Hodprot: Hot to Bot

Eugene Rodionov, Malware Researcher, ESET

Aleksandr Matrosov, Senior Malware Researcher, ESET

Dmitry Volkov, Cyber crimes investigation division leader, Group-IB

CONTENTS

1 Banking fraud in Russia

As discussed in our presentation at CARO2011 on "Cybercrime in Russia: Trends and issues", the number of Russian cybercrimes related to financial fraud and stealing money from bank accounts increased significantly in the last year. Moreover we can see accelerated growth in the number of cybercrimes related to banking fraud in the second half of 2011. The most common malware families involved in incidents of banking fraud in Russia are:

- Win32/Carberp
- Win32/Shiz
- Win32/Hodprot
- Win32/Sheldor
- Win32/RDPdoor

Here are the major regions of distribution of these banking Trojans:

1. Russia
2. Ukraine
3. Kazakhstan

Attackers have focused on these countries because similar banking software and mechanisms for financial transactions are in use there. In the late spring and early summer of 2011, according to statistics of incidents provided by Group-IB, one of the most-used families of malware is Win32/Hodprot. This is an interesting family of Trojans which merits further discussion: it implements many sophisticated algorithms and anti-forensic mechanisms.

2 Win32/Hodprot Overview

The Win32/Hodprot family of malware seems to have been undergoing something of resurgence, as indicated in our previous blog article at http://blog.eset.com/2011/07/18/hodprot-is-a-hotshot. Statistics on bank fraud provided by Group-IB (our partners in Russia dealing with cybercrime investigation) and our own research indicate that the bot was particularly active in the spring of 2011.

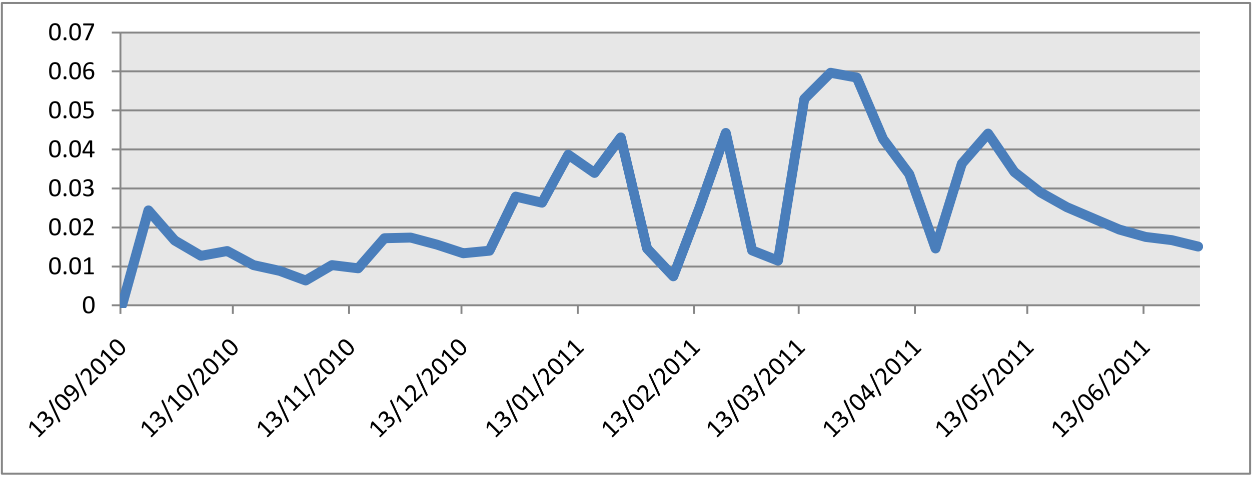


Figure 1 – Win32/Hodprot Detection Statistics From ThreatSense.NET (01.01.2010-30.06.2011)

This family supports additional modules and trojan programs that target the most popular online Russian banking systems (BSS, IBank, Inist). This suggests that Win32/Hodprot targets Russia and the former Soviet republics.

Figure 2 shows the distribution of the bot by region:

