Shifu – the rise of a self-destructive banking trojan

By Floser Bacurio Jr Fortinet, SingaporeWayne LowFortinet, SingaporeEditor: Martijn Grooten

Archived: 2026-04-05 22:52:11 UTC

2015-11-02

Abstract

The banking trojan Shifu appears to inherit some of its features from several other well-known banking trojans. Floser Bacurio and Wayne Low decided to take a close look at one of its droppers.

Copyright © 2015 Virus Bulletin

Following takedown operations against various notorious banking trojans, including Zeus, Dridex and Gozi, a freshly brewed banking trojan, Shifu, has recently made the news; we believe it inherits some features from the earlier well-known banking trojans. We decided to dive deeper into one of its droppers (MD5: E60F72FFA76386079F2645BE2ED84E53; SHA1: 963BFC778F94FE190FDD1DD66284E9BC9DD2BED6). A number of features caught our eye when looking at its underlying code.

Dropper

Exploiting CVE-2015-0003

On our first look at the dropper, we observed that the dropper code is not heavily obfuscated, although most of the strings are encoded; it turns out that the strings can easily be decoded using a simple XOR operation.

At the entry point, we can immediately tell that Shifu attempts to exploit a local privilege escalation vulnerability. The vulnerability was assigned the CVE number CVE-2015-0003 in February 2015 and can be used to elevate the privilege of a process to system privilege on *Windows 7* and above. The exploit code can easily be found on the Internet, so it is not surprising that the malware attempts to exploit this (patched) vulnerability. This means that the malware may not execute properly without sufficient privileges and also serves as a reminder that *Windows* users should always install the latest *Windows* updates.

After performing the local privilege escalation routine, we arrive at the code where the malware will extract the payloads embedded in its binary. The payloads consist of two aPLib compressed blobs for 32-bit and 64-bit platforms, which is a very common technique used by malware nowadays. The use of the aPLib compressor suggests that Shifu might have adopted some of the techniques used by malware like Zeus (aka Zbot) or Rovnix (and thus Carberp), for which the source code has been leaked.

Malware don’t like HIPS

One of the most notable things observed in the code injection routine is an attempt to obfuscate the MZPE header by overwriting random bytes in it. This action does not affect the execution of the payload since the payload code will be injected into the memory of the remote process and will be executed directly in the context of the remote process via an asynchronous thread. The purpose of overwriting bytes in the MZPE header is to defeat behavioural analysis systems and HIPS technology – with a corrupted MZPE header, the sample does not look like a legitimate binary file and could thus bypass some trivial PE signature checks.

