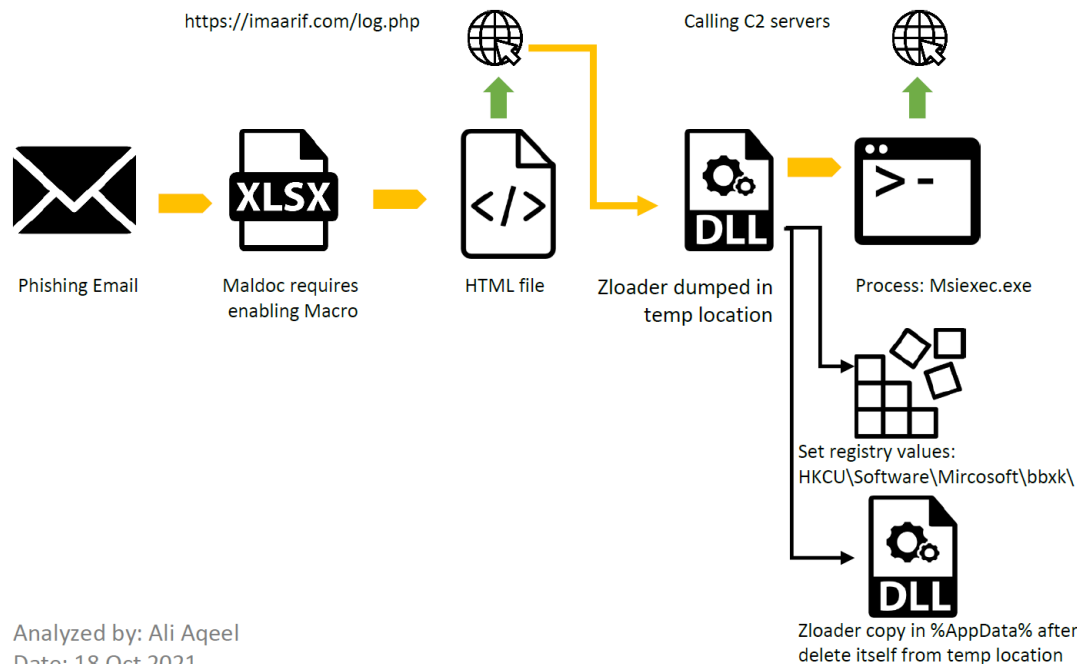


Zloader Reversing

aaqeel01.wordpress.com/2021/10/18/zloader-reversing/

Ali Aqeel

October 18, 2021



Aka: ZeusLoader, Deloader, Terdot, Zbot is a malware family that downloads Zeus OpenSSL. Parts of the source code of Zeus were leaked back in 2010 [1] and since couple of versions been forked. Each of the version has its malicious capabilities, but all in common do info stealing specially banking information. Zeus in its core does wild stuff from stealing HTTPS session before being encrypted; to split stolen data and send it in multiple channels over different C2 server based on the stolen info-type [2]. The sent data is being encrypted using RC4 algorithm. Given that major parts of the Zeus being well known and very detectable by almost every AV; Zloader is not just a loader/packer to Zeus core functionality. There are some complicated obfuscation techniques and visual encryption implemented on every single unpacked version of Zloader that bypass security and difficulty extracting configuration. Uncommon attack vector like using Google AdSense has been observed lately [3] also attacker signs Zloader with a certificate compromised from legitimate software in order to evade detection. In this post, we gonna take a look of common Zloader 123 botnet attack that uses maldoc vector. Quickly analyze maldoc, downloader, and the well known unpacking technique with observed behavior which simple and not quite interesting. However, the second part is going to be deep dive into analyzing and reversing techniques of Zloader unpacked version.

Maldoc

SHA256 500856ee3fc13326cad564894a0423e0583154ef10531de4ab6e6d5df90d4e31

File Type Office Open XML Spreadsheet

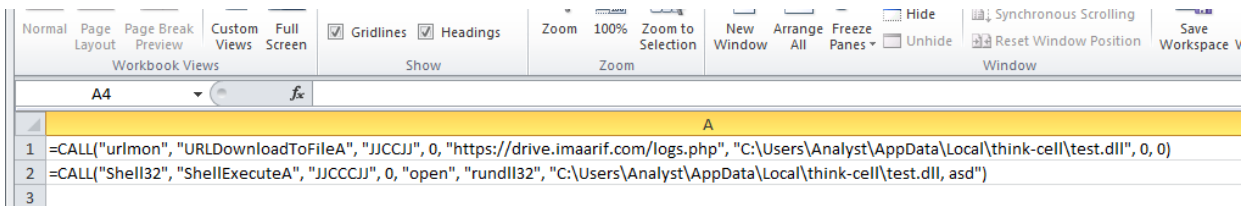
Name tn4598151.xlsm

Size 182.62 KB (187002 bytes)

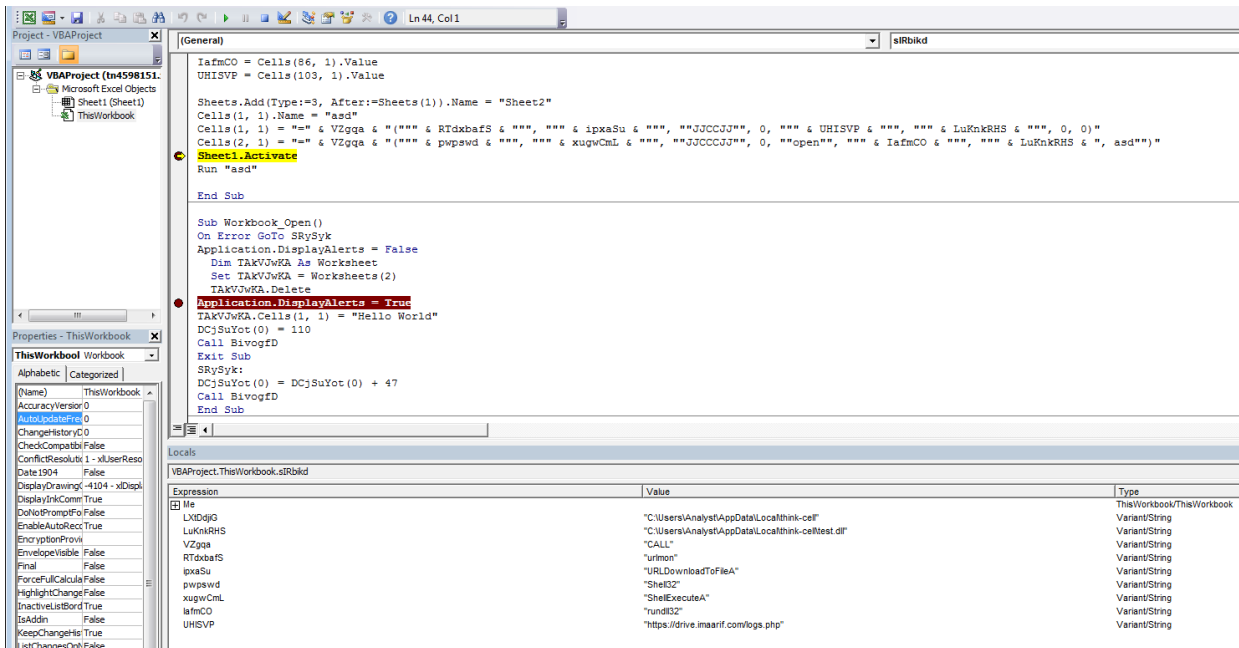
Creation Time 2021-10-04 13:17:51

Links [MalwareBazaar](#), [VirusTotal](#), [Any.run](#)

In clear text on sheet2, the maldoc give away downloader URL, directory where it's been dropped, and shell command to run a Dll which is the Zloader.



Enable macro is required to run the above in VBA script.



Downloaded *test.dll* is just an HTML! that downloads *logs.php* which is a Zloader Dll file!

Static Discovering
logs.php

Indicators:

Text report

logs.php

▲ Saved response data
🔍 Look up on [VirusTotal](#)

Mime: text/html
Size: 48.22 Kb

TrID - File Identifier	Hashes
100% HyperText Markup Language	MD5 <input type="checkbox"/> 476CDFD678724C6E86861345675FD37 SHA1 <input type="checkbox"/> 5A8C9595888A12A19386F8AEB091CC487995833 SHA256 <input type="checkbox"/> B134A592FF8DE6C5B4AD76F9C3609E56BA3CC924E1F2F1EEFAC0CC2C0F30ACA5 SSDEEP <input type="checkbox"/> 768 :HEV0sbUGU9WBHI1JcQG0RTU2pQIsAqG1UFQhbrHec :cofQU9WhGG0RG2ahdG1prHec

PREVIEW **HEX**

```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-CY"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="FrA16uECGd1RXgaUsPdB6A==">(function(){window.google={kEI:'_W1bYaLeC6HTz7sPgeal-Ak',kEXPI:'0,1302536,56873,1709,4349,207,4804,2316,383,246,5,1354,5251,1122515,1197753,648,328866,51223,16115,17444,1954,9286,17572,4858,1362,9290,3029,17580,4020,978,13228,3847,4192,6430,7432,11612,2778,919,5081,887,706,1279,2212,530,149,1103,840,6297,3514,606,2023,1777,520,14670,3227,2845,7,5599,6755,5096,15767,553,908,2,941,2614,13142,3,346,230,1014,1,5444,149,11323,2652,4,1528,2304,1236,5226,2634,2626,2015,18375,2658,4243,3114,30,11412,2216,2305,638,1494,12852,3934,2521,3276,2560,4094,17,3121,6,908,3,3541,1,14710,1815,282,38,874,5992,14659,672,2,1395,1715,2,15953,784,255,2870,1680,744,2350,3502,8873,1590,1160,6700,2378,2720,4546,3,5473,4149,486,230,3374,2,1,6,176,2,807,6768,4569,2577,369,2755,551,2075,401,1518,3,23,2,1,3,2612,267,430,1,104,2,1,2396,665,2,3179,5,603,4,2135,1142,229,600,114,117,893,2909,2073,757,92,3063,1589,40,503,302,151,274,393,375,78,57,192,686,837,113,860,98,963,674,515,229,926,69,1572,5,918,271,519,113,630,139,183,2,83,95,148,2,696,324,4,274,399,8,2,90,581,53,77,1,230,1072,185,403,2,465,468,36,220,912,110,23,6,489,407,467,319,211,4,319,206,627,351,5559791,3904,190,256,161,1802728,95,4193885,2800696,882,444,1,2,80,1,1796,1,9,2553,1,748,141,795,563,1,4265,1,1,2,1331,4142,2609,155,17,13,72,139,4,2,20,2,169,13,19,46,5,39,96,548,29,2,2,1,2,1,2,2,74,1,2,2,2,2,2,2,353,513,186,1,1,158,3,2,2,2,2,2,4,2,3,3,269,551,13,4,1,3,5,3,98,3,12,38,9,1,8,11,45,2,8,10,2,1,2,23953892,4041352,277,61,3,2414,1008,483,9,1434,1516,293,1532,889,851,1568,805559',kBL:'sQm0'};google.sn='webhp';google.kHL='en-CY';})();(function(){var f=this||self;var h,k=[];function l(a){for(var b;a&&!a.getAttribute||!b=a.getAttribute("eid"))a=a.parentNode;return b||h}function m(a){for(var b=null;a&&!a.getAttribute||
```

Dll Zloader

SHA256 c4ab81d7b7d44dd6dfc4f2b69dbe3f22fbf23c1ae49ab8edac2d26f85ae4514d

File Type Win32 DLL

Names suqyatda.dll, ewviv.dll, ehv.dll, cyvi.dll, logs.php

Size 1.13 MB (1189888 bytes)

