Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 75 (2005) 227–234
The Journal of Systems and Software
www.elsevier.com/locate/jss

A generic anti-spyware solution by access control list at kernel level

Sherman S.M. Chow *, Lucas C.K. Hui, S.M. Yiu, K.P. Chow, Richard W.C. Lui

Department of Computer Science and Information Systems, The University of Hong Kong, Pokfulam Road, Pokfulam, Hong Kong

Received 8 November 2003; received in revised form 30 April 2004; accepted 20 May 2004
Available online 15 July 2004

Abstract

Spyware refers to programs that steal the user information stored in the user's computer and transmit this information via the Internet to a designated home server without the user being aware of this transmission. Existing anti-spyware solutions are not generic and flexible. These solutions either check for the existence of *known* spyware or try to block the transmission of the private information at the packet level. In this paper, we propose a more generic and flexible anti-spyware solution by utilizing an access control list in kernel mode of the operating system. The major difference between our approach and the existing approaches is that instead of asking a guard to look for the theft (spyware) or control the exit of the computer (and hence giving the spyware enough time to hide the information to be transmitted), we put a guard besides the treasure (the private information) and carefully control the access to it in the kernel mode. We also show the details of an implementation that realizes our proposed solution.

1. Introduction

1.1. Background

Spyware refers to programs that steal the user information stored in the user's computer and transmit this information via the Internet to a designated home server without the user being aware of this transmission. These malware compromise every Internet user's privacy by collecting detailed user profiles that can be used for commercial or any other purposes. The stolen information includes the user's e-mail address, geographic location, web-surfing habits, etc.

Users may not be aware that there is a lot of private information stored in their computers, or they cannot see the value of information from the point of view of marketing companies. For example, a user's name and affiliation may enable a company to globally and uniquely identify the user and "offer" tailored marketing plan for the user. Web-surfing behavior and recently accessed files also reveal the user's interests and provide a good source of valuable marketing information. For example, a user who accesses LaTeX files very often is either a publisher or a researcher with high probability.

1.2. Working mechanism

The working mechanism of common spyware is rather simple. They try to find out their interested information from your files in the hard disk and system's settings (e.g. Microsoft Windows's registry), according to the predefined list of locations. More "intelligent" spyware can do so by getting instructions from a central server depending on the system information collected (e.g. version of operating systems). After they have collected enough information, they will transfer the stolen data back to the home server.

1.3. Spreading media

The popularity of the Internet speeds up the widespread of spyware. Nowadays, many Internet companies

* Corresponding author. Tel.: +852 28578263; fax: +852 29155702.
E-mail addresses: smchow@csis.hku.hk (S.S.M. Chow), hui@csis.hku.hk (L.C.K. Hui), smyiu@csis.hku.hk (S.M. Yiu), chow@csis.hku.hk (K.P. Chow), wclui@csis.hku.hk (R.W.C. Lui).

0164-1212/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2004.05.027

offer free services or free download of shareware to attract customers or visitors. The spyware usually embed itself in some of these free utilities such as Peer-to-Peer (P2P) file sharing programs, desktop customization tools, etc. They may also hide themselves in run-time libraries used by the utilities. By installing these programs into the computer, the bundled spyware is planted into the system too. These unethical shareware perform spyware activities that compromise the user's privacy apart from the advertised function. In order to make the communication with the home server as stealthy as possible, the program in which the spyware is attached is usually some program that requires access of the Internet such as P2P file sharing utility. Although there are no formal statistics, we believe that a significant proportion of spyware exists as an embedded component of Internet tools.

2. Related work

2.1. Existing anti-spyware techniques

There are many anti-spyware products in the market. [1] The mechanism of these anti-spyware products can mainly be classified into two categories: *Signature scanning* and *Network filtering.*

2.1.1. Signature scanning

The technique of signature scanning is similar to the same technique applied in the anti-virus software. [2,3] Basically, for each known spyware, a sequence of bytes, called the spyware signature, is identified. These signatures are stored in a database. The anti-spyware program checks the suspected software against this database. If the software matches any signature, it is considered to be a spyware. Signature scanning can only identify known spyware and users have to update the database frequently.

