Author

Global Research & Analysis Team (GReAT), Kaspersky Lab

miniFlame aka SPE: "Elvis and his friends"

- Introduction
- Executive summary
- The history of SPE discovery
  - Connections between Flame and SPE
  - Connections between Gauss and SPE
- SPE timeline
- SPE workflow
- SPE versions info
- Infection statistics (KSN data)
- Sinkhole statistics
- lcsvnt32.ocx (main module)
  - Export "RegisterService"
- lcsvntu32.ocx (the USB infector)
  - DllMain
  - Data collection routine
  - Disinfection routine
- Conclusions
- References

Introduction

While analyzing the Flame malware [1] that we detected in May 2012, Kaspersky Lab experts identified some distinguishing features of Flame's modules. Based on those features, we discovered that in 2009, the first variant of the Stuxnet worm included [4] a module that was created based on the Flame platform. This indicates that there was some form of collaboration between the groups that developed the Flame and Tilded [5] (Stuxnet/Duqu) platforms.

Based on the results of a detailed analysis of Flame, we continued to actively search for new, unknown components. A more in-depth analysis conducted in June 2012 resulted in the discovery of a new, previously unknown malware which we named Gauss [2]. Gauss uses a modular structure resembling that of Flame, a similar code base and system for communicating with C&C servers, as well as numerous other similarities to Flame.

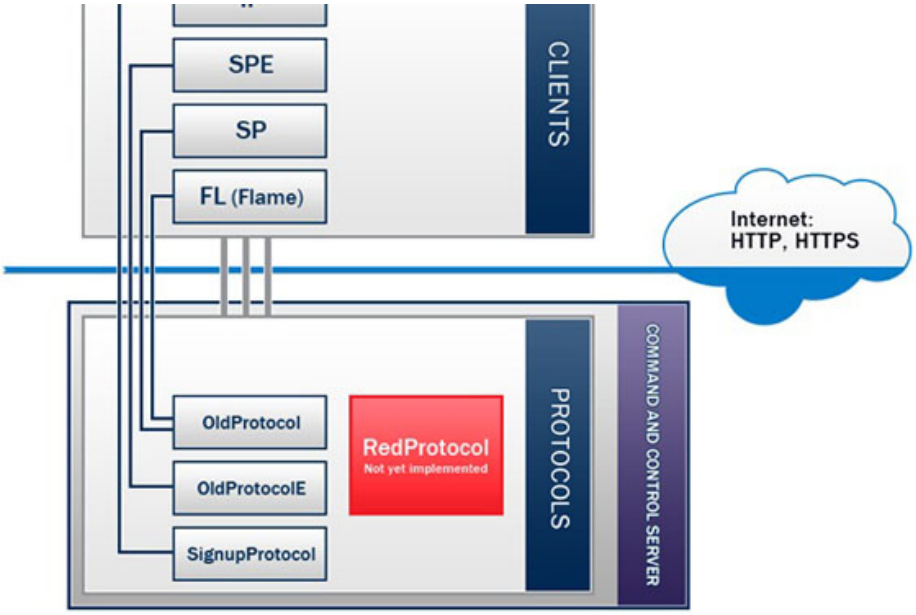
We also published our analysis of the Flame command-and-Control (C&C) servers based on external observations and publicly available information. That helped our understanding of where the C&C servers were located and how they were registered. In September 2012 we are released new information that was collected during forensic analysis of the Flame C&C servers. This investigation was done in partnership with Symantec, ITU-IMPACT and CERT-Bund/BSI.

The analysis of the Flame's C&C modules show that the code can understand several communication protocols to talk to different "clients" or malware:

- OldProtocol
- OldProtocolE
- SignupProtocol
- RedProtocol (mentioned but not implemented)

A close look at these protocol handlers revealed four different types of clients: SP, SPE, FL and IP.

We can confirm that the Flame malware was identified as client type FL. Obviously, this means there are at least three other undiscovered cyber-espionage or cyber-sabotage tools created by the same authors: SP, SPE and IP.



Clients and Protocols relations found in C&C

With help from our partners, Kaspersky Lab set up a sinkhole for Flame. We previously published statistics from the sinkhole data in our previous research papers [3].

Perhaps the most interesting thing is that based on the C&C code of Flame, we were able to catalog the connections received by our sinkhole into two main categories:

- OldProtocol connections, coming from Flame
- OldProtocolE connections, used by SPE

Thus, as of September 2012 we confirmed the existence “in the wild” of at least one unknown program created within the framework of the Flame platform.

In early July 2012, we discovered a smaller Flame module, which appeared to be able to work by itself. The module had many similarities with Flame, so we thought it might simply be an earlier version. In the months that followed, we not only studied the connection of this malware with Flame, but also came across examples of this module being used concurrently with Gauss and being controlled by the Gauss main module. After we analyzed the Flame Command and Control servers, we were surprised to discover this module appeared to use OldProtocolE connections, which are used by the mysterious SPE malware. We therefore understood this small Flame plugin was in fact a standalone malware: the one known as SPE by the Flame C2. This paper covers the history of SPE discovery and the functionality of the malware.

Executive summary

The SPE malware, which we call “miniFlame”, is a small, fully functional cyber-espionage malware designed for data theft and direct access to infected systems.

The miniFlame malware is in fact based on the Flame platform but is implemented as an independent module. It can operate either independently, without the main modules of Flame in the system, or as a component controlled by Flame.

Also it is a notable fact that miniFlame can be used in conjunction with another espionage program, namely Gauss. As many readers will remember, it has been assumed that Flame and Gauss were parallel projects that did not have any modules or C&C servers in common. The discovery of miniFlame, which works with both these espionage projects, proves that we were right when we concluded that they had come out of the same ‘cyber-weapon factory’.

Apparently, the development of miniFlame began several years ago and continued until 2012. Based on the C&C code, protocols serving SP and SPE were created before or at the same time with the communication protocol used by FL (Flame), i.e., in 2007 at the least.

We believe that the developers of miniFlame created dozens of different modifications of the program. At this time, we have “only” found six of these, dated 2010-2011.

In some cases, dedicated C&C servers were used exclusively to control the SPE operation. Concurrently with that, some SPE variants worked with the servers that communicated to Flame.

The miniFlame/SPE malware is different from Flame and Gauss in that the number of infections is significantly smaller. While we estimate the total number of Flame/Gauss victims at no less than 10,000 systems, SPE has been detected only in a few dozen systems in Western Asia. This indicates that SPE is a tool used for highly targeted attacks, which was used only against

dozen systems in Western Asia. This indicates that SPE is a tool used for highly targeted attacks, which was used only against objects having the greatest significance and posing the greatest interest to the attackers.

When we compare the number of SPE infections with those of other programs discovered earlier that have either common components or structures, we get the following figures:

Name	Incidents (KL stats)	Incidents (approx.)
Stuxnet	More than 100 000	More than 300 000
Gauss	~ 2500	~10 000
Flame (FL)	~ 700	~5000-6000
Duqu	~20	~50-60
miniFlame (SPE)	~10-20	~50-60

Unlike Flame, the vast majority of incidents were recorded in Iran and Sudan, and unlike Gauss, which was mostly present in Lebanon, SPE does not have a clear geographical bias. However, we are inclined to believe that the choice of countries depends on the SPE variant. For example, the modification known as “4.50” is mostly found in Lebanon and Palestine. The other variants were found in other countries, such as Iran, Saudi Arabia and Qatar.

The original SPE distribution vector is unknown. However, since it is known to have worked both as part of Flame and as part of Gauss and since it shares its C&C servers with Flame, we believe that in most cases SPE was installed from C&C servers onto systems that were already infected by Flame or Gauss.

It should be noted however that SPE was not on the list of files predefined in known Flame configurations and was not removed by the browse32.ocx module that was distributed by Flame authors in May 2012 in order to uninstall Flame from infected systems.

The history of SPE discovery

Connections between Flame and SPE

Flame implements an interesting configuration structure for the entire malware system, which is organized along the same lines as the system registry in Windows. The configuration defines not only the list and composition of all available modules and files, but also their parameters, lists of temporary files, lists of security programs, etc. The total number of parameters defined in the configuration can reach several thousand.

