Festi botnet analysis and investigation

Aleksandr Matrosov, Eugene Rodionov

Keywords: Festi, spam, botnet, rootkit, DDoS, OOP, HIPS, firewall

Abstract. The botnet Festi has been in business since the autumn of 2009 and is currently one of the most powerful and active botnets for sending spam and performing DDoS attacks [1]. Festi is an interesting and untypical malware family implementing rootkit functionality with strong protection against reverse engineering and forensic analysis. It is capable of bypassing sandboxes and automated trackers using some advanced techniques such as inserting timestamps in its communication protocol, detecting virtual machines, and subverting personal firewalls and HIPS systems [2].

The bot consists of two parts: the dropper, and the main module, the kernel-mode driver, which is detected by ESET as Win32/Rootkit.Festi. The malware's kernel-mode driver implements backdoor functionality and is capable of:

- Updating configuration data from the C&C (command and control server);
- Downloading additional dedicated plugins.

In our presentation we will concentrate on the latest Festi botnet update from June 2012 and offer comprehensive information gleaned from our investigations, furnishing details on developers of the botnet and reverse engineering of the bot's main components – the kernel-mode driver and the plugins (DDoS, Spam). The presentation starts with a description of our investigation and an account of how the Festi botnet evolved over time. We will present a binary analysis kernel-mode driver and downloaded plugins – volatile kernel-mode modules which aren't saved on any storage device in the system, but in memory, making forensic analysis of the malware significantly more difficult. The presentation also covers such aspects of Festi as its ability to bypass personal firewalls and HIPS systems that may be installed on the infected machine. We will give details of the Festi network communication protocol architecture, based on using the TCP/IP stack implementation in the Microsoft Windows Operating System to communicate with C&C servers, send spam and perform DDoS attacks. And finally, we will describe several self-protective features and techniques of the botnet communication protocol used to bypass sandboxes and trackers.

First published in the AVAR 2012 Conference Proceedings

Introduction

The botnet Win32/Festi started up in business in autumn of 2009 and at the present time it is one of the most active botnets, sending spam and performing DDoS attacks. The bot consists of two parts: the dropper and the kernel-mode driver – the main module - which is detected by ESET as Win32/Rootkit.Festi. In 2009 and in the beginning of 2010 the bot was leased out for spam distribution, but later it was only used for the benefit of major partners in spamming. Nowadays this is one of the most powerful spam botnets and is included among the three most active spam botnets all over the world, according to statistics from M86 Security Labs (Fig. 1).

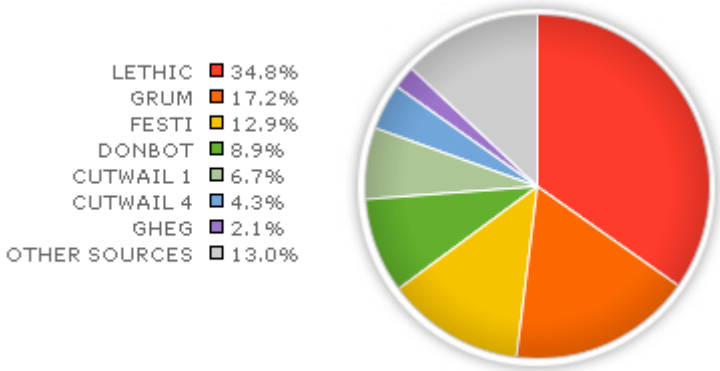

Figure 1 – Statistics of generated spam for each spambot type

In the autumn of 2011 the botnet migrated its C&C servers to new domain names (Fig. 2). All the previously-used domains are still alive and are being kept in reserve in case the primary domain/servers don't respond.

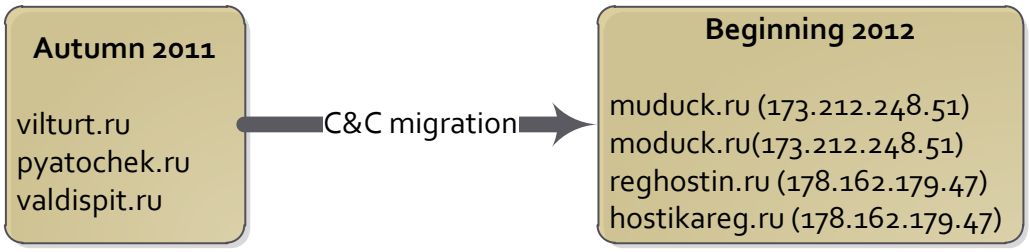

Figure 2 – Migration scheme to new domain names

The botnet periodically migrates to new hosting and domain names in order to decrease the rate at which it is detected using C&C URLs and corresponding IP addresses. The bot's binary contains only C&C domain names with no IP addresses.

The previous versions of the bot communicated with C&C servers over HTTP (Hypertext Transfer Protocol) by encrypting POST requests. At the beginning of 2012 an updated version of the bot employed a new communication protocol which is capable of bypassing IPS and IDS systems operating at the network layer. In this report we analyze the latest version of the bot which appeared in February, and is detected by ESET products as Win32/Rootkit.Festi.

We haven't seen the DDoS bot being leased out and at the present time it is used only for targeted attacks. For instance, one such attack, performed by means of the bot, targeted Assist, the company that was processing payments for Aeroflot, Russia's largest airline.

