FEATURE 2

HOW 'DARE' YOU CALL IT SPYWARE!

Prabhat K. Singh, Fraser Howard and Joe Telafici
McAfee, AVERT

Anti-virus vendors frequently receive queries, objections and legal notices from software vendors whose applications are detected as spyware or adware. As a result, the task of 'proving' whether a given sample is a spyware/adware or clean application demands careful judgment on the part of the researcher.

Several such applications cannot be ignored as benign software just because of the legal risks associated with detection. Adware and spyware may be described as applications that may carry out one or more unsolicited actions, such as spying on a user's PC activities, gathering data about the user's browsing habits or pushing unwanted and/or offensive advertisements into the user's system.

The majority of researchers in the AV industry are aware of these descriptions, yet almost everyone has a different interpretation of adware/spyware behaviour and companies tend to add detection for applications that may not warrant inclusion in this category, or vice versa.

Despite a rash of legislative activity in the US and the EU, definitions of spyware, adware and associated terminology vary widely. This article presents a description of spyware/adware behaviour and visits a few criteria that may be used by researchers to prove that a program can be detected as adware/spyware. We break malware behaviour into six areas to achieve an environment within which spyware/adware can be compared with malware programs.

MALWARE PROGRAM BEHAVIOUR

In the following sections we will discuss six areas of malware behaviour and structure: installation, survey, replication, concealment, injection and payload (for more information on this method of malware analysis see http://downloads.securityfocus.com/library/masterthesis.pdf).

Installation

An installer creates and maintains an installation qualifier so that the malware can execute on the victim system, and ensures the automatic interpretation of malware code.

In this definition there are two criteria for a segment of code in the malware to qualify as an installer. First, the code should cause a permanent change in the machine's security state. Second, the code may ensure that the program is invoked after every time the system is restarted or on the occurrence of some system event. Hence, one or more of the following activities may be observed during spyware/adware installation:

Installation of a service to ensure that the application is active even when a user is not logged on

The information relating to the services installed on the local machine is stored in the Service Control Manager (SCM) database. The Win32 SCM APIs may be used to add, query or control the status of the services installed.

COM class registration

Frequently, spyware/adware applications use COM objects to achieve software reusability and adaptability. An understanding of the relevant parts of the system which may be altered during a COM object's installation is important in order to understand the behaviour of the spyware/adware program.

A COM server is a binary file that contains the code for methods used by one or more COM classes. This server can be packaged either as a DLL or as an executable file. There are two types of activation request for an object: an in-process activation request and an out-of-process activation request. An in-process activation request requires a DLL-based version of the COM server to be present and loaded in the client's memory space. An out-of-process activation request requires the executable to be used to start the server process.

In order to allow client programs to activate objects without concern for which type of package is used or where the executables or DLLs are located, COM stores configuration information in the registry that maps the class IDs (CLSIDs) onto the server that implements that class. Whenever an activation request is made for a CLSID in a given machine, the registry is consulted. If the configuration information is not available in the registry, the request may be redirected to a remote host from which the relevant code may be downloaded and installed. COM stores most of the configuration information in the registry key 'HKLM\Software\Classes', although most programs use the more convenient alias 'HKCR'. COM keeps machine-wide information related to CLSIDs in the registry key 'HKCR\CLSID COM' and also looks into per-user class information in the 'HKCU\Software\Classes\CLSID' key.

COM stores all the locally known CLSIDs under either of the above two keys, with one subkey per CLSID. For example, let there be a class named 'JoeSpy' which is used to implement spyware functionality.

This will have a registry entry as:

```
[HKCR\CLSID\{571F1680-CC83-11d0-8C48-0080C73925BA}]
@="JoeSpy"
```

In order to allow local activation of 'JoeSpy' objects, JoeSpy's CLSID entry in the registry will have a subkey that indicates which file contains the executable code for the JoeSpy class's methods. If the COM server is packaged as a DLL, the following entry will be required in the registry:

```
[HKCR\CLSID\{571F1680-CC83-11d0-8C48-
0080C73925BA}\InprocServer32]
@="C:\JoeSpyware.dll"
```

If the COM server is a package that uses an executable file, the following entry will be required in the registry:

```
[HKCR\CLSID\{571F1680-CC83-11d0-8C48-
0080C73925BA}\LocalServer32]
@=" C:\JoeSpyware.exe"
```

If the environment does not cope easily with raw CLSIDs to make activation calls, a programmer may use ProgIDs. The CLSID to ProgID translation is achieved through the following registry entry:

```
[HKCR\CLSID\{571F1680-CC83-11d0-8C48-
0080C73925BA}\ProgID]
@="JoeSpyware.JoeSpy.1"
```

Conversely, to support the reverse mapping (ProgID to CLSID), the following keys are needed:

```
[HKCR\JoeSpyware.JoeSpy.1]
@="JoeSpy"
[HKCR\JoeSpyware.JoeSpy.1\CLSID]
@="{571F1680-CC83-11d0-8C48-0080C73925BA}"
```

A well-implemented COM server will implement the following two functions to support installation and uninstallation:

```
DllRegisterServer(void);
DllUnregisterServer(void);
```

However, most adware/spyware programs do not follow this practice.