A related anti-virus technique is called *heuristics scanning* (Skormin et al., 2003), [2, 3] which searches for specific instructions within a program instead of specific signatures. However, heuristic scanning is not directly applicable in the anti-spyware software. For anti-virus, instructions like writing the boot sector may be used as an indicator of virus. For anti-spyware, one cannot easily determine whether the access of private information (for examples, accessing Microsoft Windows's registry, a large repository of system settings as well as private information) is a legitimate operation or not.

Moreover, in case these spyware scanning tools are not executed in the kernel's mode, their processes may be stopped by the spyware. An example (McWilliams, 2002) is RadLight Player, [4] a multimedia player comes with spyware. It detects and removes a popular anti-spyware tool called AdAware. [5] Despite of the above facts, most existing anti-spyware tools adopt the scanning approach.

2.1.2. Network filtering

Firewall is for protecting and controlling the interface between a private network and an insecure, public network. Packet sniffer enables users to view the packet being transported out of the computer. In these years, network filtering technologies such as personal firewall and packet sniffer are also used as an anti-spyware solution. [6] Personal firewall enables users to know which process is accessing the Internet and disallows some of the Internet operations. However, they may not be good anti-spyware solutions. Packet sniffers do not offer any protection, but only monitoring. Some personal firewalls lack packet sniffing facilities. Besides, to effectively use the firewall as an anti-spyware tool, one has to understand the mechanism of the Internet and also the packet level data, which is often difficult to be comprehended by normal users. If the spyware scrambles the data to be transmitted, it is not easily detectable. In addition, *flexible* control of the Internet access operation is difficult to achieve. For a network program that has spyware embedded, disallowing its Internet access may disallow the normal functioning of the program too.

To summarize, existing products are not generic and flexible. They either can only check for the existence of known spyware or it is difficult to only block the spyware-related activities rather than all related Internet access activities from the program in which the spyware is embedded. Some anti-spyware tools simply remove the programs with spyware attached from the system, and hence users suffer from losing the original ''good'' functions of the program.

2.2. Our contributions

In this paper, we propose a new generic and flexible anti-spyware solution. The core of the solution is based on the concept of access control list (ACL) at kernel level to safeguard the private information. Basically, whenever a process tries to access the private information, we check this access against the ACL. To avoid our process being stopped by the spyware, the checking is executed in the kernel mode of the operating system.

[1] Spychecker—download anti-spyware and privacy related freeware and shareware. http://www.spychecker.com (accessed on 30/04/2004).

[2] PC-Cillin. http://www.antivirus.com (accessed on 30/04/2004).

[3] Symantec AntiVirus. http://www.symantec.com (accessed on 30/04/2004).

[4] RadLight.net. http://www.radlight.net (accessed on 30/04/2004).

[5] Ad-Aware, http://www.lavasoftusa.com/software/adaware (accessed on 30/04/2004).

[6] ZoneAlarm. http://www.zonelabs.com (accessed on 30/04/2004).

A major difference between our approach and the existing solutions is that we control the access to the private information to make sure that the spyware cannot access the information while existing solutions can only look for known spyware or control the exit of the computer which is shown to be ineffective. In other words, we adopt a *behavior-based approach* instead of a signature-based approach to fight against spyware. Compared with built-in access control functionalities provided by the operating system, we block the request in a *stealthy* way such that the access returns zero byte data instead of an error, and hence successfully "cheating" the spyware.

2.3. Other related works

Our idea of using ACL is similar to the *Rule Set Based Access Control* (*RSBAC*) system (Ott, 2001). The RSBAC system is motivated by the need of a high granularity mandatory control of the system. In the RSBAC system, the Generalized Framework for Access Control (GFAC) (Ott et al., 1998) approach is implemented. One of the applications of RSBAC system is to do virus detection. In our solution, we use Access Control List (ACL) in Kernel mode to implement anti-spyware function. Besides, we considered files access, registry access (we borrow the term from Microsoft's paradigm for system's settings) and the Internet access at the same time.