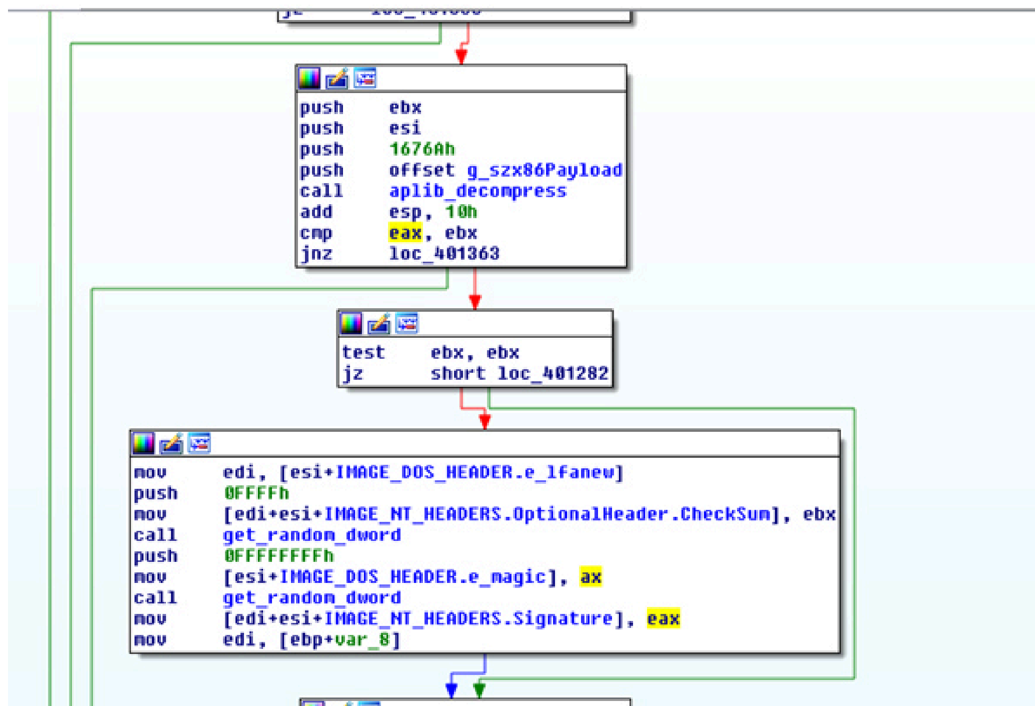

Figure 1. Corrupting the MZPE header.

Code obscuring

Analysing the decompressed buffer directly might result in an incorrect AddressOfEntryPoint as Shifu's author has intentionally obfuscated the entry point address by XORing with a widely used XOR key, 0x31337, throughout the program regardless of the dropper or payload. Shifu will first try injecting the payload code into explorer.exe. When the code injection into explorer.exe fails, to play it safe, Shifu's author implements an additional routine which will create or spawn a random *Windows* process found in C:\Windows\system32, or in C:\Windows\syswow64 if it is a 64-bit platform. Under normal circumstances, most *Windows* executable files (for instance svchost.exe) can be executed, but will quit immediately when called by non-*Windows* services. In order to circumvent this behaviour, Shifu first creates a suspended *Windows* process and then injects an infinite sleep, Sleep(-1), routine and executes it in the memory of the suspended process. After that, the suspended process will be resumed and continue execution, but it will not quit immediately unless it is forced to exit explicitly. Once the targeted process has been determined, Shifu will carry out another code injection routine to execute the payload via CreateRemoteThread or RtlCreateUserThread (Figure 2).

Figure 2. Remote code execution using CreateRemoteThread/RtlCreateUserThread.

Anti-sandbox and anti-VM

There are many anti-sandbox and anti-VM detection techniques in the dropper, as is common in mal-ware nowadays. Some of these checks have been documented in a *McAfee* blog post [1], so Table 1 lists some of the process names, as well as their hashes (computed by checking well-known process names using the *Windows* API function RtlComputeCrc32) that are 'blacklisted' by Shifu. Shifu checks a list of active processes running on the machine, as well as the sample's file name, against hard-coded CRC32 hashes in order to avoid the malware being analysed by a sandbox or by virtual machines that are built to perform dynamic analysis of malware samples.

Process name	**CRC32 hash**
vmwareuser.exe	0x99DD4432
vmwaretray.exe	0x1F413C1F
vboxservice.exe	0x64340DCE
vboxtray.exe	0x63C54474
wireshark.exe	0x77AE10F7

Process name	CRC32 hash
procmon.exe	0x5BA9B1FE
procexp.exe	0x3CE2BEF3
fortitracer.exe	0x332FD095
ollydbg.exe	0xAF2015F2
python.exe	0xD2EFC6C4
sysanalyzer.exe	0x4231F0AD
sniff_hit.exe	0xD20981E0
joeboxserver.exe	0x2AAA273B
joeboxcontrol.exe	0x777BE06C

Table 1. Process names that are 'blacklisted' by Shifu.

It also employs a check against the file names shown in Table 2.

Dropper's file name	CRC32 hash
sample.exe	0xE84126B8
malware.exe	0x3C164BED
test.exe	0xC19DADCE

Table 2. File names that are checked.

Payload

On analysing the decompressed payload using a disassembler, it turns out that we have landed at an invalid code entry point, as mentioned in the previous section, and the disassembler will complain that the import address table is corrupted. The payload cannot be analysed in a disassembler directly without first 'fixing' the file. We later realized that, besides the code entry point obfuscation, Shifu's author has also deployed some other trivial tricks to the payload to mislead analysts:

- Obfuscating the import table address – the original import table address can be restored by XORing with the key 0x31337.
- Obfuscating import function names (Figure 3 and Figure 4) – the function names are encoded using the static XOR key 0xFF. (In the Appendix, we provide a simple IDAPython script to fix the function names under *IDA Pro*.)