Our analysis of more than ten different Flame variants (of which the earliest dates back to 2008) has demonstrated that during July and August 2010, developers implemented a transition from what can be referred to as version “A” to version “B”. Until that moment, the Flame configuration mentioned only files with identifier “A”:

Fragment of Flame configuration dated 21.07.2010

```
SUICIDE.RESIDUAL_FILES.A15 : %COMMONPROGRAMFILES%
\Microsoft Shared\MSAudio\dstrlog.dat
SUICIDE.RESIDUAL_FILES.A14 : %COMMONPROGRAMFILES%\Microsoft
Shared\MSSecurityMgr\dstrlog.dat
SUICIDE.RESIDUAL_FILES.A13 : %SYSTEMROOT%\Temp\~8C5FF6C.tmp
SUICIDE.RESIDUAL_FILES.A12 : %windir%\system32\mssvc32.ocx
SUICIDE.RESIDUAL_FILES.A11 : %COMMONPROGRAMFILES%\Microsoft
 Shared\MSSecurityMgr\rccache.dat
SUICIDE.RESIDUAL_FILES.A10 : %windir%\system32\winconf32.ocx
SUICIDE.RESIDUAL_FILES.A9 : %windir%\system32\watchxb.sys
SUICIDE.RESIDUAL_FILES.A8 : %windir%\system32\sdclt32.exe
SUICIDE.RESIDUAL_FILES.A7 : %windir%\system32\scaud32.exe
SUICIDE.RESIDUAL_FILES.A6 : %windir%\system32\mssui.drv
SUICIDE.RESIDUAL_FILES.A5 : %windir%\system32\modevga.com
SUICIDE.RESIDUAL_FILES.A4 : %windir%\system32\indsvc32.ocx
SUICIDE.RESIDUAL_FILES.A3 : %windir%\system32\indsvc32.dll
SUICIDE.RESIDUAL_FILES.A2 : %windir%\system32\comspol32.ocx
SUICIDE.RESIDUAL_FILES.A1 : %windir%\system32\comspol32.dll
```

Identifier “B” first appeared in Flame samples dated August 1, 2010 and was used in all subsequent variants, including the latest ones in 2011.

Fragment of Flame configuration dated 01.08.2010

```
SUICIDE.RESIDUAL_FILES.B9 : %windir%\system32\advnetcfg.ocx
SUICIDE.RESIDUAL_FILES.B8 : %windir%\system32\nteps32.ocx
SUICIDE.RESIDUAL_FILES.B7 : %temp%\~HLV473.tmp
```

```
SUICIDE.RESIDUAL_FILES.B6 : %temp%\~HLV927.tmp
SUICIDE.RESIDUAL_FILES.B5 : %temp%\~HLV294.tmp
SUICIDE.RESIDUAL_FILES.B4 : %temp%\~HLV084.tmp
SUICIDE.RESIDUAL_FILES.B3 : %temp%\~KWI989.tmp
SUICIDE.RESIDUAL_FILES.B2 : %temp%\~KWI988.tmp
SUICIDE.RESIDUAL_FILES.B18 : %systemroot%\system32\msglu32.ocx
SUICIDE.RESIDUAL_FILES.B17 : %temp%\~dra53.tmp
SUICIDE.RESIDUAL_FILES.B16 : %temp%\~rf288.tmp
SUICIDE.RESIDUAL_FILES.B15 : %windir%\system32\advpck.dat
SUICIDE.RESIDUAL_FILES.B14 : %windir%\system32\ntaps.dat
SUICIDE.RESIDUAL_FILES.B13 : %windir%\system32\soapr32.ocx
SUICIDE.RESIDUAL_FILES.B12 : %windir%\system32\rpcnc.dat
SUICIDE.RESIDUAL_FILES.B11 : %windir%\system32\boot32drv.sys
SUICIDE.RESIDUAL_FILES.B10 : %windir%\system32\ccalc32.sys
SUICIDE.RESIDUAL_FILES.B1 : %temp%\~HLV751.tmp
SUICIDE.RESIDUAL_FILES.B : %temp%\~DFL542.tmp
```

Naturally, we tried to find all the files with names that had been used in Flame since 2008. In early June 2012 we found the file "watchxb.sys" (SUICIDE.RESIDUAL_FILES.A9 : %windir%\system32\watchxb.sys), which is one of the files used in Flame "A" until August 2010.

When we analysed the file "watchxb.sys", we were in for a surprise.

The file is encrypted using the simple xor 0xFF algorithm. It is a configuration file structured similarly to the configuration of Flame: it specifies its own files, registry keys, lists of security programs and temporary file names. It was among these file names that we found **"icsvnt32.ocx"**, which we had not seen before.

Fragment of the watchxb.sys file

```
Files.10 : %windir%\system32\mspbee32.ocx
Files.9 : %windir%\system32\mspbee32.dll
Files.8 : %windir%\system32\comspol32.ocx
Files.7 : %windir%\system32\comspol32.dll
Files.6 : %windir%\system32\commgr32.ocx
Files.5 : %windir%\system32\commgr32.dll
Files.4 : %windir%\system32\icsvnt32.ocx
Files.3 : %windir%\system32\icsvnt32.dll
Files.2 : %windir%\system32\mssvc32.ocx
Files.1 : %windir%\system32\mssvc32.dll
RegKeys.5 : HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation\StandardSize
RegKeys.4 : HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SeCEdit\LastUsedIdentifier
RegKeys.3 : HKLM\SOFTWARE\Classes\CLSID\{4E14FBA2-2E22-11D1-9964-
00C04FBBB345}\InprocServer32\#icsvnt32.ocx
RegKeys.2 : HKLM\SOFTWARE\Classes\CLSID\{4E14FBA2-2E22-11D1-9964-
00C04FBBB345}\InprocServer32\#icsvnt32.dll
RegKeys.1 : HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Drivers32\wave9#%commonprogramfiles%\Microsoft Shared\MSAudio\wavesup3.drv
```

Up to that point, we had not seen that file name in any of the known Flame modules; it had not been used in any known Flame configurations and was not included in the Flame self-removal file list in the browse32.ocx module.

Searching further, we managed to find a file named **"icsvnt32.ocx"**. During the same period, we were searching for and studying Gauss, another cyber-espionage tool. The place where we discovered was however surprising.

Connections between Gauss and SPE

Gauss is a modular system. The number and combination of modules may change from one infected system to another. In the course of our research, we discovered the following modules: **Cosmos, Godel (Kurt), Tailor, McDomain, UsbDir, Lagrange, Gauss** and **ShellHW**.

The configuration of a specific combination of modules for each system is described in a special registry key. This technique, as well as the configuration structure itself, is similar to that used in Stuxnet/Duqu (storing of the configuration in the Windows registry) and Flame (configuration structure).

We created a special detection routine which helped us to discover various Gauss configurations based on registry settings on infected machines. We detected about 1700 such configurations in total, which revealed a picture of modules propagation.

And here, we were in for another surprise. In some systems in Lebanon, the Gauss configuration included one extra module codenamed **John** in addition to those mentioned above.

codenamed **John** in addition to those mentioned above.

In those Gauss configurations, the module pointed to the file **%systemroot%\system32\icsvnt32.ocx** and was called by the **RegisterService** function.

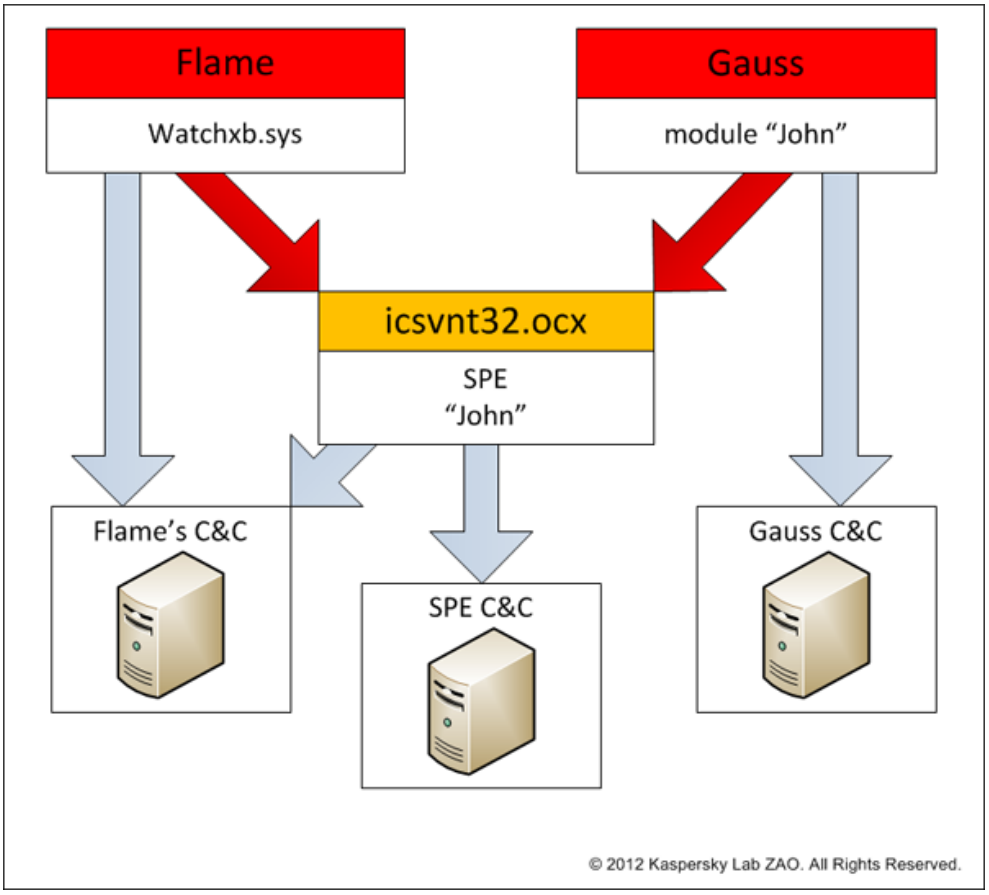
Sample Gauss configuration file

```
Gauss 1.0.8
ShellNotifyUser
ShellNotifyUserEx
SetWindowEvent
InitShellEx
%systemroot%\system32\winshell.ocx
%systemroot%\temp\ws1bin.dat

Godel
InitCache
RevertCache
ValidateEntry
CreateEntry
%windir%\system32\dskapi.ocx
%temp%\~gdl.tmp

John
RegisterService
%systemroot%\system32\icsvnt32.ocx

UsbDir
%windir%\system32\smdk.ocx
%temp%\~mdk.tmp
```