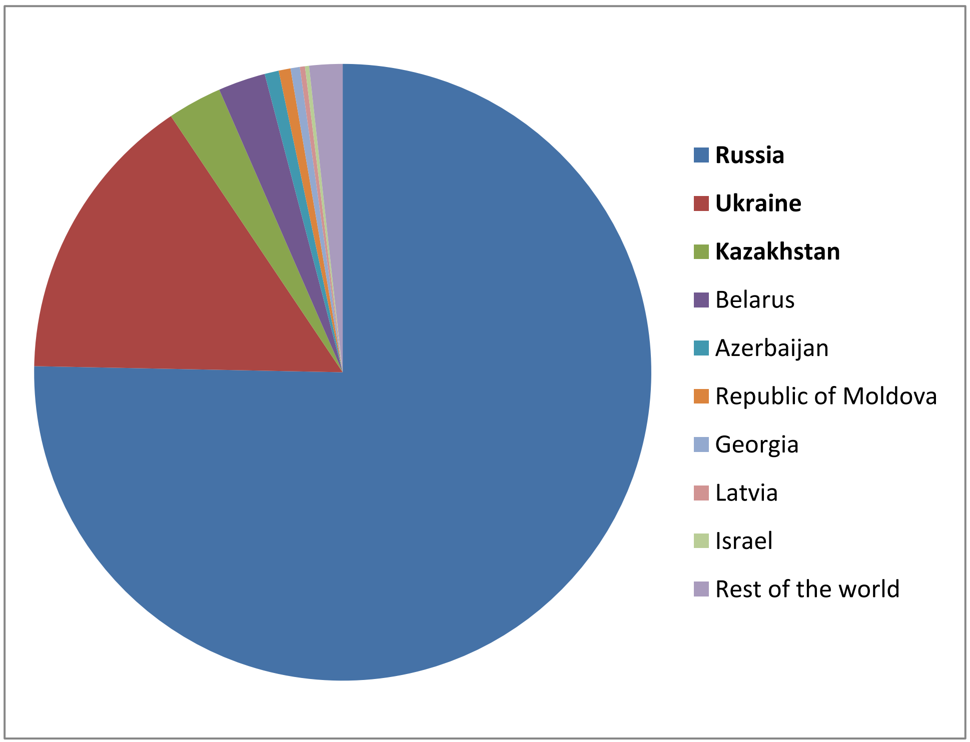


Figure 2 – Win32/Hodprot Statistics by Region From ThreatSense.NET (01.01.2010-30.06.2011)

In all cases of bank fraud involving Win32/Hodprot, a great deal of money has been stolen. T These fraudulent operations have, on average, netted several hundred thousand USD.

Even more interestingly, the Win32/Hodprot botnet is associated with other botnets specializing in bank fraud in Russia and its neighbours. Notably, Win32/Hodprot was used to download Win32.Sheldor, Win32/RDPdoor and Win32/Platcyber onto the victims' machines. The period of time when Win32/Sheldor and Win32/RDPdoor appear to have been particularly active matches that of Win32/Hodprot and shows similar distribution patterns (see Figure 3 and Figure 4).

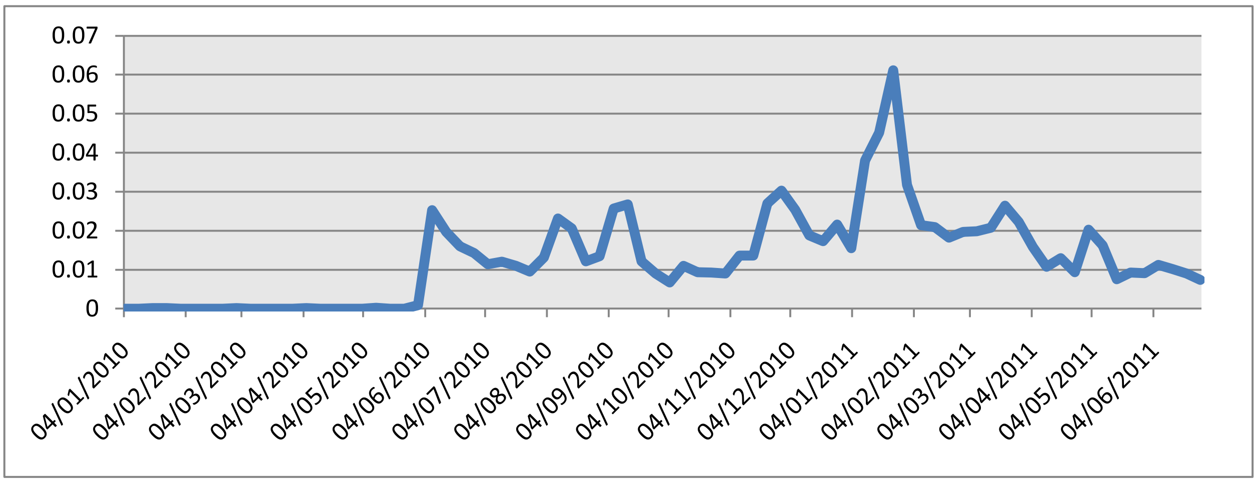


Figure 3 – Win32/Sheldor Statistics by Region From ThreatSense.NET (01.01.2010-30.06.2011