Our idea also has a certain degree of resemblance to the WindowBox model (Balfanz and Simon, 2000), a mechanism supporting the complete separation of working environments (desktop in their terms) and each working environment can only access the resources associated with it. Our system can be considered as one large desktop containing many different sandboxes (Wahbe et al., 1993), while in the WindowBox model, there are many desktops, and each desktop contains a large sandbox. If users want to use WindowBox to protect their computer against spyware, they need to know exactly what each program is doing since one malicious program in a desktop compromises the security of the whole system. Besides, users need to make many decisions such as putting which applications into which desktop, what are the authenticated Internet connection for each desktop, and what files are accessible in each desktop, etc. If users made wrong decision in setting up their desktops, they may found that the necessary files are missing during the course of their work, or they cannot find the necessary information from the Internet since general browsing of Internet may not be possible in each desktop. An over-restrictive desktop may even make the applications malfunction since the files required are isolated in another desktop. In short, the flexibility and the security of the system are highly dependent on the user's domain knowledge. Most importantly, their solution does not address the problem that the spyware activity is performed with normal functions of the program. Using our tool, only the spyware activity is blocked but not the original good function of the program, e.g. you can still download files from a P2P network with a spyware attached P2P file sharing program.

The rest of the paper is organized as follows. Our proposed solution is presented in Section 3. Section 4 shows an implementation that realizes our approach, together with an experiment validating and verifying our implementation. Analysis of our proposed solution is presented in Section 5. We conclude our paper and give possible research directions in Section 6.

3. Our proposed solution

3.1. Scope

In our discussion, we limit our scope to only executable or library form of spyware. We do not consider the type of spyware that modifies and hence spoils the system library such as WebHancer. [7] Infectious behavior of spyware can be dealt with anti-virus techniques. Apart from executable or library of some programs, spyware also exists in other forms such as Active X control, [8] web page (with the help of cookies and web bugs), and browser helper objects, etc., [9] all of them are related to web access. For users to gain more control over their privacy information web sites they visit, initiative like Platform for Privacy Preferences Project [10] has been implemented already.

As discussed in Section 2, it is difficult to determine whether the access of private information is a legitimate operation or not. By *behavior-based approach*, we detect spyware by its behavior of attempting to access private information and sending out the data, but not based on the semantics of the operations prepared by a program, i.e. knowing the usage that the program will do with the data collected.

3.2. The framework

We observe that for a spyware to work, they must perform three kinds of functions: accessing the system's data, accessing files with sensitive information, and

[7] Counterexploitation: Foistware/Spyware: WebHancer. http://www.cexx.org/webhancer.htm (accessed on 30/04/2004).

[8] Microsoft COM Technologies. http://www.microsoft.com/com/tech/ActiveX.asp (accessed on 30/04/2004).

[9] Spywarelnfo: Browser Helper Objects (BHOs). http://www.spywareinfo.com/articles/bho (accessed on 30/04/2004).

[10] P3P: Platform for Privacy Preferences Project. http://www.w3.org/P3P (accessed on 30/04/2004).

calling back their server for transmission of stolen data. Hence our goal is to monitor and block these access according to the predefined ACL.

In our anti-spyware tool, two types of access control lists, the white list and the black list, are maintained. Basically, if a process is allowed to access a piece of private information, then an entry corresponding to this process and the private information will be stored in the white list. For example, the e-mail client should be allowed to access the user's e-mail, so a corresponding entry can be found in the white list. The black list, on the other hand, stores the information of black-listed processes so as to prevent process(es) from accessing sensitive private information. For example, known spyware attached in the downloaded program will be put in the black list. In our design, the white list takes precedence over the black list, as there will only be a limited number of programs that are allowed to access the private resources.

