

# [RE026] A Deep Dive into Zloader - the Silent Night

---

← blog.vincss.net/2022/04/re026-a-deep-dive-into-zloader-the-silent-night.html

## 1. Overview

---

Zloader, a notorious banking trojan also known as **Terdot** or **Zbot**. This trojan was first discovered in 2016, and over time its distribution number has also continuously increased. The Zloader's code is said to be built on the leaked source code of the famous ZeuS malware. In 2011, when source code of ZeuS was made public and since then, it has been used in various malicious code samples.

Zloader has all the standard functionality of a trojan such as being able to fetch information from browsers, stealing cookies and passwords, capturing screenshots, etc. and for making analysis difficult, it applies advanced techniques, including code obfuscation and string encryption, masking Windows APIs call. Recently, CheckPoint expert published an analysis of a Zloader distribution campaign whereby the infection exploited Microsoft's digital signature checking process. In addition, Zloader has also recently partnered with different ransomware gangs are Ryuk and Egregor. This can indicate that the actors behind this malware are still looking for different ways to upgrade it to bypass the defenses. Here is the ranking of Zloader according to the rating from the AnyRun site:

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

---

Source: <https://any.run/malware-trends/zloader>

Most recently, multiple telecommunication providers and cybersecurity firms worldwide partnered with Microsoft's security researchers throughout the investigative effort, including ESET, Black Lotus Labs, Palo Alto Networks' Unit 42, and Avast. They took legal and technical steps to disrupt the ZLoader botnet, seizing control of 65 domains that were used to control and communicate with the infected hosts.

In this article, we will provide detailed analysis and techniques that Zloader uses, including:

- How to unpack to dump Zloader Core Dll.
- The technique that Zloader makes difficult as well as time consuming in the analysis process.
- Decrypt strings used by Zloader by using both IDAPython and AppCall methods.
- Apply AppCall to recover the Windows API calls.

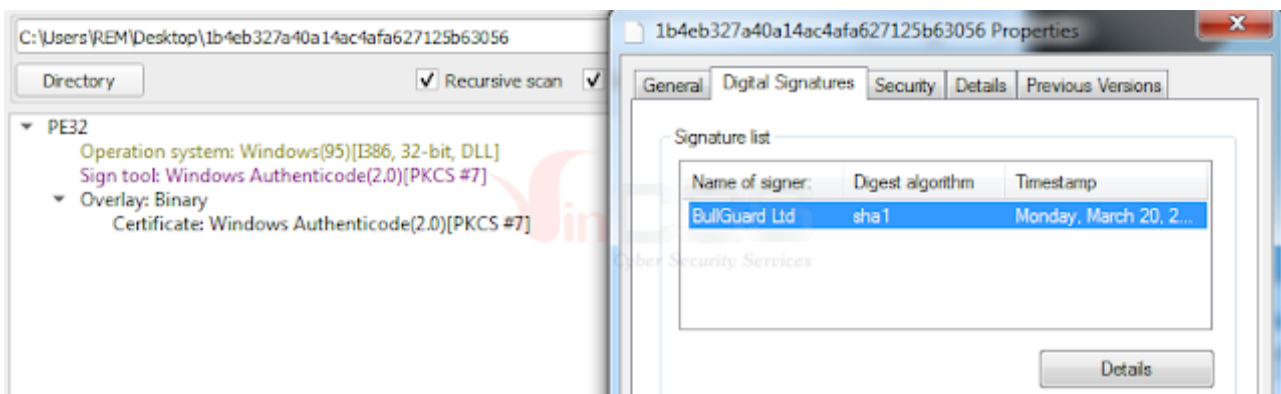
- Process Injection technique that Zloader uses to inject into the **msiexec.exe** process.
- Decrypt configuration information related to C2s addresses.
- How Zloader collects and saves information in the Registry.
- The Persistence technique.

The analyzed sample used in the article:

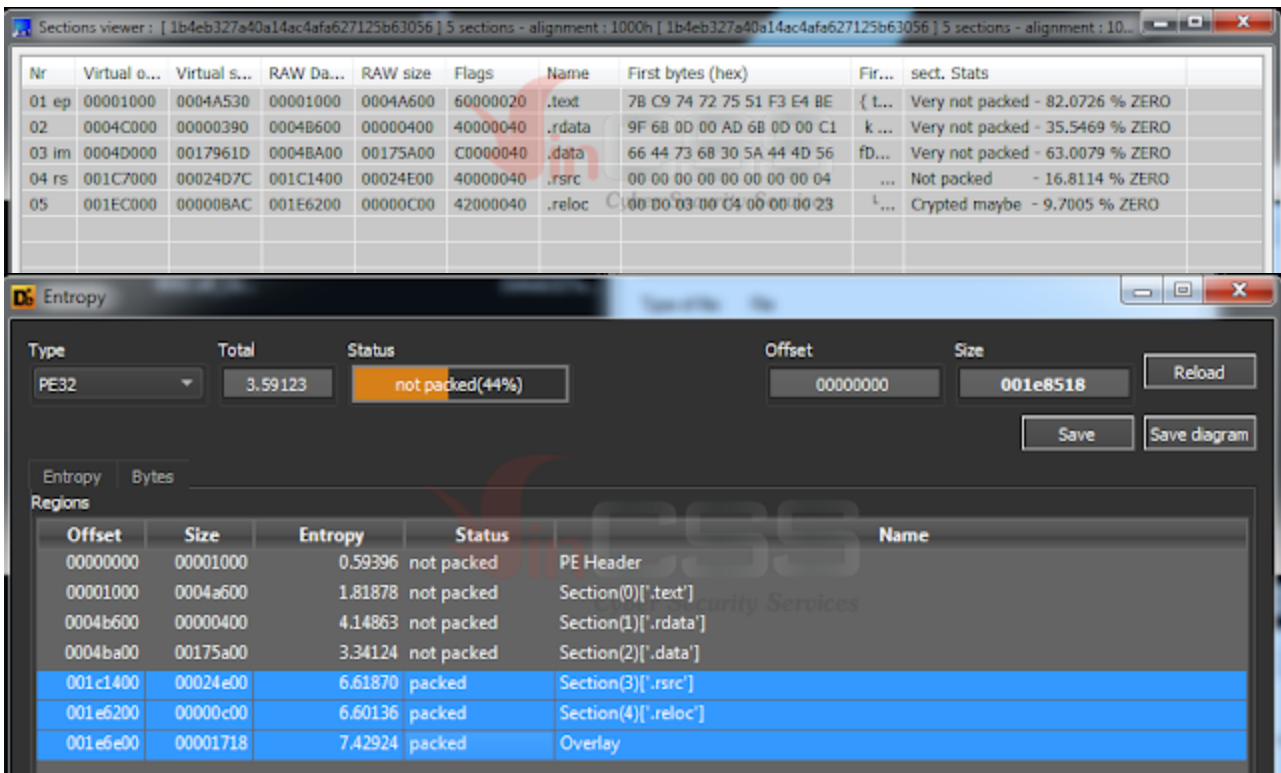
[034f61d86de99210eb32a2dca27a3ad883f54750c46cdec4fcc53050b2f716eb](https://www.virustotal.com/gui/file/034f61d86de99210eb32a2dca27a3ad883f54750c46cdec4fcc53050b2f716eb)