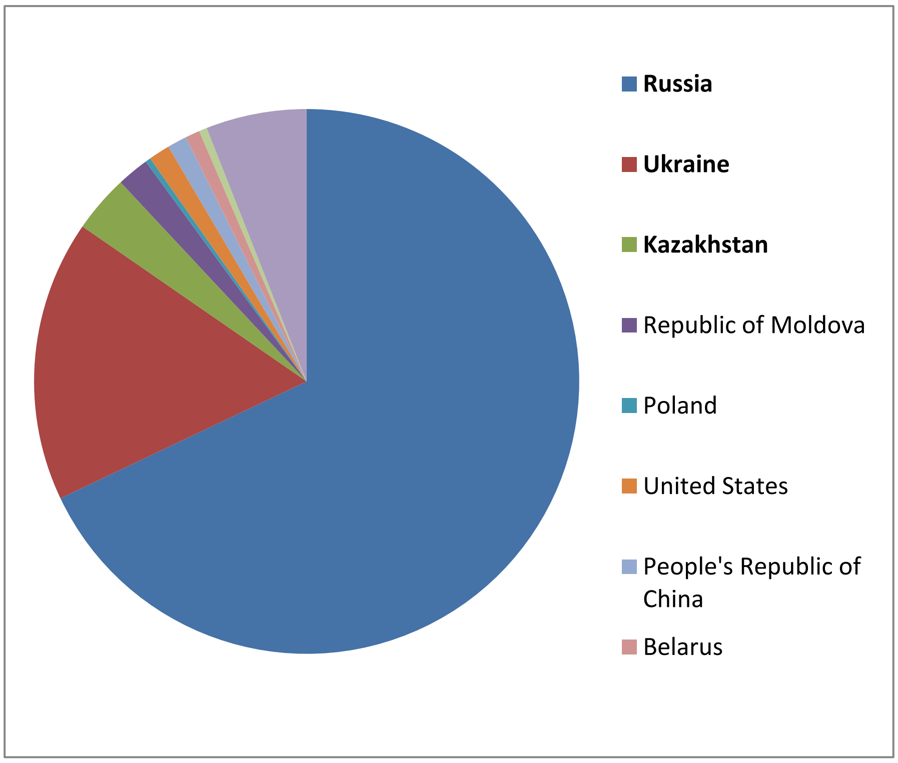


Figure 4 – Win32/Sheldor Statistics by Region From ThreatSense.NET (01.01.2010-30.06.2011)

3 Installation Details

When Win32/Hodprot is executed by a user the first thing it does is to install its components onto the system. Instead of immediately connecting to the C&C server and requesting a payload to download it penetrates deeply into the system by installing several modules that help it stay undetected by antivirus software, and bypass IDS systems while communicating with C&C servers. The next figure represents an overview of the installation algorithm described in depth later:

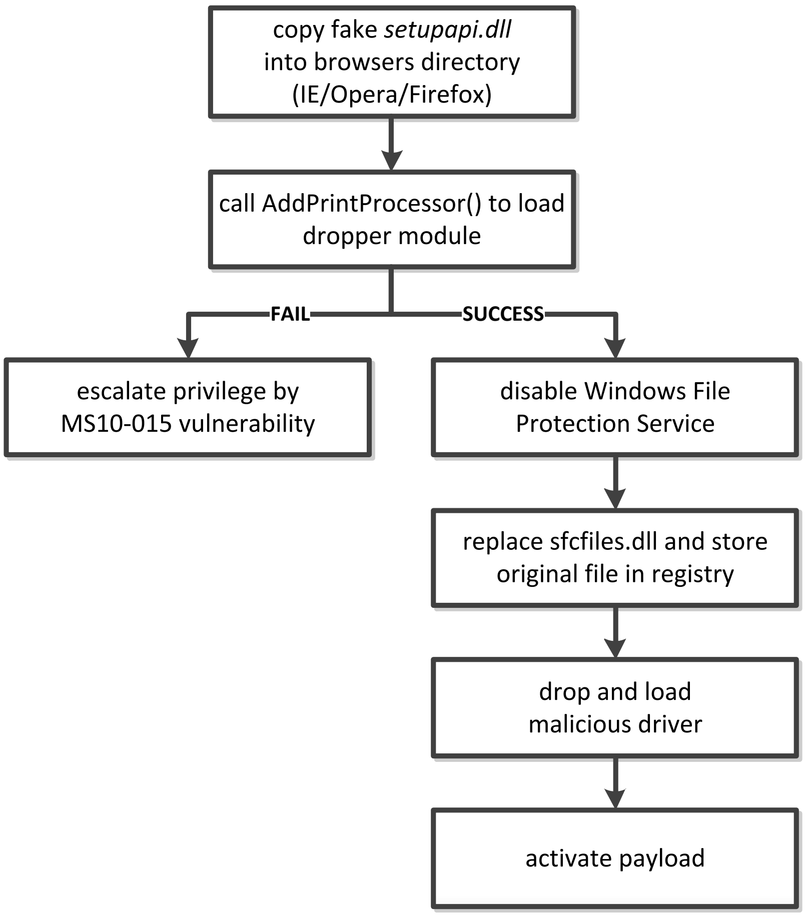

copy fake setupapi.dll into browsers directory (IE/Opera/Firefox)
call AddPrintProcessor() to load dropper module
FAIL
SUCCESS
escalate privilege by MS10-015 vulnerability
disable Windows File Protection Service
replace sfcfiles.dll and store original file in registry
drop and load malicious driver
activate payload

Figure 5 – Win32/Hodprot Installation Algorithm

When the malware is initially run it drops one of its components as a file named *setupapi.dll* in the following directories:

- *C:\Program Files\Internet Explorer*
- *C:\Program Files\Opera\*
- *C:\Program Files\Mozilla Firefox*

Then it tries to register itself as a print processor by calling *AddPrintProcessor* API so that it can perform the rest of the job within the context of the trusted system process *spoolsv.exe*. It allows the malware to bypass HIPS systems and AV software since all the actions of the bot appear to be performed by the system print spooler. You may recall that this trick was originally employed by the TDL4 rootkit (aka Win32/Olmarik). In addition, adding itself as a print processor ensures that Win32/Hodprot is executed each time the system starts up.