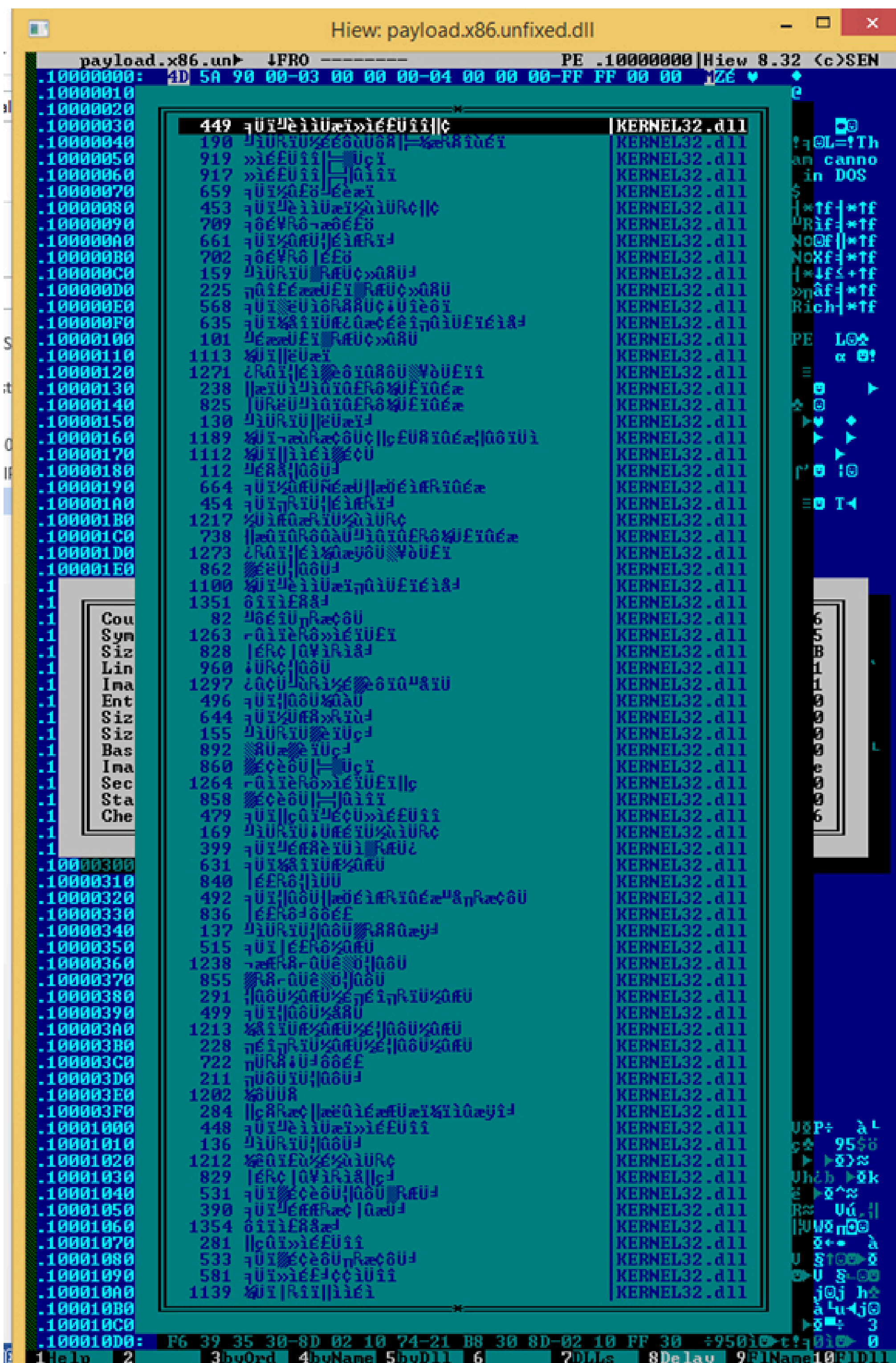

Figure 3. Obfuscated import function names.

Figure 4. Shifu deobfuscating the import table address before loading the DLL by name.

(Click here to view a larger version of Figure 4.)

