focus on events around this time. After this script landed on disk, we can see Prefetch files being created for 7zip, suggesting that the threat actor installed 7zip to extract the archive. Following these events, we can see the creation of an archive called "4359ce22-2f6d-43b8-b34d-063d74715562.7z":

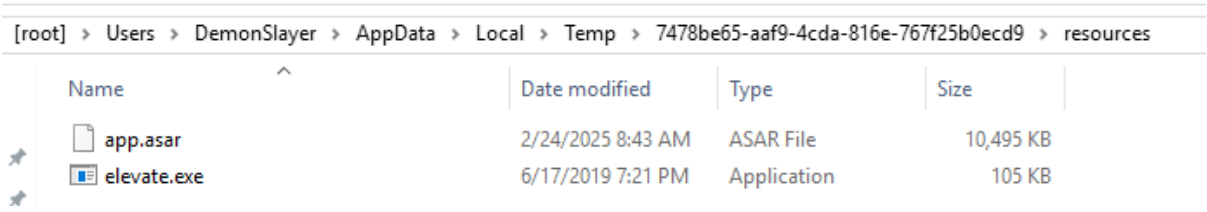
File Name	Extension	Is Directory	Has Ads	Is Ads	File Size	Created0x10
c	c	■	■	■	=	= 2025-07-11 00:00:00
DFBE70A7E5CC19A398EBF1B96859CE5D		☐	☐	☐	11136	2025-07-11 15:16:16
SECHEALTHUI.EXE-332BD1F3.pf	.pf	☐	☐	☐	25168	2025-07-11 15:16:17
SECURITYHEALTHHOST.EXE-633FDACE.pf	.pf	☐	☐	☐	4478	2025-07-11 15:16:19
RUNTIMEBROKER.EXE-8CEE565D.pf	.pf	☐	☐	☐	10173	2025-07-11 15:16:19
SVCHOST.EXE-AD192156.pf	.pf	☐	☐	☐	4346	2025-07-11 15:16:19
3715238b5cf855ae.automaticDestinations-ms	.automaticD…	☐	☐	☐	3584	2025-07-11 15:16:22
Script2Scene-main.lnk	.lnk	☐	☐	☐	1208	2025-07-11 15:16:22
Script2Scene-main (2).lnk	.lnk	☐	☐	☐	931	2025-07-11 15:16:22
Clcw.ps1.ps1	.ps1	☐	☐	☐	2320	2025-07-11 15:16:23
7478be65-aaf9-4cda-816e-767f25b0ecd9		☑	☐	☐	0	2025-07-11 15:16:24
7zInstaller.exe	.exe	☐	☐	☐	1589510	2025-07-11 15:16:27
PY.EXE-D47F3CF1.pf	.pf	☐	☐	☐	3955	2025-07-11 15:16:32
sw.txt	.txt	☐	☐	☐	8534	2025-07-11 15:16:33
tg.txt	.txt	☐	☐	☐	15127	2025-07-11 15:16:33
tk.txt	.txt	☐	☐	☐	9231	2025-07-11 15:16:33
uz-cyrl.txt	.txt	☐	☐	☐	15167	2025-07-11 15:16:33
7-zip.dll	.dll	☐	☐	☐	101376	2025-07-11 15:16:33
7ZINSTALLER.EXE-ACE0B5CF.pf	.pf	☐	☐	☐	10026	2025-07-11 15:16:33
4359ce22-2f6d-43b8-b34d-063d74715562.7z	.7z	☐	☐	☐	71997350	2025-07-11 15:16:35

Answer: 4359ce22-2f6d-43b8-b34d-063d74715562.7z

File hashes are unique identifiers that aid in tracking and correlating malware across different environments. What is the MD5 hash of the JavaScript code embedded within the core Electron application?

TLDR: Navigate to the password protected archive discovered previously and extract the core Electron .asar file. Use the Get-FileHash cmdlet to generate the MD5 hash of the main.js file.