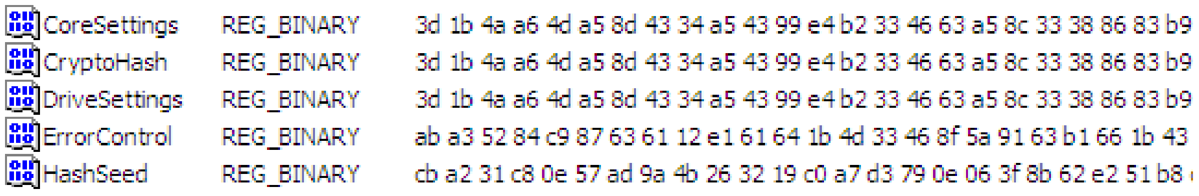
After being loaded into the *spoolsv.exe* address space, the Win32/Hodprot bot continues the infection process by modifying *sfcfiles.dll* file in the system directory. This is part of Windows File Protection mechanism (Windows File Protection is used to prevent software from modifying system protected executable). The original *sfcfiles.dll* is stored in the *CryptoHash* value of the *HKLM\SOFTWARE\Settings* registry key. Then the malicious *sfcfiles.dll* overwrites the original so that the size of the file isn't changed and the file time attributes are also preserved.

After modifying *sfcfiles.dll* the bot installs driver *sfc.sys* in the system in a rather unusual way: it doesn't create a corresponding service in the system. Usually malware installs a kernel-mode driver by creating a corresponding service entry in the registry and then executing the *ZwLoadDriver* routine exported from *ntdll.dll*, which is often monitored by HIPS software. In order to bypass this kind of protection Win32/Hodprot loads the driver by executing *ZwSetSystemInformation,* an undocumented native API routine passing as its first parameter the *SystemAddDriver* (0x26) constant, and as its second parameter the path to the driver. The driver's image is stored in the *DriveSettings* value of the *HKLM\SOFTWARE\Settings* registry key.

```
NTSTATUS
NtSetSystemInformation
                    (
                    IN SYSTEM_INFORMATION_CLASS SystemInformationClass,
                    IN PVOID SystemInformation,
                    IN ULONG SystemInformationLength
                    );
```

As a result the driver is loaded into kernel-mode address space and executes its entry point. Being executed with highest possible privileges only then the driver registers itself as a service with start type SERVICE_SYSTEM_START which signifies that the driver will be loaded during OS startup process.

The heart of the bot – the main module which communicates with C&C servers and downloads the payload – is never stored as a file in the file system. Instead the bot keeps its encrypted binary in the system registry key *HKLM\SOFTWARE\Settings* (see Figure below).



```
CoreSettings    REG_BINARY    3d 1b 4a a6 4d a5 8d 43 34 a5 43 99 e4 b2 33 46 63 a5 8c 33 38 86 83 b9
CryptoHash      REG_BINARY    3d 1b 4a a6 4d a5 8d 43 34 a5 43 99 e4 b2 33 46 63 a5 8c 33 38 86 83 b9
DriveSettings   REG_BINARY    3d 1b 4a a6 4d a5 8d 43 34 a5 43 99 e4 b2 33 46 63 a5 8c 33 38 86 83 b9
ErrorControl    REG_BINARY    ab a3 52 84 c9 87 63 61 12 e1 61 64 1b 4d 33 46 8f 5a 91 63 b1 66 1b 43
HashSeed        REG_BINARY    cb a2 31 c8 0e 57 ad 9a 4b 26 32 19 c0 a7 d3 79 0e 06 3f 8b 62 e2 51 b8
```

Figure 6 – Win32/Hodprot's registry keys

Here is a list of values of the key created by the malware during installation process:

- *ErrorControl* – contains encrypted loader code
- *CoreSettings* – contains encrypted main binary
- *HashSeed* – contains list of encrypted URLs to communicate to C&C.

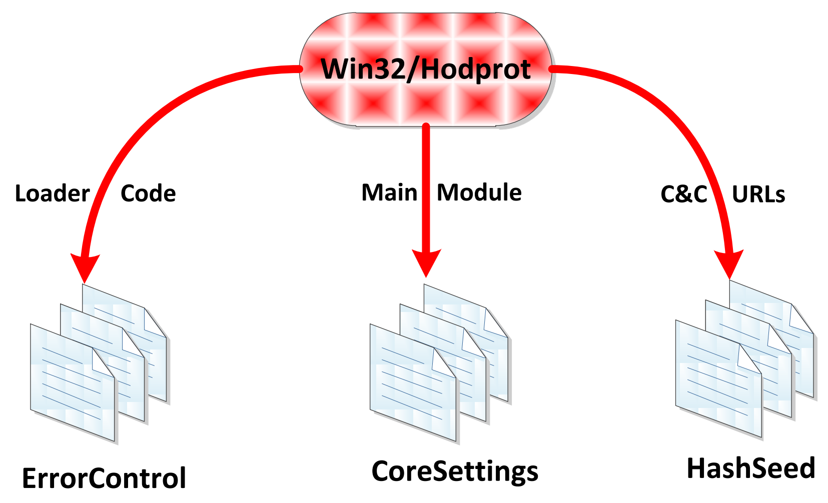


Figure 7 – Win32/Hodprot's Components

All these values are encrypted with a custom encryption algorithm. This consists of sequential XOR-ing and ROR-ing of each byte with the corresponding byte of the key. The key is generated based on the information obtained from the *ProductId* value of the registry key *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion*.