Shifu's author also utilized multi-threading to add complexity to the malware and make dynamic analysis more challenging. On the other hand, from a static analysis perspective, the malware's encoding/encryption routine is insignificant and most of the strings in the binary can trivially be decoded and fixed within IDA Pro with a few lines of code (Figure 5). However, some of the strings, such as the command-and-control server name, are encrypted using RC4.

Figure 5. A simple script to decode the encoded string.

(Click here to view a larger version of Figure 5.)

Multiple process code injection

Before Shifu is able to carry out its wide range of information-stealing activities, it needs to make sure its payload code will be injected into the relevant processes. When the payload is injected by the dropper, the very first thing it will do is traverse the running processes and transfer its code into any process that does not already contain it. In other words, all the active processes apart from the system processes will contain a copy of the payload code whenever the machine is infected.

Afterwards, a named pipe will be created in order to allow communication between the payloads in different processes.

Comprehensive data theft features

Keylogger and screenshot capture

Looking into Shifu's data theft features, we could see that the malware is equipped with some of the keylogging features found in traditional keyloggers, as well as having the ability to steal numerous banking credentials from the victim. Shifu also uses the *Windows* API-hooking technique in order to steal other credentials. Table 3 shows a short summary of the hooked *Windows* APIs along with their respective purposes.

Hooked Windows API	**Purpose**
User32!GetClipboardData	Captures the ASCII and UNICODE text currently saved in the clipboard window
User32!TranslateMessage	Captures the keyboard's keystrokes
User32!GetMessageA/User32!GetMessageW	The hook's handler performs the same thing as the user32!TranslateMessage hook's handler

Table 3. Hooked Windows APIs and their purposes.

Under normal circumstances, these *Windows* API hooks should be able to capture the keystrokes on an infected machine. However, Shifu also tries to capture virtual keyboards, which are commonly used in Internet banking, by taking screenshots of the infected machine whenever the malware detects a mouse click. It is also noteworthy that the virtual keyboard screen will only be grabbed when the malware detects an opened screen with one of the following titles, all of which are used by Italian Internet banking websites:

- Password
- Telemaco
- Scelta e Login dispositivo
- TLQ Web
- db Corporate Banking Web
- SecureStoreCSP - enter PIN

Certificate capture

Shifu manipulates *Windows* API hooks in order to intercept the certificate password when a certificate is being imported to the certificate store. All the certificate blob data and passwords found in the Crypt32!PFXImportCertStore API will be intercepted, unless the process contains the string 'torrent'.

The imported public keys on the infected machine will also be captured. This is possible, using the *Windows* Crypt32!CertEnumSystemStore API, without having access to the private key.

These hooks may be useful to the attackers when the victim imports certificates using a *Windows* PGP client like *Gpg4win*; the hook handlers could intercept the private key and certificate and save it to Shifu's specified log directory as 'randomhexavalue_cert.pfx'. Furthermore, it could be useful if the attackers want to access the cryptocurrency wallet downloaded from the victim machine, which is encrypted using an RSA key pair.

Shifu's author and its operators will be able to abuse the stolen certificates for nefarious purposes.

Other data thefts

In line with the current cryptocurrency hype, Shifu also targets Bitcoin and Litecoin wallet files found on the victim's machine. Shifu tries to steal VPN and VNC login credentials by checking the command line of running executables. If a remote desktop protocol (RDP or VNC) or VPN process is found with the configuration file name specified in the command line, Shifu attempts to save a copy of the configuration file.

It appears that this trojan steals far more information than a typical banking trojan would: from keylogging, screenshot capture, certificate capture and cryptocurrency wallet grabbing, to FTP and POP3 credentials grabbing. The malware also appears to target point-of-sale terminals for payment card data as well as some financial institutions themselves. It scans the machines if one of the following strings is found in the path of the executable file of the current process:

- tellerplus
- bancline
- fidelity
- micrsolv
- bankman
- vanity
- episys
- jack henry
- cruisenet
- gplusmain
- silverlake
- v48d0250s1

When a potential POS machine is found, the malware will send a flag, 'ETC', back to its C&C server. Based on the malware code, there is no immediate action after the machine has been recognized as a POS system; perhaps a memory-scrapping module will be deployed by the botnet operators to this machine at a later time.

Stealthy banking trojan stays under the radar?