We maintain a separate white list and a separate black list for file access, registry access, and the Internet access. Each entry in the list is a three-tuple: (*Program/Process*,*Resources*,*Action*). *Resources* refers to the file/directory/drive name, registry entry / branches, and IP address/network address; for file, registry and the Internet ACL respectively. A special value *ALL* is used to denote all files/registry entries/IP addresses. *ALL* can be used in *Program/Process* to denote all programs/processes too. In white list, *Action* can only takes the value of *allow*. In black list, *Action* can take the value of *block* and *ask*. If an *ask* value appears in the entry's *Action* field, it means that our program will ask for the user's permission to accept/deny the operation whenever the specified process is accessing the specified resources. As an example, if we found (*Spyware.exe, C:/windows/cookies/index.dat, block*) in the black list, then the file reading request issued by the process *Spyware.exe* on the file *C:/windows/cookies/index.dat* will be denied.

Notice that we are not merely combing a file system locker and firewall into a two-in-one anti-spyware tool. If we use firewall, the spyware embedded process get blocked from Internet access and hence cannot perform legitimate functions. While using our solution, the spyware embedded process cannot get the privacy data at the first place, so it is harmless to let it to have Internet access and hence the legitimate functions can be performed perfectly. In other words, our solution not only protects against spyware that exists as a separate executable that accompany otherwise legitimate applications, but also spyware that has been embedded with another originally reliable program.

Fig. 1 shows the conceptual architecture of our proposed anti-spyware tool. We implement the access control module of our anti-spyware tool as a filter driver on top of the file access, registry access and the Internet access driver in order to block the respective requests. By doing so, private information will not be leaked to outside. At the same time, spyware bundled with the program that requires Internet access can still work.

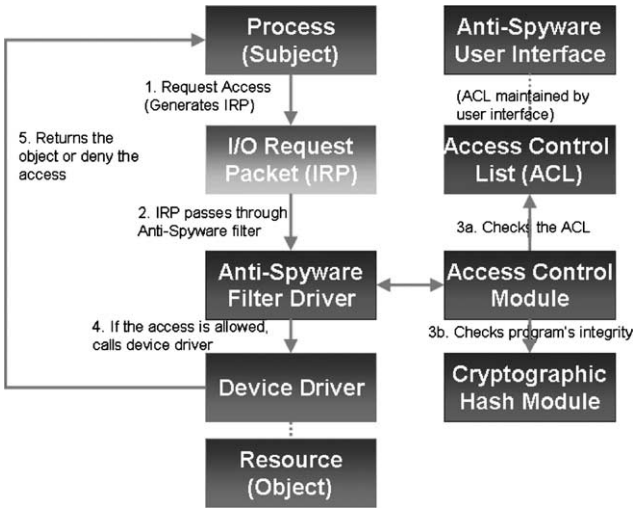

Fig. 1. Conceptual architecture of our proposed anti-spyware tool.

Consider the situation that you finally found a seminal paper in privacy protection you searched for a long time using a P2P program $Proc_P$. However, this P2P program is attached with spyware $Proc_A$ which aims at stealing your browser's cookies $File_C$. By its design, $Proc_P$ calls $Proc_A$ before you download any located object. In this case, you can simply block the process $Proc_A$ from accessing the file $File_C$ in the black list, and not putting $Proc_P$ in the black list. Since only the file reading request is blocked, but not the file open request, the program $Proc_P$ will not hang and wait indefinitely, but it can only send out zero-size data to the home server and starts downloading the useful paper for you. Notice that our solution works for the case $Proc_P = Proc_A$.

This above mentioned feature is possible only when using filter driver but not the underlying access control module of the system (e.g. in Balfanz and Simon, 2000). Our solution blocks the request in a stealthy way, while the latter will block the access of the protected resource and notify the program about the failure.

4. Implementation and experiments

We implemented our proof-of-concept anti-spyware tool that works in the Microsoft Windows platform (98/2000/ME/XP/2003) for the purposes of experimentation and validation.