Win32/Festi architecture

The malware consists of a single kernel-mode driver which is installed into the system by its dropper. The kernel-mode component is registered in the system as a *SYSTEM_START* kernel-mode driver with a randomly generated name. As a result, it is loaded and receives control in the course of the *IoInitSystem* routine during the system initialization process. In the Fig. 3 the driver's entry point call graph is presented.

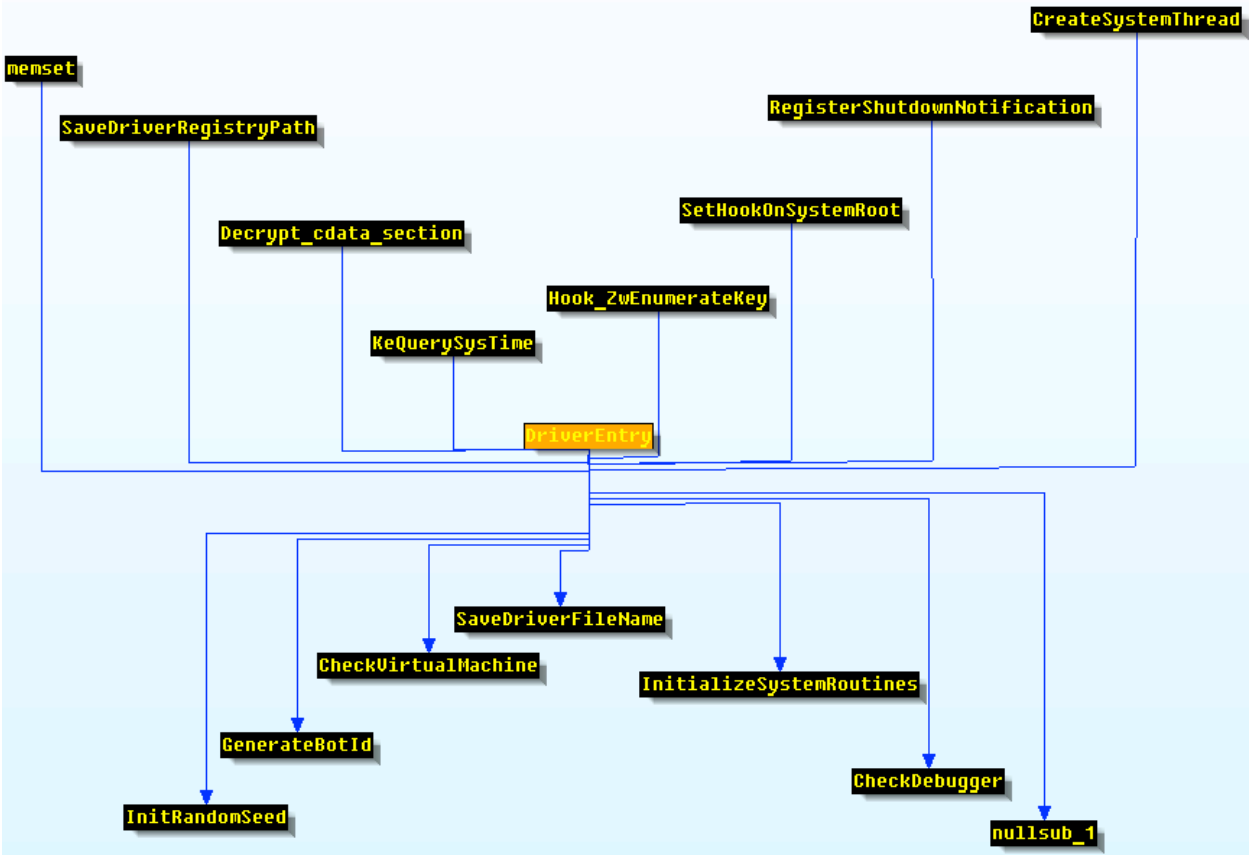

Figure 3 –The call graph of the rootkit driver entry point

The kernel-mode driver implements backdoor functionality and is capable of:

1) Updating configuration data from C&C;
2) Downloading additional dedicated plugins.

It periodically contacts the C&C server and requests plugins and configuration information. The downloaded plugins perform the bot's main job. The plugins are kernel-mode drivers that aren't saved on any storage device in the system and stay volatile in memory. Thus, when the infected computer is switched off or rebooted the plugins have vanished from system memory. This makes forensic analysis of the malware significantly harder since the only file stored on the hard drive is the main kernel-mode driver, and this contains neither the payload nor information regarding which sites to attack or to which to send spam.

Each plugin is dedicated to performing certain kinds of job: namely, performing DDoS attacks against s specified network resource, or sending spam. The plugins communicate with the main

driver through a well-defined interface. The Fig. 4 illustrates how the bot penetrates into the system and performs its malicious activity.