After the driver installation. in order to activate modules already dropped, the dropper executes the following command:

rundll32 url.dll, FileProtocolHandler http://www.google.com

This starts the default web browser and visits the google web page.

4 The Dropped Modules

As described in the previous section, during the installation Win32/Hodprot drops several modules to enable it to inject its code into the web browser's process address space and survive a reboot. In this section each of the dropped modules is analyzed.

4.1 Setupapi.dll – DLL Hijacking

The main purpose of the dropped library *setupapi.dll* is to execute the main bot binary in the context of the web browser. This module is dropped into directories corresponding to the following browsers: IExplorer, Opera and Firefox. When a user launches the browser executable, *setupapi.dll* is loaded into its address space due to the import dependencies. Win32/Hodprot employs quite an elegant method of injecting its code into the address space of the target process; let's look at how it works. The Windows PE image loader starts searching for the module's dependencies within the current directory of the process. If it fails to find the target library the loader proceeds to search directories listed in the PATH environment variable, which is certain to include the system directory. Since the process's current directory contains the malicious *setupapi.dll*, then instead of loading the legitimate *setupapi.dll* from the system directory, the malicious DLL is loaded. This technique is often referred to as DLL Hijacking.

When the malicious *setupapi.dll* is loaded in the browser address space it reads information from the registry so as to assemble the payload to execute in the context of the running process. In the figure below you can see the layout of the payload as it appears in the process address space:

Loader Code (ErrorControl)	**Main Module (CoreSettings)**	**Encrypted URLs (HashSeed)**	**Additional Modules (PnPData)**

Figure 8 – The Injected Payload Layout

It begins with the loader code which is read from the *ErrorControl* value of the *HKLM\SOFTWARE\Settings* registry key and is intended to load the main bot module: map the image, apply relocations, initialize import address table, call its entry point and execute its exported routine *DllRegisterServer*. After executing the payload the malicious *setupapi.dll* loads the original library from the system directory.

The main module in turn implements the bot functionality and uses a list of URLs to communicate with C&C servers. In addition there might be an extra module to load into the address space of the current process. The main module is described in detail in the corresponding section.

4.2 Sfcfiles.dll – Checking Kernel-mode Driver

Sfcfiles.dll is part of Windows File Protection and contains a list of files to be protected against modification. This library is loaded into the address space of the *winlogon.exe* process – this is a trusted system process launched at system start up. By overwriting this file malware thus ensures that it will be loaded into the *winlogon.exe* address space each time the Operating System starts.

Winlogon executes the routine *SfcGetFiles* exported from the *sfcfiles.dll* library to get the list of the files for which to enforce Windows File Protection, of which *sfcfiles.dll* is a component. The Windows File Protection mechanism prevents certain system files from being modified or removed by other software running on the machine. In order to conceal itself from the protection mechanism the modified library performs the following tasks on entry into *SfcGetFiles*: it loads the original library (recall that the original *sfcfiles.dll* is kept in the *CryptoHash* value of the *HKLM\SOFTWARE\Settings* registry key), executes its *SfcGetFiles* routine and excludes itself (*%systemroot%\system32\sfcfiles.dll*) from the returned list. This means that the system doesn't detect *sfcfiles.dll* modification.

The main purpose of the modified *sfcfile.dll* is to load the kernel-mode driver *sfc.sys* if it isn't loaded yet. If for some reason it can't be loaded, then *sfcfiles.dll* connects to the C&C servers and requests the bot's updated modules from C&C servers listed in the *HashSeed* registry value.:

- loader code – (*ErrorControl* value)
- main module –(*CoreSettings* value)
- kernel-mode driver (*DriveSettings* value)

In the event that it fails to obtain C&C URLs from registry *sfcfiles.dll*, it requests updates from URLs that are hardcoded into the binary:

- http://fg****ks.com/sign/page.php
- http://fy****ts.com/sign/page.php

4.3 Sfc.sys – Kernel-mode Injector

This kernel-mode driver essentiallyimplements the same functionality that the dropped *setupapi.dll* does. Its main purpose is to inject the payload into the address space of the processes corresponding to web browsers. It does so by registering *LoadImageNotifyRoutine.* The payload is injected into processes containing one (or more) of the following libraries:

- *wininet.dll* (Windows Internet Explorer)
- *ws2_32.dll* (Windows Socket Implementation)
- *iertutil.dll* (Windows Internet Explorer)
- *msvbvm60.dll* (Visual Basic Virtual Machine)

When a process is started the driver waits until the *ntdll.dll* library is loaded, then it allocates a buffer for the payload and maps it into user-mode address space. Upon loading one of the aforementioned libraries the driver splices the process's executable entry point so that the payload will be executed.

In the next figure you can see how the described modules are interconnected. From this we can conclude that the central component of Win32/Hodprot is the system registry: this is where the bot's critical components are kept. The approach of using registry as storage for malicious components makes forensic analysis more difficult.