The tool is composed of five parts:

1. Graphical user interface (GUI): The user can setup the black list/white list in GUI. Process with suspected spyware actions will be displayed. The user

can also view the information related to the resources, and decide whether to allow/block the access or kill the process. (See Fig. 2)
2. User-Kernel mode communication module: This module is responsible for the communication between filter drivers and GUI.
3. Registry filter: Device driver that employs the technique of system-call hooking to intercept any registry access operation, implemented in kernel mode.
4. File I/O filter: Filter device driver in kernel mode, intercepts any file I/O Request Packet (IRP).
5. TDI filter: Filter device driver in kernel mode, intercepts any packet of transport driver interface (TDI) protocol in network stack of Windows. (The networking components of Windows are integrated with the I/O system and the Win32 API, and the network stack adopted is not the same as the OSI reference model defined by ISO.)

A device driver is a set of routines that the operating system can call upon to perform various functions related to a particular kind of hardware device. Our device drivers are implemented in the Windows Driver Model (WDM) platform (Oney, 1999). Windows Driver Model (WDM) is the unified driver architecture which supports Windows 98, Windows ME, Windows 2000, Windows XP and Windows 2003. WDM adopts a layered driver architecture, in which we can insert filter driver above and below functional device driver. Kernel-mode drivers are packet-driven, i.e. all I/O requests are submitted using IRP. When filter driver is used, each IRP that goes down or up the driver stack will be processed by the filter drivers. By installing TDI filter, file I/O filter and Register filter, all the running processes in Windows are monitored.

4.1. Experiments

To the best of authors' knowledge, there is no other tools providing the same set of functions, so we devised our own experiments.

To show our tool's capability, we implemented a simple program *SpywareX* to test the registry filter. *SpywareX* reads the web-surfing behavior of the user by reading the entries in the registry *HKey_Current_User/Software/Microsoft/InternetExplorer/TypedUrl* (which store URL typed by the user in Microsoft Internet Explorer) and saves them in a local text file. We also use *ftp.exe* provided in the Windows package to test our file I/O filter driver and TDI filter driver.

We set our black list as follows. We disallowed *SpywareX* from reading the entry *url*1 of the above mentioned registry branches (i.e. all typed URLs except the most recently typed one can be accessed). We also disallowed *ftp.exe* from reading *cookies.dat* (storing information related to cookies) and accessing *gatekeeper* (gateway to the Internet) in *csis.hku.hk* domain. The first experiment