If you navigate to the archive discovered in the previous question, we can find an interesting file within the resources directory called "app.asar":



After a quick google search, we can determine that this is an archive file used by applications built with the Electron.js framework. To extract this file, we can use the following command:

```
npx asar extract app.asar extracted_out
```

Within this directory, we can find three files:

```
Length Name
------ ----
       node_modules
483637 main.js
   564 package.json
```

After doing some research, we can see that main.js is the main script of the application:

Basic Electron Structure

A minimal Electron application requires only a few files in its root directory to function:

```
/
├── index.html
├── main.js
├── package.json
└── node_modules/
```

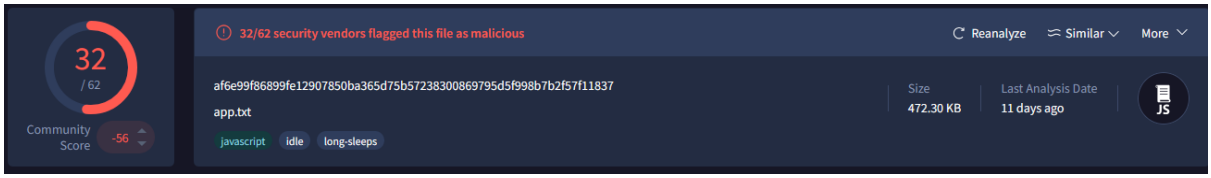
- ***package.json*** : The application's manifest file. It defines the entry point for the main process (commonly `main.js`) in its `main` field, as well as scripts and dependencies.
- ***main.js*** : The **main process script** and entry point of the application. It runs in a Node.js environment, manages the app lifecycle, creates windows (each running a renderer process), and handles native APIs.
- ***index.html*** : The main UI entry point. Each window in an Electron app is a renderer process that loads an HTML file.

To generate the MD5 hash of this file, we can use the Get-FileHash cmdlet:

- `Get-FileHash -Algorithm MD5 -Path .\main.js`

```
Algorithm       Hash
---------       ----
MD5             BD8CB26657654E9CF107AF2218564AB2
```

Upon submitting this hash to VirusTotal, we can see that it receives a significant amount of detection:



Answer: BD8CB26657654E9CF107AF2218564AB2

Persistence

Recognizing the method of persistence helps in effective eradication and ensures no remnants of the malware survive reboot. What are the names of the three scheduled tasks created by the malware?

To find scheduled tasks created by the malware, we can load the SOFTWARE registry hive using Registry Explorer and navigate to the following key:

- `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Schedule\Taskcache\Tasks`

Here I discovered three interesting scheduled tasks created around the time of intrusion:

\Microsoft\Windows\Device Guide\RegisterDeviceSecurityAlert	2025-07-11 15:19:01
\Microsoft\Windows\Device Guide\RegisterDevicePowerStateChange	2025-07-11 15:19:01
\Microsoft\Windows\Device Guide\RegisterDeviceNetworkChange	2025-07-11 15:19:01

All of which were created in the same second.

Answer: RegisterDeviceSecurityAlert, RegisterDevicePowerStateChange, RegisterDeviceNetworkChange

Understanding the exploitation of vulnerable binaries is crucial to defend against privilege escalation attacks that lead to system-level access. Which Windows system binary was exploited by the JavaScript code to achieve privilege escalation?

After exploring PowerShell event logs (Event ID 400) I noticed the following command being executed:

```
HostApplication=powershell.exe -ExecutionPolicy Bypass -Command Start-Process 'C:\Windows\System32\ComputerDefaults.exe'
```

After doing some research, I discovered that ComputerDefaults can be used as a UAC bypass technique. Within the manifest of ComputerDefaults.exe is a property that automatically grants it elevated privileges without showing a UAC prompt. After this command was executed, I also observed some defence evasion techniques that used the runas command, suggesting that privilege escalation was successful:

```
HostApplication=PowerShell -NoProfile -ExecutionPolicy Bypass -WindowStyle Hidden -Command Start-Process PowerShell -ArgumentList '-NoProfile -ExecutionPolicy Bypass -File "C:\Users\DemonSlayer\.vs-script\disabledefender.ps1"' -WindowStyle Hidden -Verb RunAs
```