Screenshot of a decrypted key in a Gauss configuration which uses the “John” module:

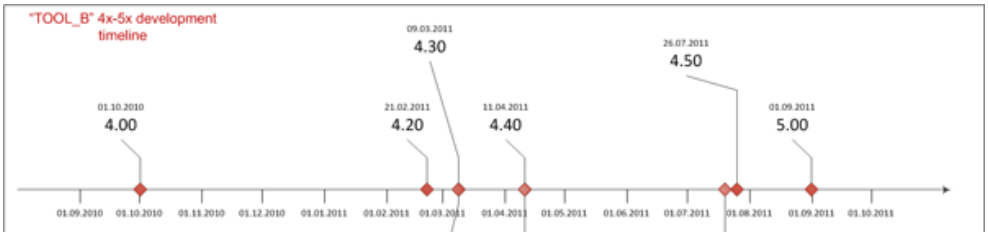
This provided confirmation of the fact that icsvnt32.ocx is a unique kind of module, used both in conjunction with Flame and in conjunction with Gauss.

It is a link connecting these two projects, while remaining independent. It uses either its own C&C servers or common servers with Flame.



SPE timeline

We detected six different versions of SPE in total. All of them were created over the period from 1 October 2010 to 1 September 2011. For three of these SPE versions, their so-called “U” modules, responsible for working with USB disks, were also discovered.



The above timeline shows that there was only one instance (4.50) when the “U” module was created on a different day from the main module.

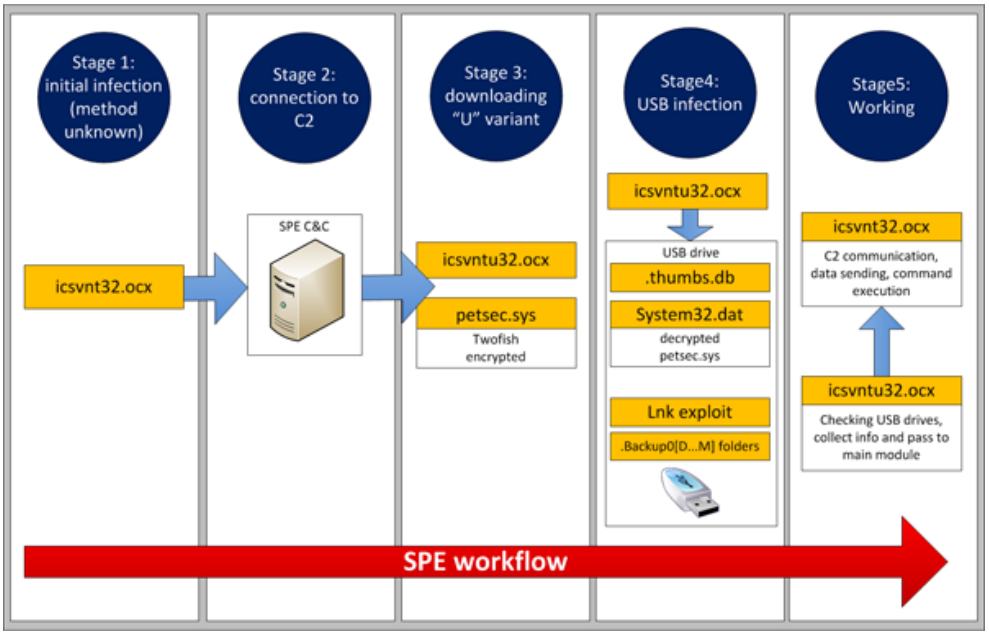
The existence of SPE versions earlier than 4.x or older than 5.x has not yet been established. It is possible that the earlier versions were in fact the program “SP” which was also supported on the C&C servers. Assuming that developing each major version of the program took one year, version 1.x could have been created as early as 2007.

At the time of writing this paper, version 4.50 is the most widespread in the wild according to statistics from the Kaspersky Security Network.

From a number of indirect indications, we can make the assumption that further program development was discontinued after version 5.00.

SPE workflow

By analyzing the SPE modules, operation logic and the capabilities of the C&C servers, we pieced together a diagram of the general algorithm showing how SPE works in an infected system:



The first stage is the initial infection of the system. The infection method is unknown; however, given the known facts about the previously seen relationship between SPE and Flame/Gauss, we deem it probable that icsvnt32.ocx can be loaded to and installed on a previously infected system from one of the Flame/Gauss servers upon the C2 operator’s command.

It is also possible that SPE is part of some sort of main Flame dropper (as yet undiscovered), or is in fact the unknown encrypted payload which was distributed by Gauss on USB disks (see [2] and [7]).

After infecting the system, SPE starts to communicate with the C&C server and sends the information it collects there. At this stage, the collected data is probably analyzed by operators: if the affected system is suitable for the deployment of the “USB module”, it is installed (icsvntu32.ocx), and so is the file petsec.sys.

At each subsequent stage, the two modules (the main module and the USB infector) operate independently from each other. The main module is responsible for sending data (including that collected with the help of the USB module) to the C&C server and executing the commands that are sent, while the USB module infects removable disks and collects data from them.

It is important to note that the USB module is not always installed. For example, if a Gauss infection is active in the system alongside SPE, operations with USB disks are performed by the appropriate Gauss module.

SPE versions info

As mentioned earlier, we detected a total of six different modifications of miniFlame which pertain to two major versions – 4.x and 5.x.

Icsvnt32.ocx (main module)

Version	Compile date	MD5	Size
4.00	10.10.2010	6F5ACDC848508C33F15634B1A068B16D	75264
4.20	21.02.2011	11C845B2C254C4170E9E49177F5053BB	89680
4.30	09.03.2011	16C986E14D34C7881E16186384DAB968	76288
4.40	11.04.2011	3091B15D27EEEE830FF85C50D50B3A05	97280
4.50	26.07.2011	B3E630714BF2526D3AA70370D2AC54B7	96768
5.00	01.09.2011	256469662C493731D4CEB003FC4783B1	104448

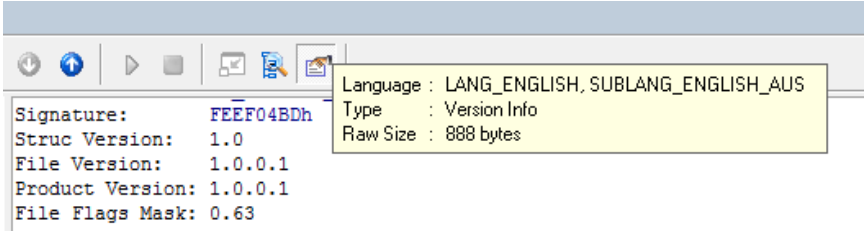
Icsvntu32.ocx (USB module)

Version	Compile date	MD5	Size
4.30	09.03.2011	523C6D9229B5656942B2CADEA3F0824C	108544
4.40	11.04.2011	E4EA1110E5915B7B66B405979E586887	113152
4.50	20.07.2011	A4C2DD6F3998A7625196DC79B1954150	112128

In all the 4.x versions, the developers used the same “version info” file.

```
Length Of Struc: 0378h
Length Of Value: 0034h
Type Of Struc:   0000h
Info:            VS_VERSION_INFO
Signature:       FEEF04BDh
Struc Version:   1.0
File Version:    1.0.0.1
Product Version: 1.0.0.1
File Flags Mask: 0.63
File Flags:
File OS:         NT (WINDOWS32)
File Type:       DLL
Language/Code Page: 3081/1200
CompanyName:        Microsoft Corporation
FileDescription:    icsvnt32
FileVersion:        1, 0, 0, 1
InternalName:       icsvnt32 (or icsvntu32 for U-module)
LegalCopyright:     Copyright ? 2010
LegalTrademarks:
OriginalFilename:   icsvnt32.ocx  (or icsvntu32 for U-module)
PrivateBuild:
ProductName:        Microsoft Corporation icsvnt32  (or icsvntu32 for U-module)
ProductVersion:     1, 0, 0, 1
SpecialBuild:
Child Type:         VarFileInfo
Translation:        3081/1200
```

The most notable detail here is the information about the language configured in the system where the contents of “version info” were modified. It corresponds to code page 3081, which is ENG_AUS (English (Australia)):

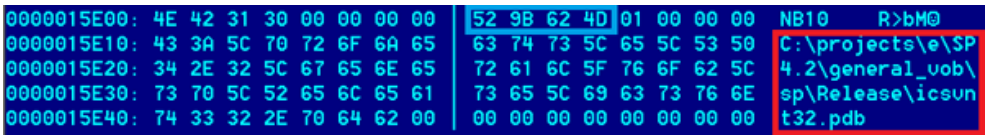


None of the Flame or Gauss modules known to us contained any information about ENG_AUS. Typically, they used ENGLISH_US or NEUTRAL.