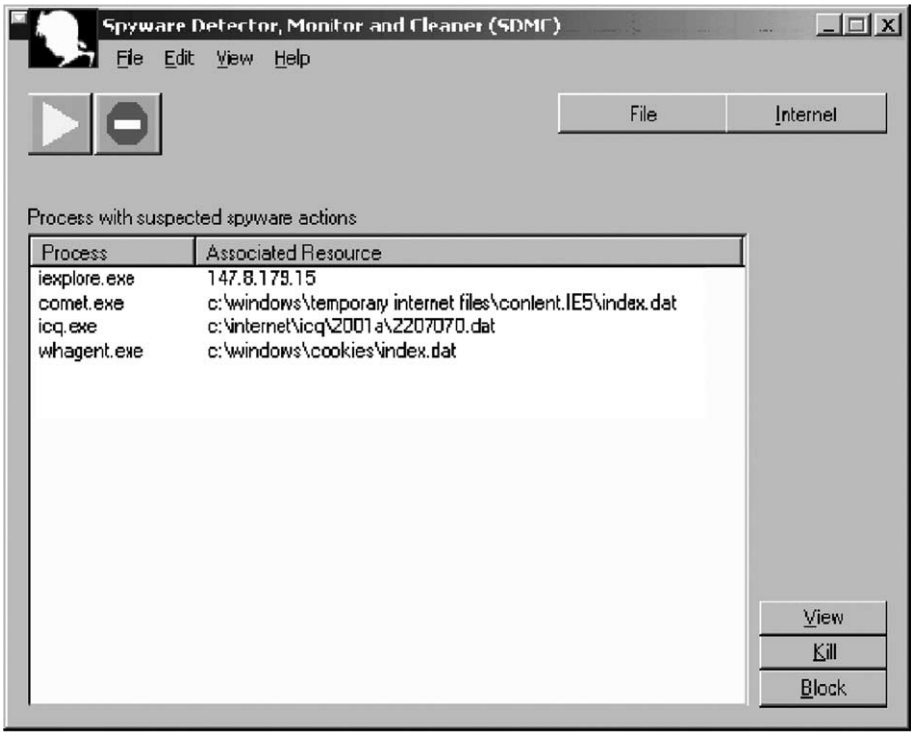

Fig. 2. Proof-of-concept prototype.

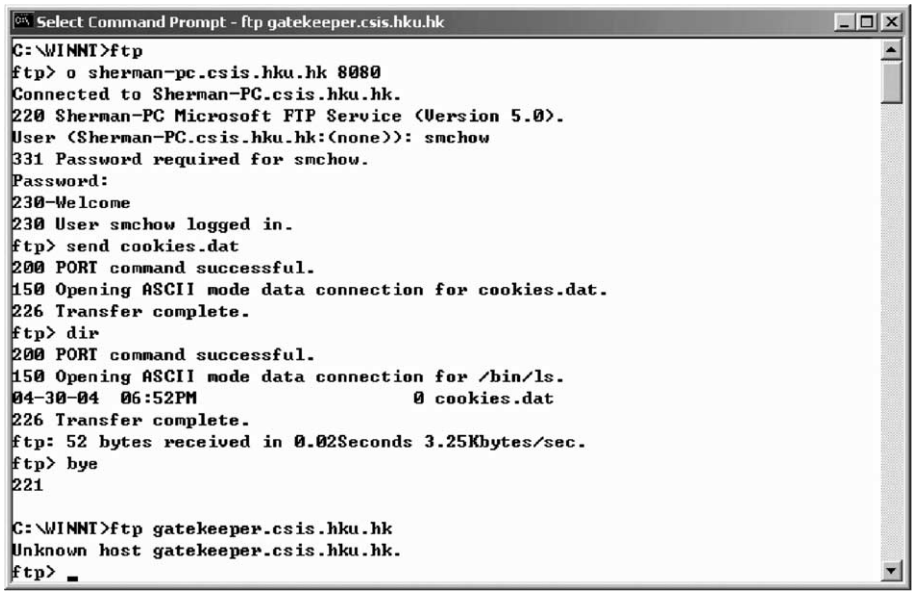

Fig. 3. Result of experiments.

tests the blocking ability of registry filter. We executed *SpywareX*, and found that the recently typed URLs were saved in the text files, except the most recently entered one. The second experiment tests the file access blocking ability of file I/O filter. We executed *ftp.exe* to upload *cookies.dat*. We found that the upload operation was reported as "successful" by *ftp.exe*; however, only zero bytes were transferred, which means our file I/O filter blocked the access of the file successfully. The third experiment tests the Internet access blocking ability of the Internet filter. We used *ftp.exe* to access *gatekeeper.csis.hku.hk*, as expected, connection fails. The results of the second and the third experiments are shown in Fig. 3. To conclude, all experiments show positive results and our filters are successfully implemented.

Regarding the performance issue, we believe that our anti-spyware process is an economic method since the loading of one more filter driver and the checking of ACLs at kernel level will not make notable performance degradation.