Browser Helper Object (BHO) installations

These are frequently used by adware applications for installing toolbars or redirecting network traffic in *Internet Explorer* and the *Windows Explorer*. The BHO is installed as a COM in-process server registered under the 'HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects' registry key, and registers the CLSIDs for all the BHOs installed in the system.

Assuring execution at system initialization

This is usually achieved by using Run and RunOnce registry keys. These keys are present in the HKCU and the HKLM hives of the registry. Generally either 'HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run' or 'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run' is used.

Both of these key entries ensure that the spyware/adware program is executed every time the user logs into their account. The difference between the two keys is that the HKLM key executes the program *before* the appearance of the desktop, while the HKCU key executes the program *after* the appearance of the desktop.

LSP and NSP installation

Spyware and adware installations (e.g. a version of *MarketScore*) use Winsock 2 Layered and Network Service Provider implementations to redirect network traffic to specific sites. The network data emanating from and entering the machine can be sent to a 'central' site and then sent to the user's application.

For example, when the user types 'www.google.com' into their Internet browser, the browser will connect silently to www.marketscore.com and send part of this data, *before* connecting to www.google.com. This is a more powerful mechanism than that achieved through a BHO because the interception of network traffic is not application-specific (i.e. not limited to *IE* using a BHO). This is because LSPs work at the TCP implementation level.

LSP is a component that intercepts winsock2 calls and has the ability to manipulate them, and then optionally pass them to the winsock2 provider. There are two kinds of LSP, the transport service provider and the name space service provider. The former is used frequently by adware/spyware for implementing LSPs. An LSP is implemented as a DLL and should be registered with the system. The usual registry location is 'HKLM\System\CurrentControlSet\Services\Winsock2'.

The LSP will always export the WSPStartup() function. This function is invoked whenever a user calls the WSAStartup() function. LSPs are organized in a chain wherein a network-related function call invokes the first node in the chain. This node processes the call and invokes the next node in the chain and so on, until the last node invokes the winsock2 function. An exhaustive treatment of this topic can be found at http://www.microsoft.com/msj/0599/LayeredService/LayeredService.aspx.

Survey

Survey behaviour identifies appropriate targets, network hosts or objects and their locators so that other behavioural units can perform correctly. Here, a locator is an address or path information to the target.

Survey behaviour is the reconnaissance activity that malware programs carry out to find more vulnerable host addresses for the purpose of replication. This activity is observed in viruses and worms and is *absent* from adware

and spyware. This activity should not be confused with the spyware behaviour that involves collecting information from the user's PC for market survey-related purposes.

Replication

Replication behaviour provides the logistic mechanisms for the transfer of malware code. Logistic mechanisms are technical and/or social engineering methods for the transfer of malware from an infected host to another target host.

Although spyware and adware may carry out a lot of network activity, they do not exhibit replication behaviour as defined above. Replication using social engineering should not be confused with installation using social engineering methods where the user is duped into installing the program onto their PC. Currently, it is safe to say that adware and spyware are characterized by the *absence* of replication behaviour – otherwise they could be put into the category of self-propagating programs (worms) and there would be no question about the legality of their detection by any AV software.

Concealment

Concealment behaviour prevents the discovery of the activity and structure of a malware program in order to avoid detection, forensics and removal.

Software forensics can be used for author identification and characterization. The absence of author information or presence of conflicting information may be a strong indication of concealment. The use of concealment in spyware/adware is intended to increase the complexity of analysis and thus increases the difficulty in 'proving' an adware/spyware program. In many cases, variable CLSID values are generated during several instances of installation of the same sample. The proving process may not be reliable if we base our conclusions only on the CLSID 'blacklist' to identify known malicious COM objects (e.g. a version of *Virtumonde*).

Concealment has been classified into two categories, namely, concealment that prevents any dissemination of program information (PPID) and concealment that is achieved through attacks on the system's security mechanism (ASM). Currently, the concealment approaches for achieving PPID in spyware/adware are trivial and nearly all of these have been borrowed from the virus-writing domain.

The first approach to achieving PPID 'blocks' the availability of a program's executable image to the researcher. In this case the executable image of the program remains in the memory of the infected system. Once the system is rebooted, the malicious program needs to be installed on the system again. Such attacks are used more often in the conventional worm and virus-style attacks than in adware/spyware.