Compiler Time-stamp Mon Sep 23 01:29:14 2019

First Submission 2021-10-04 18:23:00

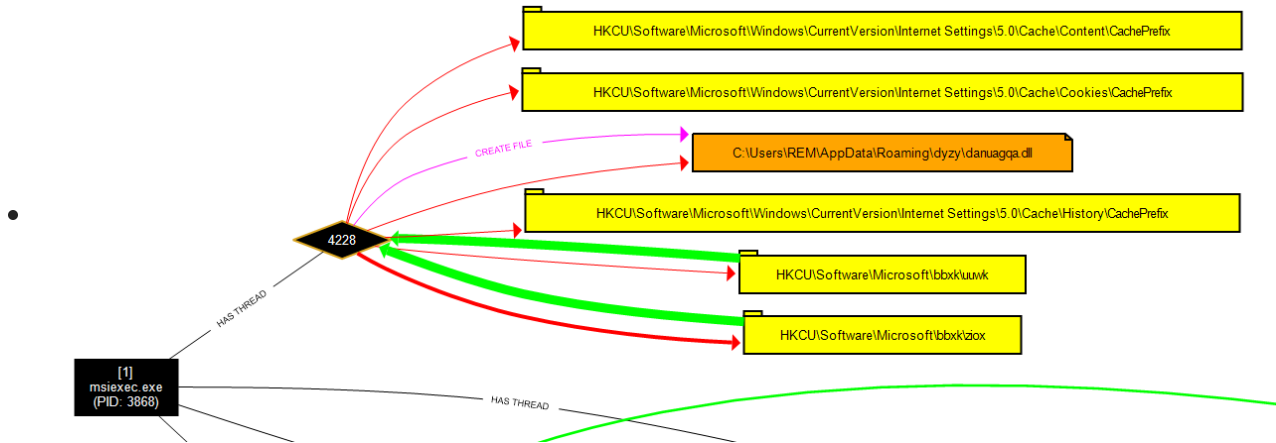
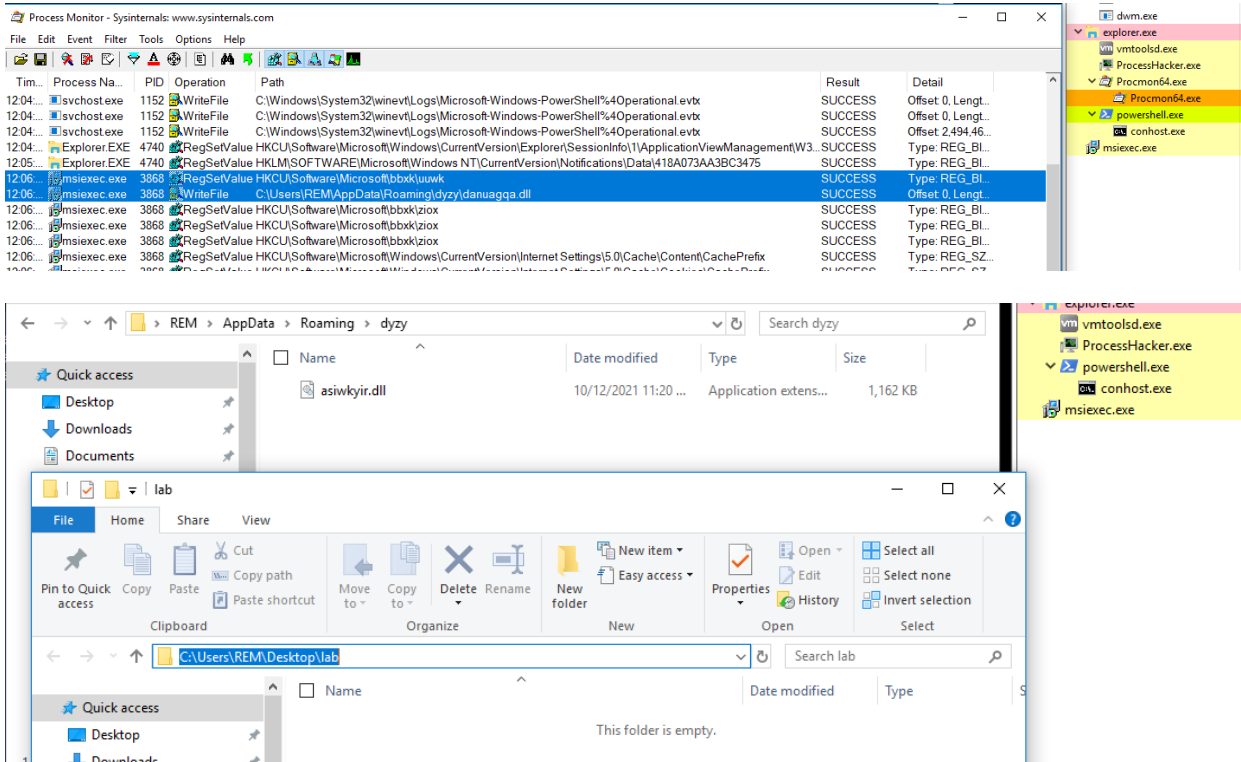
Links [MalwareBazaar](#), [VirusTotal](#), [Tria.ge](#)

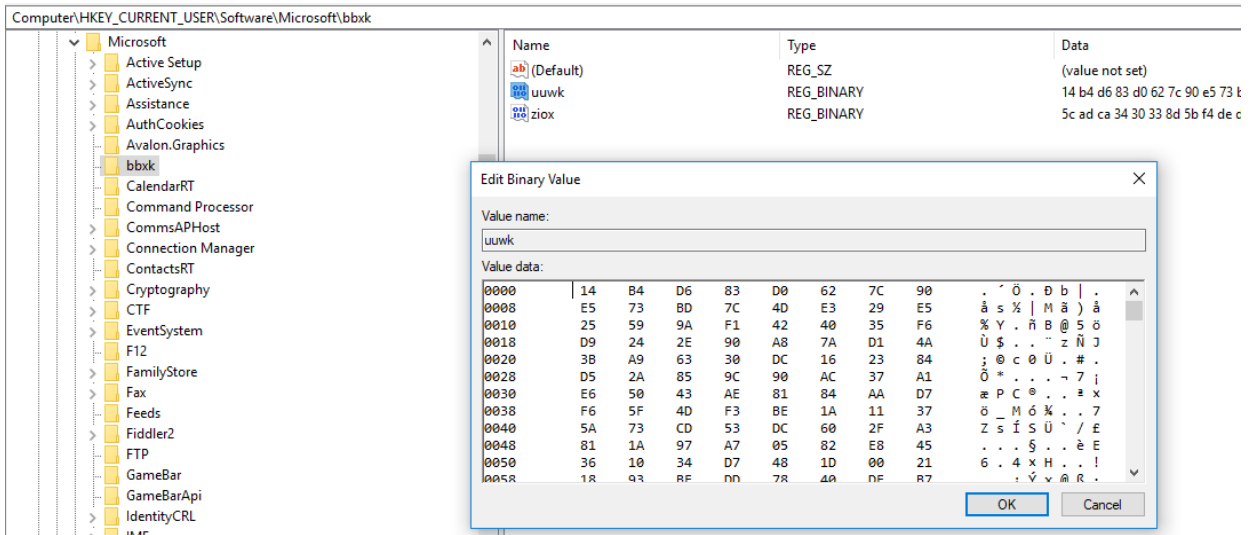
Zloader Dll file is been downloaded and runned in temp location. After Zloader runs:

1. Create new process *msiexec.exe* and inject its loader in it.
2. Loader sets new registry values using random hive and key names in:
 - o HKCU\Software\Mircosoft\bxbk\uuwk
 - o HKCU\Software\Mircosoft\bxbk\xioz

3. Deletes original downloader and copy itself to %AppData%\Roaming*random name*.dll

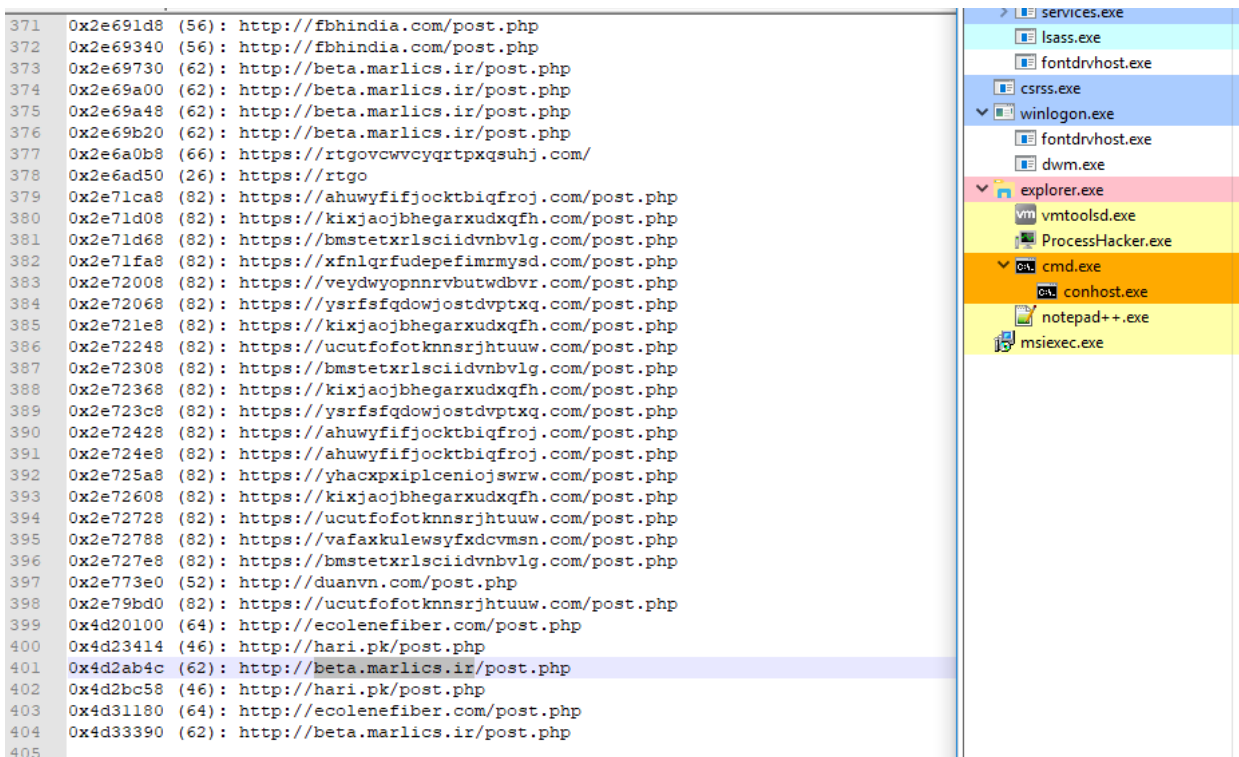
The registry value calls the new directory for persistent in case of host rebooted. Both registry values are encrypted with RC4 but more to that in next section





Running Process and Registry Value change

When checking memory strings for any forensics it spells out great number of C2 values. Noticed that 20 URLs has random name with fixed length. Those are called Domain Generated Algorithm DGA, unlike hardcoded C2 URLs those are queried during running. More to that later in next section.



Using *pe-sieve64* tool is good way to dump the unpacked Zloader from the running process which is valid PE file to be analyzed. However, just a quick debugging would give same result. In SquirrelWaffle and QakBot recent analysis [4] [5] it's been observed that Zloader among other malwares are using same crypters/decryptor for unpacking mechanism for their loaders before injecting them in process. Following the same debugging method in [4] would reveal the packed Zloader.

File View Debug Trace Plugins Favourites Options Help Mar 4 2018

CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Snowman Handles

00FD02EA 88 5D 10 mov ebx,dword ptr ss:[ebp+10]
 00FD02ED 66 01 DA add dx,bx
 00FD02F0 66 F7 DA neg dx
 00FD02F3 68 D2 03 imul edx,edx,3
 00FD02F6 C1 CA 06 ror edx,6
 00FD02F9 89 55 10 mov dword ptr ss:[ebp+10],edx
 00FD02FC 30 10 xor byte ptr ds:[eax],dl
 00FD02FE 40 inc eax
 00FD02FF E2 E2 jmp FD02E3
 00FD0302 C2 0C 00 ret c
 00FD0305 83 EC 10 sub esp,10
 00FD0308 53 push ebx
 00FD0309 55 push ebp
 00FD030A 56 push esi
 00FD030B 8B 74 24 24 mov esi,dword ptr ss:[esp+24]
 00FD030F 30 C9 xor cl,cl
 00FD0311 31 ED xor ebp,ebp
 00FD0313 31 DB xor ebx,ebx
 00FD0315 30 C0 xor al,al
 00FD0317 85 F6 test esi,esi
 00FD0319 57 push edi
 00FD031A 8B 4C 24 13 mov byte ptr ss:[esp+13],cl
 00FD031E 0F 86 98 00 00 00 jbe FD038F
 00FD0324 EB 04 24 28 jmp FD032A
 00FD0326 8B 74 24 28 mov esi,dword ptr ss:[esp+28]
 00FD032A 84 C9 test cl,cl
 00FD032C 74 2F jle FD0350
 00FD032E 0F B6 4C 24 13 movzx ecx,byte ptr ss:[esp+13]
 00FD0330 8B 74 24 30 mov edx,dword ptr ss:[esp+30]
 00FD0332 01 D1 add ecx,edx
 00FD0333 8B 79 01 00 cmp byte ptr ds:[ecx+1],0
 00FD033D 8A 11 mov dl,byte ptr ds:[ecx]
 00FD033F 75 0E jne FD034E
 00FD0341 0F B6 CA movzx ecx,dl
 00FD0344 01 CB add ebx,ecx
 00FD0346 C6 44 24 13 00 mov byte ptr ss:[esp+13],0
 00FD0348 30 C9 xor cl,cl
 00FD034D EB 66 jmp FD0385
 00FD034F 80 44 24 13 01 add byte ptr ss:[esp+13],1
 00FD0354 0F B6 CA movzx ecx,dl
 00FD0357 01 CB add ebx,ecx

eax: "" "" "" ""
 eax: "" "" "" ""