When analysing Shifu's *Windows* API-hooking mechanisms, we also discovered that the malware tries to remain hidden from the victim. The malware conceals its presence from the running processes by hijacking the *Windows* ntdll!ZwQuerySystemInformation API, which is called whenever a user-mode program attempts to enumerate a list of active processes using one of various process enumeration APIs. However, this is a well-known technique and is defeated by all modern security tools. Apparently, the purpose of this trick is to remain concealed from non-tech-savvy users, however what confused us is that the malware also hijacks *Windows* API calls used for DNS resolution, such as ws2_32!gethostbyname,

ws2_32!getaddrinfo and ws2_32!GetAddrInfoExW, to redirect URLs that contain the pattern 'secure\..*\.moz\..*' to 'google.com'. Essentially, this seems to dismiss the idea of the malware being stealthy, as hijacking a website is often a clear sign of infection.

Home sweet home

When a new machine is infected, the malware will report the new victim to the command-and-control (C&C) server by connecting to a domain that is hard-coded in the code, using the path '/news/userlogin.php'. The following is the information on the machine that will be stored in the botnet's control panel (also see Figure 6):

- botid – username and computer name
- ver – botnet version
- up – uptime of the infected machine
- os – operating system identifier of the infected machine
- ltime – local timestamp of the infected machine
- token – existence of smart card information
- cn – unknown
- av – name of the security solution installed
- dmn – domain name of the workstation

Address	Hex dump	ASCII
002BFA98	62 6F 74 69 64 3D 44 59 49 54 55 53 45 52 37 33	botid=
002BFAA8	32 21 44 59 49 54 57 49 4E 37 58 38 36 21 44 39	
002BFAB8	39 37 35 45 33 35 26 76 65 72 3D 31 2E 35 31 38	&ver=1.518
002BFAC8	26 75 70 3D 32 34 33 31 26 6F 73 3D 36 31 30 30	&up=2431&os=6100
002BFAD8	26 6C 74 69 6D 65 3D 25 32 62 38 26 74 6F 6B 65	<ime=%2b8&toke
002BFAE8	6E 3D 30 26 63 6E 3D 61 33 26 61 76 3D 26 64 6D	n=0&cn=a3&av=&dm
002BFAF8	6E 3D 00 00 00 00 00 00 00 00 00 00 00 00 00 00	n=..............
002BFB08	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
002BFB18	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
002BFB28	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
002BFB38	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
002BFB48	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
002BFB58	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

Figure 6. Machine information sent back to the C&C control panel.

All the data sent and received is obfuscated to prevent direct exposure by a packet sniffer program. The data is encoded and decoded using the RC4 algorithm with the key 'a7zoSTHljZylEx4o3mJ2eqIdsEguKC15KnyQdfx4RTc5sjH'.

When information is collected on the machine, it is immediately sent back to the C&C server via the path '/news/userpanel.php'. A fake HTTP referrer, 'www1.google.com', is used throughout the C&C communications.

When there is a need to generate a random domain name via a domain generation algorithm (DGA), the malware contacts the master C&C server first to retrieve some configuration data via the path '/news/users.php'.

In the payload of the most recently distributed Shifu, with a compilation date of 06 Oct 2015, we have noticed a subtle update: it no longer connects to the C&C server on machines that are found to have Man-in-the-Middle (MitM) interception for HTTPS connections. It makes this check by comparing the certificate's MD5 fingerprint with those of some well-known websites (see Figure 7):