Answer: ComputerDefaults.exe

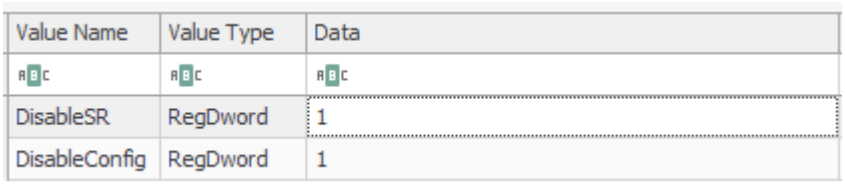
Defence Evasion

Understanding registry modifications ensures complete cleanup and system restoration. When were the registry values modified to disable system backup functionality?

Within the SOFTWARE hive is a SystemRestore key located at:

- `Policies\Microsoft\Windows NT\SystemRestore`

This key allows you to enable, disable, or configure System Protection. Threat actors often disable System Restore to hinder recovery efforts. If you navigate to this key in Registry Explorer, we can see that two keys were modified, with the last write timestamp corresponding to when the key was last modified:



Value Name	Value Type	Data
DisableSR	RegDword	1
DisableConfig	RegDword	1

These registry modifications disable system restore entirely and disables access to system restore configuration UI.

Answer: 2025-07-11 15:19

Anti-analysis checks for antivirus presence help attackers avoid detection and improve their evasion techniques. What is the complete file path of the antivirus software that the PowerShell script checks for during execution?

If you explore the RegistryDeviceSecurityAlert schedule task we identified previously, you can see that it executes a PowerShell script called “mbam.ps1”:

```
<Exec>
  <Command>powershell</Command>
  <Arguments>-ExecutionPolicy Bypass -File "C:\Users\DemonSlayer\AppData\Local\Programs\Common\OneDriveCloud\mbam.ps1"</Arguments>
</Exec>
```

If you navigate to the location of this script and view it, we can see that it tests whether mbuns.exe is present on disk, if it does not exist, the script exits, otherwise it runs silently if Malwarebytes is present.

```
$mbunsPath = "C:\Program Files\Malwarebytes\Anti-Malware\mbuns.exe"
if (Test-Path $mbunsPath) {
    $process = Start-Process -FilePath $mbunsPath -ArgumentList "/silent" -Wait -PassThru
    if ($process.ExitCode -ne 0) {
        exit
    }
}
else {
    exit
}
```

Answer: C:\Program Files\Malwarebytes\Anti-Malware\mbuns.exe

Understanding how threat actor deploys payloads helps track infection vectors and anticipate further stages of the attack. What are the three encrypted filenames that were decrypted by the malware loader?

TDLR: Analyse PowerShell script block logs (Event ID 4104).

If you examine PowerShell script block logs (Event ID 4104 in PowerShell operational logs), we can find the contents of a PowerShell script called "d0561448-8738-44f3-b528-cfd4dcdada56.ps1". This script contains an AES decryption routine:

```
function AES-Decrypt {
    param (
        [byte[]]$EncryptedBytes,
        [string]$Key
    )
    try {
        if (-not $EncryptedBytes) {
            throw "The EncryptedBytes parameter cannot be null or empty."
        }
        if (-not $Key) {
            throw "The key parameter cannot be null or empty."
        }
        $SaltBytes = [byte[]](26, 20, 202, 234, 136, 123, 69, 47)
        $PasswordBytes = [Text.Encoding]::UTF8.GetBytes($Key)
        $KeyAndIv = New-Object System.Security.Cryptography.Rfc2898DeriveBytes(
            $PasswordBytes,
            $SaltBytes,
            1000
        )
```

Going through the logs, we can see that it decrypts three files:

```
'C:\Users\DemonSlayer\AppData\Local\Programs\Common\OneDriveCloud\resources\manifest\boot_f'

'C:\Users\DemonSlayer\AppData\Local\Programs\Common\OneDriveCloud\resources\manifest\kernel_f'

'C:\Users\DemonSlayer\AppData\Local\Programs\Common\OneDriveCloud\resources\manifest\thread_f'
```

Answer: boot_f, kernel_f, thread_f

Masquerading malicious binaries as legitimate applications helps attackers evade detection and maintain persistence. What is the name of the executable that masquerades as a legitimate application but was actually dropped by the Electron-based malware?