4.2. A more realistic example

In the above experiments, we demonstrate that our anti-spyware tool can actually filter out illegal operations. So, the black list works properly. In the following, we further demonstrate how to use the white list and black list together to allow legitimate process to access private information while blocking other processes from accessing the information. In this example, we use the message dialogue history of ICQ (a popular Internet instant messenger) [11] as an illustration. The user wants to maintain his privacy regarding his message dialogue history of ICQ. Suppose his ICQ number is 763092, he firstly add the file entries (*ALL*, *C:/Program Files/ICQ/2003a/763092.dat*, *block*) and (*ALL*, *C:/Program Files/ICQ/2003a/763092.idx*, *block*), together with the registry list entry (*ALL*, *HKEY_CURRENT_USER/Software/Mirabilis/ICQ*, *block*) to the black list, where the *.dat* and *.idx* files stores ICQ's message dialogue history. Then, the corresponding files entries (*icq.exe*, *C:/Program Files/ICQ/2003a/763092.dat*, *allow*), (*icq.exe*, *C:/Program Files/ICQ/2003a/763092.idx*, *allow*), and the registry list entry (*icq.exe*, *HKEY_CURRENT_USER/Software/Mirabilis/ICQ*, *allow*) are added to the white list. Since the white list takes precedence over the black list, all programs except *icq.exe* is not allowed to access this user message dialogue history and ICQ's setting. No spyware can steal his ICQ message dialogue history from his computer successfully.

4.3. Extra features

In our implementation of the anti-spyware, we implemented the following extra features.

4.3.1. Integrity checking

Suppose *icq.exe* is allowed to access the file storing your ICQ's message dialogue, a spyware knowing this fact may attach itself to *icq.exe* to steal your message dialog, i.e. a good process is no longer reliable now. To address this attack, we employ integrity checking on the executable in the white list. Any unmatched hash value of the executable is detected.

[11] ICQ.com. http://www.icq.com (accessed on 30/04/2004).

4.3.2. Protection of privacy information

Apart from specific files and registry entries to be protected, we also filter the packets containing the user specified string like credit card number; preventing the situation that there is some private information stored in a location not protected by the ACL. However, this approach gives a new line of attack that the spyware can directly access the file storing these user specified strings. We solve this problem by disallowing programs other than our tool to access the file.

4.3.3. Wizard mode for setting ACL

Usually the user only needs to protect a few set of files, so a wizard mode is provided to aid the user in setting up the ACL quickly. In the wizard mode, the user can specify the set of sensitive files (e.g. e-mail in mail client, work related data of the user's enterprise, etc.) and the corresponding legitimate programs, then ACL entries that deny the access to these files by all other programs are automatically added.

5. Analysis

5.1. Usability analysis

The use of predefined ACL is *not* the same as the use of signature database. In the signature database, we store the signatures of all known spyware. Updating the signature database requires the help of domain expert. On the other hand, using ACL in our system, we store the private information to be protected, which is under control of the information creator, who is the user himself/herself. Basically, our scheme can protect the private information from new or unknown spyware provided the ACL is set appropriately. In practice, one concern is the location of these private information. As raised by (Balfanz and Simon, 2000), maintaining ACL is a complicated task for the average user. However, a rule-of-thumb is to allow only the program creating/ modifying the file to access the file, e.g. only *icq.exe* can access file under the folder *C:/Program Files/ICQ/2003a*. Moreover, the number of files storing sensitive data is not large for common applications. (For examples, ICQ stores the message dialogue in the above two mentioned files, Microsoft Outlook stores the personal e-mail in a single file called Outlook.pst. [12]

We hope that every software vendor to publish this information together with their software so that users can set the ACL accordingly, or making the process of setup ACL automated. Indeed, in today's networked computing environment, it is essential for the software vendor to tell users what data its program stored for them, as normal users may face the threat that an innocuous looking file is being stolen by social engineer (Mitnick and Simon, 2002).