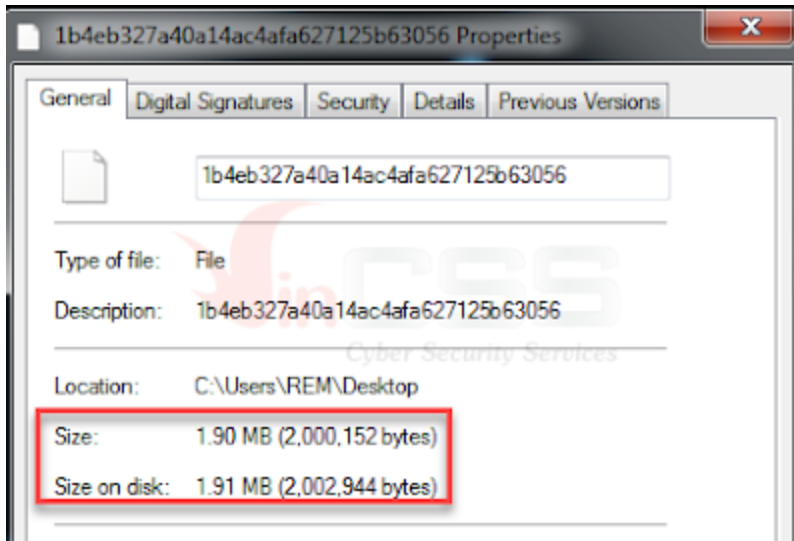
## 2. Unpacking Zloader Core DLL

First, check the sample with **Nauz File Detector**:



By collecting and combining information about sections from **ExeInfo**, entropy in **DiE** as well as the size of the DLL file, we can confirm that this DLL is packed:





For unpacking, use **x64dbg** to load Dll file, set a **bp NtAllocateVirtualMemory**. Then, modify the breakpoint's condition as follows:

### Syntax

C++

```

_kernel_entry NTSYSCALLAPI NTSTATUS NtAllocateVirtualMemory(
[in] HANDLE ProcessHandle,
[in, out] PVOID *BaseAddress,
[in] ULONG_PTR ZeroBits,
[in, out] PSIZE_T RegionSize,
[in] ULONG AllocationType,
[in] ULONG Protect
);

```

Type	Address	Module/Label/Exception	State	Disassembly	Hits
Software	1004A87E	<1b4eb327a40a14ac4afa627125b63056.dll.EntryPoint>	One-time	MOV EAX, ECX	0
	759343BF	<kernel32.dll.ResumeThreadStub@4>	Enabled	MOV EDI, EDI	0
	759438C3	<kernel32.dll.CreateProcessInternalW@48>	Enabled	PUSH 0x624	0
	7594D9A8	<kernel32.dll.WriteProcessMemoryStub@20>	Enabled	MOV EDI, EDI	0
	7723FAB0	<ntdll.dll.ZwAllocateVirtualMemory@24>	Enabled	MOV EAX, 0x15	0

Edit Breakpoint 7723FAB0 <ntdll.ZwAllocateVirtualMemory@24>

Break Condition: [esp+18]==00000040

Log Text:

Log Condition:

Command Text:

Command Condition:

Name:

Hit Count: 0

Singleshoot  Silent  Fast Resume Save Cancel

Execute with **F9** and wait until the breakpoint is hit (after about 1126120 hits):

Type	Address	Module/Label/Exception	State	Disassembly	Hits	Summary
Software	759948BF	<kernel32.dll!_ResumeThreadStub@>	Enabled	mov edi, edi	0	
Software	75943BC3	<kernel32.dll!_CreateProcessInternalW@>	Enabled	push 0x624	0	
Software	759409A8	<kernel32.dll!_WriteProcessMemoryStub@2@>	Enabled	mov edi, edi	0	
Software	7723FA80	<ntdll.dll!_ZwAllocateVirtualMemory@24>	Enabled	mov eax, 0x1	1126120	Kernel32!_ZwAllocateVirtualMemory@24

Following the allocated memory regions, after the 3rdrd hit, the core DLL of Zloader will be unpacked:

```
00000000  44 50 78 00 01 00 00 00 04 00 00 00 00 00 00 00 00
00000004  00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
00000008  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000014  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000018  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000001C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000024  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000028  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000002C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000034  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000038  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000003C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000044  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000048  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000004C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000054  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000058  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000005C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000064  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000068  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000006C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000074  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000078  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000007C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000084  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000088  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000008C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000094  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000098  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000009C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000AC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000BC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000CC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000DC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000E8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000EC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000F8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000FC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000104  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000108  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000010C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000114  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000118  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000011C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000124  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000128  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000012C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000134  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000138  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000013C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000144  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000148  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000014C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000154  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000158  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000015C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000164  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000168  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000016C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000174  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000178  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000017C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000184  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000188  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000018C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000194  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000198  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000019C  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001A4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001A8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001AC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001B8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001BC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001C8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001CC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001D8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001DC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001E4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001E8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001EC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001F4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001F8  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000001FC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000200  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Dump this DLL to disk, the file has MD5: 9b5589fcd123a3533584a62956f2231b.

```
zloader_core_dll.bin
DOS Header
DOS stub
NT Headers
Signature
File Header
Optional Header
Section Headers
Sections
.text
.rdata
.data
.reloc
```

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder
1FD8E	KERNEL32.dll	18	FALSE	21010	0	0
1FD92	USER32.dll	14	FALSE	2105C	0	0
1FDE6	GDI32.dll	2	FALSE	21098	0	0

```
KERNEL32.dll [ 18 entries ]
Call via Name Ordinal Original Thunk Thunk Forwarder
210A4 CloseHandle - 21138 21138 -
210A8 CreateFileW - 21146 21146 -
210AC DeleteCriticalSe... - 21154 21154 -
210B0 DeleteFileW - 2116C 2116C -
210B4 GetFileAttribute... - 2117A 2117A -
210B8 GetLastError - 21190 21190 -
210BC GetModuleHan... - 211A0 211A0 -
210C0 GetProcessHan... - 211B4 211B4 -
```

### 3. Anti-analysis

To consume time of the analyst, Zloader uses meaningless functions, or rewrites functions that look very complicated but only to perform simple tasks such as **AND**, **OR**, **XOR**, **ADD**, **SUB**, etc.

For example, a function that does a meaningless task, however it can cause a delay in execution in a sandbox environment:

```
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) + 0xB000000) >> 0x18));
    tmp2 = tmp1 * (g_0C80441ADh + 0x12D);
    InsertMenuItemW(
        ((g_0C80441ADh + 0x12D) & (((g_0C80441ADh << 0x18) + 0xB000000) >> 0x18)),
        tmp1 * (g_0C80441ADh + 0x12D),
        (g_0C80441ADh + 0x12D) & (((g_0C80441ADh << 0x18) + 0xB000000) >> 0x18),
        ((g_0C80441ADh + 0x12D) & (((g_0C80441ADh << 0x18) + 0xB000000) >> 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;
}
```

Functions that perform **AND**, **OR** operations:

```
char __cdecl f_zl_and(char num1, char num2)
{
    int v2; // ebx
    int v3; // edi
    int v4; // edi
    const @CHAR *v5; // ebx
    signed int v7; // [esp+0h] [ebp-14h]
    HDC hdc; // [esp+0h] [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_0xF623385A(0x843A5CA6) && sub_10006100(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 - 0x181)
        + (f_zl_xor_arg_with_0xF623385A(0x843A5CA6) + num1 + 0xCA * (v3 & 8) * v3 - 0x181)
        + 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. Use IDAPython

All strings that the core DLL uses are encrypted. The wide string decoder function will take two parameters as input:

- **First parameter:** the address containing the encrypted string.
- **Second parameter:** the address where the string is stored after decoding.

```
.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_371]
```

```
7 calls, 0 strings
calls:
020 call f_zl_return_0x2D5_if_arg1_equal_arg2_else_0x0
01C call f_zl_xor_arg_with_0xF623385A
020 call f_zl_add_arg1_with_arg2
01C call f_zl_xor_with_0x385A
020 call f_zl_and_arg1_with_arg2
020 call f_zl_sub_arg1_from_ar2
020 call f_zl_sub_arg1_from_ar2
```