To look for evidence of execution, we can parse the Prefetch files located at:

- `%SYSTEMROOT%\Prefetch`

Prefetch is a feature in Windows that can improve the performance of the Windows boot process and reduce the time it takes for programs to start. To parse the Prefetch directory, we can use a tool called PECmd:

- `.\PECmd.exe -d "Windows\Prefetch" --csv . --csvf pre_out.csv`

To view the output, we can use Timeline Explorer. After exploring the output, I noticed the NVIDIA CONTROL PANEL was executed multiple times on the 11th:

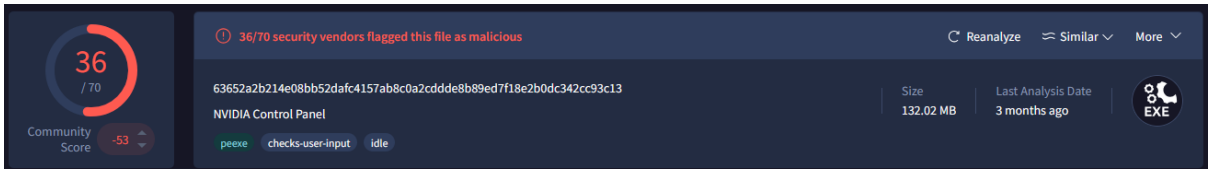
NVIDIA CONTROL PANEL.EXE	2	8515C67D	130736	Windows …	2025-07-13 08:40:54	2025-07-11 15:20:24
SMARTSCREEN.EXE	15	3A39E32D	60792	Windows …	2025-07-13 08:40:54	2025-07-13 08:35:09
NVIDIA CONTROL PANEL.EXE	2	8515C67F	53150	Windows …	2025-07-13 08:40:55	2025-07-11 15:20:25
NVIDIA CONTROL PANEL.EXE	2	8515C685	32832	Windows …	2025-07-13 08:40:55	2025-07-11 15:20:25

NVIDIA Control Panel is a legitimate application that is used to configure settings for NVIDIA GPUs, however, given that it was executed around the time of the intrusion, its likely masquerading as legitimate software. Filtering for this filename in the MFT, we can see that it's located at:

- `.\Users\DemonSlayer\AppData\Local\Microsoft\Vault\UserRoamingTiles\NVIDIAContainer`

Parent Path	File Name
	NVIDIA control
.\Users\DemonSlayer\AppData\Local\Microsoft\Vault\UserRoamingTiles\NVIDIAContainer	NVIDIA Control Panel.exe

If you generate the SHA256 hash for this file and submit it to VirusTotal, we can see that it is clearly malicious and not the legitimate NVIDIA Control Panel:



Answer: NVIDIA CONTROL PANEL.EXE

Discovery & C2

Attackers often leave ransom and contact links for ransom negotiations, which can be crucial for attribution and negotiation strategies. What is the domain name of the command and control server used for data exfiltration?

Event ID 5156 in the Security event logs is generated each time a successful network connection has been established. Using Event Log Explorer, we can filter for network connections established by NVIDIA CONTROL PANEL.EXE. Here we can find it made two connections, one over port 443, and the other over port 80:

```
Application Information:
        Process ID:             175316
        Application Name:       \device\harddiskvolume4\users\demonslayer\appdata\local\microsoft\vault\userroamingtiles\nvidiacontainer\nvidia control panel.exe

Network Information:
        Direction:              Outbound
        Source Address:         172.28.187.167
        Source Port:            53239
        Destination Address:    149.154.167.220
        Destination Port:       443
        Protocol:               6
```