Decryper loop

Hide FPU
 EAX 04F95C14 "" "" ""
 EBX C4115104 "" "" ""
 ECX 00000000 "" "" ""
 EDX A1631607 "" "" ""
 EBP 00FAF68C "" "" ""
 ESP 00FAF68C "" "" ""
 ESI 00000001 "" "" ""
 EDI 00FE0000 "" "" ""
 EIP 00FD0301 "" "" ""
 EFLAGS 00000304 "" "" ""
 ZF 0 PF 1 AF 0 "" "" ""
 OF 0 SF 0 DF 0 "" "" ""
 CF 0 TF 1 IF 1 "" "" ""
 LastError 00000057 (ERR) "" "" ""
 LastStatus C0070057 "" "" ""
 GS 002B FS 0053 "" "" ""
 ES 002B DS 002B "" "" ""
 CS 002B SS 002B "" "" ""
 X87r0 0000000000000000 "" "" ""
 X87r1 0000000000000000 "" "" ""
 X87r2 0000000000000000 "" "" ""
 X87r3 0000000000000000 "" "" ""
 X87r4 0000000000000000 "" "" ""
 X87r5 0000000000000000 "" "" ""
 X87r6 0000000000000000 "" "" ""
 X87r7 0000000000000000 "" "" ""
 X87TagWord FFFF "" "" ""
 X87TW_0_3 (Empty) X87 "" "" ""
 X87TW_2_3 (Empty) X87 "" "" ""
 Default (stocal)
 1: [esp+4] 00FD0797 "" "" ""
 2: [esp+8] 04F80000 "" "" ""
 3: [esp+C] 00015C14 "" "" ""
 4: [esp+10] A1631607 "" "" ""
 5: [esp+14] 00FE0000 "" "" ""

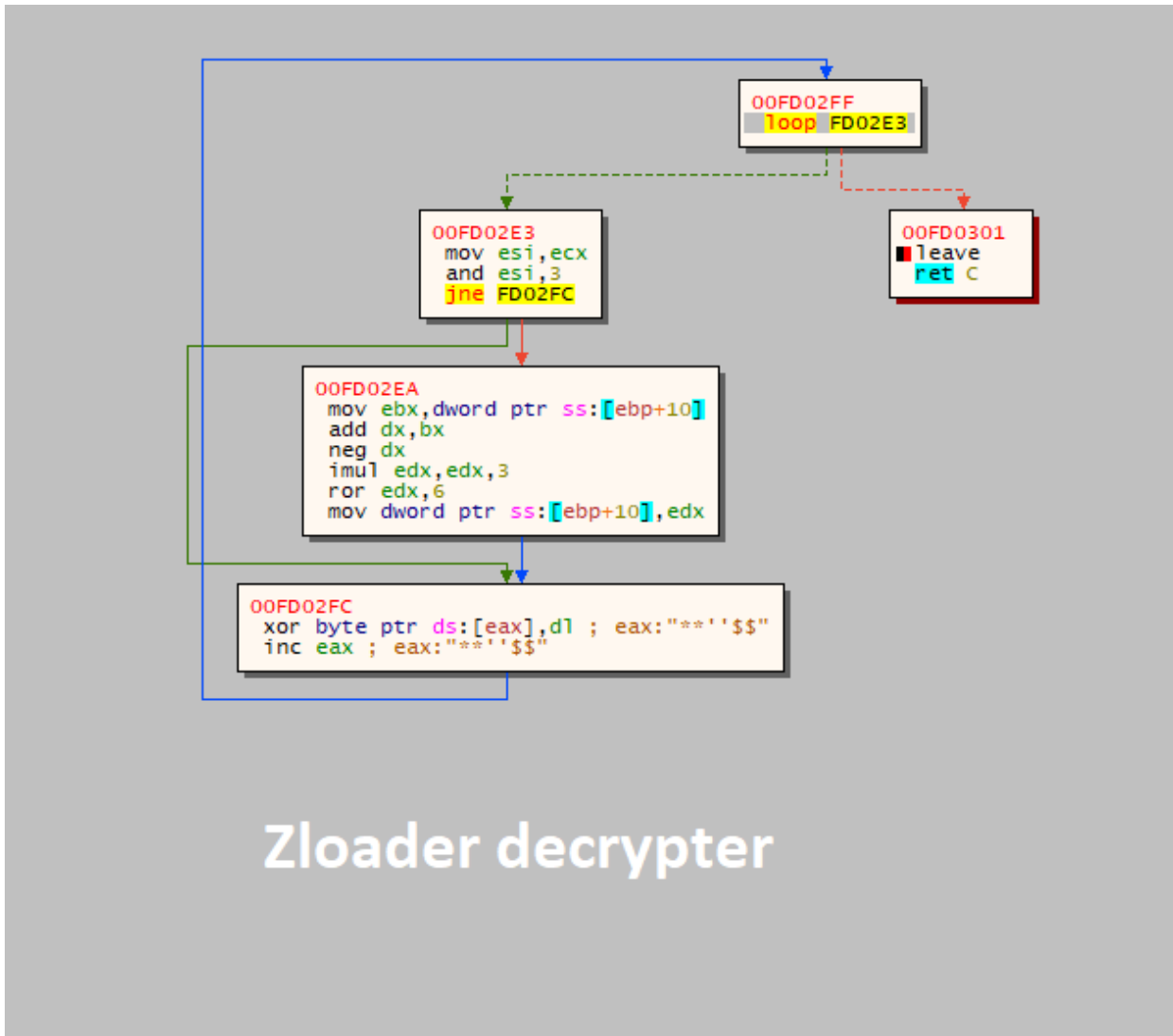
00FD0301

Address Hex ASCII
 04F80000 4D 38 5A 90 38 03 66 02 04 09 71 FF 81 B8 C2 91 M&Z.8.f...qy..A.
 04F80010 01 40 C2 15 C6 C8 09 1C 0E 1F 8A F8 00 84 09 CD .0A.AE...?b. i
 04F80020 21 88 01 4C CD 0A 54 68 89 73 20 0E 70 72 6F 67 i..Lk.This -p00
 04F80030 67 61 6D 87 63 47 6E 1F 4F 74 E7 62 65 AF CF 75 gam.cgn.Otçbe Iu
 04F80040 5F 98 69 06 44 4F 7E 53 03 6D 6F 64 65 2E 0D 89 .1.D0-5;mode...
 04F80050 0A 24 4C 44 CC 01 98 15 EE 88 79 78 8D 58 04 ED .LOI...i u{Xk.
 04F80060 0A 9F 7A 8C 98 0C 8C 7C 53 DB 14 30 89 09 22 A7 ..Zk.WjSÜ.0."\$
 04F80070 79 50 8C 08 52 14 69 63 68 14 42 8C A1 50 45 01 YP&R.1ch.B.iPE.
 04F80080 4C 01 40 C6 C0 D2 EE 56 60 14 38 E0 05 02 21 08 L.0&MIV".SA..i.
 04F80090 01 1C 9F 1E 12 32 08 5C 90 14 D0 3D CC 08 10 092...D=i...
 04F800A0 30 D1 28 0A 41 0C 94 17 06 34 D1 08 9C 19 A0 28 ON(A...4N...+
 04F800B0 04 19 41 20 40 01 A1 68 38 23 08 84 0F 3C 80 FC .A.B.k&#. .x.u
 04F800C0 64 49 60 C8 BE 3C 8C 90 40 80 70 08 AD 01 52 22 .E-.0'p..R"
 04F800D0 A8 70 01 B5 1F 03 2E 74 65 78 9C 42 E8 10 02 B1 p.u...tex.Bè..z
 04F800E0 91 12 4E 88 46 01 30 20 06 60 2E 72 1C 64 64 74 .N.F.0...dat
 04F800F0 81 12 A4 22 5D FC 48 81 16 F1 4F 28 32 40 07 2E a."Um..Ro(20..
 04F80100 93 27 81 48 28 89 08 60 02 1B 1C 82 3A BE 28 60 .'.H+...:X(

00FAF68C 00FAF6DC
 00FAF694 00FD0797 return to 00FD0797 from 0
 00FAF698 04F80000
 00FAF69C 00015C14
 00FAF6A0 A1631607
 00FAF6A4 00FE0000 "" "" ""
 00FAF6A8 00FD0060 "" "" ""
 00FAF6AC 00FAF6DC "" "" ""
 00FAF6B0 00FAF6C0 00FAF6C0
 00FAF6B4 00C49791 "" "" ""
 00FAF6B8 00000000 "" "" ""
 00FAF6BC 6F1C0000 "" "" ""
 00FAF6C0 00000004 "" "" ""
 00FAF6C4 AC4A32F9 "" "" ""
 00FAF6C8 00000004 "" "" ""
 00FAF6CC 706E3370 L"C:\WINDOWS\system32"
 00FAF6D0 0000000F "" "" ""
 00FAF6D4 00000007 "" "" ""
 00FAF6D8 00000040 "" "" ""

Value from EAX register

Comment:



Unpacked Zloader

SHA256

3A4CA58B0A2E72A264466A240C6636F62B8742FFBC96CE14E2225F0E57012E96

File Type Win32 DLL

Name unpacked_zloader_21_10_4.dll,

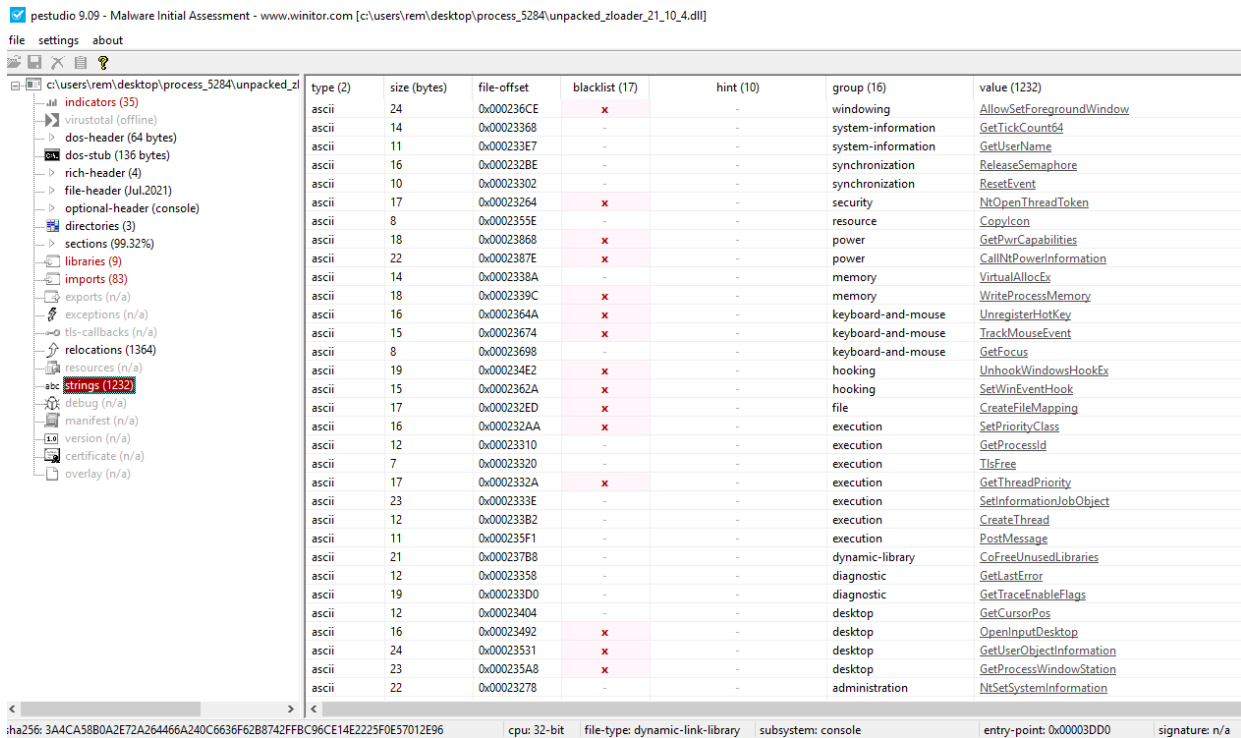
Size 146.00 KB (149504 bytes)

Compiler Time-stamp Wed Jul 14 08:04:16 2021

First Submission 2021-10-18 15:32:37

Links [MalwareBazaar](#), [VirusTotal](#), [Tria.ge](#)

The unpacked Zloader is a master piece of obfuscated functions that waste lots of analysis time to dig into. API strings among other static indicators would not be a good clue for analyzing Zloader. Beside, this malware family is known for API hashing, Visual Encryption using XOR, and RC4 encryption to encrypt strings.