The use of several of the dropped components to inject the payload make Win32/Hodprot a very resilient malicious program anddifficult toremove..

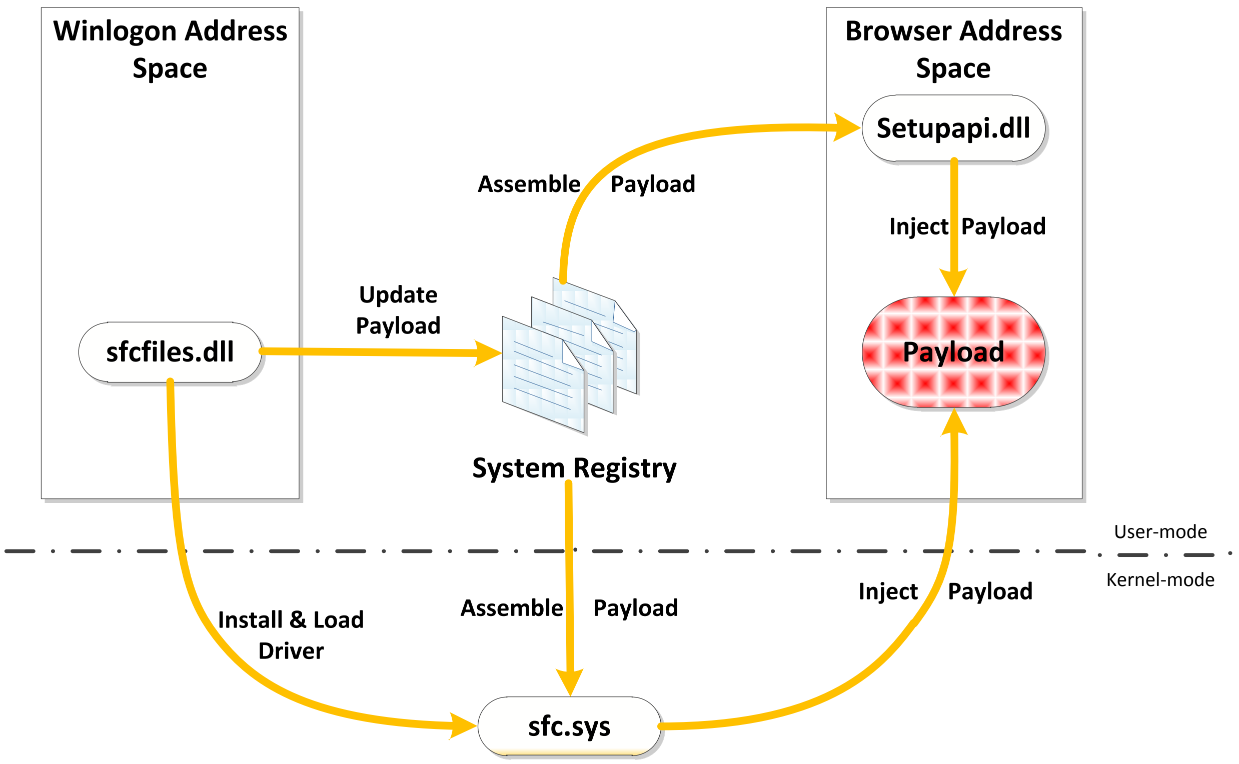
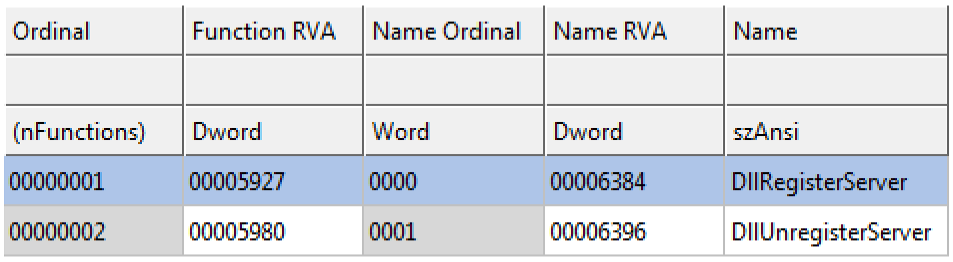


Figure 9 –Win32/Hodprot Dropped Modules Interconnection

5 The bot

5.1 The main module

The general purpose of the bot is to download malware to the infected system and run it. The main module of the bot is a DLL which in addition to its entry point exports two routines:



Ordinal	Function RVA	Name Ordinal	Name RVA	Name
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00005927	0000	00006384	DllRegisterServer
00000002	00005980	0001	00006396	DllUnregisterServer

Figure 10 – The Bot's Main Module Export Address Table

The core functionality is implemented in the *DllRegisterServer* routine. When the main module of the bot is executed it checks modules loaded into the current process address space for the presence of the following sequence of bytes:

0x41 , 0x00, 0x56, 0x00. 0x5a, 0x00, 0x00, 0x00

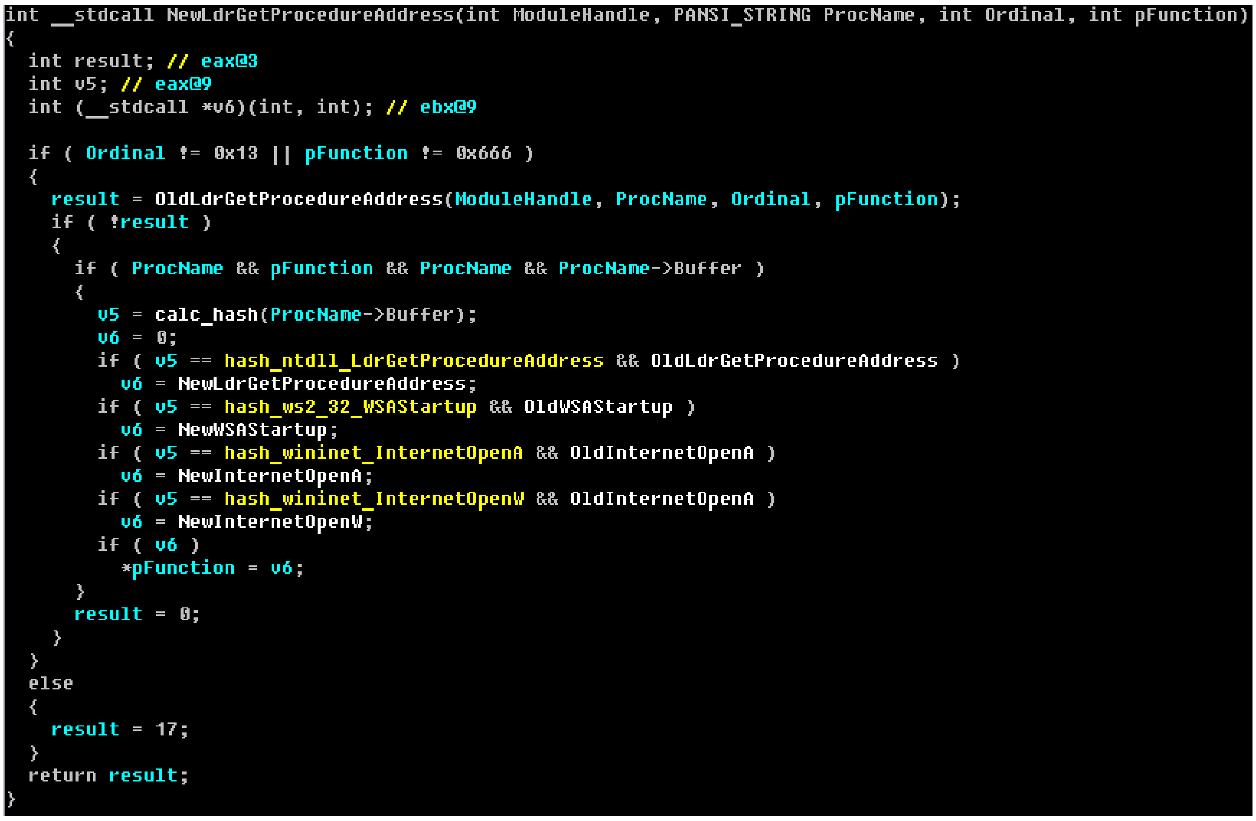
This can be interpreted as the Unicode string "*AVZ\0*". If it finds a module matching this signature it simply returns control to the caller. At the next step it checks whether the current process is already injected with the module by calling *LdrGetProcAddress* routine from *ntdll.dll* library:

```
NTSTATUS
LdrGetProcAddress
               (
               IN ULONG_PTR ImageBase,
               IN ULONG ImageSize,
               IN PSZ SymbolName,
               OUT PVOID *SymbolAddress
               )
```

It passes the following parameters:

LdrGetProcAddress(NULL, 0, 0x13, 0x666)

If the returned value equals 0x11 then there is an instance of the main module in the current process and control is returned to the caller (see Figure 11).



```
int __stdcall NewLdrGetProcedureAddress(int ModuleHandle, PANSI_STRING ProcName, int Ordinal, int pFunction)
{
  int result; // eax@3
  int v5; // eax@9
  int (__stdcall *v6)(int, int); // ebx@9

  if ( Ordinal != 0x13 || pFunction != 0x666 )
  {
    result = OldLdrGetProcedureAddress(ModuleHandle, ProcName, Ordinal, pFunction);
    if ( !result )
    {
      if ( ProcName && pFunction && ProcName && ProcName->Buffer )
      {
        v5 = calc_hash(ProcName->Buffer);
        v6 = 0;
        if ( v5 == hash_ntdll_LdrGetProcedureAddress && OldLdrGetProcedureAddress )
          v6 = NewLdrGetProcedureAddress;
        if ( v5 == hash_ws2_32_WSAStartup && OldWSAStartup )
          v6 = NewWSAStartup;
        if ( v5 == hash_wininet_InternetOpenA && OldInternetOpenA )
          v6 = NewInternetOpenA;
        if ( v5 == hash_wininet_InternetOpenW && OldInternetOpenA )
          v6 = NewInternetOpenW;
        if ( v6 )
          *pFunction = v6;
      }
      result = 0;
    }
  }
  else
  {
    result = 17;
  }
  return result;
}
```

Figure 11 – The hook for LdrGetProcAddress routine

If the process isn't injected with the bot's main module then the malware loads and reads the list of C&C URLs according to the image of the main module in memory and additional modules obtained from *PnPData* in the *HKLM\SOFTWARE\Settings* registry key.