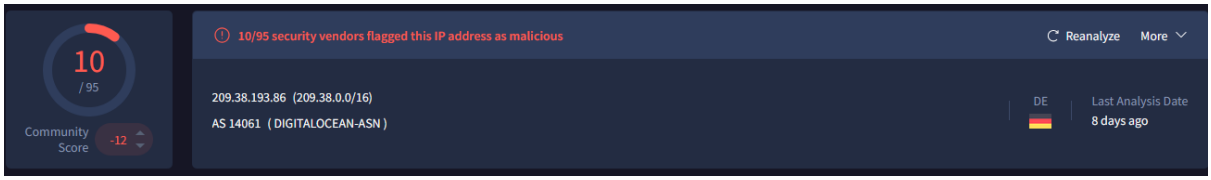
```
Application Information:
        Process ID:             175316
        Application Name:       \device\harddiskvolume4\users\demonslayer\appdata\local\microsoft\vault\userroamingtiles\nvidiacontainer\nvidia control panel.exe

Network Information:
        Direction:              Outbound
        Source Address:         172.28.187.167
        Source Port:            53238
        Destination Address:    104.26.13.205
        Destination Port:       80
        Protocol:               6
```

If you submit the destination IP associated with the 443 connection to VirusTotal and navigate to the Relations tab, we can see that this resolved to api.telegram.org:

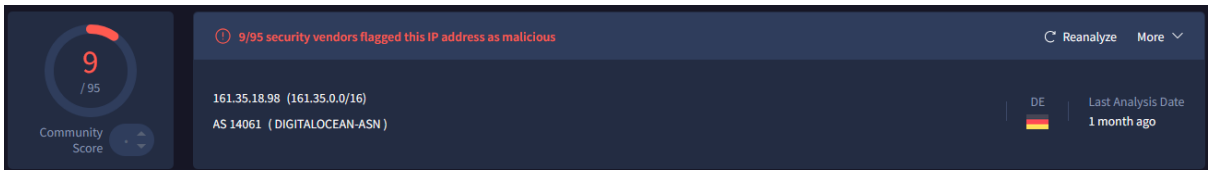
Answer: api.telegram.org

After successful payload injection, the compromised system process established outbound connections to multiple suspicious IP addresses. When analyzing one of these destination IPs on VirusTotal, several related malware samples are discovered. What is the SHA1 hash of the executable sample among the related files?

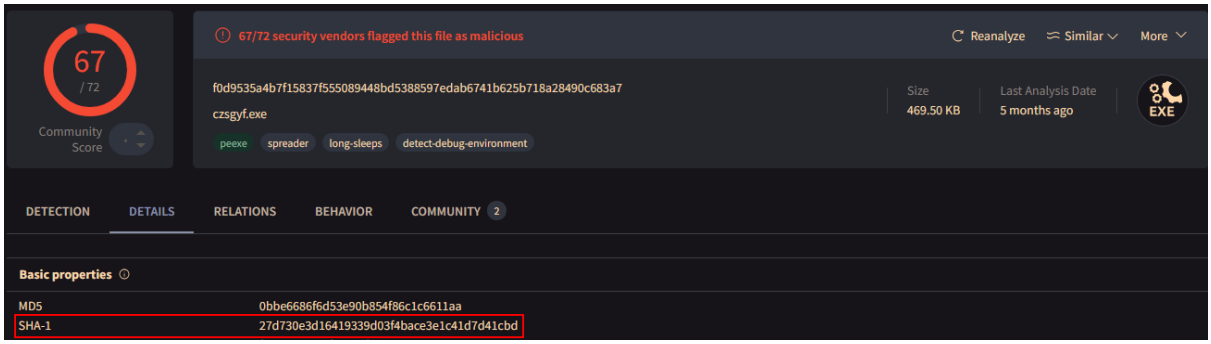
Continuing with exploring 5156 logs, I noticed that bulk of the network connections (over 4000) were made by regasm.exe on the day of the intrusion. These connections were made to two destination IPs and ports. If you submit the first IP to VirusTotal, we can see that it receives multiple detections and is associated with the hosting provider Digital Ocean.



Similarly, the second IP receives a decent number of detections:



The most recent executable sample among the related files is czsgyf.exe:



Answer: 27d730e3d16419339d03f4bace3e1c41d7d41cbd