There are five main topics we are going to discuss in this section when reversing Zloader: API hashing, XORing string, extracting Configuration, DGA routine, and Zeus function.

API Hashing:

Statically analyzing Zloader is a bit of a challenge. However, with a new amazing IDA plugin called *HashDB* from OpenAnalysis Labs [6] it's amazing how much obfuscated strings get out the way when reversing Zloader. Just to show a case of what HashDB can do before and after shots of hashed values in a random function. The hashes been checked among large database of hashes with good prediction of hashing algorithms been used.

IDA View-A, Pseudocode-A

Strings

Hex View-1

Structures

Enums

Imports

IDA View-A

```

; Attributes: bp-based frame
; int_cdecl sub_171E70(int)
sub_171E70 proc near

var_28= dword ptr -28h
var_1c= dword ptr -1Ch
arg_0= dword ptr 8

push ebp
mov ebp, esp
push ebx
push edi
push esi
sub esp, 1Ch
push 6AA0E84h
push 0
call sub_17C3F0
add esp, 8
push 0
push 4
call eax
cmp eax, 0FFFFFFFh
jz short loc_171F03

```

100.00% (-22,22) (476,419) 00011279 00171E79: sub_171E70+ (Synchronized with Ps

IDA View-A

```

mov esi, eax
mov [ebp+var_28], 1Ch
push 0C75BEA4h
push 0
call sub_17C3F0
add esp, 8
lea ebx, [ebp+var_28]
push ebx
push esi
call eax
xor edi, edi
test eax, eax
jz short loc_171EEF

```

100.00% (-22,22) (476,419) 00011279 00171E79: sub_171E70+ (Synchronized with Ps

Pseudocode-A

```

1 int_cdecl sub_171E70(int a1)
2 {
3   int (__stdcall *v1)(int, _DWORD); // eax
4   int v2; // eax
5   int v3; // esi
6   int (__stdcall *v4)(int, int *); // eax
7   int v5; // edi
8   int (__stdcall *v6)(int, int *); // eax
9   void (__stdcall *v7)(int); // eax
10  int v9[10]; // [esp+0h] [ebp-28h] BYREF
11
12  v1 = (int (__stdcall *) (int, _DWORD))sub_17C3F0(0, 111808132);
13  v2 = v1(4, 0);
14  if ( v2 == -1 )
15    return -1;
16  v3 = v2;
17  v9[0] = 28;
18  v4 = (int (__stdcall *) (int, int *))sub_17C3F0(0, 209043108);
19  v5 = 0;
20  if ( v4(v3, v9) )
21  {
22    do
23    {
24      v5 -= sub_165C80(0, v9[3] == a1);
25      v6 = (int (__stdcall *) (int, int *))sub_17C3F0(0, 13158276);
26    }
27    while ( v6(v3, v9) );
28  }
29  v7 = (void (__stdcall *) (int))sub_17C3F0(0, 193887669);
30  v7(v3);
31  return v5;
32 }

```

00011279 sub_171E70:12 (171E79) (Synchronized with IDA View-A)

IDA View-A, Pseudocode-A

Strings

Hex View-1

Structures

Enums

Imports

IDA View-A

```

; Attributes: bp-based frame
; int_cdecl sub_171E70(int)
sub_171E70 proc near

var_28= dword ptr -28h
var_1c= dword ptr -1Ch
arg_0= dword ptr 8

push ebp
mov ebp, esp
push ebx
push edi
push esi
sub esp, 1Ch
push 6AA0E84h
push 0
call sub_17C3F0
add esp, 8
push 4
call eax
cmp eax, 0FFFFFFFh
jz short loc_171F03

```

100.00% (-56,18) (646,399) 0001129C 00171E9C: sub_171E70+ (Synchronized with Ps

IDA View-A

```

mov esi, eax
mov [ebp+var_28], 1Ch
push 0C75BEA4h
push 0
call sub_17C3F0
add esp, 8
lea ebx, [ebp+var_28]
push ebx
push esi
call eax
xor edi, edi
test eax, eax
jz short loc_171EEF

```

100.00% (-56,18) (646,399) 0001129C 00171E9C: sub_171E70+ (Synchronized with Ps

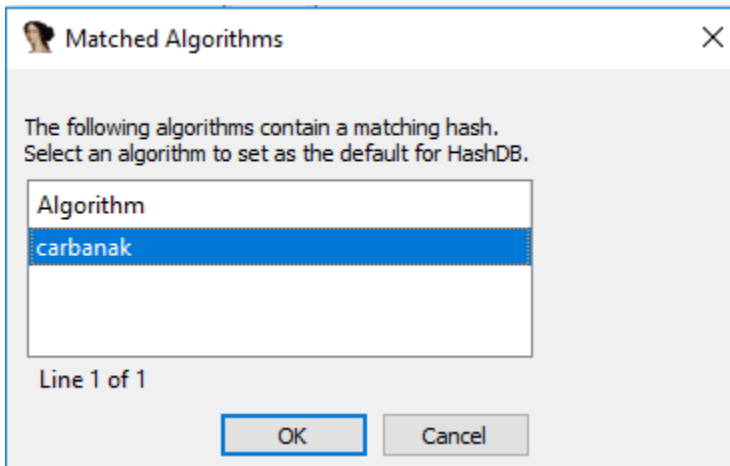
Pseudocode-A

```

1 int_cdecl sub_171E70(int a1)
2 {
3   int (__stdcall *v1)(int, _DWORD); // eax
4   int v2; // eax
5   int v3; // esi
6   int (__stdcall *v4)(int, int *); // eax
7   int v5; // edi
8   int (__stdcall *v6)(int, int *); // eax
9   void (__stdcall *v7)(int); // eax
10  int v9[10]; // [esp+0h] [ebp-28h] BYREF
11
12  v1 = (int (__stdcall *) (int, _DWORD))sub_17C3F0(0, createtoolhelp32snapshot);
13  v2 = v1(4, 0);
14  if ( v2 == -1 )
15    return -1;
16  v3 = v2;
17  v9[0] = 28;
18  v4 = (int (__stdcall *) (int, int *))sub_17C3F0(0, thread32first);
19  v5 = 0;
20  if ( v4(v3, v9) )
21  {
22    do
23    {
24      v5 -= sub_165C80(0, v9[3] == a1);
25      v6 = (int (__stdcall *) (int, int *))sub_17C3F0(0, thread32next);
26    }
27    while ( v6(v3, v9) );
28  }
29  v7 = (void (__stdcall *) (int))sub_17C3F0(0, closehandle);
30  v7(v3);
31  return v5;
32 }

```

0001129C sub_171E70:18 (171E9C) (Synchronized with IDA View-A)



XORing Strings:

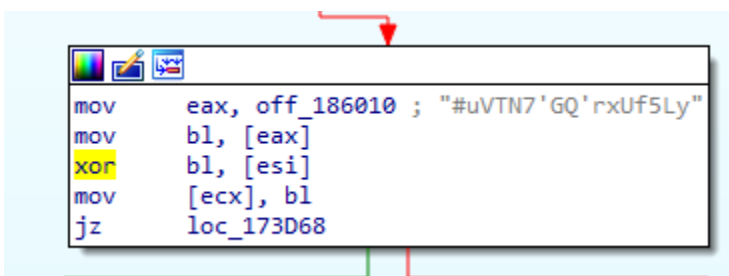
With API hashing out of the way. It's important to get reversing tricks to dig into the main functions and extract configurations. There're very limited hardcoded strings in Zloader that can be clues like those.

```

.text:00161ED9      call     sub_1741A0
.text:00161EDE      call     sub_170ED0
.text:00161EE3      push    offset aQhpacozsstaznu ; "qhpacozsstaznupphhedjtuoww"
.text:00161EE8      push    offset unk_184404
.text:00161EED      call     sub_1656B0
.text:00165911      cmp     [ebp+arg_4], 0
.text:00165915      jz      loc_165A46
.text:0016591B      mov     eax, off_186010 ; "#uVTN7'GQ'rxUf5Ly"
.text:00165920      movzx  ebx, word ptr [esi]
.text:00165923      movsx  esi, byte ptr [eax]
.text:00165926      mov     dword ptr [ebp+var_10], ebx

```

First, let's look at **Off_186010** which is an offset of an offset of a memory location **rdata:00183D80** with literal string (#uVTN7'GQ'rxUf5Ly). When cross referencing this offset it's been used 4 times in two different functions. And there's some sort of XOR function in both subroutines which reveal this is the key literal string could be an XOR key value.



```

loc_165990:
push  0A95168F5h
push  edi
call  sub_161DE0
add   esp, 8
lea   edi, [eax+56AE970Ch]
mov   esi, eax
mov   edx, 0F0F0F0F1h
mov   ecx, off_186010 ; "#uVTN7'GQ'rxUf5Ly"
mov   eax, edi
mul   edx
shr   edx, 4
mov   eax, edx
shl   eax, 4
add   eax, edx
neg   eax
lea   eax, [esi+eax+56AE970Ch]
movsx eax, byte ptr [ecx+eax]
mov   ecx, [ebp+arg_0]
movzx ecx, word ptr [ecx+esi*2-52A2D1E8h]
mov   ebx, ecx
xor   ebx, eax
push  eax
push  ecx
call  sub_161D70
add   esp, 8
mov   ecx, [ebp+arg_4]
test  bx, bx
mov   [ecx+esi*2-52A2D1E8h], bx
jz    short loc_165A48

```

```

if ( (_WORD)v7 )
{
    v8 = v7;
    v9 = 0;
    while ( 1 )
    {
        v13 = sub_167E80(12429);
        v14 = -sub_163120(-(__int16)v8, -v13);
        sub_167E80(12429);
        if ( (unsigned __int16)v14 >= 0x5Fu )
        {
            if ( (unsigned __int16)v8 > 0xDu )
                break;
            v15 = 9728;
            if ( !_bittest(&v15, v8) )
                break;
        }
        v10 = sub_161DE0(v9, -1454282507);
        v9 = v10 + 1454282508;
        v11 = v10;
        v12 = off_186010[v10 - 17 * ((v10 + 1454282508) / 0x11u) + 1454282508];
        v8 = v12 ^ (unsigned __int16)a1[v11 - 693201140];
        sub_161D70(a1[v11 - 693201140], v12);
        v3 = a2;
        a2[v11 - 693201140] = v8;
        if ( !(_WORD)v8 )
            return v3;
    }
    return a1;
}
else
{

```

Direction	Typ	Address	Text
	r	sub_1658F0+2B	mov eax, off_186010; "#uVTN7'GQ'rxUf5Ly"
Do...	r	sub_1658F0+BB	mov ecx, off_186010; "#uVTN7'GQ'rxUf5Ly"
Do...	r	sub_173C90+1C	mov eax, off_186010; "#uVTN7'GQ'rxUf5Ly"
Do...	r	sub_173C90+55	add ecx, off_186010; "#uVTN7'GQ'rxUf5Ly"

Line 1 of 4

OK Cancel Search Help

Cross referencing both **sub_1658F0** and **sub_173C90** routines would shows that over 120 times those functions has been called. Randomly checking any of the cross referencing like below