The pseudocode at the `f_zl_decrypt_wstring` decryption function looks confusing, but if we look closely, the function performs a simple xor loop with the decryption key is "PgtrIPF-2ftOj000x":

```

// xor_key = "PgtrIPF-2ft0j000x"
dec_char = *g_PgtrIPF2ft0j000x;
LOWORD(dec_char) = *encString ^ dec_char;
*decString = dec_char;
// 1st, dec_char = 0x50 (0)
// decString[0] = encString[0] ^ dec_char

if ( !(_WORD)dec_char )
{
    return ptr_decString;
}
i = 0;
while ( 1 )
{
    val_0x20 = f_zl_sub_arg1_from_ar2(0, 0xFFE0);
    if ( (unsigned __int16)f_zl_sub_arg1_from_ar2(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 ^ xor\_key\_val

Based on the above pseudocode, the python code that performs decryption as follows:

```

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')

```

With the help of IDAPython, we can automate the whole process of string decoding and add annotations at the decryption functions in IDA for further analysis. The entire python code is as follows:

```

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 = ''
    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 idutils.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 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()

```

The results before and after the script execution will make the analysis easier:



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_1000CC00+...	call f_zl_decrypt_wstring	Down	p	sub_1000CC00+...	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: Proxyfier.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: %s_%08X%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_100189B0+18	call f_zl_decrypt_wstring	Down	p	sub_100189B0+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_1001BCC0+...	call f_zl_decrypt_wstring	Down	p	sub_1001BCC0+...	call f_zl_decrypt_wstring: tmp
Down	p	sub_1001BCC0+...	call f_zl_decrypt_wstring	Down	p	sub_1001BCC0+...	call f_zl_decrypt_wstring: %s%08x
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



Before

After

## 4.2. Use IDA AppCall

If you don't have time to dig into the decryption implementation of the function, or when the algorithm is too complex, we can use IDA's useful feature known as AppCall, to help decrypt the data. Basically, Appcall is a mechanism used to call functions inside the debugged program from the IDA debugger. Before applying AppCall, the first thing is to given a function with a correct prototype. For example, the function **f\_zl\_decrypt\_wstring** has the following prototype:

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

Note again that in order to use AppCall, the program must be debugged. As shown below, IDA is stopping at the breakpoint set at **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 8Ch
.text:72A7C470 lpReserved= dword ptr 10h
.text:72A7C470
EIP> .text:72A7C478 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)

```

Then execute the below python script to decode and add comments related to decoded strings at the functions:

```

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('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

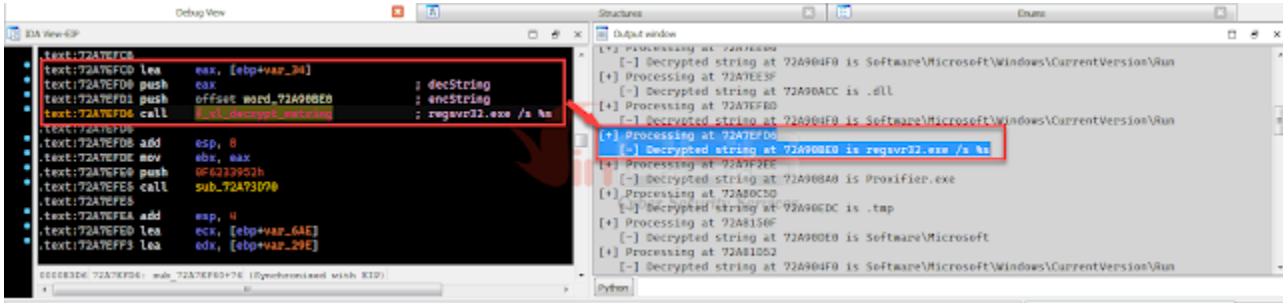
def search_inst(ea, inst):
    """
    Return the address of wanted instruction
    """
    while True:
        if idc.print_insn_mnem(ea) == inst:
            return ea
        ea = idc.prev_head(ea)

# 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)

```

The final result should be similar to the image below:

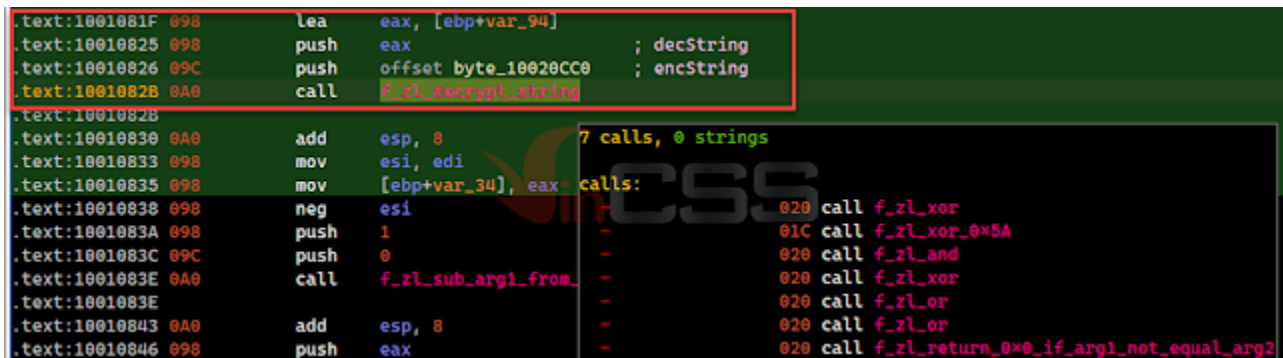


## 5. Decrypt ansi string

### 5.1. Use IDAPython

Besides the function to decode wide strings, Zloader also uses the function to decode ansi strings. This function also accepts two arguments:

- **First parameter:** the address containing the encrypted string.
- **Second parameter:** the address where the string is stored after decoding.



Similar to the above `f_zl_decrypt_wstring` function, the pseudocode of the `f_zl_decrypt_string` function looks quite messy, but it still uses an xor loop to decrypt with the decryption key still "PgtrIPF-2ftOj000x":

```

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[0xFFFFFFFF * (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;

```

Here is the full python code to automate the whole process of decoding strings and adding comments at functions:

```

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 F0 F7 E1 0F B6 C3 89 D6 6A ??" #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()