- microsoft.com

- dropbox.com
- twitter.com
- sendspace.com
- etrade.com
- facebook.com
- instagram.com
- github.com
- icloud.com
- python.org

```
  BYTE *v2; // eax@4
  BYTE *v3; // eax@5
  BYTE *v4; // eax@6
  BYTE *v5; // eax@7
  BYTE *v6; // eax@8
  BYTE *v7; // eax@9
  BYTE *v8; // eax@10
  BYTE *v9; // eax@11
  char result; // [sp+0h] [bp-20Ch]@2
  char v12; // [sp+104h] [bp-108h]@2
  int v13; // [sp+208h] [bp-4h]@1

  v13 = 1;
  if ( check_internet_connectivity() )
  {
    decode_string(&result, (int)microsoft_com, 0xDu, -122);
    v0 = decode_string(&v12, (int)g_CertHashMicrosoft, 0x20u, -122);
    if ( !is_same_certhash((int)&result, (const char *)v0) )
    {
      decode_string(&result, (int)dropbox_com, 0xBu, -122);
      v1 = decode_string(&v12, (int)g_CertHashDropbox, 0x20u, -122);
      if ( !is_same_certhash((int)&result, (const char *)v1) )
      {
        decode_string(&result, (int)twitter_com, 0xBu, -122);
        v2 = decode_string(&v12, (int)&g_CertHashTwitter, 0x20u, -122);
        if ( !is_same_certhash((int)&result, (const char *)v2) )
        {
          decode_string(&result, (int)sendspace_com, 0xDu, -122);
          v3 = decode_string(&v12, (int)&g_CertHashSendSpace, 0x20u, -122);
          if ( !is_same_certhash((int)&result, (const char *)v3) )
          {
            decode_string(&result, (int)etrade_com, 0xAu, -122);
            v4 = decode_string(&v12, (int)&g_CertHashEtrade, 0x20u, -122);
            if ( !is_same_certhash((int)&result, (const char *)v4) )
            {
              decode_string(&result, (int)facebook_com, 0xCu, -122);
              v5 = decode_string(&v12, (int)&g_CertHashFacebook, 0x20u, -122);
              if ( !is_same_certhash((int)&result, (const char *)v5) )
              {
                decode_string(&result, (int)instagram_com, 0xDu, -122);
                v6 = decode_string(&v12, (int)&g_CertHashInstagram, 0x20u, -122);
                if ( !is_same_certhash((int)&result, (const char *)v6) )
                {
                  decode_string(&result, (int)github_com, 0xAu, -122);
                  v7 = decode_string(&v12, (int)&g_CertHashGithub, 0x20u, -122);
                  if ( !is_same_certhash((int)&result, (const char *)v7) )
                  {
                    decode_string(&result, (int)icloud_com, 0xAu, -122);
                    v8 = decode_string(&v12, (int)&g_CertHashIcloud, 0x20u, -122);
                    if ( !is_same_certhash((int)&result, (const char *)v8) )
                    {
                      decode_string(&result, (int)python_org, 0xAu, -122);
                      v9 = decode_string(&v12, (int)&g_CertHashPython, 0x20u, -122);
                      if ( !is_same_certhash((int)&result, (const char *)v9) )
                        v13 = 0;
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
  return v13;
}
00007595 check ssl mitm cert:59
```

Figure 7. Checking the existence of MitM interception on HTTPS.

The purpose for this update is believed to be to avoid the malware's SSL traffic being intercepted and analysed by researchers and by intrusion detection systems that typically have SSL traffic inspection capabilities.

One browser plug-in kills them all

It seems that Shifu's author feels nostalgic about the *PhishWall* anti-phishing solution from *SecureBrain*. All third-party browser plug-ins will be disabled immediately via a single registry value, 'Enable Browser Extensions', located in HKCU\Software\Microsoft\Internet Explorer\Main, if *PhishWall* is found to be installed as an *Internet Explorer* plug-in.

This makes sense, given that Shifu was first found to be actively spread in Japan, as *SecureBrain* is a Japanese security provider. Moreover, it appears that Shifu's author is cautious with the *SecureBrain* solution and does not want to create obvious noise – for example by disabling all browser plug-ins – that could easily alert non-tech-savvy victims.

In addition to disabling third-party *IE* plug-ins, it also disables the pop-up blocker in *IE* through a registry key:

```
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\Zones\3
"1406" = 0
```

Botnet-like banking trojan

Last but not least, Shifu also allows the botnet operator to download and execute additional modules, hence it supports a limited set of commands. The following is a list of commands currently supported by the malware:

webinject

Webinjection to the targeted Internet banking sites is carried out through a local *Apache* server installed silently by Shifu. In the underlying code, we realized that the botnet operators issue mitm_mod and mitm_script commands to download a copy of the *Apache* web server software as well as the webinjection script and its associated configuration file, config.xml. When these files are in place on the infected machine, the malware will first modify and make sure the *Apache* server is properly configured by setting the server's address to localhost (127.0.0.1) using a random port for both HTTP and HTTPS. After the server's configuration file is set, the server's process will be started and its status will be monitored periodically through an asynchronous thread every second to make sure it keeps running.