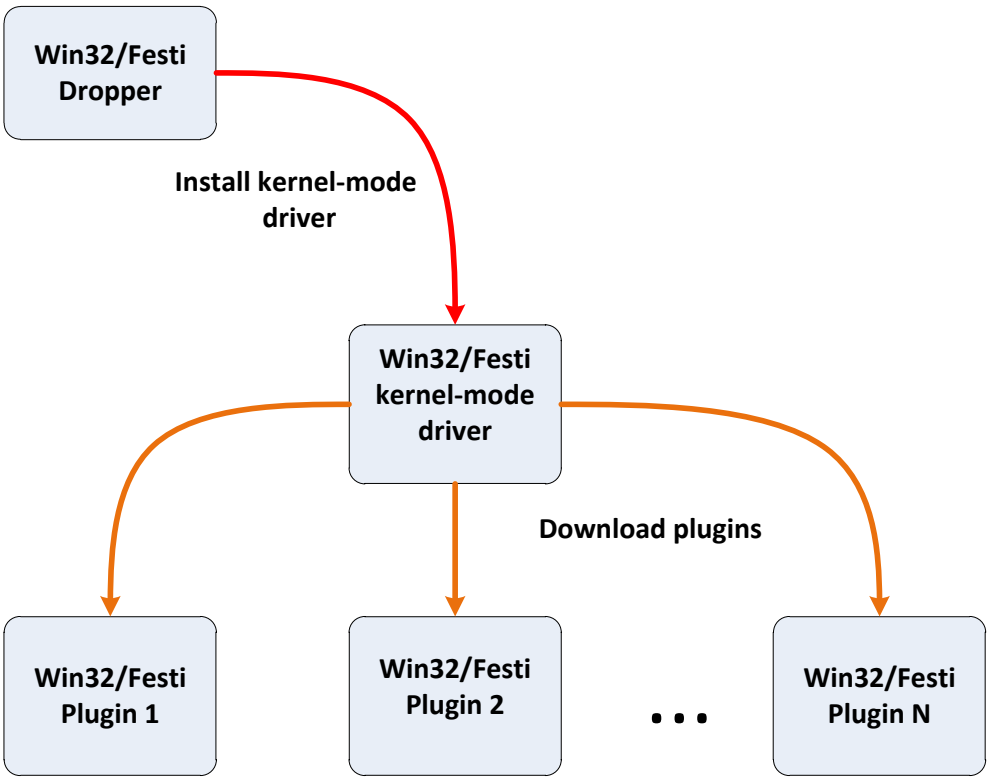

Figure 4 – The bot infection algorithm

OOP Framework

One of the remarkable features of the bot is that its driver is developed using an object oriented programming (OOP) language and has a corresponding architecture. This is something not very common for kernel-mode drivers as they are usually written in plain C. Here is the list of main components (classes) implemented by the malware:

- Memory manager – to allocate/release memory buffers;
- Network sockets – to send/receive data over the network;
- C&C protocol parser – to parse C&C messages and execute received commands;
- Plugin manager – to efficiently manage downloaded plugins;

The interconnection of the components listed above is presented in the Fig. 5. As we can see Memory Manager is the central component used throughout the bot.

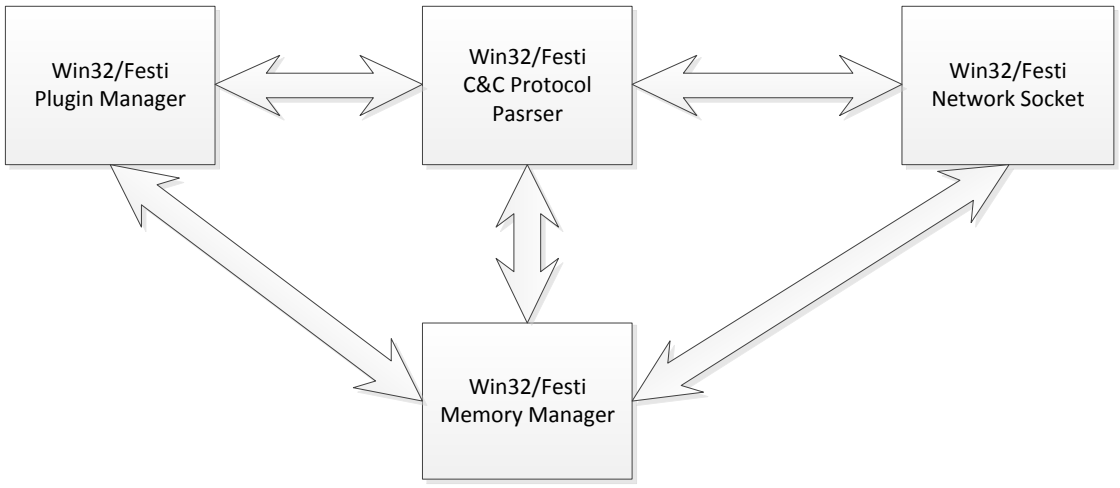

Figure 5 –Win32/Festi object oriented architecture

The design principles of the malware make it extremely portable to other platforms like Linux, for instance. The system-specific code is isolated by the component's interface and may be easily changed to support other platforms. For instance, downloaded plugins that are dedicated to performing a specific task rely almost completely on the interfaces provided by the main module. They rarely use routines provided by the system to do system-specific operations.

Managing plugins

To be able to manage downloaded plugins efficiently the bot maintains an array of pointers to a specially defined structure. The structure describes a plugin and provides the bot with specific entry points for plugins – routines responsible for handling data received from C&C (Fig. 6).