```

The results before and after the script execution

xrefs to f_zl_decrypt_string				xrefs to f_zl_decrypt_string			
Direction	Type	Address	Text	Direction	Type	Address	Text
Up	p	f_zl_setup_URL_ca...	call f_zl_decrypt_string	Up	p	f_zl_setup_URL_ca...	call f_zl_decrypt_string /%a
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 .cob
Up	p	sub_1300F270+4F	call f_zl_decrypt_string	Up	p	sub_1300F270+4F	call f_zl_decrypt_string BOT-INFO
Up	p	sub_1300F270+42	call f_zl_decrypt_string	Up	p	sub_1300F270+62	call f_zl_decrypt_string It's a debug version.
Up	p	sub_1300F270+AD	call f_zl_decrypt_string	Up	p	sub_1300F270+AD	call f_zl_decrypt_string BOT-INFO
Up	p	sub_1300F270+C3	call f_zl_decrypt_string	Up	p	sub_1300F270+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 msioexec.exe
Up	p	sub_13001010+18	call f_zl_decrypt_string	Up	p	sub_13001010+18	call f_zl_decrypt_string [SSS]shwuyy[SSSSSSS]-70ABCDEFGHIJKLMNOPQRSTUVWXYZ[!*_abcdefghijklmnopqrstuvwxyz]
Up	p	sub_13001070+F9	call f_zl_decrypt_string	Up	p	sub_13001070+F9	call f_zl_decrypt_string %d
Up	p	sub_130011500+25	call f_zl_decrypt_string	Up	p	sub_130011500+25	call f_zl_decrypt_string be
Up	p	sub_130011500+30	call f_zl_decrypt_string	Up	p	sub_130011500+30	call f_zl_decrypt_string he
Up	p	sub_130011500+78	call f_zl_decrypt_string	Up	p	sub_130011500+78	call f_zl_decrypt_string tr
Up	p	sub_130011500+A3	call f_zl_decrypt_string	Up	p	sub_130011500+A3	call f_zl_decrypt_string tl
Up	p	sub_130011500+CB	call f_zl_decrypt_string	Up	p	sub_130011500+CB	call f_zl_decrypt_string div
Up	p	sub_130011500+F6	call f_zl_decrypt_string	Up	p	sub_130011500+F6	call f_zl_decrypt_string h1
Up	p	sub_130011500+121	call f_zl_decrypt_string	Up	p	sub_130011500+121	call f_zl_decrypt_string h2
Up	p	sub_130011500+14C	call f_zl_decrypt_string	Up	p	sub_130011500+14C	call f_zl_decrypt_string h3
Up	p	sub_130011500+177	call f_zl_decrypt_string	Up	p	sub_130011500+177	call f_zl_decrypt_string h
Up	p	sub_130011500+19F	call f_zl_decrypt_string	Up	p	sub_130011500+19F	call f_zl_decrypt_string h5
Up	p	sub_130011500+1C7	call f_zl_decrypt_string	Up	p	sub_130011500+1C7	call f_zl_decrypt_string h6
Up	p	sub_130011500+1EF	call f_zl_decrypt_string	Up	p	sub_130011500+1EF	call f_zl_decrypt_string k
Up	p	sub_130011500+360	call f_zl_decrypt_string	Up	p	sub_130011500+360	call f_zl_decrypt_string s
Up	p	sub_130011500+445	call f_zl_decrypt_string	Up	p	sub_130011500+445	call f_zl_decrypt_string i
Up	p	sub_130011500+495	call f_zl_decrypt_string	Up	p	sub_130011500+495	call f_zl_decrypt_string s
Up	p	sub_130011500+507	call f_zl_decrypt_string	Up	p	sub_130011500+507	call f_zl_decrypt_string tl
Up	p	sub_130011500+80	call f_zl_decrypt_string	Up	p	sub_13001090+00	call f_zl_decrypt_string .com
Up	p	sub_130013C00+3E0	call f_zl_decrypt_string	Up	p	sub_130013C00+3E0	call f_zl_decrypt_string .me
Up	p	sub_130013C00+4A2	call f_zl_decrypt_string	Up	p	sub_130013C00+4A2	call f_zl_decrypt_string .dl
Up	p	sub_130013C00loc_...	call f_zl_decrypt_string	Up	p	sub_130013C00loc_...	call f_zl_decrypt_string .me
Up	p	sub_130014300+221	call f_zl_decrypt_string	Do...	p	sub_130014300+221	call f_zl_decrypt_string 6.3
Up	p	f_zl_send_request_...	call f_zl_decrypt_string	Do...	p	f_zl_send_request_...	call f_zl_decrypt_string "?"
Up	p	f_zl_send_request_...	call f_zl_decrypt_string	Do...	p	f_zl_send_request_...	call f_zl_decrypt_string HTTP/1.1
Up	p	f_zl_send_request_...	call f_zl_decrypt_string	Do...	p	f_zl_send_request_...	call f_zl_decrypt_string .
Up	p	f_zl_send_request_...	call f_zl_decrypt_string	Do...	p	f_zl_send_request_...	call f_zl_decrypt_string Connection: close
Up	p	sub_130014F00+56	call f_zl_decrypt_string	Do...	p	sub_130014F00+56	call f_zl_decrypt_string /post.php
Up	p	sub_130014F00+F2	call f_zl_decrypt_string	Do...	p	sub_130014F00+F2	call f_zl_decrypt_string https://
Up	p	sub_130017200+15	call f_zl_decrypt_string	Do...	p	sub_130017200+15	call f_zl_decrypt_string ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/-
Up	p	sub_130017400+1C	call f_zl_decrypt_string	Do...	p	sub_130017400+1C	call f_zl_decrypt_string kern32.dll
Up	p	sub_130019000+10	call f_zl_decrypt_string	Do...	p	sub_130019000+10	call f_zl_decrypt_string Basic
Up	p	sub_13001E070+1A	call f_zl_decrypt_string	Do...	p	sub_13001E070+1A	call f_zl_decrypt_string bcdfghijklmnopstvwxyz
Up	p	sub_13001E070+34	call f_zl_decrypt_string	Do...	p	sub_13001E070+34	call f_zl_decrypt_string .anxoy

## 5.2. Use IDA AppCall

To use AppCall, same as above, need to define correctly the prototype for the `f_zl_decrypt_string` function as follows: `char *__cdecl f_zl_decrypt_string(char *encString, char *decString);`

Slightly modified the script used for decoding the wide strings above:

```

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)

```

Result after running the script:

The screenshot shows a debugger window with two panes. The left pane displays assembly code for a function, with instructions like `mov dword_72023C9C, eax` and `lea eax, [ebp+var_31]`. The right pane shows the output of the script, listing decrypted strings at various memory addresses. One string is highlighted in red: `[-] Decrypted string at 72028590 is Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36`. Other strings include `[-] Decrypted string at 72028200 is EOT-INFO` and `[-] Decrypted string at 72028220 is It's a debug version.`

## 6. List of DLLs used by Zloader

In the list of strings decrypted by the `f_zl_decrypt_string` function above, there is a string after the decryption that is quite meaningless. Going to this address, after diving into it I noticed that the first parameter passed to the function is an array containing the addresses of the encrypted strings. Based on the corresponding `index` value of the array will access the address containing the corresponding encrypted string:

```

text:1000E6FF 9AB lea  eax, [ebp+dec_str]
text:1000E705 9AB push eax                ; dec_str
text:1000E706 9AC push ds:g_ptr_enc_dll_str[ebx*4] ; enc_str
text:1000E70D 980 call  f_zl_decrypt_string ;_cvb

```

```

40  if ( f_zl_and_ex(v21, (val_0=3A60E8DA | 0=3A60E8DA) * 0=709F1728) )
41  {
42  { sz_dll_name = f_zl_decrypt_string((g_ptr_enc_dll_str)[arg_dll_index], dec_str);
43  f_zl_strcpy(v22, sz_dll_name, 0xFFFFFFFF);

```

Going to the `g_ptr_enc_dll_str` array (*renamed above*) will see a list of addresses as shown below:

```

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

```

points to encrypted string

Modify the script to decode the specific Dll strings, the results obtained when executing the script are as follows:

```

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 ; iphlpapi.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

```



To summarize, we have a list of **indexes** corresponding to the DLLs that Zloader can use to retrieve the addresses of APIs:

<b>Index</b>	<b>Dll Name</b>
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	wtsapi32.dll
18	mpr.dll
19	wininet.dll
20	userenv.dll
21	bcrypt.dll

## 7. Dynamic APIs resolve

Similar to other advanced malware... Zloader will also get the address of API function(s) through searching by pre-computed hash value based on API function name.

```

.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

```

As shown in the above figure, the `f_zl_resolve_api_func_ex` function takes two parameters:

- (1): The first parameter is `dll_index`. Based on this parameter, the function will decode the name of the corresponding Dll, then call the `LoadLibraryA` function to get the base address of this Dll.

```

{
  // decrypt Dll name based on Dll index
  sz_dll_name = f_zl_decrypt_string((&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): The second parameter is **pre\_api\_hash**. This parameter is the pre-computed hash of the API function name. The function **f\_zl\_resolve\_api\_func\_ex** will call **f\_zl\_resolve\_api\_func** to retrieve the corresponding API address:

```

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

```

The pseudocode at the **f\_zl\_resolve\_api\_func** function as follows:

```

export_dir_va = (dll_base_addr + export_dir_rva);
export_dir_size = pOptionalHeaders->DataDirectory[0].Size;
AddressOfNameOrdinals_sub_0x31705B64 = dll_base_addr + export_dir_va->AddressOfNameOrdinals - 0x31705B64;
val_0xCE82A49C = f_zl_xor_arg_with_0xF6233B5A(0x38A19FC6);
pOrdinalsTbl = f_zl_sub_arg1_from_arg2(AddressOfNameOrdinals_sub_0x31705B64, 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) + 0x74C0298C, dll_base_addr) - 0x74C0298C;

```

convert API name to lowercase

compare calculated hash to pre\_hash

The entire pseudocode of the function that performs the hash calculation by the API function name is as follows:

```

int __fastcall f_zl_calc_hash(char *inString, int strlen)
{
    int calced_hash; // edi MAP06T
    unsigned int i; // edx MAP06T
    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_0xF623385A(0x7472BBA7);
            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_0xF623385A(0xE9A4F0B8) - 0x1F89C6E8;
        ++inString;
    }
    while ( i != strlen );
    return calced_hash;
}

```

Based on the above pseudocode, re-implement using Python code as follows:

```

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

```

Results when using the above function to find API functions corresponding to hash values hash 0xFDA8B77, 0xB1C1FE3, 0x8ADF2D1:

<pre> v1 = f_zl_resolve_api_func_exe(0, 0xFDA8B77); (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); </pre>	<pre> python .\zloader_brute_api_funcs.py API hash: 0xFDA8B77 --&gt; API found: GetModuleFileNameW API hash: 0xB1C1FE3 --&gt; API found: GetProcAddress API hash: 0x8ADF2D1 --&gt; API found: LoadLibraryA </pre>
--	---

With all the above analysis results, it is possible to write an IDAPython script to recover all the APIs that Zloader uses. However, to avoid having to dig into Zloader's hashing algorithm for each analysis, here I will use AppCall to do this task. The python code that uses AppCall is as follows:

```

import idc, idaapi, idautils

def resolve_n_comment(func, func_name):
    """
    Resolve API
    """
    for xref in idautils.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 ({})(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)

```

Note, Zloader has many areas of code that call to the `f_zl_resolve_api_func_ex` function, but there will be areas of code that do not have any reference to it and that area has not been defined as a complete function. Therefore, to be able to run the above script, it is necessary to create functions for those first. The final result after executing the script will be as follows:

The screenshot displays two windows from IDA Pro. On the left is the assembly view for the function `f_zl_resolve_api_func_ex`. The instruction at address `724E02A8` is `call f_zl_resolve_api_func_ex ; kernel32!GetModuleFileNameW`, which is highlighted with a red box. On the right is the console window showing the script's output. It shows the script processing the instruction at `724E02A8`, identifying the module index as `00000000`, the precalculated hash as `0FD08077`, and the resolved API as `kernel32!GetModuleFileNameW`. A yellow arrow points from the console output to the assembly instruction.

xrefs to f_zl_resolve_api_func_ex				xrefs to f_zl_resolve_api_func_ex			
Direction	Typ	Address	Text	Direction	Typ	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; shlwapiPathUnquoteSpacesW
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_compone...	call f_zl_resolve_api_func_ex	Up	p	f_zl_setup_URL_compone...	call f_zl_resolve_api_func_ex; wininetInternetCrackURLA
Up	p	sub_10001700+6E	call f_zl_resolve_api_func_ex	Up	p	sub_724D1700+6E	call f_zl_resolve_api_func_ex; ws232WSASetLastError
Up	p	sub_10001700+8D	call f_zl_resolve_api_func_ex	Up	p	sub_724D1700+8D	call f_zl_resolve_api_func_ex; ws232accept
Up	p	sub_10001900+2F	call f_zl_resolve_api_func_ex	Up	p	sub_724D1900+2F	call f_zl_resolve_api_func_ex; ws232select
Up	p	sub_10001900+71	call f_zl_resolve_api_func_ex	Up	p	sub_724D1900+71	call f_zl_resolve_api_func_ex; ws232recv
Up	p	sub_10001900+A6	call f_zl_resolve_api_func_ex	Up	p	sub_724D1900+A6	call f_zl_resolve_api_func_ex; ws232send
Up	p	sub_10001900+F9	call f_zl_resolve_api_func_ex	Up	p	sub_724D1900+F9	call f_zl_resolve_api_func_ex; ws232select
Up	p	sub_10001D80+1A	call f_zl_resolve_api_func_ex	Up	p	sub_724D1D80+1A	call f_zl_resolve_api_func_ex; ole32CoCreateInstance
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; kernel32SetFileTime
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; kernel32GetFileAttributesW
Up	p	sub_10002270+80	call f_zl_resolve_api_func_ex	Up	p	sub_724D2270+80	call f_zl_resolve_api_func_ex; shlwapiPathAddExtensionW
Up	p	sub_10002640+1F	call f_zl_resolve_api_func_ex	Up	p	sub_724D2640+1F	call f_zl_resolve_api_func_ex; ws232getsockname
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; ntdllRtlAllocateHeap
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; ws232WSAIoctl
Up	p	sub_10003000+64	call f_zl_resolve_api_func_ex	Up	p	sub_724D3000+64	call f_zl_resolve_api_func_ex; shlwapiURLunescapeA
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; kernel32Process32FirstW
Up	p	sub_10003600+85	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+85	call f_zl_resolve_api_func_ex; kernel32Process32NextW
Up	p	sub_10003600+A6	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+A6	call f_zl_resolve_api_func_ex; kernel32OpenProcess
Up	p	sub_10003600+C4	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+C4	call f_zl_resolve_api_func_ex; kernel32CloseHandle
Up	p	sub_10003600+E	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+E	call f_zl_resolve_api_func_ex; kernel32OpenMutexW
Down	p	sub_10003600+2D	call f_zl_resolve_api_func_ex	Up	p	sub_724D3600+2D	call f_zl_resolve_api_func_ex; kernel32CloseHandle
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; advapi32RegCreateKeyExW
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; advapi32RegSetValueExW
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
Down	p	sub_100042D0+2B	call f_zl_resolve_api_func_ex	Up	p	sub_724D42D0+2B	call f_zl_resolve_api_func_ex; shlwapiwvnsprintfA
Down	p	sub_10004810+37	call f_zl_resolve_api_func_ex	Up	p	sub_724D4810+37	call f_zl_resolve_api_func_ex; ntdllRtlAllocateHeap
Down	p	sub_10004810+4E	call f_zl_resolve_api_func_ex	Up	p	sub_724D4810+4E	call f_zl_resolve_api_func_ex; ntdllRtlAllocateHeap
Down	p	sub_10005690+13	call f_zl_resolve_api_func_ex	Up	p	sub_724D5690+13	call f_zl_resolve_api_func_ex; kernel32GetTempPathW
Down	p	sub_10005830+12	call f_zl_resolve_api_func_ex	Up	p	sub_724D5830+12	call f_zl_resolve_api_func_ex; shlwapiSHDeleteKeyW
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; kernel32WaitForSingleObject
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; wininetInternetReadFile
Down	p	sub_10006E80+17	call f_zl_resolve_api_func_ex	Up	p	sub_724D6E80+17	call f_zl_resolve_api_func_ex; ws232shutdown
Down	p	sub_10006E80+2C	call f_zl_resolve_api_func_ex	Up	p	sub_724D6E80+2C	call f_zl_resolve_api_func_ex; ws232closesocket
Down	p	sub_100071A0+9C	call f_zl_resolve_api_func_ex	Up	p	sub_724D71A0+9C	call f_zl_resolve_api_func_ex; shlwapiwvnsprintfA
Down	p	sub_10007EF0+14	call f_zl_resolve_api_func_ex	Up	p	sub_724D7EF0+14	call f_zl_resolve_api_func_ex; ws232shutdown
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; kernel32GetFileSizeEx
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; kernel32CloseHandle
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; kernel32VirtualAlloc
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

However, as shown in the figure there are still places where the API function can't be recovered, that's because Zloader has performed the previous calculation of the `dll_index` and `pre_api_hash` values and saved them in the register. After that, call the `f_zl_resolve_api_func_ex` function:

```

setz  dl
push  @F6233853h ; InVal
call  f_zl_xor_arg_with_0xF623385A
add   esp, 4
mov   esi, eax
push  @F5322733h ; InVal
call  f_zl_xor_arg_with_0xF623385A
add   esp, 4
push  eax ; pre_hash
push  esi ; module_index
call  f_zl_resolve_api_func_ex

```

```

13 {
14     return 0;
15 }
16 ReqQueryValueExW = f_zl_resolve_api_func_ex(9u, 0x8897C7u);
17 LOBYTE(samDesired) = ReqQueryValueExW(hKey, lpValueName, 0, 0, 0) == 0;
18
19 module_idx_9 = f_zl_xor_arg_with_0xF623385A(0xF6233853);
20 pre_hash_0x3111C69 = f_zl_xor_arg_with_0xF623385A(0xF5322733);
21 RegCloseKey = f_zl_resolve_api_func_ex(module_idx_9, pre_hash_0x3111C69);
22 RegCloseKey(hKey);
23 return samDesired;

```

## 8. Process Injection Technique

Zloader, when executed, will inject Core Dll into the `msiexec.exe` process. The whole process is as follows:

Use the **CreateProcessA** API function to create the **msiexec.exe** process in the **SUSPENDED** state.