Webinjection through the local HTTP server won't work without setting up a hook on the browser process. However, the hook implementation is simpler than that of traditional banking trojans. Table 4 shows a summary of the hooks on the Winsock API. Besides hooking the Winsock API, it also hijacks the SSL verification API used by the browser process so that the verification is always successful regardless of whether the presented SSL certificates are valid.

Hook Windows API	Purpose
ws2_32!connect ws2_32!connectEx	Redirect all the HTTP/HTTPS traffic to a local HTTP server to carry out man-in-the-middle operation
crypt32!CertVerifyCertificateChainPolicy nss3.dll!SSL_AuthCertificateHook nspr4.dll!SSL_AuthCertificateHook	Always return success when the browser's SSL verification process takes place

Table 4. Hooks on the Winsock API.

wipe_cookies

It is assumed that Shifu is mainly distributed via Flash-based exploit kits. This module allows the botherder to make the botnet clean up Flash cookies found in the %APPDATA%\Macromedia folder to cover the presence of the exploited Flash files.

update

Like most software, the botnet also supports automatic updates. As the malware uses a trivial protection method to prevent the removal of the malware file – through an opened file handle – it must first close the file handle before being able to replace the new binary file.

load

This allows for the execution of arbitrary executables downloaded from the C&C server.

kill_os

The real reason why the botnet supports the self-destruction feature (which destroys both the malware and the operating system) is still a mystery to us. But based on the nature of the botnet – it copies lots of ideas and codes from different notorious malware – and the fact that it tries to evade analysis by both auto-analysis systems and manual analysis, we can safely assume that the self-destruction routine will be executed when it is found to be executing on an unwilling platform.

The self-destruction routine (see Figure 8) is pretty straightforward:

- Remove all the files attached to removable drives, for instance thumb drives.
- Corrupt the filesystem (e.g. NTFS/FAT) by overwriting its boot sector.
- Shut down the machine.
- In case the shutdown operation is not successful, it terminates itself.

```
14   char Buffer[260]; // [sp+20Ch] [bp-310h]@1
15   CHAR FileName; // [sp+310h] [bp-20Ch]@9
16   char szWindowsSystem32; // [sp+414h] [bp-108h]@7
17   DWORD NumberOfBytesWritten; // [sp+518h] [bp-4h]@2
18
19   SetErrorMode(1u);
20   hThread = GetCurrentThread();
21   SetThreadPriority(hThread, 2);
22   v1 = GetLogicalDriveStringsA(0x104u, Buffer);
23   v2 = v1;
24   if ( v1 )
25   {
26     NumberOfBytesWritten = 0;
27     if ( v1 )
28     {
29       v3 = 0;
30       do
31       {
32         v4 = &Buffer[v3];
33         if ( GetDriveTypeA(&Buffer[v3]) == DRIVE_REMOVABLE )
34           recur_find_file_and_delete(v4);
35         NumberOfBytesWritten += 4;
36         v3 = (unsigned __int16)NumberOfBytesWritten;
37       }
38       while ( (unsigned __int16)NumberOfBytesWritten < v2 );
39     }
40     GetSystemWindowsDirectoryA(&szWindowsSystem32, 0x104u);
41     v5 = alloca(12);
42     v6 = (const char *)&v10;
43     if ( &v10 )
44       v6 = decode_string(&v10, (int)C_ROOT, 0xAu, 0x91);
45     snprintf(&FileName, 0x104u, v6, szWindowsSystem32);
46     hNtfsBootSec = CreateFileA(&FileName, 0xC0000000, 3u, 0, 3u, 0, 0);
47     if ( hNtfsBootSec != (HANDLE)-1 )
48     {
49       NumberOfBytesWritten = 0;
50       memset(szEmptyByte, 0, sizeof(szEmptyByte));
51       SetFilePointer(hNtfsBootSec, 0x200, 0, 0);
52       WriteFile(hNtfsBootSec, szEmptyByte, 0x200u, &NumberOfBytesWritten, 0);
53       FlushFileBuffers(hNtfsBootSec);
54       CloseHandle(hNtfsBootSec);
55     }
56   }
57   if ( !shut_down() )
58   {
59     elevate_SeDebugPrivilege();
60     v8 = GetDesktopWindow();
61     GetWindowThreadProcessId(v8, &NumberOfBytesWritten);
62     if ( NumberOfBytesWritten )
63       terminate_itself(NumberOfBytesWritten);
64   }
65   return self_destructor();
66 }

00010D59 self_destructor:61
```

Figure 8. Shifu's self-destruction routine.