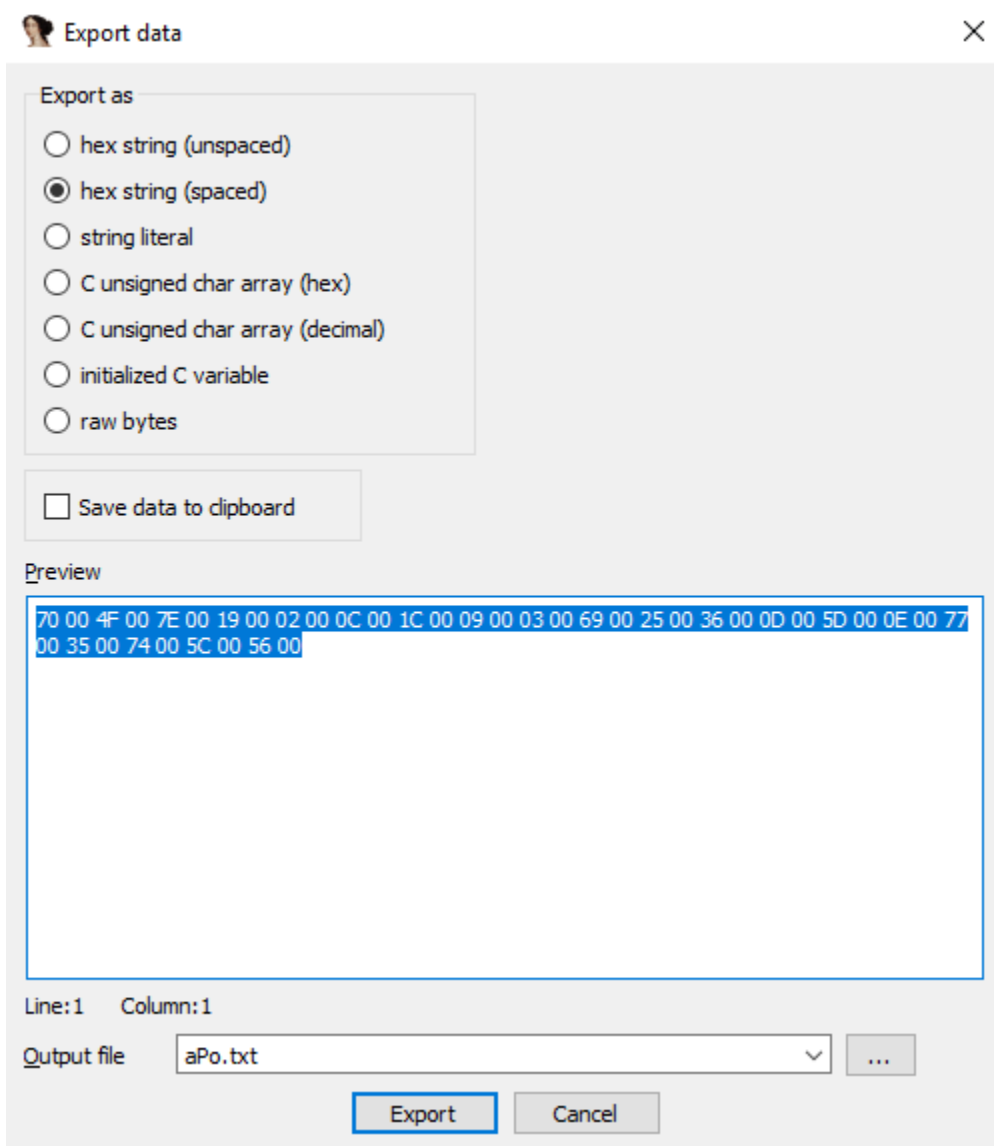
```

....skipped lines.....
text:001610E0 push offset unk_183E8E
.text:001610E5 call sub_1658F0
....skipped lines.....
text:0016444E push offset unk_184310
text:00164453 call sub_173C90
..... skipped lines.....
text:00165685 push offset unk_184040
text:0016568A call sub_1658F0

```

We noticed both subroutines been called after a push of unknow offsets

Let's use XOR key (#uVTN7'GQ'rxUf5Ly) with **offset unk_184040** value.



Shift+E over unknow

offset location

Hex: 70 00 1A 00 30 00 20 00 39 00 56 00 55 00 22 00 0D 00 6A 00 1B 00 1B 00 27 00 09 00 46 00 23 00 1F 00 57 00 29 00 56 00

key: #uVTN7'GQ'rxUf5Ly

Result: SuLT~7.Gh'\$x.f.Lt#.VON,'Q.r>UE5Sytu.T.7

The XORed value/result doesn't make sense. If anything noticeable that the Hex values has zeros in sequence. Which indicate **sub_1658F0** is for wide character and this makes and **sub_173C90** for normal character. let's try again deleting all repeated zeros and XOR with the key

Hex: 701A3020395655220D6A1B1B270946231F572956

key: #uVTN7'GQ'rxUf5Ly

Result: Software\Microsoft\

It's not just strings that been obfuscated, some API calls been XORed too. Almost 120 offset being pushed in stack which means 120 strings are being XORed and to make it readable; *Appendix – A* contains all the strings with addresses after been XORed.

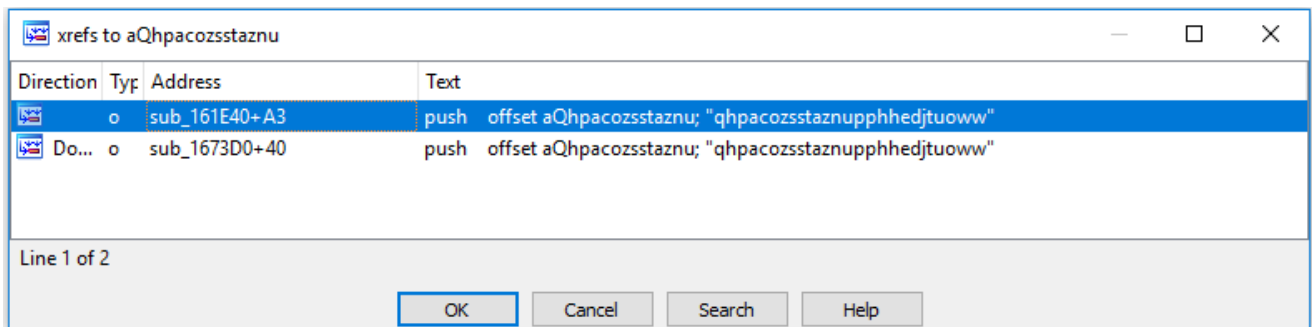
```

.text:00164449      lea    eax, [esp+44h+var_25]
.text:0016444D      push  eax
.text:0016444E      push  offset NtWriteVirtualMemory
.text:00164453      call  wide_string_deobfuscation
.text:00164458      add   esp, 8
.text:0016445B      mov   esi, eax
.text:0016445D      call  sub_178400
.text:00164462      push  esi
.text:00164463      push  edx
.text:00164464      push  eax
23 char v26[8]; // [esp+17h] [ebp-2Dh] BYREF
24 char v27[37]; // [esp+1Fh] [ebp-25h] BYREF
25
26 if ( !qword_187480 )
27 {
28     v6 = wide_string_deobfuscation(NtWriteVirtualMemory, v27);
29     v7 = sub_178400();
30     qword_187480 = sub_17E6B0(v7, SHIDWORD(v7), v6);
31     if ( !qword_187480 )
32         return 0;

```

Configuration:

The other string 'qhpacozsstaznupphhedjtuoww' is 26 length. It's crossed referenced twice in two separate routines.



snipped assembly from **sub_161E40** and **sub_1673D0** routines

```

.text:00161EE3 push offset aQhpacozsstaznu ; "qhpacozsstaznupphhedjtuoww"
.text:00161EE8 push offset unk_184404
——skipped lines——
text:001673D0 sub_1673D0 proc near ; CODE XREF: sub_171400+40↓p
.text:001673D0 push ebp
.text:001673D1 mov ebp, esp
.text:001673D3 push edi
.text:001673D4 push esi
.text:001673D5 mov esi, ecx
.text:001673D7 call sub_1809A0
.text:001673DC mov edi, [eax+30h]
.text:001673DF mov ecx, esi
.text:001673E1 call sub_1809A0
.text:001673E6 add eax, edi
.text:001673E8 push 36Fh
.text:001673ED push offset unk_184404
.text:001673F2 push eax
.text:001673F3 call sub_171D80
.text:001673F8 add esp, 0Ch
.text:001673FB mov ecx, esi
.text:001673FD call sub_1809A0

```

```
.text:00167402 mov edi, [eax+34h]
.text:00167405 mov ecx, esi
.text:00167407 call sub_1809A0
.text:0016740C add eax, edi
.text:0016740E push 64h ; 'd'
.text:00167410 push offset aQhpacozsstaznu ; "qhpacozsstaznupphhedjtuoww"
```

In both routines notice a repeated push to an offset **unk_184404**. This offset contains configurations. Noticed that both offset passed into a function **sub_1656B0** (*name decrypting_rc4*)

```

1 char __usercall C2@<a1>(int a1@<ebx>)
2 {
3     unsigned __int8 *v1; // eax
4     unsigned int v2; // eax
5     void (__cdecl *v3)(int (__stdcall *)(int)); // eax
6     unsigned int v4; // eax
7     void (__cdecl *v5)(int); // eax
8     char v7[13]; // [esp+3h] [ebp-Dh] BYREF
9
10    if ( !sub_16EDD0() )
11        return 0;
12    v1 = wide_string_deobfuscation(kernel32_dll_0, v7);
13    if ( !sub_175E70(lpLibFileName, v1) )
14        return 0;
15    if ( !LoadLibraryA(lpLibFileName) )
16        return 0;
17    call_get_proc_heap();
18    v2 = sub_167890(-1514247953);
19    v3 = Resolve_api(0, v2);
20    v3(exit_thread);
21    v4 = sub_167890(-1432131992);
22    v5 = Resolve_api(0, v4);
23    v5(32775);
24    call_internet_set_option();
25    sub_174FA0();
26    sub_170ED0();
27    decrypting_rc4(&config, "qhpacozsstaznupphhedjtuoww");
28    sub_16F5D0(a1);
29    if ( !call_getModule_name() || !call_get_length_sid() || !sub_168830() )
30        return 0;
31    call_getcurrentprocessid();
32    sub_174E80();
33    return 1;
34 }

```

Pseudo code from **sub_1656B0** routine

Decrypting_rc4 function calls multiple function and those are calling other functions. What we are looking here is RC4 algorithm.