The second approach to achieving PPID uses code evolution. We define code evolution as the process of creating program equivalents in such a way that, given an identical input sequence of symbols, they produce an identical output sequence of symbols. The variations of this are: polymorphism, metamorphism and packing – these are well known by virus researchers and will not be discussed in this article.

Injection

The injector behaviour causes an injection of malware components into the victim object such that one or more of the malware components is placed in the execution space of the victim object. The copy of the malware may be exact or an evolved instance of the original malware, after being processed by the concealment behaviour. The execution space of an object is the code/text segment of the victim object or the environment in which the interpretation of the object will take place. Injection may or may not be present in adware/spyware.

While carrying out concealment and installation, a malicious program may need to inject all or parts of its components into another program in the memory. Common activities may involve DLL injection into the address space of another process. This is usually achieved using the following methods:

DLL injection using the registry

This is achieved by adding the DLL pathname as a data value for a subkey in the following registry key:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs

During system initialization when user32.dll is mapped into a process's address space it automatically reads these registry key entries and loads any DLL mentioned in the data value field.

DLL injection using Windows hooks

This is achieved by implementing a system-wide hook using the SetWindowsHookEx() function. When a system-wide hook is called from the context of another process, the system loads the DLL containing the hook into the process's address space.

DLL injection using remote threads

Most *Windows* functions will allow a process to manipulate its own address space. Some functions do exist that can be used for manipulating other processes. In order to load a DLL in another process and get loadlibrary() to load a DLL

into the other process, the program creates a new thread in the other process using the CreateRemoteThread() function. Since the CreateRemoteThread()API is not present in *Windows 9x*, this method will not work on all platforms.

Injection targets

Spyware/adware programs usually inject themselves into processes like *Internet Explorer* and *Windows Explorer*.

Payload

The payload is a behaviour programmed into the malware that is used to achieve a specific purpose for which the malware was created.

Two types of payload behaviour are observed both in spyware/adware and in malware.

Host payload behaviour

This type of payload carries out activities such as displaying a text message that may be a threat, warning, joke or an advertisement banner on the user's computer. Payloads may include excessive consumption of computation resources leading to a denial of service or complete destruction of a specific resource on the user's computer.

Network payload behaviour

Network payloads collect information from the host computer and send information to another host on the network. The main difference between replication and payload-related network behaviour is shown in Figures 1 and 2. In both cases we observe the use of email addresses, URLs, IP addresses or IRC channels for connecting to network targets, but in the case of replication, network addresses are variable in nature (obtained from the infected host itself). In network-based payloads, the network targets are usually hard-coded in the malware program or are received in the form of command/control information from fixed source hosts on the Internet (these may come bundled in the malware code). Network-based payloads may be further categorized on the basis of the direction of relevant information flow occurring during the network activity:

- Outward flow – the information flows from the infected object to a remote object.
- Inward flow – the information flows from the remote object to the infected object.

The outward information flow is observed more in spyware, while the latter activity has been prevalent in adware.

CRITERIA DISCUSSION

In the previous section we observed that adware and spyware programs display only four of the six areas of

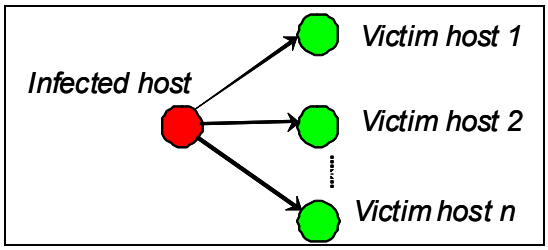

Figure 1. Replication network behaviour.

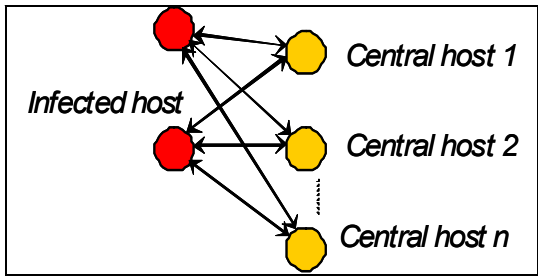

Figure 2. Payload network behaviour.

malware behaviour: installation, injection, concealment and payload behaviour. The presence of survey and replication behaviour is sufficient to eliminate a sample from the adware/spyware category. In this section we discuss some criteria that will aid the decision process.

Each criterion in this section is based on at least one of the following factors: integrity, privacy of user information and uninterrupted availability of system resources. Two types of criteria are discussed here, one is based on the program structure (achieved through static analysis) and the other is based on observations of program execution (achieved through dynamic program analysis).

Static analysis

A product should not misrepresent its creator. For example, if the sample has a resource section that falsifies a CompanyName resource attribute, it should be interpreted as abuse of user's trust and hence violation of system integrity through concealment. We have seen this in at least one spyware sample.