In the event that all the checks are passed successfully the bot starts downloading the malware.

5.2 Network communication

To download the payload the bot once each day runs through the following list of URLs:

- http://fa****g.org/action/page.php
- http://fa****g.net/ad/page.php
- http://fa****g.info/pic/page.php
- http://fa****r.biz/info/page.php
- http://fa****l.com/forum/page.php

The download request to C&C server is encapsulated with the following information:

- Bot Identifier;
- A 32-bit integer.

Here is what the request looks like when sent to C&C:

http://target_url?para_1=value_1&para_2=value_2&...&para_N=value_N

N is a randomly chosen integer from 5 to 9, *para_X* is a string randomly chosen from the following list:

id, cookie, n, session, hl, client, lr, article, key, do, page, query, uid, var, link.

value_X may be randomly chosen from the following list:

rand, rnd, query, opera, ie, mozzila, a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, unknown, en, us, index, title;

Alternatively, it may be a bot ID sent to the server or the 32-bit integer.

These manipulations of the parameters are intended to prevent recognizable patterns in requests sent to the C&C and being detected IDS systems. The assembled URL is passed as second parameter to the *URLDownloadToCacheFileA* API function. The following diagram shows communication between the bot and the C&C server.

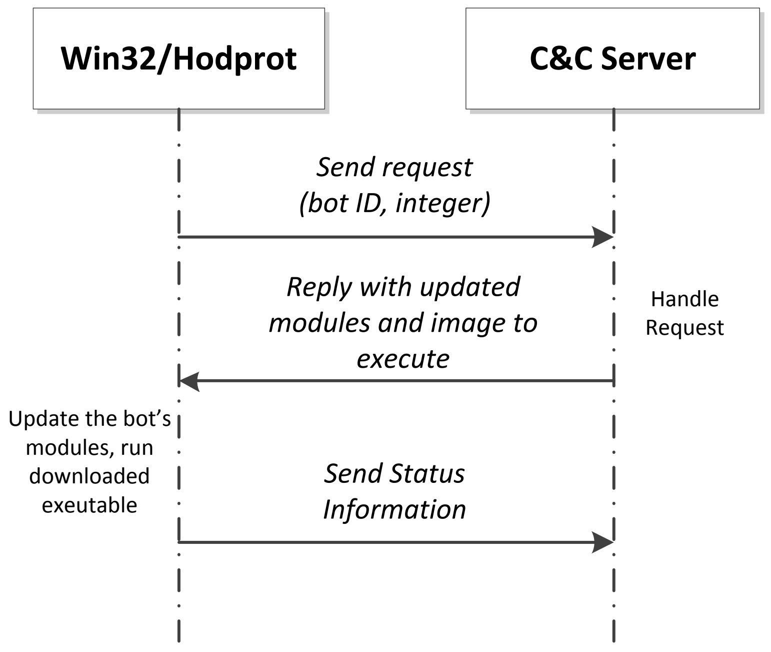

Win32/Hodprot
C&C Server
Send request (bot ID, integer)
Reply with updated modules and image to execute
Handle Request
Update the bot's modules, run downloaded exeutable
Send Status Information

Figure 12 – Communication Workflow between the Bot and the C&C Server

In case of success the bot receives the data structure as follows:

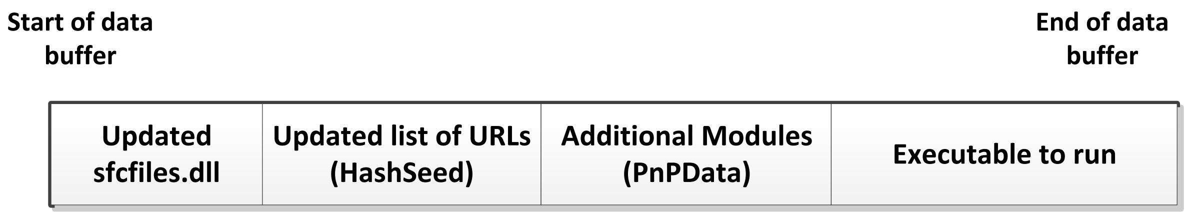


Figure 13 – The Downloaded Data From C&C Servers

The downloaded data start with an updated *sfcfiles.dll*, followed by the list of URLs and extra modules to be injected into processes, along with the main bot module. All the data blocks in buffer are compressed with the aPLib library.

The last item in the received data buffer is the executable to be launched. The executable is copied into a file in a temporary directory and run by executing the *CreateProcessA* routine. The status information relating to the running of the downloaded module is stored in the registry key *HKLM\Software\Microsoft\Windows* and subsequently sent to C&C server.

Afterword

Win32/Hodprot is an outstanding example of a persistent threat which delivers other malware onto the victim's machine. It was specifically designed to withstand forensic analysis and bypass security systems. It is involved in many incidents of banking fraud in Russia in conjunction with other banking Trojans which are proving profitable for cybercrime groups and they are thus motivated to heavily invest in the malware developing process. All this makes Win32/Hodprot very dangerous threat.