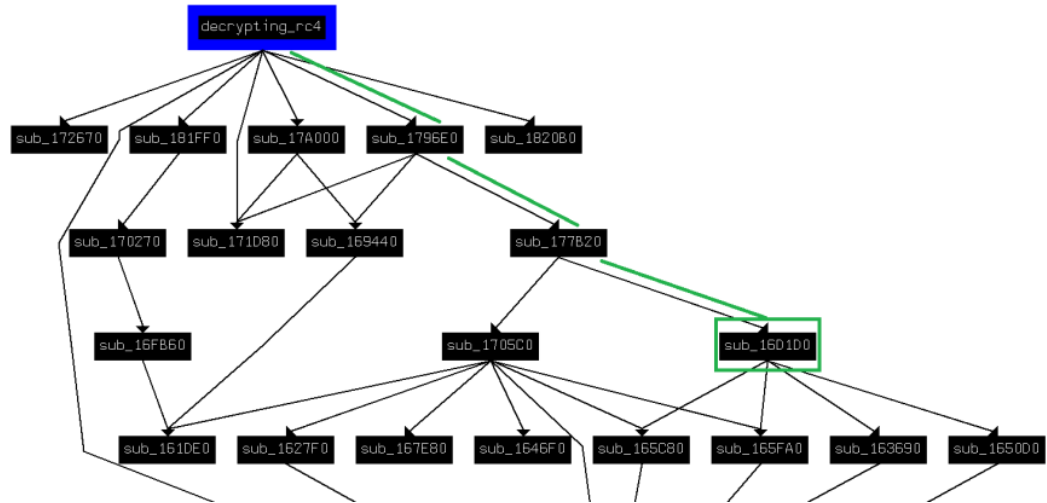
```

IDA View-A
ext:0016D1D4      push     edi
ext:0016D1D5      push     esi
ext:0016D1D6      sub      esp, 10h
ext:0016D1D9      mov      ecx, [ebp+arg_8]
ext:0016D1DC      mov      esi, [ebp+arg_4]
ext:0016D1DF      mov      dl, [ecx+100h]
ext:0016D1E5      mov      al, [ecx+101h]
ext:0016D1E8      test     esi, esi
ext:0016D1ED      jz       loc_16D2A1
ext:0016D1F3      mov      edi, [ebp+arg_0]
ext:0016D1F6      mov      [ebp+var_18], edi
ext:0016D1F9      nop
ext:0016D1FA      nop
ext:0016D1FB      nop
ext:0016D1FC      nop
ext:0016D1FD      nop
ext:0016D1FE      nop
ext:0016D1FF      nop
ext:0016D200      loc_16D200:
ext:0016D200      inc      dl
ext:0016D202      mov      [ebp+var_1C], esi
ext:0016D205      movzx   edi, dl
ext:0016D208      mov      [ebp+var_E], dl
ext:0016D20B      movzx   ebx, byte ptr [ecx+edi]
ext:0016D20F      mov      edx, ebx
ext:0016D211      add     dl, al
ext:0016D213      movzx   eax, al
ext:0016D216      mov      [ebp+var_D], dl
ext:0016D219      push    eax
ext:0016D21A      push    ebx
ext:0016D21B      mov      esi, ecx
ext:0016D21D      call   sub_165FA0
ext:0016D222      add     esp, 8
ext:0016D225      movzx   eax, [ebp+var_D]
ext:0016D229      movzx   ecx, byte ptr [esi+eax]
ext:0016D22D      mov      [esi+edi], cl
ext:0016D230      mov      [esi+eax], bl
ext:0016D233      movzx   eax, byte ptr [esi+edi]
ext:0016D237      mov      [ebp+var_14], al
ext:0016D23A      push    ebx

Pseudocode-A
19  unsigned __int8 v20; // [esp+Fh] [ebp-Dh]
20
21  v3 = a3;
22  v4 = a2;
23  v5 = *(a3 + 256);
24  result = *(a3 + 257);
25  if ( a2 )
26  {
27  {
28  {
29  v7 = v5 + 1;
30  v15 = v4;
31  v8 = v7;
32  v19 = v7;
33  v10 = *(v3 + v7);
34  v9 = v10;
35  v20 = result + v10;
36  v11 = v3;
37  sub_165FA0(v10, result);
38  *(v11 + v8) = *(v11 + v20);
39  *(v11 + v20) = v9;
40  v17 = *(v11 + v8);
41  v12 = sub_165C80(0, v9);
42  LOBYTE(v9) = *(v11 + sub_163690(~(v12 + ~v17), -1));
43  v18 = *a1;
44  v13 = sub_1650D0(*a1, 255);
45  v3 = v11;
46  v14 = v20;
47  v5 = v19;
48  LOBYTE(v9) = v9 & v13 | v18 & ~v9;
49  result = v20;
50  *a1 = v9;
51  v4 = v15 - 1;
52  ++a1;
53  }
54  while ( v15 != 1 ) ;
55  }
56  else
57  {
58  v14 = *(a3 + 257);
59  }
60  *(v3 + 256) = v5;

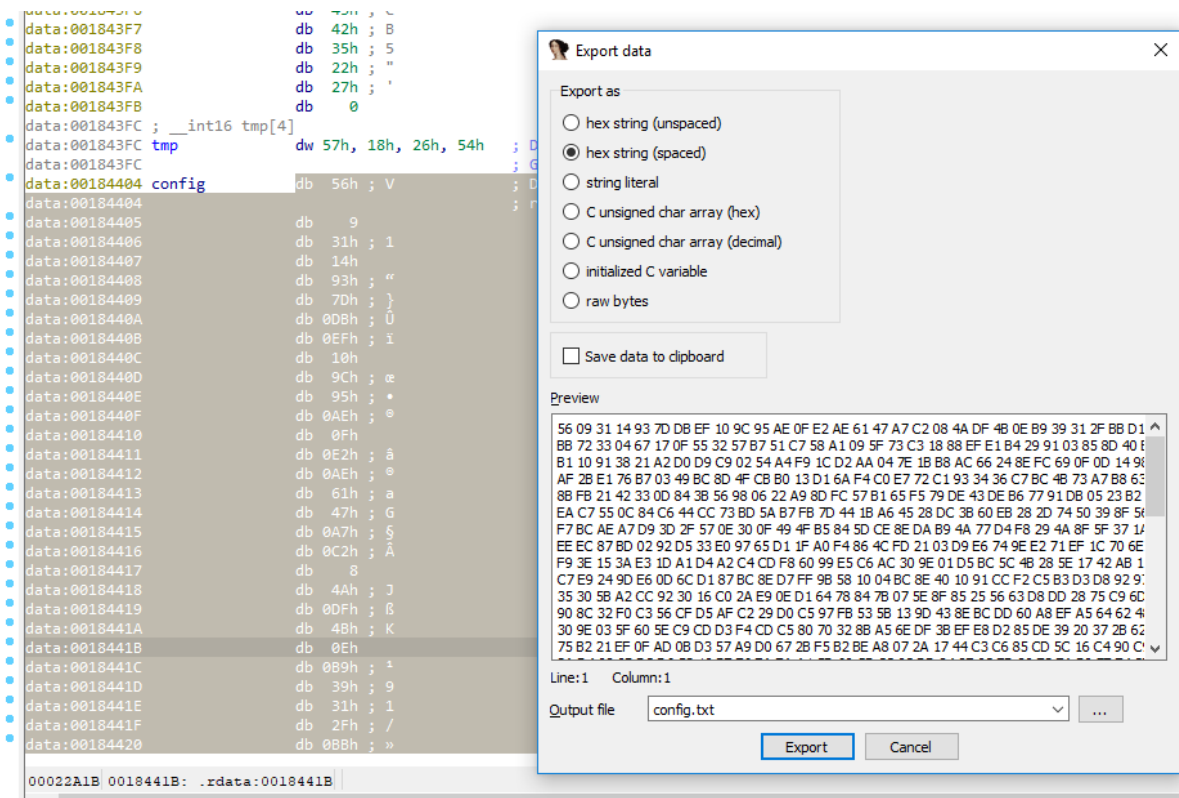
```

To have mind map where RC4 algorithm location lets Xref-from Decrypting_rc4 function where Config strings and key retrieved



Xref_from **sub_1656B0** (decrypting_rc4)

Now let's go back to the configuration 'config' offset in data block and copy its hex value to CyberChef and use RC4 algorithm to decrypt it with the key (qhpacozsstaznupphhedjtuoww)



Notice three things: got C2 URLs, list in Table-1, and 123 which is ID for this variant of Zloader, and at the tail there's this value (djfsf02hf832hf03) which is another RC4 key that decrypt the registry values in HKEY_CURRENT_USER\Software\Microsoft\bbxk and also encrypt traffic with C2 [7].

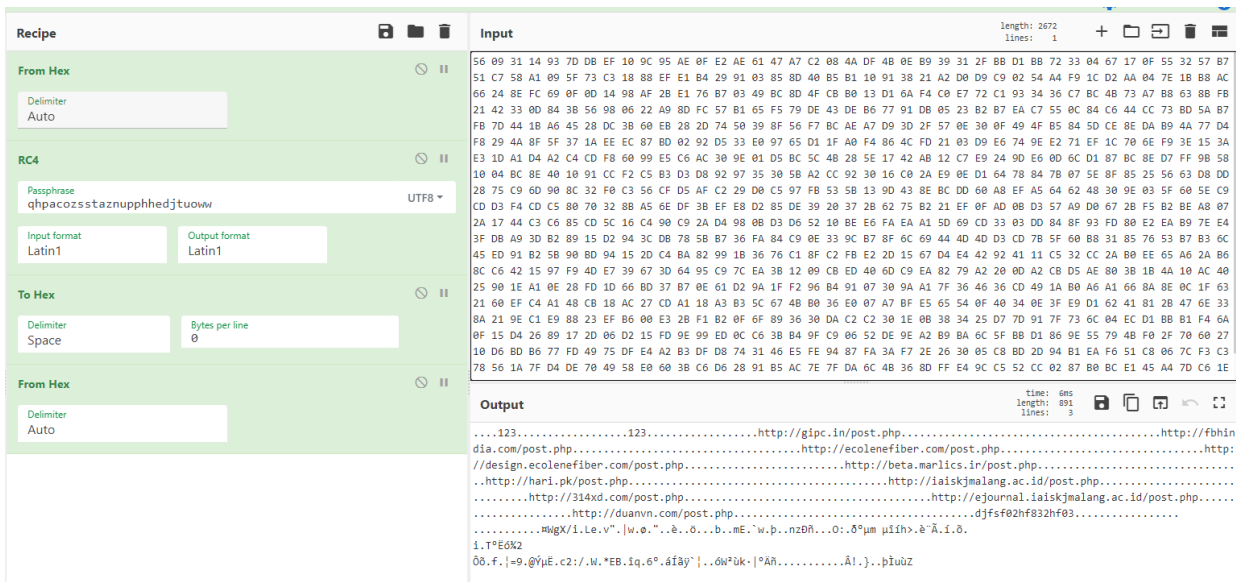


Table-1

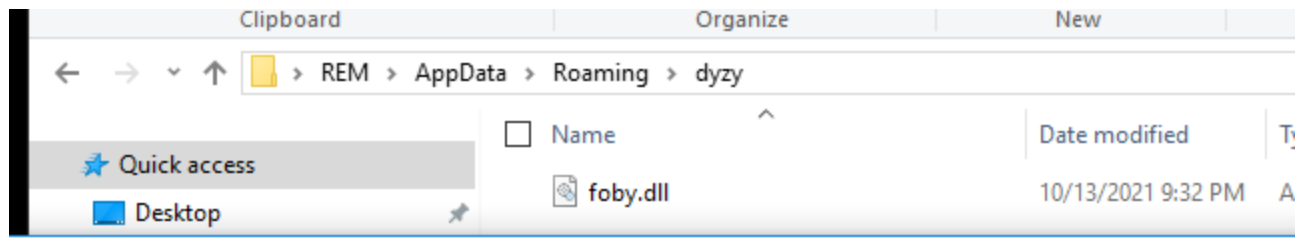
123

hxxp://gipc.in/post[.]php

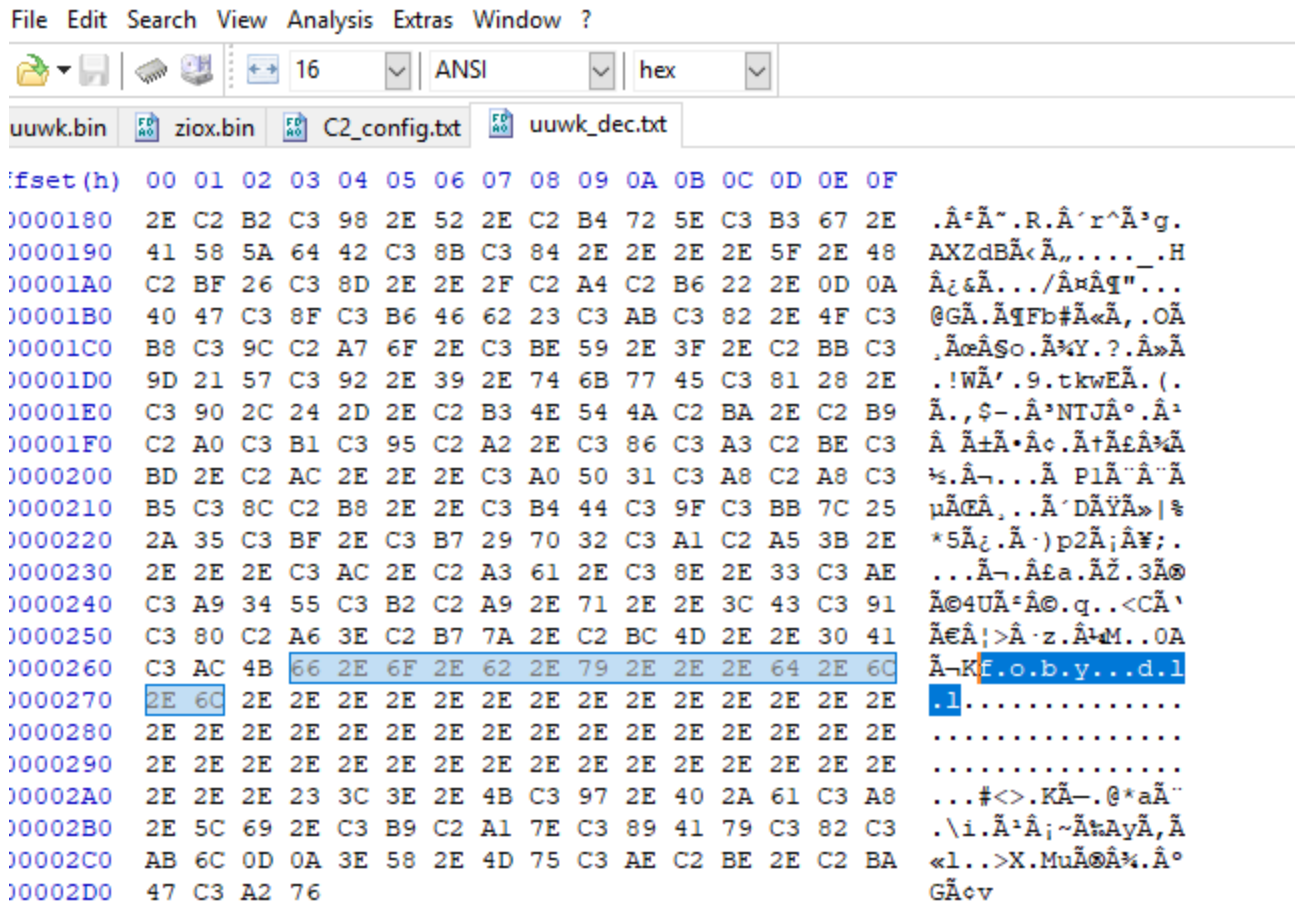
hxxp://fbhindia.com/post[.]php

hxxp://ecolenefiber.com/post[.]php
 hxxp://design.ecolenefiber.com/post[.]php
 hxxp://beta.marlics.ir/post[.]php
 hxxp://hari.pk/post[.]php
 hxxp://iaiskjmalang.ac.id/post[.]php
 hxxp://314xd.com/post[.]php
 hxxp://ejournal.iaiskjmalang.ac[.]id/post.php
 hxxp://duanvn.com/post[.]php
 djfsf02hf832hf03

Decrypted registry key value contains host name and the Zloader in %AppData% directory.