Conclusion

In conclusion, Shifu is an enhanced or improved piece of banking malware that has borrowed a lot of techniques from its predecessors; it rectified and refined the weaknesses possibly found in other renowned competitors. The author clearly has a good understanding of how to deal with thread synchronization in multi-threaded applications – which could indicate that he/she is an experienced programmer. However, the use of some old-school techniques in Shifu can be easily spotted and blocked by many security products.

Bibliography

Appendix

Table A1 shows the sample SHA1 used in the analysis.

Compilation timestamp	Dropper's SHA1	X86 Payload's SHA1
18 August 2015	963BFC778F94FE190FDD1DD66284E9BC9DD2BED6	16E4476146511F6B9D8DDF4B232D896D7EC91F
06 October 2015	B4ED692D6E8C35F3C611084E6785972CCAE8DCDC	8FC58220FD84F3A59F20D52F4A07F0765747446

Table 5. Sample SHA1 used in the analysis.

```
shifu_fix_iat.py

import idaapi
import idautils

# Global variables
IMG_BASE = idaapi.get_imagebase()
list_seg = []
for seg in idautils.Segments():
 list_seg.append(seg)
IMG_END = idc.SegEnd(list_seg[len(list_seg)-1])

def decrypt(ea, key):

 # Virtual address to IMAGE_IMPORT_DESCRIPTOR->FirstThunk
 va_iat = 0
 # Virtual address to IMAGE_IMPORT_DESCRIPTOR->OriginalFirstThunk
 va_int = 0
 tmp_ea = ea

 # Back-tracing to locate the IMAGE_IMPORT_DESCRIPTOR from import address table passed from the callback
 for xref in idautils.XrefsTo(ea, 0):
  if XrefTypeName(xref.type) == 'Data_Offset':
   va_iat = xref.frm - 0x10

 if va_iat != 0:
```

```
  print “Import Name Table->%08x” % (idaapi.get_long(va_iat) + IMG_BASE)
  va_int = idaapi.get_long(va_iat) + IMG_BASE
 else:
  return

 if va_int != 0:
  va_itd = idaapi.get_long(va_int)
  # Enumerate array of IMAGE_THUNK_DATA
  while va_itd != 0:
   va_itd = va_itd + IMG_BASE
   if va_itd > IMG_BASE and va_itd <= IMG_END:
    print «Image thunk data->%08x» % va_itd
    va_ibn = va_itd + 2
    ch = idaapi.get_byte(va_ibn)
    str = ‘’
    while ch != 0 and ch != 255:
   str += chr(ch ^ key)
   va_ibn += 1
   ch = idaapi.get_byte(va_ibn)

   # Save the decoded import name
   print «IMAGE_IMPORT_BY_NAME->Name (%08x): %s» % (va_itd+2, str)
   idc.MakeName(tmp_ea, str)
   tmp_ea += 4

  # Next IMAGE_THUNK_DATA
  va_int += 4
  va_itd = idaapi.get_long(va_int)
 else:
  return

def imp_cb(ea, name, ord):
 if not name:
  print «%08x: ord#%d» % (ea, ord)
 else:
  print «%08x: %s (ord#%d)» % (ea, name, ord)

 # The decrypt function will be responsible to enumerate IMPORT_DESCRIPTOR_TABLE to decode all the function na
 decrypt(ea, 0xFF)
 # We only want to callback once for every imported DLL
 return False

# Main
nimps = idaapi.get_import_module_qty()

for i in xrange(0, nimps):
 name = idaapi.get_import_module_name(i)
 if not name:
  print «Failed to get import module name for #%d» % i
  continue

 print «Walking-> %s» % name
 idaapi.enum_import_names(i, imp_cb)
```

```
print «All done...»
```

Source: https://www.virusbulletin.com/virusbulletin/2015/11/shifu-rise-self-destructive-banking-trojan