```
// msiexec.exe
sz_msiexec = f_zl_decrypt_string(asc_749805F3, v30);
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) )
{
```



Name	PID	Private Bytes	Working Set	Session ID	Architecture	Company Name	Product Name
rundll32.exe	1064	1.09 MB	REM-PC/REM	0	64-bit	Windows	host process
msiexec.exe	2192	372 kB	REM-PC/REM	0	64-bit	Microsoft	Windows® installer

Get **SizeOfImage** value of Zloader DLL being loaded by **rundll32.exe/regsrvr32.exe**. Use the **VirtualAllocEx** API function to allocate new memory inside the **msiexec.exe** process:

```
zl_size_of_image = f_zl_retrieve_size_of_image(zl_base_addr);
val_0x8CAE838 = f_zl_xor_arg_with_0xF623385A(0xFEE90362);
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 )
```

Allocate heap memory, copy the entire contents of the DLL into this heap:

```
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);
```

Generate a random number and use it to encrypt the entire payload stored in the 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 );
}

```

Cyber Security Services

Hex View-1 Hex View-2 Hex View-3 Hex View-4

00628AB0	6D 00 73 00 5C 00 46 00	69 00 64 00 64 00 6C 00	m.s.\.F.i.d.d.l.
00628AC0	65 00 72 00 00 00 AB AB	AB AB AB AB AB AB EE FE	e.r...aaaaaaiþ
00628AD0	00 00 00 00 00 00 00 00	3D 3B 15 48 D9 8D 00 1C	.....=;.HÜ...
00628AE0	57 38 8C DA 1B 62 F4 DA	1E 62 F4 DA 1A 62 F4 DA	W8EU.bðU.bðU.bðU
00628AF0	1A 62 F4 DA 1A 62 F4 DA	5A 62 F4 DA 1A 62 F4 DA	.bðU.bðUzbðU.bðU
00628B00	1A 62 F4 DA 1A 62 F4 DA	1A 62 F4 DA 1A 62 F4 DA	.bðU.bðU.bðU.bðU
00628B10	1A 62 F4 DA 1A 62 F4 DA	1A 62 F4 DA 62 62 F4 DA	.bðU.bðU.bðUbbðU
00628B20	14 7D 4E D4 1A D6 FD 17	3B DA F5 96 D7 43 A0 B2	.)N0.öÿ.;üö-xc ²
00628B30	73 11 D4 AA 68 0D 93 A8	7B 0F D4 B9 7B 0C 9A B5	s.0*h."{.0¹{.3µ
00628B40	6E 42 96 BF 3A 10 81 B4	3A 0B 9A FA 5E 2D A7 FA	nB-¿:..":.šü^-\$ü
00628B50	77 0D 90 BF 34 46 F4 DA	4A 27 F4 DA 56 63 F0 DA	w..¿4FðÜJ'ðÜVcðÜ
00628B60	22 85 78 85 1A 62 F4 DA	1A 62 F4 DA FA 62 F6 FB	"...x...bðU.bðUðbðÜ
00628B70	11 63 AA C3 1A 88 F5 DA	1A 42 F4 DA 1A 62 F4 DA	.c#Ä.7ðU.BðU.bðU

UNKNOWN 00628AE0: debug186:unk\_628AE0

encrypted DLL in allocated heap

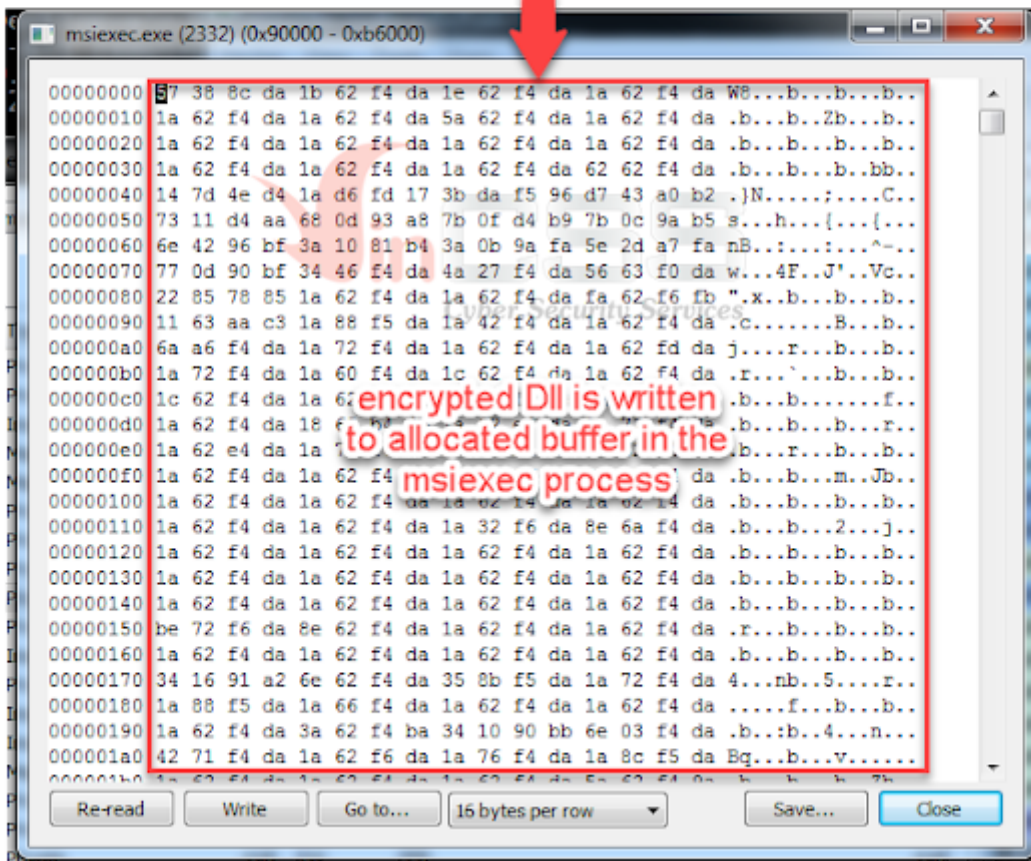
Use the **WriteProcessMemory** API function to write the entire encrypted payload from the heap to the previously allocated memory in the **msiexec.exe** process:



```

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) )
{

```

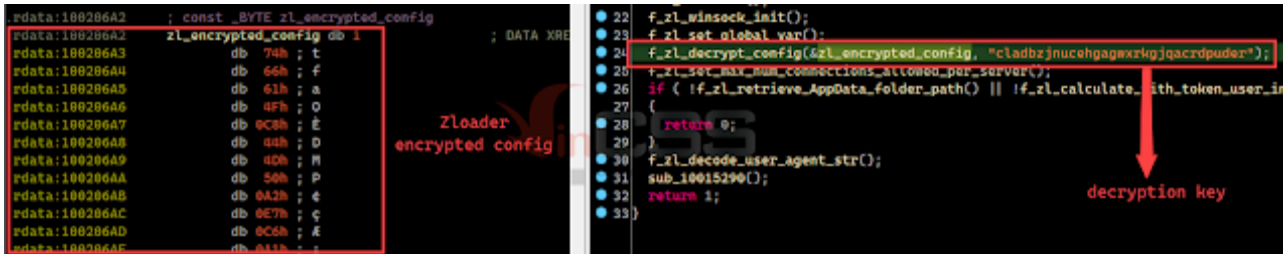


Continue to use the **VirtualAllocEx** API function to allocate a second memory region has size of region are 66 bytes in the **msixec.exe** process. This memory region will be used to decrypt the entire encrypted Dll above. Update the **STARTUPINFO** structure created by the **CreateProcessA** function before, the data here are the assembly code that will be used to decrypt the encrypted Dll. Then, call the **WriteProcessMemory** function to write the updated contents of **STARTUPINFO** to the newly created memory region.