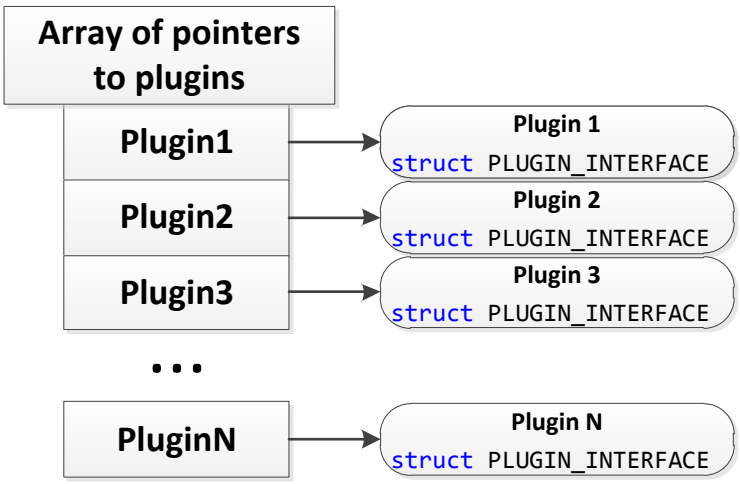

Figure 6 – The plugins interconnection

Below you can see the layout of the structure describing the interface that a plugin should make available to the main module:

```
struct PLUGIN_INTERFACE
{
    // Initialize plugin
    PVOID Initialize;
    // Release plugin, perform cleanup operations
```

```
    PVOID Release;
    // Get plugin version information
    PVOID GetVersionInfo_1;
    // Get plugin version information
    PVOID GetVersionInfo_2;
    // Write plugin specific information into tcp stream
    PVOID WriteIntoTcpStream;
    // Read plugin-specific information from tcp stream and parse data
    PVOID ReadFromTcpStream;
    // Reserved fields
    PVOID Reserved_1;
    PVOID Reserved_2;
};
```

When the bot transmits data to the C&C server it runs through the array of pointers to the plugin interface and executes the *WriteIntoTcpStream* routine of each registered plugin passing a pointer to a TCP stream object as a parameter. On receiving data from the C&C server the bot executes the plugins' *ReadFromTcpStream* routine, so that the registered plugins can get parameters and plugin-specific configuration information from the network stream. As a result the data sent over the network are structured as described in Fig. 7.

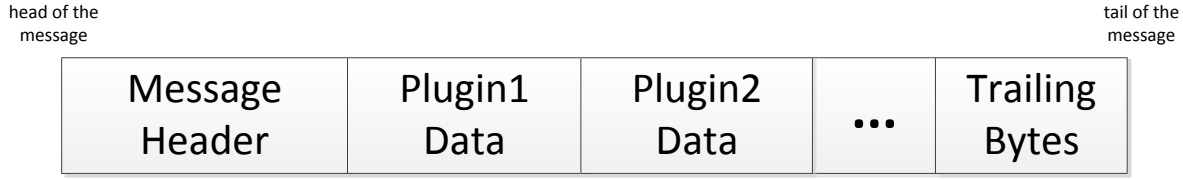

Figure 7 – C&C network packet layout

Built-in plugins. When the bot is installed into the system, the main kernel-mode driver already contains two built-in plugins, namely:

- The configuration information manager;
- The bot plugin manager.

Configuration manager. This plugin is responsible for requesting configuration information from the C&C server.

Plugin manager. The plugin is responsible for maintaining an array of downloaded plugins for the bot. It is able to load/unload a specific plugin onto the system when it receives a remote command. It receives compressed plugins from the C&C server. Each plugin is a DLL exporting two routines:

- *PLUGIN_INTERFACE *CreateModule(PVOID DriverInterfaces);*
- *VOID DeleteModule().*

The *CreateModule* routine is executed on plugin initialization and returns a pointer to the interface described above. It takes as a parameter a pointer to the set of interfaces provided by the main module. The plugin uses these interfaces to interact with the main module and C&C servers over the network. The *DeleteModule* routine is executed when the plugin is unloaded and is used to free all the previously allocated resources. On the Fig. 8 you can see a description of the algorithm for loading a downloaded plugin:

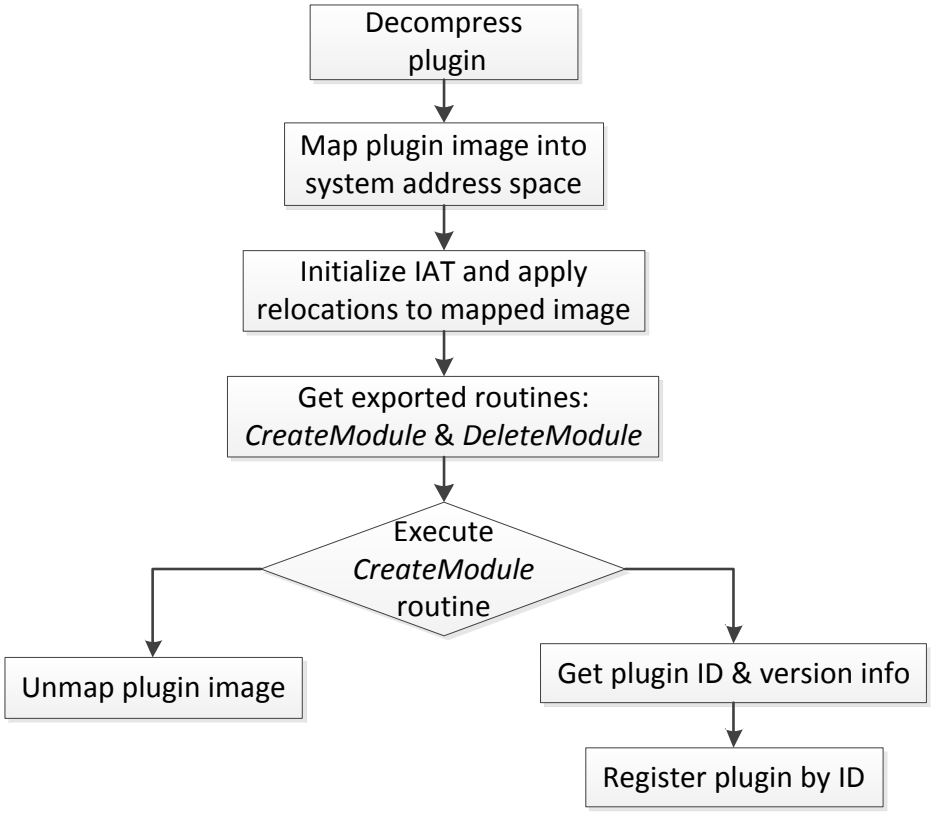

Figure 8 – The algorithm of loading plugins

Bypassing security software and anti-forensics

Personal firewalls & HIPS. One of the interesting features of Win32/Festi is that it is able to bypass personal firewalls and HIPS systems installed on the infected machine. So as to be able to communicate over the network with C&C servers and send spam and perform DDoS attacks, it relies on a TCP/IP stack implemented in the Microsoft Windows Operating System in kernel-mode. By doing it this way the malware is able to send TCP/UDP packets and at the same time isn't burdened with constructing packets of these types manually (as is the case with NDIS drivers that operate at link layer).

In order to send/receive packets the malware opens *\Device\Tcp* or *\Device\Udp* devices depending on the protocol type being used. Most personal firewalls and HIPS systems intercept *IRP_MJ_CREATE_FILE* requests sent to the transport driver on opening these devices. This allows the security software to ascertain who (i.e. which process) is going to communicate over the network. Generally speaking, there are two ways of achieving this:

- Hooking the *ZwCreateFile* system service handler to intercept all attempts to open the devices;
- Attaching to *\Device\Tcp* or *\Device\Udp* in order to intercept all the IRP requests sent.

Let's see how Win32/Festi bypasses both techniques to establish connection with a remote host over the network.

Instead of using a system implementation of the *ZwCreateFile* system service it implements its own service, with almost the same functionality as the original. The Fig. 9 describes the custom implementation of the *ZwCreateFile* routine:

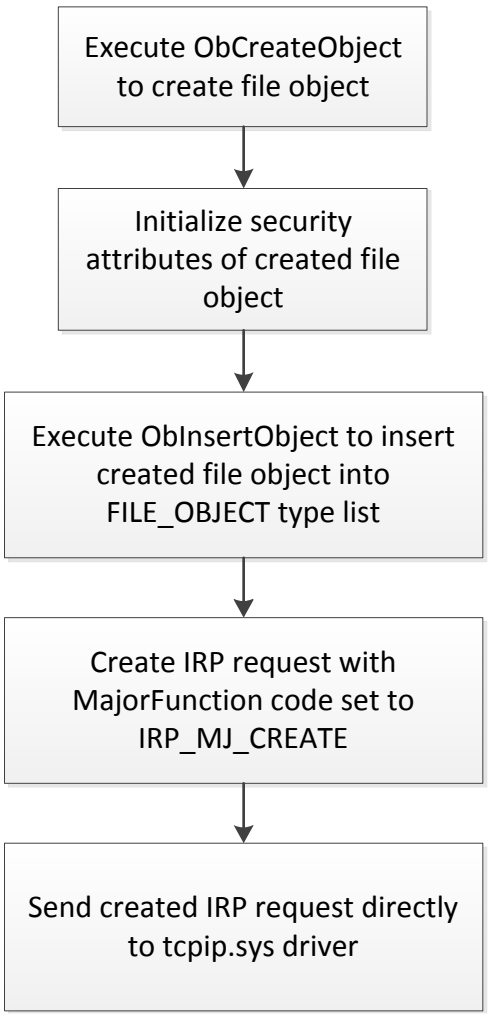

Figure 9 – Custom implementation of ZwCreateFile system routine

From the figure we can see that Win32/Festi manually creates a file object to communicate with the device being opened and sends an *IRP_MJ_CREATE* request directly to the transport driver. Thus, all the devices attached to *\Device\Tcp* or *\Device\Udp* will miss the request and as a result this operation is unnoticed by security software. This is clarified in the Fig. 10.