HxD - [C:\Users\REM\Desktop\zloader disassembled\uuwk_dec.txt]



DGA:

Zloader know for using DGA algorithm and we notice above some of the generated 32 character length URLs. To find the DGA function in this we can look for .com or post.php strings that been deobfuscated in the previous section of XORing strings.

371	0x2e691d8	(56):	http://fbhindia.com/post.php
372	0x2e69340	(56):	http://fbhindia.com/post.php
373	0x2e69730	(62):	http://beta.marlics.ir/post.php
374	0x2e69a00	(62):	http://beta.marlics.ir/post.php
375	0x2e69a48	(62):	http://beta.marlics.ir/post.php
376	0x2e69b20	(62):	http://beta.marlics.ir/post.php
377	0x2e6a0b8	(66):	https://rtgovcvcyqtrtpxqsuhj.com/
378	0x2e6ad50	(26):	https://rtgo
379	0x2e71ca8	(82):	https://ahuwyfifjocktbiqfroj.com/post.php
380	0x2e71d08	(82):	https://kixjaojbhegarxudxqfh.com/post.php
381	0x2e71d68	(82):	https://bmstetxrlsciidvnbvlg.com/post.php
382	0x2e71fa8	(82):	https://xflnqrfudepefimrmysd.com/post.php
383	0x2e72008	(82):	https://veydywopnrvbutwdbvr.com/post.php
384	0x2e72068	(82):	https://ysrfsfqdowjostdvptxq.com/post.php
385	0x2e721e8	(82):	https://kixjaojbhegarxudxqfh.com/post.php
386	0x2e72248	(82):	https://ucutfofotknnsrjhtuuw.com/post.php
387	0x2e72308	(82):	https://bmstetxrlsciidvnbvlg.com/post.php
388	0x2e72368	(82):	https://kixjaojbhegarxudxqfh.com/post.php
389	0x2e723c8	(82):	https://ysrfsfqdowjostdvptxq.com/post.php
390	0x2e72428	(82):	https://ahuwyfifjocktbiqfroj.com/post.php
391	0x2e724e8	(82):	https://ahuwyfifjocktbiqfroj.com/post.php
392	0x2e725a8	(82):	https://yhacxpjplceniojswrw.com/post.php
393	0x2e72608	(82):	https://kixjaojbhegarxudxqfh.com/post.php
394	0x2e72728	(82):	https://ucutfofotknnsrjhtuuw.com/post.php
395	0x2e72788	(82):	https://vafaxkulewsyfxdcvmsn.com/post.php
396	0x2e727e8	(82):	https://bmstetxrlsciidvnbvlg.com/post.php
397	0x2e773e0	(52):	http://duanvn.com/post.php
398	0x2e79bd0	(82):	https://ucutfofotknnsrjhtuuw.com/post.php
399	0x4d20100	(64):	http://ecolenefiber.com/post.php
400	0x4d23414	(46):	http://hari.pk/post.php
401	0x4d2ab4c	(62):	http://beta.marlics.ir/post.php
402	0x4d2bc58	(46):	http://hari.pk/post.php
403	0x4d31180	(64):	http://ecolenefiber.com/post.php
404	0x4d33390	(62):	http://beta.marlics.ir/post.php
405			

when cross referencing .com from rdata:001849B4 location we find that it's been called by one function and let's name that function the_dga

<pre> ext:00176868 call sub_180AE0 ext:0017686D movsx ebx, [ebp+var_D] ext:00176871 add ebx, esi ext:00176873 mov esi, ebx ext:00176875 not esi ext:00176877 and esi, 8EBDE18h ext:0017687D push 0FFFFFFFh ext:0017687F push 8EBDE18h ext:00176884 call sub_1627F0 ext:00176889 add esp, 8 ext:0017688C and ebx, eax ext:0017688E or ebx, esi ext:00176890 push eax ext:00176891 push [ebp+dwSeed] ext:00176894 call sub_163690 ext:00176899 add esp, 8 ext:0017689C or esi, eax ext:0017689E mov esi, [ebp+dwSeedANDED] ext:001768A1 xor esi, ebx ext:001768A3 mov ebx, 51EB851Fh ext:001768A8 dec edi ext:001768A9 jnz short loc_176B40 ext:001768AB lea eax, [ebp+szTLD] ext:001768AE push eax ext:001768AF push offset_com ext:001768B4 call wide_string_deobfuscation ext:001768B9 add esp, 8 ext:001768BC lea edi, [ebp+var_24] ext:001768BF mov ecx, edi ext:001768C1 push eax ext:001768C2 call concatenate ext:001768C7 mov ecx, [ebp+arg_8] ext:001768CA push edi ext:001768CB call save_in_array ext:001768D0 mov ecx, edi ext:001768D2 call sub_181F60 ext:001768D7 push 0D673F2BAh ext:001768DC call sub_167890 ext:001768E1 add esp, 4 ext:001768E4 mov ecx, [ebp+iDomainNum] ext:001768E7 mov edi, ecx </pre>	<pre> 7 int v6; // eax 8 unsigned int v7; // ebx 9 int v8; // eax 10 unsigned __int8 *szTLD_1; // eax 11 unsigned int v10; // eax 12 char szTLD[5]; // [esp+3h] [ebp-29h] BYREF 13 int v12[3]; // [esp+8h] [ebp-24h] BYREF 14 unsigned int dwSeedANDED; // [esp+14h] [ebp-18h] 15 int iDomainNum; // [esp+18h] [ebp-14h] 16 char v15[13]; // [esp+1fh] [ebp-0h] BYREF 17 18 if (nNumberOfDomains) 19 { 20 r = dwSeed; 21 result = 0; 22 dwSeedANDED = ~dwSeed & 0x8EBDE18; 23 do 24 { 25 iDomainNum = result; 26 sub_180E80(v12); 27 i = 20; 28 do 29 { 30 sub_1646F0(12); 31 v15[0] = r % 0x19 + 97; 32 sub_180AE0(v15); 33 the_letter = r + v15[0]; 34 v6 = sub_1627F0(-1897144808, -1); 35 v7 = ~the_letter & 0x8EBDE18 v6 & the_letter; 36 v8 = sub_163690(dwSeed, v6); 37 r = v7 ^ (dwSeedANDED v8); 38 --i; 39 } 40 while (i); 41 szTLD_1 = wide_string_deobfuscation(com, szTLD); 42 concatenate(szTLD_1); 43 save_in_array(v12); 44 sub_181F60(v12); 45 v10 = sub_167890(-697044294); 46 result = iDomainNum - v10 + 2082815448; 47 sub_161DE0(iDomainNum, 1); 48 } </pre>
---	---

The `_dga` function has been called one by another function. Based on [8], the caller of DGA routine does it math calculating values called Seed based on time and RC4 key (djfsf02hf832hf03) (second key). So the values generated are much different each day passed in used `GetLocalTime` and `SystemTimeToFileTime` APIs. Notice that the `_dga` function has passed value of 32 which is the same length of the URL string with Seed value which in this case makes the entire caller function to calculate Seed value. followed by `post.php` and `https` while loop. The caller function got many obfuscated function that slows down analysis and it get complicated calculating generated domains manually.

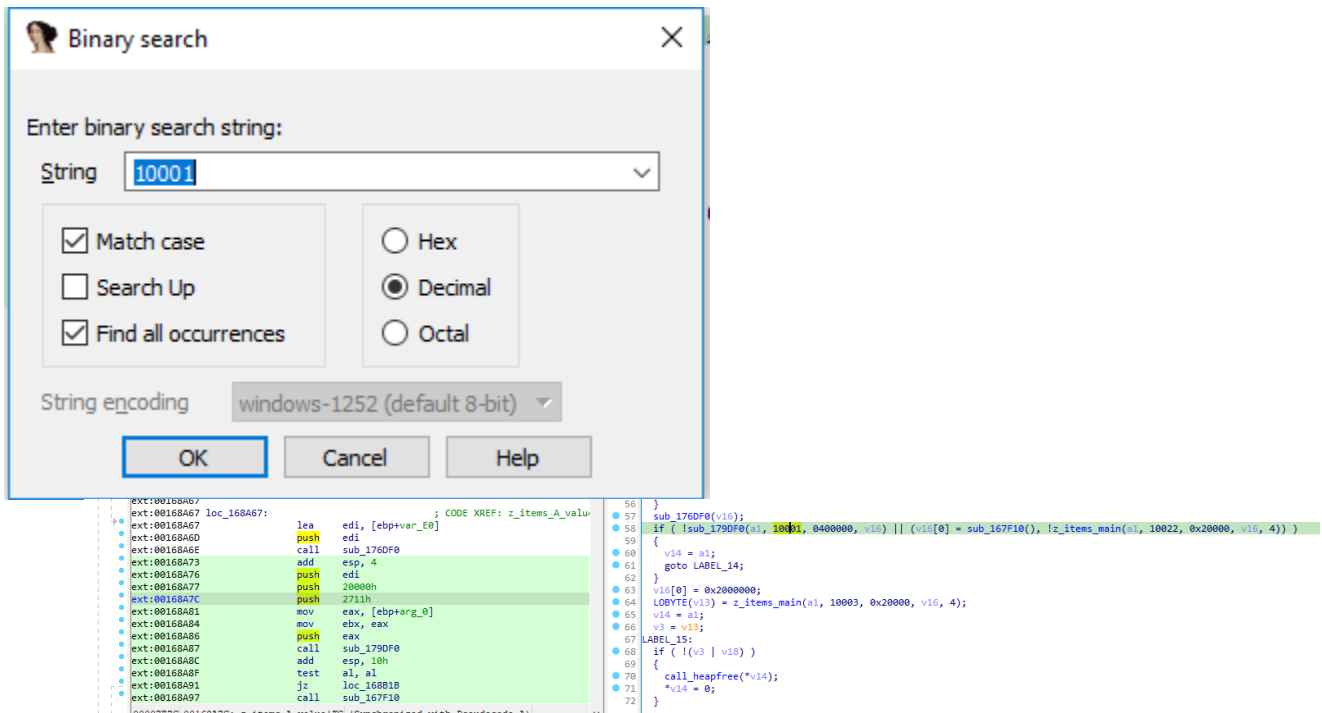
Zeus Items:

Zeus uses item ID as list below which is the main one, there are more extended list based on Zloader version [1] [2] [7]. Each ID passed into a function and dissect information from victim machine. When that information stored in attacker SQL filed it show retrieved info about the host.

Item ID	Value
10001	SBCID BOT ID
10002	SBCID BOTNET
10003	SBCID BOT VERSION
10005	SBCID NET LATENCY
10006	SBCID TCPPOINT S1
10007	SBCID PATH SOURCE

Item ID	Value
10008	SBCID PATH DEST
10009	SBCID TIME SYSTEM
10010	SBCID TIME TICK
10011	SBCID TIME LOCALBIAS
10012	SBCID OS INFO
10013	SBCID LANGUAGE ID
10014	SBCID PROCESS NAME
10015	SBCID PROCESS USER
10016	SBCID IPV4 ADDRESSES
10017	SBCID IPV6 ADDRESSES
10018	SBCID BOTLOG TYPE
10019	SBCID BOTLOG

To get to Zeus item values and function we need to search strings in IDA to find one the common ID values since they are constant.



Notice that most the calls are **sub_1657B0**, let's call it **z_items_main**, it's been crossed referenced 17 times. List of Zeus items being found in 123 variant.