Several known adware and spyware programs have been found to be packed using a packer or an obfuscator. Packers such as UPX, do not fall into this category since these come with an unpack option. However, there are patched UPX variants that have been used in packing adware and spyware programs – these will be considered malicious. A number of executable packers are used solely or almost entirely by malicious software. Thus the use of 'questionable' packers or obfuscators, such as morphine, UPX (Redir), telock, petite, pepatch, fsg, pecompact, exestealth, obsedium, ezip,

pespin, krypton, exe32pack, pex and pediminisher, may be a strong indicator of questionable intent. Packing is used to conceal the URL strings or IP addresses embedded in the data, resource or text section of the sample. These are addresses of sites to which the adware program will connect. A good example for this is a version of *Adware-Lop*, which encrypts and stores all URL strings (in the resource section of the program) to which it may connect.

Dynamic analysis

During the installation phase, the product should inform the user (through the end user licence agreement, or EULA) exactly what the program will do. Sufficient information in the EULA should be provided, which can convince the user that the program will not invade their privacy or tamper with the system's integrity (with respect to security).

The presence of 'fine print funny business' in the EULA is a strong indicator that an adware/spyware program will be installed (see http://grc.com/oo/cbc.htm). Also, we need to check for the availability of a privacy notice from the developer during the installation process, or at least some information that points to the developer's website indicating the same.

The privacy policy and the EULA should hold true for all the components that are installed by the product at a current or later stage. Programs which announce and do not respect well-defined privacy and anonymity constraints will generally be classified as adware.

If the program exhibits a 'silent' installation behaviour, wherein the components are downloaded and installed on the user's PC while the user is browsing a website, this is a strong indicator of spyware/adware. This violates the system integrity of the user's machine and also consumes network and system resources which may be paid for unknowingly by the user.

The product should not install and execute hidden processes. Also, the product should not install and run programs that masquerade as well known system programs (e.g. svchost.exe). This is a form of concealment that violates the system's integrity since the information on the system is altered without the user's knowledge or consent. If the program modifies the browser settings or launches a browser automatically and displays arbitrary content, it is a violation of system integrity policy and warrants detection as adware.

For a product to qualify as 'well-behaved', it is very important that an uninstall feature be available and functional. Some adware programs, such as a version of *CommonName*, will automatically conceal or 'morph' themselves to an undetectable form when their uninstaller is executed.

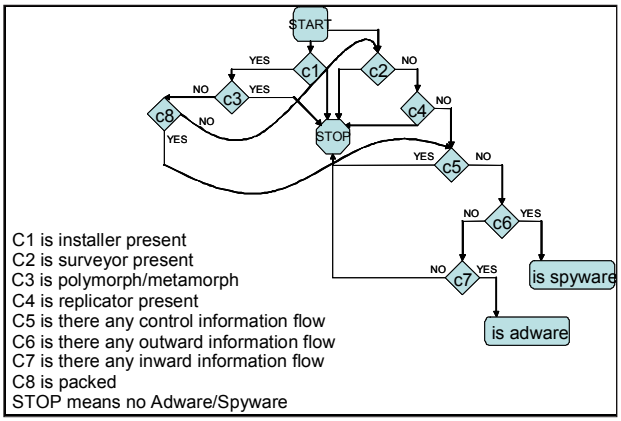

Figure 3. A possible procedure for determination of spyware or adware.

If the program sends information over the network without the user's consent (e.g. email address, banking or credit card information, account names or passwords), this is an indicator of spyware behaviour. Sometimes information gathered about the user's browsing habits may also be transmitted by spyware, which may not be acceptable to the user. If the injection of the program or one of its components is done in *Explorer* or a browser program, there is strong reason to believe that this is done to achieve concealment from outgoing firewall rule sets. This activity may carry out transmission of sensitive data which would have been blocked by a firewall. These types of activity violate the user's privacy. If the program receives command and control information from a machine outside the user's domain of control, this indicates a network payload behaviour which eventually violates the privacy, integrity and resource usage policy of the user.

Based on the behaviour studied in the previous section Figure 3 is a flow chart that summarizes the criteria we have discussed.

CONCLUSION

At the time of writing, adware and spyware account for nine of the top 20 threats detected by *McAfee* users (see http://vil.mcafee.com/mast/viruses_by_continent_internal.asp). Large ISPs and IHVs report that a large portion of their technical support calls arise from the side effects or degraded performance resulting from the presence of spyware or adware programs on their users' systems.

While legislative activity and litigation may alter definitions of spyware and adware in the future, it behooves us as an industry to develop consistent criteria and definitions. This will ensure better customer understanding and reduce legal exposure. It also allows us to assist those vendors who are making an honest effort to clean up their acts.