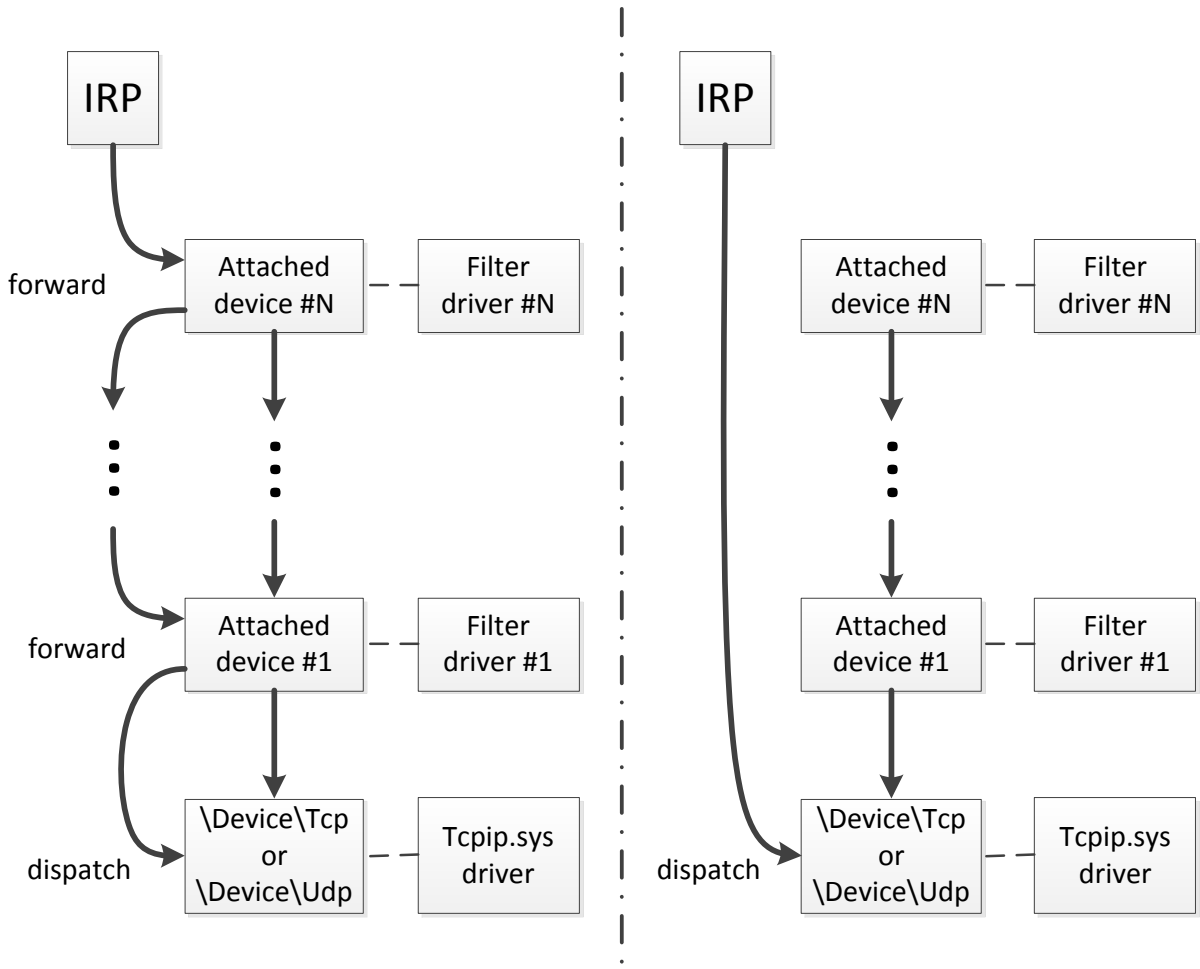

Figure 10 – Bypassing filter drivers attached to network stack

So as to be able to send the request directly to *\Device\Tcp* or *\Device\Udp* the malware requires pointers to corresponding device objects. It obtains a pointer to the *tcpip.sys* driver object by executing

```
NTSTATUS
ObReferenceObjectByName (
IN PUNICODE_STRING ObjectName,
IN ULONG Attributes,
IN PACCESS_STATE AccessState OPTIONAL,
IN ACCESS_MASK DesiredAccess OPTIONAL,
IN POBJECT_TYPE ObjectType,
IN KPROCESSOR_MODE AccessMode,
IN OUT PVOID ParseContext OPTIONAL,
OUT PVOID *Object
);
```

This is an undocumented system routine passing it as parameter pointer to a Unicode string with the target driver name. Then the malware iterates through the list of device objects corresponding to the driver object and compares its names with *"\Device\Tcp"* or *"\Device\Udp"*. The fragment of code responsible for this maneuver is presented in the Figure 11.

```
_this = this;
if ( !this->DeviceTcp )
{
  if ( this->tcp_drv_ver < 6u )
    RtlInitUnicodeString(&DriverName, L"\\Driver\\Tcpip");
  else
    RtlInitUnicodeString(&DriverName, L"\\Driver\\tdx");
  RtlInitUnicodeString(&tcp_name, L"\\Device\\Tcp");
  RtlInitUnicodeString(&udp_name, L"\\Device\\Udp");
  if ( !ObReferenceObjectByName(
          &DriverName,
          64,
          0,
          0,
          0x1F01FFu,
          IoDriverObjectType,
          0,
          0,
          &TcpipDriver) )                    // get pointer to DRIVER_OBJECT by its name
  {
    ms_exc.disabled = 0;
    DevObj = TcpipDriver->DeviceObject;
    _DevObj = TcpipDriver->DeviceObject;
    while ( DevObj )                         // iterate through DEVICE_OBJECT linked list
    {
      if ( !ObQueryNameString(DevObj, &Objname, 256, &v8) )// Get DEVICE_OBJECT name
      {
        if ( RtlCompareUnicodeString(&tcp_name, &Objname, 1u) )// Check names
        {
          if ( !RtlCompareUnicodeString(&udp_name, &Objname, 1u) )
          {
            ObfReferenceObject(DevObj);
            _this->DeviceUdp = DevObj;       // Save pointer to \Device\Udp
            DevObj = _DevObj;
          }
        }
        else
        {
          ObfReferenceObject(DevObj);
          _this->DeviceTcp = DevObj;         // Save pointer to \Device\Tcp
        }
      }
      _DevObj = DevObj->NextDevice;          // get pointer to next DEVICE_OBJECT in the list
      DevObj = DevObj->NextDevice;
    }
    ms_exc.disabled = -2;
    ObfDereferenceObject(TcpipDriver);
  }
  this = _this;
}
return this->DeviceTcp != 0;
```

Figure 11 –Acquiring pointers to \Device\Tcp or \Device\Udp objects

When the malware obtains a handle for the opened device as described above, then it uses the handle to send/receive data over the network. Although the malware manages to avoid security software, we can see packets sent by the malware with network traffic filters operating at a lower level (NDIS level) than Win32/Festi.