5.2. Security analysis

We give a security analysis similar to that of (Balfanz and Simon, 2000) here. We define an implementation of our idea is *secure* if there is no *covert channel* for bypassing our access control, In our implementation, a covert channel is an I/O request that is not conveyed by using IRP, for example, accessing the file by physical access according to the file's physical location in the hard disk. [13] We considered a number of covert channels like renaming the filename of the protected file before read access, fast I/O (performed if the file is cached, the data will be accessed directly from the system cache by the Cache Manager, instead of creating IRP) and paging I/O (I/O about page swapping) in our prototype, but we cannot give a formal assurance that no other dangerous channels exists since Windows is not a open source system. However, we try to assure the security of our solution by implementing it as "low" level as possible. As suggested by (Balfanz and Simon, 2000), every program has to go through certain path of the kernel to do anything useful. In view of this, we implement our anti-spyware solution as a kernel filter driver.

Moreover, implementing access control at kernel level means the ACL is also stored in kernel level. Normal processes at user level are not capable to tamper the kernel memory, which ensure the efficacy of our method.

Indeed, the efficacy of our solution also depends on the design of the underlying operating systems. Windows's registry is a system-wide place for storing system's settings, some privacy information like the name and organization of the registered user is placed inside registry. It may be inconvenient for the user if most programs are blocked from the accessing of this information. We sincerely hope that the design of operating systems in the future will have solution to address this kind of potential privacy problems, like classifying the system settings which contain privacy info into different classes.

6. Conclusion and future work

To conclude, we proposed a generic anti-spyware tool which is based on the execution behavior of the spyware, but not any predefined or collected signatures on the spyware programs' files. Our solution is also flexible than other anti-spyware solutions such as firewall in the sense that normal functions of a program will not

[12] http://www.microsoft.com/outlook (accessed on 30/04/2004).

[13] To solve this problem, we may try to prevent the process from querying the physical location of the protected file.

be affected even its spyware activities are blocked. As a final note, many "user-friendly" malicious hacking tools like NetBus, SubSeven are easily available nowadays and many script-kiddies use these tools to hack into friends' computer to steal files such as diary, photos, assignments, business documents, ID card number, credit card number, etc. Using our anti-spyware tool, users can easily specify what processes can read these sensitive files and enable a better control of private information using our anti-spyware tool, thus preventing the script-kiddies from compromising their privacy.

We notice that a poor user interface may jeopardize the security of the system (Whitten and Tygar, 1999). In our prototype, we employ ACL, a simple data structure to implement the security policy. The use of simple ACL makes the user interface as simple as possible. A possible research direction is to choose between the complexity of security model and the complexity of the user interfaces so that a user with the least computer knowledge can still use our tool effectively.

Other future works include devising an experiment to find out the performance information of our tool, and studying the semantic of the program to devise a more intelligent behavior-based approach against spyware.

References

Dirk, B., Simon, D.R., 2000. WindowBox: a simple security model for the connected desktop. In Proceedings of the 4th USENIX Windows System Symposium.

McWilliams, B., 2002. ComputerUser.com News: Anti-spyware program targeted by multimedia player. Available from <http://www.computeruser.com/news/02/04/24>.

Mitnick, K.D., Simon, W.L., 2002. The Art of Deception. Wiley Publishing, Inc.

Oney, W., 1999. Programming the Microsoft Windows Driver Model. Microsoft Press.

Ott, A., 2001. The rule set based access control (RSBAC) Linux kernel security extension. In: Proceedings of the 8th International Linux Kongress, 2001.

Ott, A., Hübner, S.F., Swimmer, M., 1998. Approaches to integrated malware detection and avoidance. In: Proceedings of the 3rd Nordic Workshop on Secure IT Systems, Trondheim.

Skormin, V.A., Summerville, D.H., Moronski, J.S., 2003. Detecting malicious codes by the presence of their gene of self-replication. In: Computer Network Security, MMM-ACNS 2003, Russia, Proceedings, Lecture Notes in Computer Science, vol. 2776. Springer.

Wahbe, R., Lucco, S., Anderson, T.E., Graham, S.L., 1993. Efficient software-based fault isolation. In: Proceedings of the Symposium on Operating System Principles.

Whitten, A., Tygar, J.D., 1999. Why Johnny Can't Encrypt: A Usability Evaluation of POP 5.0. In: Proceedings of the 8th USENIX Security Symposium.