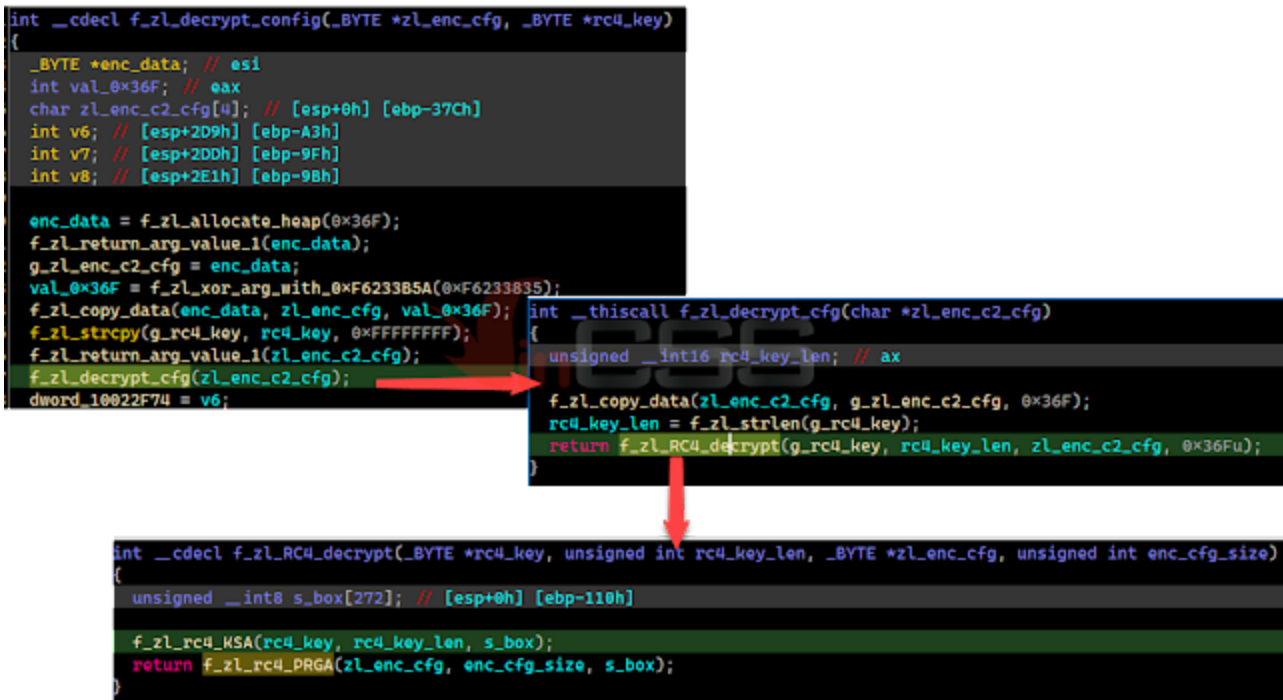
## 9. Decrypt Zloader config

The configuration info of the Zloader has been encrypted and stored in the **.rdata** section. The decrypt function takes two parameters are the encrypted configuration data and the key used to decrypt:



The screenshot shows two parts of the IDA Pro interface. On the left, the **.rdata** section is displayed, showing a list of memory addresses and their contents. A red box highlights the entry for `zloader_encrypted_config` at address `100286A2`. A red arrow points from this entry to the right. On the right, the `f_zl_decrypt_config` function is shown. A red box highlights the call to `f_zl_decrypt_config` with the key `"cladzbjnucehgagwxrlgjqacrdpuder"`. A red arrow points from this key to the text "decryption key" below it.

Inside the function `f_zl_decrypt_config` will use the RC4 algorithm to decrypt the data:



The screenshot shows the implementation of the `f_zl_decrypt_config` function. The function signature is `int __cdecl f_zl_decrypt_config(_BYTE *zl_enc_cfg, _BYTE *rc4_key)`. The function body includes several steps: allocating memory for the encrypted data, copying the key, and calling `f_zl_decrypt_cfg`. A red arrow points from the `f_zl_decrypt_cfg` call to its implementation. The implementation of `f_zl_decrypt_cfg` shows the use of the RC4 algorithm to decrypt the data. A red arrow points from the `f_zl_decrypt_cfg` call to the implementation of `f_zl_decrypt_cfg`.

With the analyzed results, we can use IDAPython code below to perform the decoding:

```

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()

```

Result after executing the script:

```

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://daksjuggdhwq.com/gate.php
    https://dkisuaggdjhna.com/gate.php
    https://eiqwuggejqw.com/gate.php
    https://dquggwjhdmq.com/gate.php
    https://djshggadasj.com/gate.php
[+] Embedded RC4 key: 03d5ae30a0bd934a23b6a7f0756aa504