Detecting Virtual Machines. Win32/Festi detects whether it is running inside a VMware virtual machine. It employs a rather well-documented technique: it executes the following instructions (Fig. 12).

```
mov eax, 'VMXh'
mov ebx, 0
mov ecx, 0Ah
mov edx, 'VX'
in  eax, dx
```

Figure 12 – Detecting VMware virtual environment

If the code is executed inside a VMware virtual environment the *ebx* register will contain the 'VMX' dword.

Anti-debugging. Win32/Festi also checks for the presence of a kernel debugger in the system by examining the *KdDebuggerEnabled* symbol. It also periodically zeroes debugging registers so as to remove the hardware breakpoint, if any (Fig. 13).

```
char __thiscall ProtoHandler_1(STRUCT_4_4 *this, PKEVENT a1)
{
  _EDX = 0;
  __asm
  {
    mov     dr0, edx
    mov     dr1, edx
    mov     dr2, edx
    mov     dr3, edx
  }
  return _ProtoHandler(&this->struct43, a1);
}
```

Figure 13 –Anti-debugging trick

OOP Reversing problem

In this section we would like to highlight some of the problems that researchers face while reversing object-oriented code. Currently, there is a lot of malware written in C++ and other high-level languages which employ an object oriented approach to implementing complex logic. Take, for instance, such threats as Duqu or Stuxnet that are overloaded with object oriented structures. As the size and number of implemented objects in the code grows, then the task of reversing is more and more challenging.

In programs written in procedural languages like C it is usually straightforward to build a control flow graph until special measures are taken to obfuscate it. Although C supports dynamic pointers to routines which complicate the whole thing, normally it's quite easy to ascertain control transfer direction.

In the case of code written in object-oriented languages, the task of building a control flow graph is not so easy. For instance, virtual functions, which implement polymorphism in the C++ language, are called by pointers. This is depicted in the Figure 14:

```
pagecode:0001BC80 68 09 46 00 00     push    4609h
pagecode:0001BC85 58                 pop     eax
pagecode:0001BC86 8B D0              mov     edx, eax
pagecode:0001BC88 8B 4F 0C           mov     ecx, [edi+0Ch]
pagecode:0001BC8B 8B 01              mov     eax, [ecx]
pagecode:0001BC8D 8D 55 F8           lea     edx, [ebp-8]
pagecode:0001BC90 52                 push    edx
pagecode:0001BC91 FF 77 10           push    dword ptr [edi+10h]
pagecode:0001BC94 FF 50 0C           call    dword ptr [eax+0Ch]
pagecode:0001BC97 84 C0              test    al, al
pagecode:0001BC99 75 0E              jnz     short loc_1BCA9
pagecode:0001BC9B 8B CF              mov     ecx, edi        ; this
pagecode:0001BC9D E8 5E FF FF FF     call    CloseTcpSocket
pagecode:0001BCA2 B8 CF 44 00 00     mov     eax, 44CFh
pagecode:0001BCA7 50                 push    eax
pagecode:0001BCA8 5B                 pop     ebx
```

Load pointer to table of virtual methods

Call virtual method

Figure 14 –Calling an object's method by pointer

Given this information, it is difficult to obtain the exact address of the routine being called. Static analysis doesn't provide a researcher with information as to the location register that eax points to. To be able to get the address, one needs to figure out where the object of specified type is created. At the time of its creation an object is initialized with a pointer to a table of virtual methods like this (Fig. 15).

```
STRUCT_4_3 *__thiscall CSocket_Ctor(STRUCT_4_3 *this)
{
  STRUCT_4_3 *v1; // esi@1

  v1 = this;
  this->vTable = &csocket_v_table;
  this->struct44 = 0;
  this->DeviceTcp = 0;
  this->DeviceUdp = 0;
  sub_19664(&this->struct41);
  sub_13E22(&v1->struct2);
  v1->SocketNumber = 0;
  v1->RefNo = 0;
  return v1;
}
```

Figure 15 –Socket object constructor

In the figure above you can see a constructor of *CSocket* class implemented in the malware. We can see that its opaque *CSocket::vTable* field is initialized with a pointer to a table of virtual methods which has the following layout (Fig. 16).

Figure 16 –Socket object table of virtual methods

Thus, when we encounter a virtual function call in static analysis, we are unable to get the address of the called routine unless we already have type information for the object. To get this info we need to find where the object is created, and this task is quite challenging. As a result, reversing object oriented code is usually a difficult and time consuming task.

Plugins

During investigation we spotted that different bots download different set of plugins. We managed to identify two sets of bots:

- Spammers – those that send spam;
- DDoS – the bots designated to perform DDoS attacks.

Spam module (BotSpam.dll). This plugin is responsible for sending junk emails. The plugin receives a list of email addresses to which it should send spam mail, and the actual text for sending. There is nothing unusual about the algorithm for sending spam. The plugin merely runs through the list of recipients and sends mails to corresponding email addresses.

The interesting thing about the plugin is the way it checks the status of sent email. The plugin scans a response from the server for specific string constants signifying that there are problems with sending email (the mail wasn't received or was classified as junk). In the course of the research we obtained two almost identical versions of the plugins where the plugin looks for different strings in the server response. Both sets of strings are used for verifying the server's reply. In the event that the plugin finds any of those strings present in the server's response it stops sending messages to that address and fetches the next address in its list.