At the same time, SPE 5.00 (which is the latest known SPE version) uses a new “version info” file, which uses the code page 1033, corresponding to ENGLISH_US.

```
Length Of Struc: 0454h
Length Of Value: 0034h
Type Of Struc:   0000h
Info:            VS_VERSION_INFO
Signature:        FEEF04BDh
Struc Version:   1.0
File Version:    2001.12.4414.320
Product Version: 3.0.0.4414
File Flags Mask: 0.63
File Flags:
File OS:         WINDOWS32
File Type:       DLL
File SubType:    UNKNOWN
Language/Code Page:  1033/1200
CompanyName:        Microsoft Coporation
FileDescription:
FileVersion:        2001.12.4414.320
InternalName:       ICSVNT32.DLL
LegalCopyright:     Copyright (C) Microsoft Corp. 1995-1999
LegalTrademarks:    Microsoft(R) is a registered trademark of Microsoft
Corporation. Windows(TM) is a trademark of Microsoft Corporation
OriginalFilename:
PrivateBuild:
ProductName:        COM Services
ProductVersion:     03.00.00.4414
SpecialBuild:
Child Type:          VarFileInfo

Translation:         1033/1200
```

One particularly interesting case is version 4.20. Here, during the build process, the author mistakenly included the path and the name of the project as CODEVIEW debug information:



The value 4D629B52 corresponds to the timestamp when the executable file linking took place, which is 21/02/2011 17:05:22.

The full path to the project debug database:

C:\projects\e\SP4.2\general_vob\sp\Release\icsvnt32.pdb

We can see the project branch is “e”, and the project name and version is “SP4.2”. The executable name “icsvnt32” is known, and was apparently kept between different versions. The meaning of “general_vob” is unclear to us.

Infection statistics (KSN data)

As we have already mentioned, the number of identified SPE infections is very low and is much closer to the number of infections with Duqu than to Flame/Gauss.

According to data received from Kaspersky Security Network, at the end of September 2012 we registered 15 reports from infected machines. They are all located in Western Asia, mostly in Lebanon.

Country	Number of incidents
Lebanon	10
Palestinian territory	3
Qatar	1
Kuwait	1

We believe that most cases of SPE infections in Lebanon are directly related with recently discovered and much more widespread infection of Gauss in this country.

To prove this theory, we combined infection reports for the main SPE module, its USB module and modules of Flame and Gauss

Gauss.

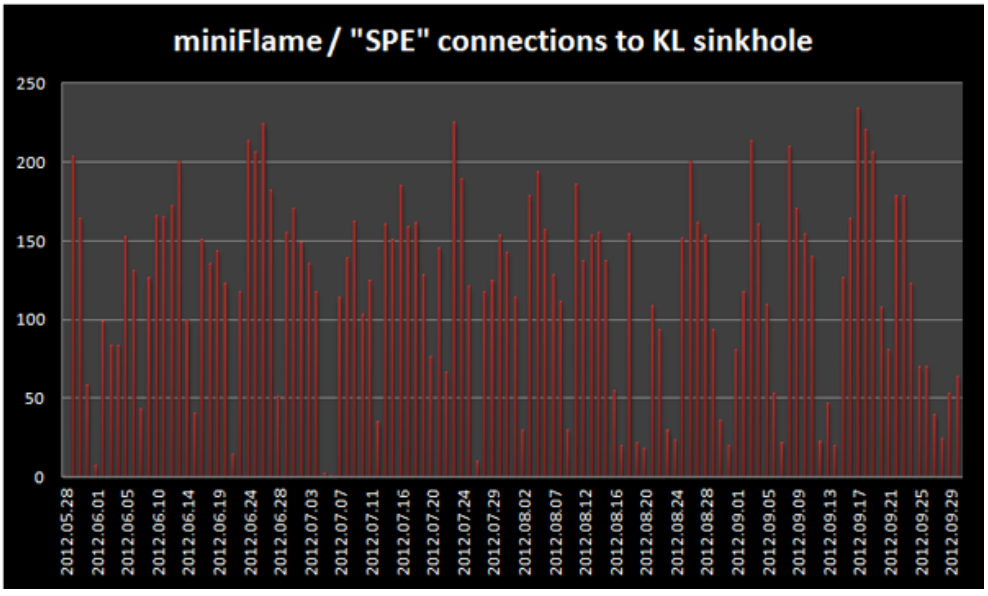
Victim of SPE	Previous Flame infection	Previous Gauss infection	SPE USB module
Lebanon#1	No	No	No
Lebanon#2	No	Yes	No
Lebanon#3	No	Yes	Yes
Lebanon#4	No	Yes	No
Lebanon#5	No	Yes	No
Palestinian territory#1	No	Yes	No
Qatar#1	No	No	Yes
Lebanon#6	Yes	Yes	No
Lebanon#7	No	No	Yes
Lebanon#8	No	No	No
Lebanon#9	Yes	Yes	No
Kuwait#1	No	No	Yes
Palestinian territory#2	No	Yes	No
Lebanon#10	Yes	Yes	Yes
Palestinian territory#3	Yes	Yes	No

As a result, it turns out that only two out of 10 victims of Gauss and SPE have the SPE USB module installed. This indicates that the Gauss “dskapi.ocx” could have been used as a substitute, since it implements similar functionality.

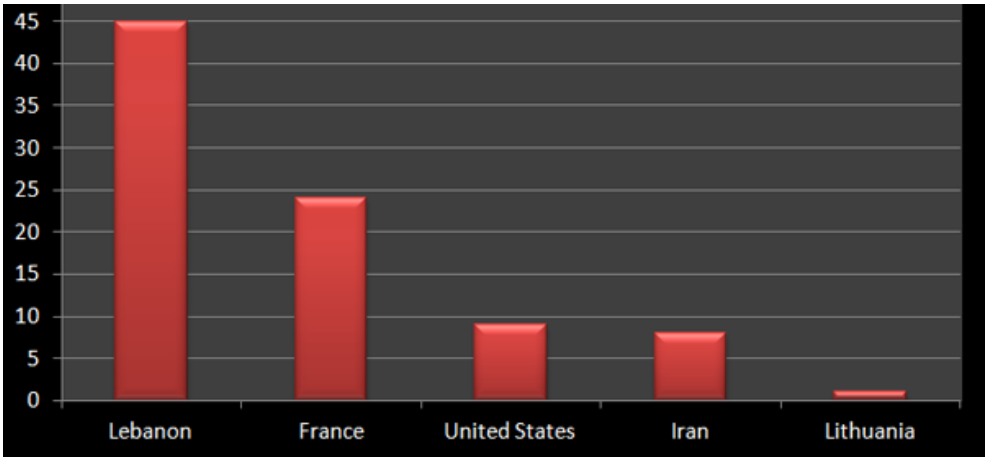
Sinkhole statistics

With the help of our partners, we set up a sinkhole for several of the Flame C&C domains as well as several miniFlame domains. The statistics below are for miniFlame connections only.

Between 28th of May 2012 and September 30th, we have counted close to 14,000 connections in total, coming from almost 90 different IPs.



Distribution of IPs of infected victims:



We were able to trace the IPs in the United States to VPN connections. Similarly, the IP in Lithuania belongs to an ISP which provides satellite internet in Lebanon. The IPs in France are the most curious ones – some do appear to be proxies or VPNs, but others are not.