Item ID	Value
10001	SBCID BOT ID
10003	SBCID BOT VERSION
10006	SBCID_PING
10007	SBCID PATH SOURCE
11014	SBCID_GET_FILE
11015	SBCID_GET_FILE_VER
11031	SBCID_LOG_ID_EXT
11032	SBCID_LOG_ERR_CODE
11033	SBCID_LOG_MSG
10022	SBCID_DEBUG
10025	SBCID_MARKER
20001	CFGID_LAST_VERSION
20000	SBCID_BOTLOG
20005	CFG_HTTP_FILTER
20006	CFGID_HTTP_POSTDATA_FILTER
20008	CFGID_DNS_LIST

Just to give an example of the level of obfuscation on every stage of Zloader. Not all the items ID values are retrieved in decimal passed to the function. Some values passed into another function and require to calculate separately like below in v29 value return from **sub_167890** .

```

if ( z_items_A_value(v41) )
{
    v29 = sub_167890(-1437145989);
    v30 = sub_167890(-1437151383);
    z_items_main(v41, v29, 0, &a1, v30);
    z_items_main(v41, 11031, 0, &a2, 4);
    z_items_main(v41, 11032, 0, &a3, 4);
    v31 = a4;
    if ( sub_1812A0(a4) )

```

```

Pseudocode-A
1 unsigned int cdecl sub_167890(int a1)
2 {
3     unsigned int result; // eax
4     unsigned int v2; // esi
5     char v3; // bl
6     unsigned int v4; // edx
7     int v5; // edi
8     void *v6; // edi
9
10    result = a1 ^ 025225547555;
11    v2 = a1 & (a1 ^ 025225547555);
12    v3 = v2 * (a1 ^ 0155) * (a1 & (a1 ^ 0155));
13    v4 = v3 + (a1 ^ 025225547555) * v2;
14    if ( v3 != a1 )
15    {
16        v2 = v4 + 973;
17        v5 = (v4 + 973) * v3;
18        v3 = result & ((v4 - 51) * v3);
19        v4 = v5;
20    }
21    v6 = 0;
22    if ( a1 ^ v3 | a1 ^ v2 )
23        v6 = v4;
24    if ( v4 == a1 )
25        v6 = v4;
26    pvReserved = v6;
27    return result;
28 }

```

To give an example of how Zeus item works let take a look at this function **sub_177110**.

```

Pseudocode-A
31 int v32; // [esp+154h] [ebp-14h] BYREF
32 _DWORD *v33; // [esp+158h] [ebp-10h]
33
34 v33 = a1;
35 sub_1615B0(v26);
36 v30 = a2;
37 if ( a2 )
38     dga_caller(v26);
39 else
40     sub_16F780(v26);
41 v3 = &v32;
42 v32 = sub_171910();
43 z_items_A_value(&v32);
44 v27 = 0;
45 z_items_main(&v32, 10006, 0, &v27, 4);
46 sub_180E50(v23);
47 v4 = sub_167890(-1437151363);
48 sub_180FB0(v4);
49 v5 = sub_1812A0(v23);
50 v6 = ret_this2(v23);
51 sub_177A00(v6, v5, 0, 0xFFu, 0);
52 v7 = sub_1812A0(v23);
53 v8 = ret_this2(v23);
54 z_items_main(&v32, 10007, 0, v6, v7);
55 decrypt_config_rc4(v23);
56 v9 = sub_17F660(&v32, 1, v21);
57 sub_181B20(v32, v32 + v9);
58 sub_180E00(v33);
59 v10 = ret_this2(v26);
60 v28 = sub_1815A0(v26);
61 if ( v10 != v28 )
62 {
63     v11 = v22;
64     do
65     {
66         sub_180E50(v11);
67         if ( sub_177750(v3, v24, v10, v11, v21, 4) )
68         {
69             v12 = ret_this2(v11);
70             v3 = v11;
71             v13 = *(v12 + 24);
72             v29 = v12;

```

sub_16F780 has (20001: CFGID_LAST_VERSION) that updates Zloader version. Looking at it in the disassembler would show so much obfuscated function, but to have an idea of what possibly this update could do let's see the leaked source code having this similar Item ID in similar fashion [9].


```

141 if(md->data != NULL &&
142 (md->size = BinStorage::_unpack(NULL, md->data, md->size, &baseConfig->baseKey)) != 0 &&
143 BinStorage::_getItemDataAsDword((BinStorage::STORAGE *)md->data, CFGID_LAST_VERSION, BinStorage::ITEMF_IS_OPTION, NULL))
144 {
145     WDEBUG(WDDT_INFO, "Configuration unpacked.");
146
147     //Update the configuration.
148     {
149         PESETTINGS pes;
150         DWORD type;
151
152         initRegistry();
153         Core::getPeSettings(&pes);
154
155         if((md->size = BinStorage::_pack((BinStorage::STORAGE **)&md->data, BinStorage::PACKF_FINAL_MODE, &pes.rc4Key)) == 0 ||
156 Registry::_setValueAsBinary(HKEY_CURRENT_USER, registryKey, registryValue, REG_BINARY, md->data, md->size) == false)
157         {
158             WDEBUG(WDDT_ERROR, "Failed to repack configuration.");
159         }
160         else
161         {
162             retVal = true;
163             tryToUpdateBot(flags & UCF_FORCEUPDATE);
164         }
165     }
166 }

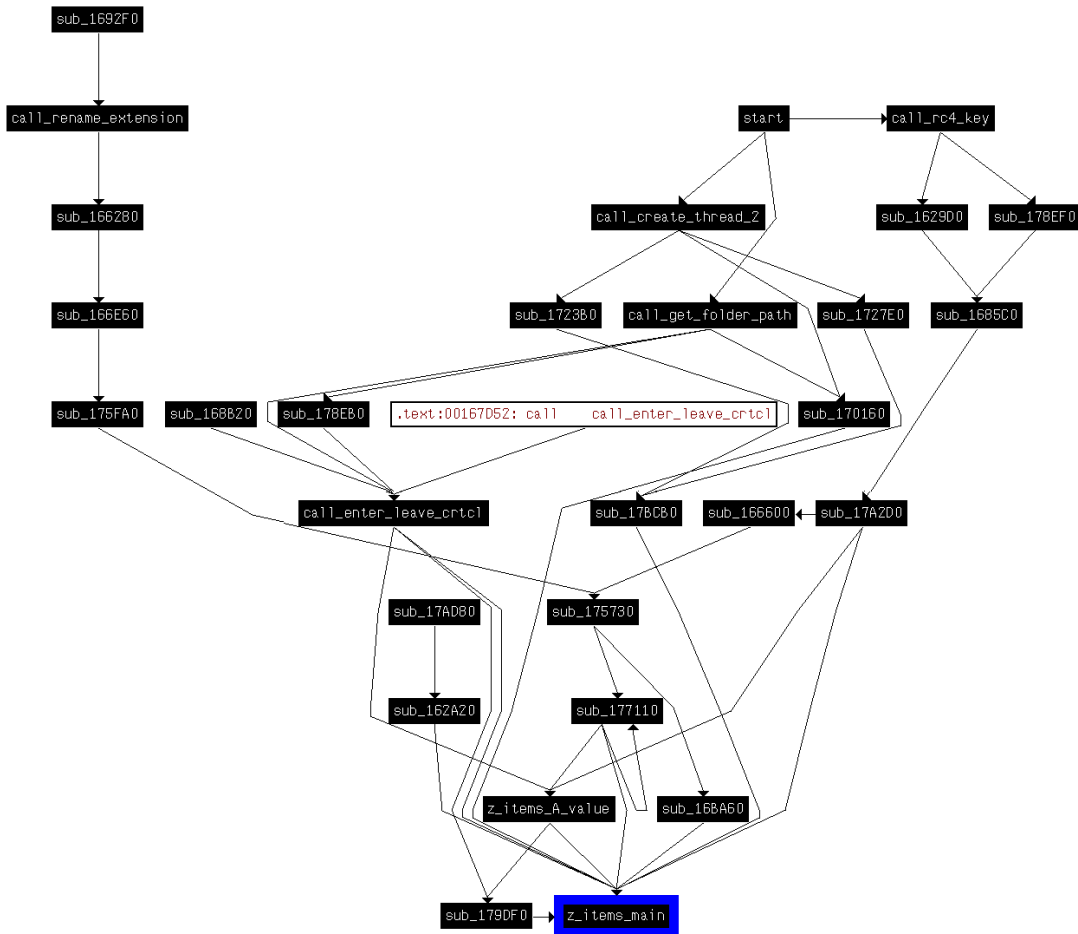
```

id: 20001

If update successful it changes registry values

The successful update would lead to update registry values in HKCU\Software\Microsoft\bboxk\ which points to %AppData% directory of possible the new Zloader that has new C2 connections

Finally, to have an idea how Zeus function being called here's a mind map when Xref-to it.



Address	XORed string
rdata:00183DED	kernel32.dll
rdata:00183DFA	http
rdata:00183E26	post
rdata:00183E37	.63
rdata:00183E3B	Wininet.dll
rdata:00183DE0	Imagehlp.dll
rdata:00183DB0	C:\Windows\SystemApps
rdata:00183D9C	Local
rdata:00183D92	.exe
rdata:00183D50	NtQueryVirtualMemory
rdata:00183D43	Bcrypt.dll
rdata:00183D38	Ftlib.dll
rdata:00183D2A	Samlib.dll
rdata:00183D20	Post.php
rdata:00183E47	Ntdll.dll
rdata:00183E5C	CmpMem64
rdata:00183E70	INVALID_BOT_ID
rdata:00183E8E	\start
rdata:00183EA0	HideClass
rdata:00183EB4	advapi32.dll
rdata:00183ED0	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
rdata:00183F21	ws2_32.dll
rdata:00183F3B	Shlwapi.dll
rdata:00183F47	crypt32.dll
rdata:00183F60	NtProtectVirtualMemory
rdata:00183F77	GetMem64

Address	XORed string
rdata:00183F90	Get
rdata:00183FA0	Software\Microsoft\Windows\CurrentVersion\Run
rdata:00183FFC	Urlmon.dll
rdata:0018400A	wtsapi32.dll
rdata:00184040	Software\Microsoft
rdata:00184068	tmp
rdata:0018407C	lphlpapi.dll
rdata:0018408C	Version.dll
rdata:0018409E	rpcrt4.dll
rdata:001840AA	Dll
rdata:00184111	wldap32.dll
rdata:00184165	ole32.dll
rdata:0018416F	psapi_dll
rdata:00184180	NtFreeVirtualMemory
rdata:001841A0	NtSetContextThread
rdata:001841B3	Winsta.dll
rdata:001841D0	user32.dll
rdata:001841E0	Software\Microsoft\WindowsNT\CurrentVersion
rdata:00184288	gdi32.dll
rdata:00184292	Gdiplus.dll
rdata:001842C0	regsvr32.exe
rdata:001842F0	RtlCreateUserProcess
rdata:00184310	NtWriteVirtualMemory
rdata:00184330	InstallDate
rdata:001843B0	NtReadVirtualMemory
rdata:001843E0	RtlCreateProcessParameters

Address	XORed string
rdata:00184780	Connection_close
rdata:00184794	Dnsapi.dll
rdata:001847BC	secur32.dll
rdata:001847D0	kernel32.dll
rdata:001847F0	NtGetContextThread
rdata:00184820	Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36.
rdata:00184892	NtResumeThread
rdata:001848B0	SeSecurityPrivilege
rdata:00184914	shell32.dll
rdata:00184920	Ntdll.dll
rdata:00184940	LdrGetProcedureAddress
rdata:00184957	netapi32.dll
rdata:00184964	Mpr.dll
rdata:0018496C	https:\\
rdata:00184975	X64Call
rdata:00184980	NtAllocateVirtualMemory
rdata:001849B4	.com
rdata:001849BA	Global
rdata:001849CA	Winscard.dll
rdata:001849D7	Cabinet.dll
rdata:001849E3	Userenv.dll
rdata:001849EF	Ncrypt.dll

References

[1] Zeus opensource, <https://github.com/Visgean/Zeus>

- [2] Titans' revenge: detecting Zeus via its own flaws, <https://www.honeynet.it/wp-content/uploads/Papers/04-Titans%20revenge.pdf>
- [3] Hide and Seek | New Zloader Infection Chain Comes With Improved Stealth and Evasion Mechanisms, <https://www.sentinelone.com/labs/hidden-and-seeK-new-zloader-infection-chain-comes-with-improved-stealth-and-evasion-mechanisms/>
- [4] The Squirrel Strikes Back: Analysis of the newly emerged cobalt-strike loader "SquirrelWaffle" , <https://elis531989.medium.com/the-squirrel-strikes-back-analysis-of-the-newly-emerged-cobalt-strike-loader-squirrelwaffle-937b73dbd9f9>
- [5] QakBot Quick analysis, <https://twitter.com/aaqeel87/status/1443255927000424449?s=20>
- [6] HashDB project, <https://hashdb.openanalysis.net/#section/Using-The-API/Hash-Format>
- [7] The "Silent Night" Zloader/Zbot , https://www.malwarebytes.com/resources/files/2020/06/the-silent-night-zloader-zbot_final.pdf
- [8] The DGA of Zloader, <https://bin.re/blog/the-dga-of-zloader/>
- [9] Zeus source code, <https://github.com/Visgean/Zeus/blob/c55a9fa8c8564ec196604a59111708fa8415f020/source/client/dynamicconfig.cpp>