```

## 10. Collect and save configuration in Registry

When first executed, Zloader will collect information about the victim including **volume\_GUID**, **Computer\_Name**, **Windows version**, **Install Date**, create random folders at **%APPDATA%**, generate a random registry key at **HKEY\_CURRENT\_USER\Software\Microsoft**, then encrypt all relevant information and save it in the created registry:

```
if ( !f_zl_gen_random_reg_key_and_retrieve_val() )
{
    f_zl_wchar_strepy( #sz_zl_dll_path, g_ #sz_zl_dll_path );
    bRet = f_zl_collect_victim_info_create_random_folders_and_store_info_in_registry( #sz_zl_dll_path, 1 );
    f_zl_free_heap_ex( #sz_zl_dll_path );
    v18 = 1;
    if ( bRet )
    {
        goto LABEL_13;
    }
    ExitProcess = f_zl_resolve_api_func( 0, 0x7F96C13u );
    ExitProcess( 0 );
}

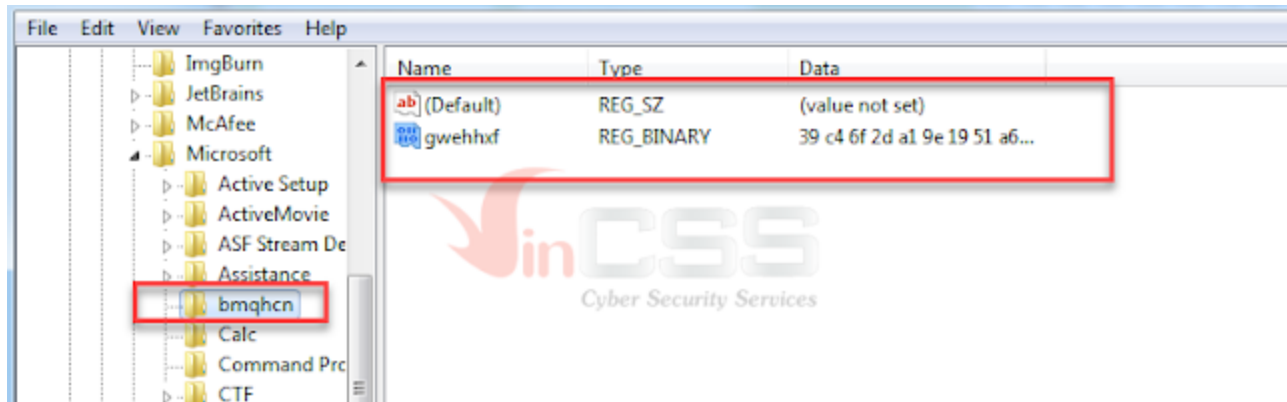
zl_victim_ctx = f_zl_allocate_heap_region( 0x300 );
f_zl_retrieve_original_cli_of_root_drive( #zl_victim_ctx -> pcli );
f_zl_get_victim_system_info( #zl_victim_ctx -> ComputerName_VersionInfo_InstallDate );
f_zl_gen_random_wstring( 2, #zl_victim_ctx -> rand_reg_key, 4u, 8u ); // ex: Iplfq
f_zl_rc4_hsa_for_embedded_key( #zl_victim_ctx -> rc4_sbox_embedded_key );

// create 12 random folders
f_zl_create_rand_directory( #sv92, #szAppDataPath, #sz_dll, 0 );
f_zl_ctor_struct( #sv99 );
f_zl_create_rand_directory( #sv79, #szAppDataPath, 0, 0 );
f_zl_ctor_struct( #sv80 );
f_zl_create_rand_directory( #sv80, #szAppDataPath, 0, 0 );
f_zl_ctor_struct( #sv81 );
f_zl_create_rand_directory( #sv81, #szAppDataPath, 0, 0 );
f_zl_ctor_struct( #sv82 );
f_zl_create_rand_directory( #sv82, #szAppDataPath, 0, 0 );
f_zl_ctor_struct( #sv83 );
f_zl_create_rand_directory( #sv83, #szAppDataPath, 0, 1 );
f_zl_ctor_struct( #sv84 );
f_zl_create_rand_directory( #sv84, #szAppDataPath, 0, 1 );
f_zl_ctor_struct( #sv85 );
f_zl_create_rand_directory( #sv85, #szAppDataPath, 0, 0 );
f_zl_ctor_struct( #sv86 );
f_zl_create_rand_directory( #sv86, #szAppDataPath, 0, 0 );
f_zl_ctor_struct( #sv87 );
f_zl_create_rand_directory( #sv87, #szAppDataPath, 0, 0 );
f_zl_ctor_struct( #sv88 );
f_zl_create_rand_directory( #sv88, #szAppDataPath, 0, 0 );
f_zl_ctor_struct( #sv89 );
f_zl_create_rand_directory( #sv89, #szAppDataPath, 0, 0 );

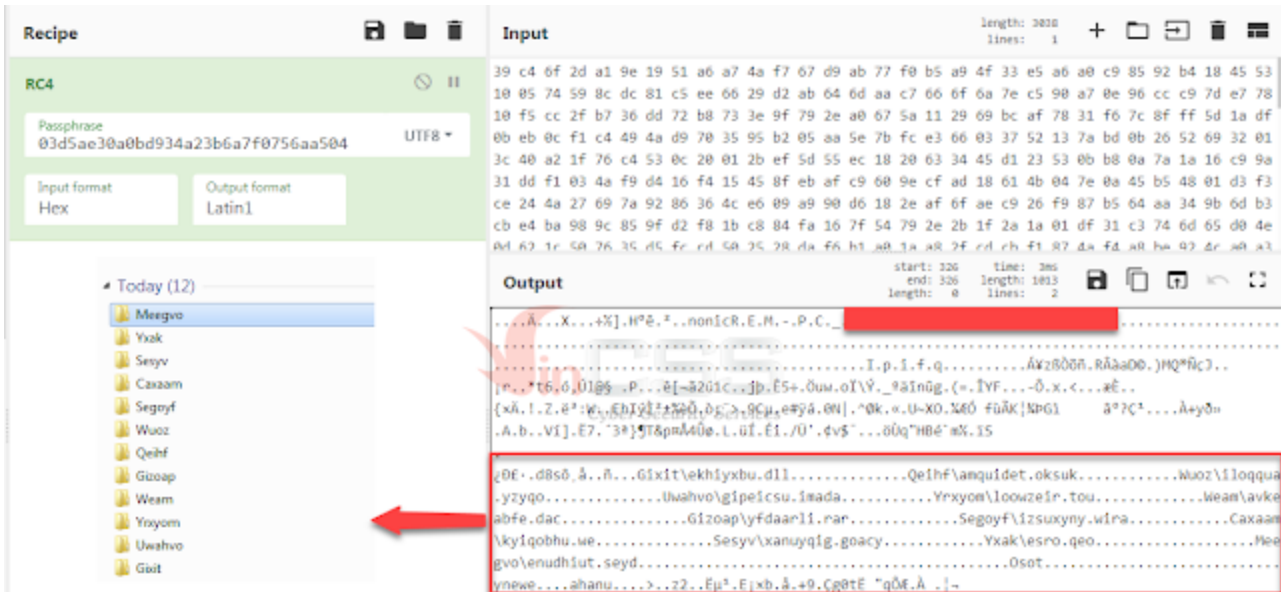
if ( !f_zl_encrypt_data_create_random_registry_and_set_registry_value( #zl_victim_ctx ) )
{
    LABEL_24:
    bRet = 0;
}

```

The information stored in the registry is similar to the following:



To decrypt the data stored in the above Registry, use the decoded embedded RC4 key above. With the support of **CyberChef**, we can easily decrypt data as follows below:

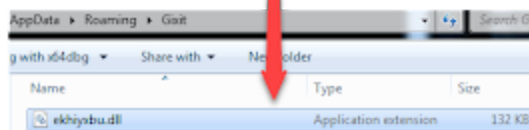


## 11. Persistence technique

Zloader reads the entire contents of the core DLL from disk into the memory region, then writes to a random dll in a directory created above at **%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(&z1_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(&z1_cloned_payload);
ptr_z1_cloned_payload = f_zl_return_buf(&z1_cloned_payload);
// ex: C:\Users\REM\AppData\Roaming\Gixit\ekhiyxbu.dll
wsz_random_dll_path = f_zl_return_struc_value(&ptr_random_dll_path);
f_zl_create_file(wsz_random_dll_path, wsz_random_dll_path, ptr_z1_cloned_payload, payload_size);
```



Create persistence key at **HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run**:

```
if ( f_zl_set_persistence_at_run_key_wrap() )  
{
```

```
int __usercall f_zl_set_persistence_at_run_key_wrap@<ebx>()  
{  
    wchar_t *wsz_RunKey; // eax  
    wchar_t *wsz_dll_extension; // eax  
    _WORD *ptr_extension_pos; // eax  
    _WORD wszFilePath[260]; // [esp+2h] [ebp-37Ah]  
    wchar_t decString[61]; // [esp+20Ah] [ebp-172h]  
    WCHAR wsz_reg_value_name[100]; // [esp+284h] [ebp-F8h]  
    wchar_t arg2[24]; // [esp+34Ch] [ebp-30h]  
  
    f_zl_return_value_name(wsz_reg_value_name);  
    // Software\Microsoft\Windows\CurrentVersion\Run  
    wsz_RunKey = f_zl_decrypt_wstring(word_B04F0, decString);  
    if ( !f_zl_retrieve_type_and_data_of_reg_value_2(HKEY_CURRENT_USER, wsz_RunKey, wsz_reg_value_name) )  
    {  
        // ex: return C:\Users\REM\AppData\Roaming\Gixit\ekhlyxbu.dll  
        if ( f_zl_decrypt_cfg_in_registry_and_build_file_path(2, wszFilePath) )  
        {  
            wsz_dll_extension = f_zl_decrypt_wstring(word_B0ACC, arg2);  
            ptr_extension_pos = f_zl_check_file_extension(wszFilePath, wsz_dll_extension);  
            f_zl_set_persistence_at_run_key_ex(wszFilePath, wsz_reg_value_name, ptr_extension_pos != 0);  
        }  
    }  
}
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
lpifq	REG_SZ	regsvr32.exe /s C:\Users\REM\AppData\Roaming\Gixit\ekhlyxbu.dll

## 12. References

Click [here](#) for Vietnamese version.

Tran Trung Kien (aka m4n0w4r)

Malware Analysis Expert

R&D Center - VinCSS (a member of Vingroup)