For instance, one of the IPs of victims in France belongs to Francois Rabelais University of Tours:

IP Information for 193.52.[redacted]

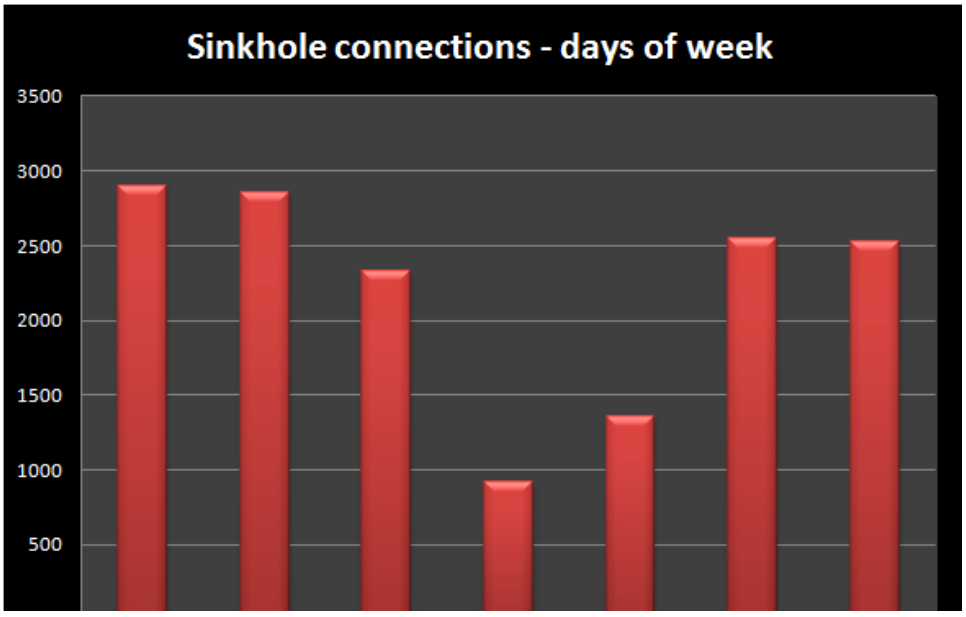
IP Location:	France Tours Universite Francois Rabelais - Tours
ASN:	AS2200
IP Address:	193.52[redacted] W R P D T

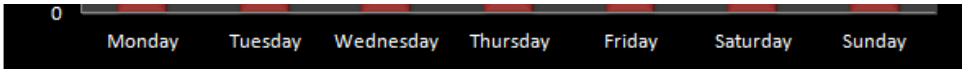
```
inetnum:        193.52.[redacted] - 193.52.[redacted]
netname:        FR-UNIV-TOURS
descr:          Universite Francois Rabelais - Tours
descr:          Service Informatique, Reseaux et TÃ©lÃ©communications
descr:          Parc de Grandmont, 37200 Tours, France
country:        FR
admin-c:        SM6134-RIPE
tech-c:         TH2076-RIPE
tech-c:         LB3797-RIPE
status:         ASSIGNED PA
mnt-by:         RENATER-MNT
source:         RIPE # Filtered
```

Other IPs in France belong to mobile internet users or free internet users.

Overall, it seems that the main two locations of victims are Lebanon and Iran.

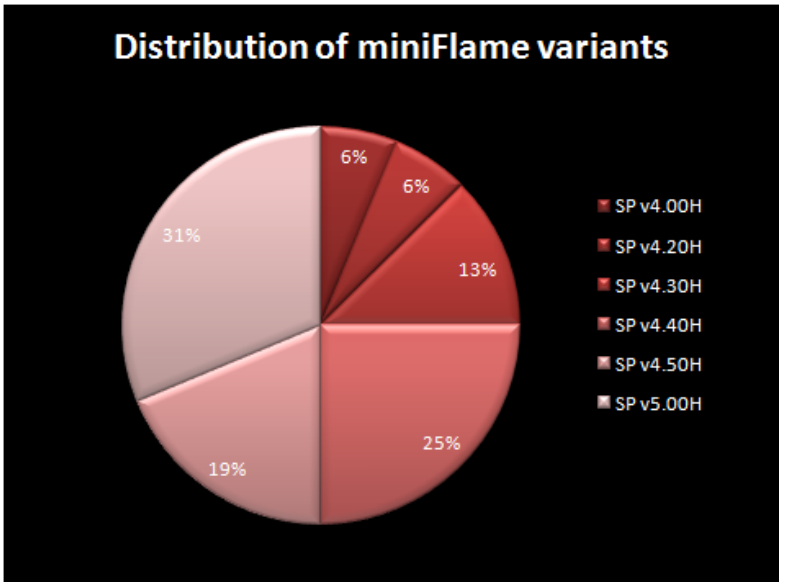
Distributed during the week, here's how the connections look like:





From the graphic above, it appears that the victims most often rest on Thursdays and Fridays.

Here's an overview in terms of versions connecting to the sinkhole:



When connecting to the sinkhole, miniFlame identifies itself as "SP vx.yz", not "SPE". It should also be noted that version 4.10 appears to be missing from the wild.

Icsvnt32.ocx (main module)

We will consider the file of version 4.50 as the reference point.

File names and paths

File names	Folders
icsvnt32.ocx	%windir%\System32
icsvnt32a.ocx	%windir%\System32
icsvntu32.ocx	"%allusersprofile%\ (Documents and Settings\All Users)
icsvntu32a.ocx	"%allusersprofile%\ (Documents and Settings\All Users)
ICSVNTU32.OCX	ProgramData
ICSVNTU32A.OCX	ProgramData

The file is a Windows PE DLL with 19 exports, compiled with Microsoft Visual Studio 6.0. Exports:

Ord	Name
1	DllGetClassObject
2	DllCanUnloadNow
3	<noname>
4	DllRegisterServer
5	DllUnregisterServer
6	NotifyLogoffUser
7	NotifyLogonUser
8	ServiceMain
9	LCEControlServer
10	RegisterTheFriggingEventServiceDuringSetup

11	RegisterTheFrigginEventServiceAfterSetup
12	RegisterTheEventServiceDuringSetup
13	RegisterTheEventServiceAfterSetup
14	RestoreMyDocsFolder
15	PerUserInit
16	DllInstall
17	CreateSharedDocuments
18	**RegisterService**
19	SvchostPushServiceGlobals

Creates events:

Event name	Comment
Global\TRStepEvent	All versions
Global\EPOAgentEvent	except version 5.00
Global\MICEvent	version 5.00 only
Global\AdvTW32SyncEvent	All versions
Global\AdvTW32ReadyXXXWfEvent	Where XXX is version number (e.g. 450)

Writes encrypted log files: “%allusersprofile%\mstlis.log”, “%allusersprofile%\datFE2B.da1”, “%temp%\daa59.tmp” (version 5.00 only).

All the actual functionality is implemented in two functions: **“DllMain”** (entry point) and **“RegisterService”** .

DllMain

The entry point routine runs in two different modes, depending on the parameters. If the parameter lpReserved is not equal to the magic number 0x1A33F1AB (true for the Windows loader), it proceeds in “loader mode”. If the parameter matches the magic number, it starts the main thread.

DllMain, “loader mode”

The module checks the version it is running on and selects its parameters for further operation:

For Windows NT 4.0 and higher:

Target process name	**svchost.exe**
Target username	SYSTEM
Target registry key	HKLM\SOFTWARE\Classes\CLSID\{4E14FBA2-2E22-11D1-9964-00C04FBBB345}\InProcServer32
Host DLL name	es.dll

For Windows 9x:

Target process name	**explorer.exe**
Target username	none
Target registry key	HKLM\SOFTWARE\Classes\CLSID\{450D8FBA-AD25-11D0-98A8-0800361B1103}\InProcServer32
Host DLL name	mydocs.dll

The module is supposed to be as a proxy for the selected DLL file. It loads the original library and resolves its exported function names to substitute.

Then, it launches the loader thread and returns from DllMain. In case of an error, it disinfects the registry by the restoring

original registry values and preventing itself from loading.

Loader thread

First, the module checks if it is running in a target process name and (if specified) by the target username. If the module or user names do not match, the thread terminates.

Then, it starts the registry monitor thread and, if succeeded, loads its own module using own PE format manipulation routines. Then, it executes the DllMain function of the loaded copy with a magic number 0x1A33F1AB, effectively starting itself in “main mode”.

Registry monitor thread

The module opens the target registry key and then waits for its modification using the API function “RegNotifyChangeKeyValue”. If the key’s default registry value was changed from pointing to the module to something else, it tries to revert the modification and increases a dedicated counter. The thread stops operation of the module and disinfects the registry if it encounters more than two modifications to the registry, or another thread sets the “Global\TRStepEvent”.

DllMain, “main mode”

When started in “main mode”, the module initializes its main object, the C&C interaction component and enters the main operation loop.

Version 5.00 only: *The module loads the optional library “%windir%\system32\msfrmt32.dll” and calls its exports “DllStartServer”, “DllStopServer” before and after the main loop, accordingly. The module also deletes this file on self-destruct. The purpose of this “msfrmt32.dll” is currently unknown as we have not been able to find a copy.*

The module continuously checks for running anti-virus processes by executable file names: “outpost.exe”, “bdagent.exe”. If any of these processes if present, it exits.

Then, it checks if its log files are big enough for exfiltration, and if true, moves each log to “%allusersprofile%\Wnm.tmp” and then transfers its contents to the first available C&C server. Connection to C&C servers occurs only if the module can fetch “http://www.google.com” first. Then, it tries to connect to a C&C server and request new commands.

C&C information

The list of C&C servers is hardcoded and consists of two arrays: the first array contains hostnames and IP addresses of the servers and the second one contains the URL corresponding to each server.