DDos module (BotDos.dll). The DDoS plugin allows the bot to perform DDoS attacks against specified hosts. The plugin supports several types of DDoS attacks, depending on configuration data received from the C&C. It is highly configurable and, as a result, may be used to mount attacks on remote hosts with different kinds of software installed, and of different architecture. Here are the types of attack implemented by the plugin:

- TCP flood;
- UDP flood;
- DNS flood;
- HTTP flood.

TCP flood. In the case of TCP flooding the bot initiates by default a large number of connections to

port 80 (the HTTP port) on the target machine. The port to connect to might be changed by corresponding configuration information from the C&C server.

UDP flood. For UDP flooding the bot sends UDP packets of randomly-generated length and filled with random data. The length of the packet lies in the range from 256 up to 1024 bytes. The attack proceeds as follows. The target port is also generated at random and is therefore unlikely to be open. As a result the attack causes the target host to generate enormous amount of ICMP Destination Unreachable packets in reply to UDP requests. Thus, the target machine becomes unavailable.

DNS flood. The bot is also able to perform a DNS flood attack. In such a case it sends high volumes of UDP packets to port 53 (DNS service) on the target host. The packets contain requests to resolve a randomly-generated domain name in the ".com" domain zone.

HTTP flood. Another feature implemented in the bot is an HTTP flood attack against web servers. The bot contains many different user-agent strings in the binary. You can find all the user-agent strings that the bot uses to attack web servers in appendix A of [2]. These strings are used to create a large number of HTTP sessions with the Web server, thus overloading the remote host. The Figure 17 contains code assembling the HTTP request to be sent.

```
user_agent_index = gen_rnd() % 0x64;
str_cpy(http_request, "GET ");
str_cat(http_request, (char *)v4 + 204 * *(_DWORD *)(v2 + 4) + 2796);
str_cat(http_request, " HTTP/1.0\r\n");
if ( *((_BYTE *)v4 + 2724) & 2 )
{
  str_cat(http_request, "Accept: */*\r\n");
  str_cat(http_request, "Accept-Language: en-US\r\n");
  str_cat(http_request, "User-Agent: ");
  str_cat(http_request, user_agent_str[user_agent_index]);
  str_cat(http_request, "\r\n");
}
str_cat(http_request, "Host: ");
str_cat(http_request, (char *)v4 + 204 * *(_DWORD *)(v2 + 4) + 2732);
str_cat(http_request, "\r\n");
if ( *((_BYTE *)v4 + 2724) & 2 )
  str_cat(http_request, "Connection: Keep-Alive\r\n");
str_cat(http_request, "\r\n");
result = str_len(http_request);
*(_DWORD *)(v2 + 16) = result;
return result;
```

Figure 17 – Assembling HTTP header to perform DDoS attack

To send the packets the plugin employs network sockets implemented by the main module of the bot. As a result the attack is performed in kernel-mode, which makes it quite stealthy. Also the bot is able to send IP packets with the protocol field set to a random value.

Conclusion

In this paper we present technical analysis of the Festi botnet, one of the most powerful botnets for sending spam and performing DDoS attacks. The bot has several striking features that distinguish it significantly from other malware samples with similar functionality. With its object-oriented architecture incorporated into the kernel-mode driver as well supporting downloadable plugins, it is provided with such features as portability and resistance to reverse engineering. The distinguishing characteristics of the bot such as strong resistance to forensic analysis and its ability to bypass IDS/IPS software make it an efficient weapon in hands of

cybercriminals. It remains one of the biggest spam botnets and, improved with the latest updates, it becomes even more dangerous.

References

[1] B. Krebs, Financial Mogul Linked to DDoS Attacks (2011) // http://krebsonsecurity.com/2011/06/financial-mogul-linked-to-ddos-attacks/

[2] E. Rodionov, A. Matrosov, King of Spam: Festi Botnet Analysis (2011) // http://blog.eset.com/wp-content/media_files/king-of-spam-festi-botnet-analysis.pdf

About the Authors

Aleksandr Matrosov has more than ten years of experience with malware analysis, reverse engineering and advanced exploitation techniques. Currently working at ESET as Senior Malware Researcher since joining the company in October 2009 as a virus researcher, and working remotely from Russia. He has worked as a security researcher since 2003 for major Russian companies. He is also a Lecturer at the Cryptology and Discrete Mathematics department of the National Research Nuclear University in Moscow, and co-author of the research papers "Stuxnet Under the Microscope" and "The Evolution of TDL: Conquering x64", and is frequently invited to speak at major security conferences (including Ekoparty, Recon and Virus Bulletin). Nowadays he specializes in the comprehensive analysis of complex threats, modern vectors of exploitation and research into cybercrime activity.

Eugene Rodionov graduated with honors from the Information Security faculty of the Moscow Engineer-Physics Institute (State University) in 2009 and successfully defended Ph.D. thesis in 2012. He has been working in the past five years for several companies, performing software development, IT security audit and malware analysis. He currently works at ESET, one of the leading companies in the antimalware industry, where he performs analysis of complex threats. His interests include kernel-mode programming, anti-rootkit technologies, reverse engineering and cryptology. He is co-author of the research papers "Stuxnet Under the Microscope" and "TDL3: The Rootkit of All Evil?". Eugene Rodionov also holds the position of Lecturer at the National Nuclear Research University MEPHI in Russia.