Version 4.00, 4.20, 4.30, 4.40

Server name	URL
webupdate.hopto.org	/cgi-bin/feed.cgi
webapp.serveftp.com	/cgi-bin/feed.cgi
web.autoflash.info	/cgi-bin/feed.cgi
web.velocitycache.com	/cgi-bin/feed.cgi
webupdate.dyndns.info	/cgi-bin/feed.cgi
cache.dyndns.info	/cgi-bin/feed.cgi

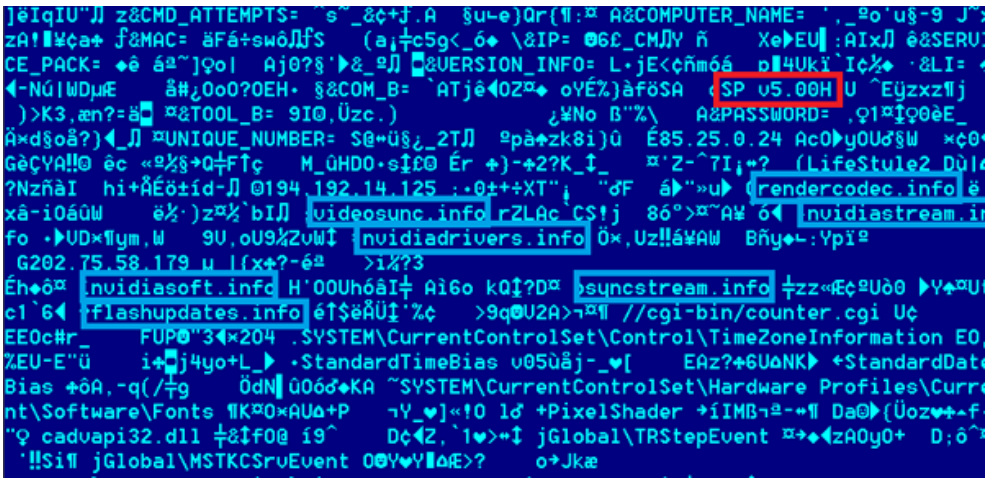
Version 4.50

Server name	URL
flashcenter.info	/cgi-bin/feed.cgi
flashrider.org	/cgi-bin/feed.cgi

Version 5.00

Server name	URL
flashupdates.info	/cgi-bin/counter.cgi
syncstream.info	/cgi-bin/counter.cgi
nvidiasoft.info	/cgi-bin/counter.cgi
202.75.58.179	/cgi-bin/counter.cgi
nvidiadrivers.info	/cgi-bin/counter.cgi
nvidiastream.info	/cgi-bin/counter.cgi
videosync.info	/cgi-bin/counter.cgi

rendercodec.info	/cgi-bin/counter.cgi
194.192.14.125	/cgi-bin/counter.cgi



Decrypted malware version string and C2 domains in a miniFlame sample

The domains used in version 5.00, among others, are the same as some of the domains used in several known versions of Flame [8]. Thus, SPE and Flame referred to the same C&C servers, and were serviced with the same software kit that we described earlier when analyzing Flame C&C servers. On the server side, the connections from miniFlame were processed by the OldProtocolE code, identifying the malware ("client") as SPE.

Communication protocol

The module selects the first available server and switches to the next one if the connection fails. During the operation, the module reads and writes its internal configuration data in the registry key:

[HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation] StandardTimeBias

Interestingly, this registry key is also known to Flame. The Flame module that carries out the unique MitM attack [6] while spreading across the local network checks if this SPE key is present in the system being infected and if so, it doesn't infect it.

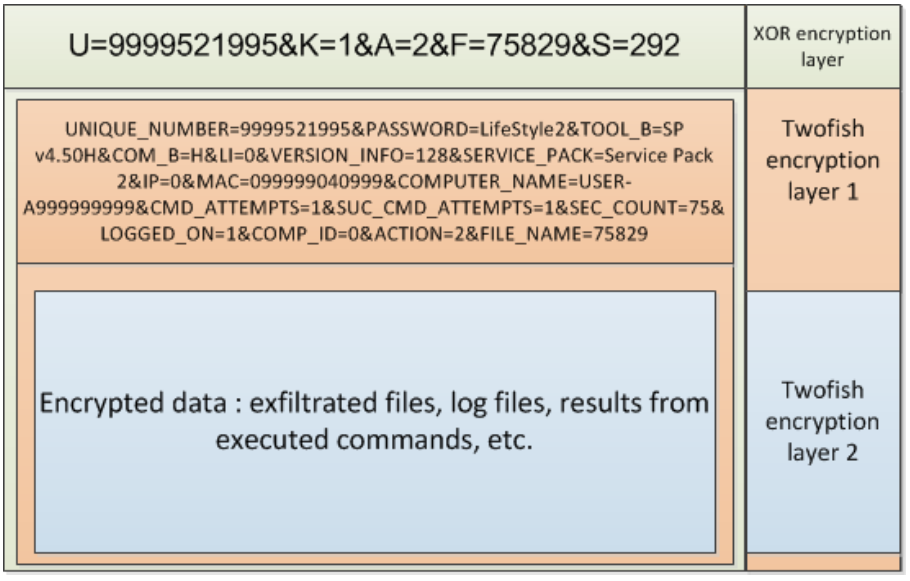
Thus, systems infected with SPE are recognized and avoided by the Flame replication code.

The C&C communication protocol is based on HTTP: the module sends both requests and replies in the body of the HTTP POST request, and the server may reply with commands to the module.

All data sent to C&C is placed in the contents of the POST request. The format is the following:

The packet starts with a 10-byte XOR key and a string encrypted using this key. It looks like a HTTP request string and contains basic information about the request:

"U=unique_victim_id&K=1&A=1 or 3&F=internal_file_ number&S=size_of_actual_request"



The request string is followed by a second part of the request that is encrypted using the Twofish algorithm. The size of the

The request string is followed by a second part of the request that is encrypted using the Twofish algorithm. The size of the encrypted part is defined by the “S=” value in the first part of the packet. The second part of the packet contains more information about the victim machine and the request:

```
UNIQUE_NUMBER=unique_victim_id
PASSWORD=LifeStyle2
TOOL_B=SP version_number
COM_B=H
LI=0
VERSION_INFO=encoded_windows_version
SERVICE_PACK=windows service pack number
IP=victim ip as uint
MAC=host MAC address
COMPUTER_NAME=windows pc name
CMD_ATTEMPTS=number of received commands
SUC_CMD_ATTEMPTS=number of successful commands
SEC_COUNT=GetTickCount()/1000
LOGGED_ON=1/0(if explorer.exe is running)
COMP_ID=value of [HKLM\SYSTEM\CurrentControlSet\Hardware Profiles\Current\Software\Fonts] PixelShade
ACTION=number
FILE_NAME=internal file number
```

Note the value of PASSWORD. It is “LifeStyle2”, the same password used in all known Flame versions for C&C communication.

The second part of the request may be followed by an optional third part that contains more data. It can be a log file, a file from the victim computer, or a log of commands that were received from the C&C server. The third part is encrypted with another layer of Twofish with the same key.

The format of the server reply is simpler: it is a buffer containing commands and their parameters encrypted with a single layer of Twofish with the same key. Every line of the buffer represents a command:

<!-- COMMAND_NAME CONTINUE_ON_ERROR(0/1) parameters … server_to_send_results port_to_send_results --> \n

Available commands:

Command name	Description
FIONA	Write file from the C&C to the victim machine
SONIA	Send a file from the victim machine to the C&C
EVE	Load a specified DLL and execute its specified export with no arguments
ELVIS	Create a process with given parameters, wait for it
DRAKE	Remove StandardTimeBias parameter from registry, signal the event “Global\TRStepEvent” (self destruct)
CHARLES	Write a new StandardTimeBias value to the registry
SAM	Sleep for a specified amount of time
ALEX (version 5.00)	Get system idle time in milliseconds
BARBARA (version 5.00)	Make a screenshot of the foreground window if it belongs to one of the predefined processes
TIFFANY (version 5.00)	Switch to the C&C server provided in the parameters of the command

After executing all the commands given by the server, the bot sends a new request to the server. This request contains a complete log of executed commands. If any of the commands require the bot to send data to the server (i.e., SONIA), they are sent in separate HTTP requests preceding the log request. These requests can also be redirected do a different C&C, if specified in the command parameters.

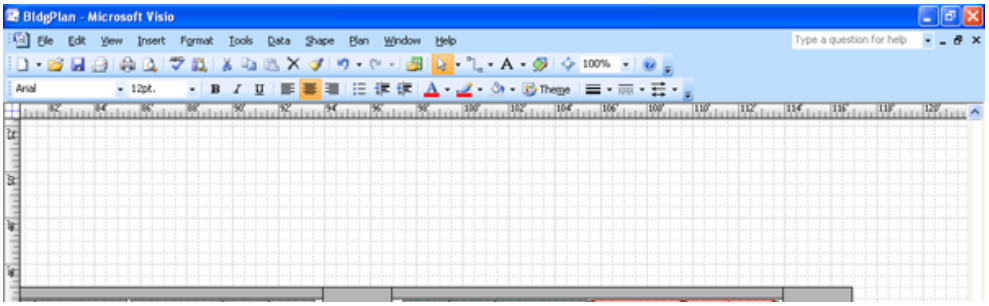
BARBARA

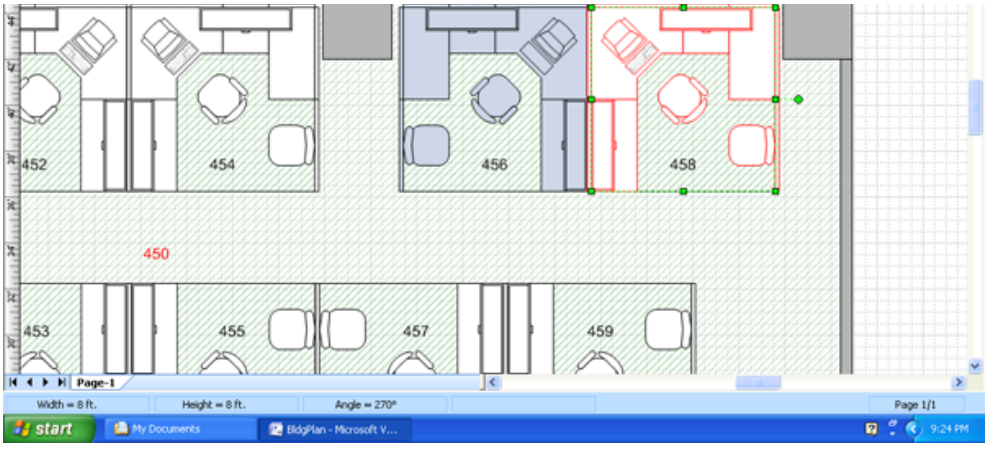
When the module receives the command “BARBARA”, it first retrieves the time of the last user input and continues only if the machine is not idle. Then, it makes a screenshot of the whole desktop in BMP format, compresses it using the PPMd algorithm (modified variant “I”) and sends the screenshot to its C&C server in a separate request.

It is worth noting that one of the modules of the Flame malware used the same algorithm (PPMd) for compressing screenshots.

The handler routine for the “BARBARA” command can be run in a different mode and can produce screenshots only if the foreground window belongs to one of the processes from the hardcoded list; however, this functionality is disabled.

Process name	Description
lexplore.exe	Internet Explorer browser
Mozilla.exe	Mozilla browser
Outlook.exe	MS Outlook
Msimn.exe	MS Outlook Express
Winword.exe	MS Word
Excel.exe	MS Excel
Msmsgs.exe	MSN Messenger
Msgplus.exe	MSN Messenger extension
Msnmsgr.exe	MSN Messenger extension
Msdev.exe	Microsoft Developers Studio
Explorer.exe	Windows Explorer
Cygwin.exe	Linux-like environment for Windows making it possible to port software running on POSIX systems (such as Linux, BSD, and Unix systems) to Windows
Acrobat.exe	Adobe Acrobat
Acrord32.exe	Adobe Acrobat Reader
Aim.exe	AOL Instant Messenger
Aim95.exe	AOL Instant Messenger
Frontpage.exe	MS FrontPage editor
Icq.exe	ICQ Instant Messenger
Icqlite.exe	ICQ Lite
Inetinfo.exe	component that hosts the IIS metabase and the non-Web services of IIS 6.0
Exceed.exe	Enterprise management system Hummingbird Ltd/OpenText
telnet.exe	Windows Telnet client
ftp.exe	Windows FTP client
Putty.exe	SSH/Telnet client
Netscape.exe	Netscape Navigator browser
Notepad.exe	Windows Notepad
Winproj.exe	Microsoft Office Project
Powerpnt.exe	Microsoft PowerPoint
Visio.exe	MS Visio
Ypager.exe	Yahoo Messenger
Mstsc.exe	Microsoft Remote Desktop connection
mmc.exe	Microsoft Management Console
Paltalk.exe	PalTalk Messenger
Onenote.exe	Microsoft Office OneNote
Onenotem.exe	Microsoft Office OneNote Quick Launcher





Example of a test screenshot made by the module after receiving the command "BARBARA" with Microsoft Visio running in the foreground

After executing all the commands given by the server, the bot sends a new request to the server. This request contains a complete log of executed commands. If any of the commands require the bot to send data to the server (i.e., SONIA), they are sent in separate HTTP requests preceding the log request. These requests can also be redirected do a different C&C, if specified in the command parameters.

Export "RegisterService"

This function may be called by an external module, i.e. the Gauss loader component. This is an installation routine: it copies the module to the system directory and modifies the registry to make it run on system startup.

The module checks if it is running on Windows 2000 or later NT version (XP, Vista, 7), returns if false. It also checks if there are running processes "outpost.exe" or "bdagent.exe". If any of the processes are running, it disinfects the registry and exits.

Then, it copies the file "%windir%\system32\icsvnt32a.ocx" to a temporary file with a prefix "%allusersprofile%\gfw". Then, it moves that temporary file to "%windir%\system32\icsvnt32.ocx". It also retrieves the creation time of the file "%windir%\system32\kernel32.dll" and sets the creation time of own file to the same value. The original file, "%windir%\system32\icsvnt32a.ocx", is scheduled to be deleted on the next reboot.

If the file "icsvnt32.ocx" is installed without errors, the module then changes the target registry key's default value to "%windir%\system32\icsvnt32.ocx". By replacing one of the system COM objects, the module actually registers to be loaded by most of the system processes.

Icsvntu32.ocx (the USB infector)

The module is based on the same source code as "icsvnt32.ocx" and many functions are identical. The file even contains the code for interpreting commands from the C&C server but the code that connects to the server is replaced by a stub that writes the data to a log file.

The program is a Windows PE DLL with 19 exports, compiled with Microsoft Visual Studio 6.0. Exports:

Ord	Name
1	DllGetClassObject
2	DllCanUnloadNow
3	<noname>
4	DllRegisterServer
5	DllUnregisterServer
6	NotifyLogoffUser
7	NotifyLogonUser
8	ServiceMain
9	LCEControlServer
10	RegisterTheFrigginEventServiceDuringSetup
11	RegisterTheFrigginEventServiceAfterSetup
12	RegisterTheEventServiceDuringSetup
13	RegisterTheEventServiceAfterSetup
14	RestoreMyDocsFolder

15	PerUserInit
16	DllInstall
17	CreateSharedDocuments
18	**RegisterService**
19	SvchostPushServiceGlobals

Creates events:

Event name	Comment
Global\TRStepEventU	All versions
Global\EPOAgentEventU	All versions
Global\TUSEventU	All versions
Global\AdvTW32AutoDetectU	All versions
Global\AdvTW32ReadyXXXWfEventU	Where XXX is version number (e.g. 450)
Global\MSTKCSrvEventU	All versions

The encrypted log files it writes are named: “%allusersprofile%\mstlis.log”, “%allusersprofile%\datFE2B.da1”, “%allusersprofile%\datFE2A.tmp”

DllMain

The entry point routine runs in two different modes, depending on the parameters. If the parameter lpReserved is not equal to the magic number 0x1A33F1AB (true for the Windows loader), it proceeds in “loader mode”. If the parameter matches the magic number, it starts the main thread.

DllMain, “loader mode”

The module checks the version it is running on and selects its parameters for further operation:

For Windows NT 4.0 and higher, Windows 9x:

Target process name	**explorer.exe**
Target username	None
Target registry key	HKLM\SOFTWARE\Classes\CLSID\{35CEC8A3-2BE6-11D2-8773-92E220524153}\InProcServer32
Host DLL name	stobject.dll

The parameters for both NT and 9x versions are the same, still the functions that initialize the parameter values are different.

The module is supposed to be as a proxy for the selected DLL file. It loads the original library and resolves its exported function names to substitute.

Then, it launches the loader thread and returns from DllMain. In case of any error, it disinfects the registry by the restoring original registry values and preventing itself from loading.

Loader thread

First, the module checks if it is running in a target process name and (if specified) by the target username. If the module or user names do not match, the thread terminates.

Then, it starts the registry monitor thread and, if it succeeds, loads its own module using own PE format manipulation routines. Then, it executes the DllMain function of the loaded copy with a magic number 0x1A33F1AB, effectively starting itself in “main mode”.

Registry monitor thread

The module opens the target registry key and then waits for its modification using the API function “RegNotifyChangeKeyValue”. If the key’s default registry value was changed from pointing to the module to something else, it tries to revert the modification and increases a dedicated counter. The thread stops operation of the module and disinfects the registry if it encounters more than two modifications to the registry, or another thread sets the “Global\TRStepEventU”.

DllMain, “main mode”

When started in “main mode”, the module initializes its main object, the substitution for the C&C interaction component (similar to the actual C&C interaction component in “icsvnt32.ocx”). It also creates a desktop named “Default3” and assigns its main thread to that desktop. Then, it enters the main operation loop.

The module continuously checks for running anti-virus processes by executable file names: “outpost.exe”, “bdagent.exe”, “antivirus.exe”. If any of these processes are present, it exits.

The main loop is a modified of the C&C interaction loop taken out of “icsvnt32.ocx”, but instead of connecting to the server it only starts the drive infection routine. During the operation, the module reads and writes its internal configuration data in the registry value:

[HKCU\Console]
StandardTimeBiasU

USB drive infection routine

The module enumerates all available USB drives. Each USB drive formatted with FAT, FAT32 or NTFS is then processed and infected.

First, the module retrieves information about the drive and stores it in “%allusersprofile%\mstlis.log”. Then, it tries to disinfect the drive potentially infected by an older variant by removing any of the following files:

System32.dat
.Catroot.tmp

It also searches for directories with names “.Backup0D” – “.Backup0M”, and for each directory, removes contained files named “target.lnk” and “desktop.ini” and the directory itself. These directories are also created during the Gauss USB infection.

If the drive contains a file named “.thumbs.db” in its root directory, the module reads its contents and writes it in the log file “%allusersprofile%\datFE2A.tmp”. The log entry is preceded by the string “USB_RESULT” and current date/time. If the log file is considered big enough for exfiltration, its contents are then written to another log file: “%allusersprofile%\datFE2B.da1”.

Then, the module infects the drive. It creates a new file named “.thumbs.db” in the root directory of the drive. The file contains the magic number 0x0EB397F2B and a TTL value of 10. This TTL counter is decremented by the payload that is executed from the USB drive, and the payload disinfects the drive when the counter reaches zero.

The format of the “.thumbs.db” file is identical to the one used by the Gauss module “dskapi.ocx”.

The module decrypts the file “%allusersprofile%\petsec.sys” using a hardcoded Twofish key and writes it to the root directory of the drive with a name “System32.dat”. The file is a PE DLL file, and is supposed to be loaded by the LNK exploit from the infected drive.

Then, it creates directories named “.Backup0D” –“.Backup0M”. In each directory it creates files named “target.lnk” and “desktop.ini”. The first file contains the LNK exploit that loads the “System32.dat” file, and the latter converts the directory into a junction point.

Export “RegisterService“

This function may be called by an external module, i.e. Gauss loader component. This is an installation routine: it copies the module to the system directory and modifies the registry.

The module checks if it is running on Windows 2000 or later NT version (XP, Vista, 7), returns if false. It also checks if there are running processes “outpost.exe”, “bdagent.exe” or “antivirus.exe”. If any of the processes are running, it disinfects the registry and exits.

Then, it copies the file “%allusersprofile%\icsvntu32a.ocx” to a temporary file with a prefix “%allusersprofile%\gfw”. Then, it moves that temporary file to %allusersprofile%\icsvntu32.ocx”. It also retrieves the creation time of the file “%windir%\system32\kernel32.dll” and sets the creation time of own file to the same value. The original file, “%allusersprofile%\icsvntu32a.ocx”, is scheduled to be deleted on the next reboot.

If the file “icsvntu32.ocx” is installed without errors, the module then changes the target registry key’s default value to “%allusersprofile%\icsvntu32a.ocx”. By replacing one of the system COM objects, the module actually registers to be loaded by

most of the system processes.

.CatRoot.tmp / System32.dat

We were able to find a copy of the file “petsec.sys” which is mentioned in the 4.00 version of SPE. File location on infected USB drive: “\.CatRoot.tmp”, “\System32.dat”

Known variants:

Version	MD5	Compilation date	Size
4	8d206625957f7c96fd468f6c176248d0	2010.09.15 15:08:17 (GMT)	13312

Creates mutex: “lsvp4003ltrEvent”.

The file is a Windows PE DLL without exports, compiled with Microsoft Visual Studio 6.0.

All the actual functionality is implemented in “DllMain” (entry point). The module’s functionality is almost identical to “System32.dat”/”System32.bin” files written by the Gauss module “dskapi.ocx”: it collects information about the system it was started on and writes it to a file on the infected USB drive.

The known variant above corresponds to the version 4.00 of the “icsvntu32.ocx” module and is supposed to have the name “.CatRoot.tmp”. Newer version of the “icsvntu32.ocx” infect USB drives with the file named “System32.dat”, however we do not have these versions of the infection payload.

DllMain

The module checks if its filename contains its complete original filename, “.CatRoot.tmp” (version 4.00). Since the module crashes when its name is different, newer modifications of the module should have used a different name, “System32.dat”.

If the filename matches, the module copies itself to “%TEMP%\%cCatRoot.tmp”, where “%c” is the name of the drive from where it started. I.e., if the module started from disk “F:”, the name is “%TEMP%\FCatRoot.tmp”. Then, it loads its new copy and returns.

If the filename contains its original name without the first symbol (“CatRoot.tmp” without the dot), the module assumes it was loaded from a local copy and proceeds with its main routine.

The module marks the system as infected by writing the following registry key:

[HKCU\Control Panel\Desktop]
WindowBuildVal = random value

Then, it reads the file “.thumbs.db” from the root directory of the infected USB drive. The 32-bit integer at offset 4 of that file contains the module’s TTL (time to live) counter. It is decremented each time, and when its value reaches zero, the module executes the disinfection routine. If it is greater than zero, it executes the data collection routine.

Data collection routine

The module collects several types of data and writes it to the end of the file “.thumbs.db” in the root directory of the infected USB drive. The file is encrypted with XOR and its format is identical to the one used by the Gauss “dskapi.ocx” module.

The module collects the following information:

- Computer name
- Windows version
- Platform type
- List of network adapters
- Contents of the ARP table
- List of loaded modules and processes
- List of files in root directories of fixed and network drives, up to 200 names for each drive
- List of files in the directories, up to 200 names for each directory:
  - “%TEMP%*”
  - “%userprofile%\Desktop*”
  - “%windir%\Prefetch*”
  - “%programfiles%*”

  List of visible network servers

Disinfection routine

The module deletes the file “.CatRoot.tmp” from the root directory of the infected drive. Then, it searches the root directory for subdirectories with names “.Backup0D” – “.Backup0M”, and for each directory:

- removes files “desktop.ini”, “target.lnk”
- removes the directory

The file “.thumbs.db” is not removed during disinfection.

Conclusions

During the analysis of the Flame C&C server side code, we identified four different malware known to the server: SP, SPE, FL

and IP. The malware known as FL is Flame. The malware known as SPE is described in this paper.

Based on our analysis, we have been able to put together several main points of SPE which we also dubbed the “miniFlame”, or “John”, as named by the corresponding Gauss configuration:

- The malware is rare, probably deployed only on a very small number of high profile victims.
- Unlike Gauss, it implements a full client/server backdoor, which allows the operator to have direct access to the infected system.
- The Flame C&C code we’ve analysed does not appear to contain specific modules to control SPE clients; we can assume other dedicated SPE servers, with special codebase exist or existed.
- The development of SPE was carried out in parallel to Flame and Gauss, during 2010-2011.
- Both Flame and Gauss make use of miniFlame/SPE as a module.
- The most recent variant of SPE is 5.00; the earliest known one is 4.00.
- The exact infection vector for SPE is unknown; it is believed that the malware gets deployed from the C2 during Flame or Gauss infections.
- Version 4.20 of the malware contains a debug path in the binary which points to “C:\projects\e\SP4.2\general_vob\sp\Release\icsvnt32.pdb”. This indicates the authors named the malware “SP4.2”, although it uses the SPE client type. It is possible that SP is probably just an earlier version of miniFlame / SPE – 1.00 to 3.x.
- SPE consolidates the theory of a strong link between Flame and Gauss teams. The miniFlame represents a common module used by both.
- All known 4.xx versions of SPE contain a version info section which references code page 3081, ENG_AUS, English (Australia).

If Flame and Gauss were massive cyber-espionage operations, infecting thousands of users, then miniFlame/SPE is a high precision, surgical attack tool. The numbers of victims are comparable to Duqu, for instance. We can assume this was part of

the same operation, used in multiple waves: first, as many potentially interesting victims as possible are infected. Secondly, data is collected from the victim, allowing the attackers to profile them and find potentially interesting targets. For these selected targets, a specialized spy tool such as SPE is deployed.

Within the Flame C&C code, two other malware files are referenced: SP and IP. If SP probably refers to an older variant of the malware described in this paper, IP is probably different and remains unknown still.
With Flame, Gauss and miniFlame, we have only just scratched surface of the massive cyber-espionage operations ongoing in the Middle East. Their full purpose remains obscure and the identity of the victims and the attackers remains unknown.

References

[1] The Flame: Questions and Answers

[2] Gauss: Abnormal Distribution

[3] Full Analysis of Flame’s Command and Control Servers

[4] Back to Stuxnet: the missing link

[5] The Mystery of Duqu: Part Seven (Back to Stuxnet, the Tilded platform)

[6] Flame: Replication via Windows Update MITM proxy server

[7] The Mystery of the Encrypted Gauss Payload

[8] The Roof Is on Fire: Tackling Flame’s C&C Servers