A Discretionary Access Control Method for Preventing Data Exfiltration (DE) via Removable Devices

Duane Wilson[1] and Michael K. Lavine[2]

[1] Network Security Branch, U.S. Army Research Laboratory, USA
[2] Information Security Institute, John Hopkins University, USA
dwilson@arl.army.mil, mlavine@jhu.edu

Abstract. One of the major challenges facing the security community today is how to prevent DE. DE is the unauthorized release of information from a computer system or network of systems. Current methods attempt to address this issue by controlling the information that is released over the Internet. In this paper, we present a host-level discretionary access control method that focuses on exfiltration via removable devices (e.g. thumb drives or external hard drives). Using XML to store extended file attributes, we classify files based on user-defined distribution levels and the community of interest to which they belong. Files are classified with a distribution statement upon creation and re-classified (if necessary) when modified. By monitoring the access to all classified files present on a file system, we allow or prevent release of this information based on predefined policies. With this approach, we show that the unauthorized release of information can be prevented by using a system of accounting that is tied to access control policies. Users are given the authority to transfer files to a removable device according to their current access rights. As a proof of concept, our method demonstrates the value of using accounting as a means of preventing data loss or theft. Our approach can be applied to a variety of data types found on a file system including: executables, archived files, images, and even audio or video files.

Keywords: Data exfiltration, extended file attributes, alternate data streams.

1 Introduction

DE is defined as the unauthorized release of information from a computer system or network of systems. This release frequently occurs as a result of a malicious attempt to steal information. However, it is possible that trusted entities (i.e. employees, supervisors) can unintentionally cause data to be stolen themselves (i.e. via negligence or violating company policy). This unintentional release of information can be prevented by instituting a few simple controls that prevent individuals from being able to transfer data from a system without having the correct permissions. These controls can also handle automated exfiltration processes by associating each transfer process with a user. The DATALOSS-db is an open security foundation for gathering information about events involving the loss, theft, or exposure of personally identifiable information [1]. The information gathered by this organization has been

S. Goel (Ed.): ICDF2C 2009, LNICST 31, pp. 151–160, 2010.

used in research by numerous educational, governmental, and commercial entities and has been collected since the year 2000. According to their research, 22% of DE incidents that have occurred in an educational, government, medical, or corporate domain can be tied to an insider (i.e. a trusted non-external entity). These incidents are categorized as both accidental (16%) and malicious (5%). One percent of the incidents are attributed to an unclassified insider. For the current year, insider-related incidents remain consistent with the trend, accounting for 20% of the DE incidents that have occurred. See Figure 1 below for the incidents recorded since 2000 and those attributable to the current year.

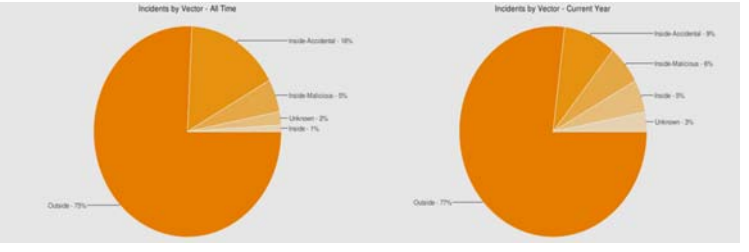

Fig. 1. DE incident statistics

These statistics demonstrate that although the majority of DE incidents are attributable to sources that are outside of an organization, insiders (i.e. trusted users) account for a large percentage of the incidents that occur on a regular basis. Therefore, it is essential to consider both outsider and insider threats as potential causes of data that is exfiltrated from an organization.

In [2], Giani, Berk, and Cybenko presented a taxonomy of the different methods of DE. According to their taxonomy, exfiltration methods can be divided into three distinct domains: Network, Physical, and Cognitive. Each domain can be further subdivided into two categories: Usually Benign or Known Malicious. Our research focuses on the Physical/Usually Benign category. The reason for focusing on this category is because the use of these devices is not usually nefarious; however, it is possible to use these devices (or methods) to exfiltrate data; intentionally or unintentionally. Typical devices in this category are: printing devices, CD/DVD, Disk, USB, and Digital Media Players.

Our proposed method is comprised of three major components: document tagging tied to distribution (release) information, file and process monitoring, and recording file access/modification/transfer actions based on the user. We defined a sample user environment of eight users (i.e. four pairs). Each pair was assigned the same classification level which corresponds to a distribution level (e.g. A, B, C, or D). This was done to demonstrate how the system would respond when an authorized or unauthorized transfer took place. Upon creation, documents were tagged with a distribution level and community of interest label. This information was used to authorize or deny a user's right to transfer a document. In order to determine when a transfer to a removable device was taking place, we monitored the file system and processes. When a transfer was initiated, we determined its validity and recorded specific information about the transfer for further analysis. Used in parallel, these

components provided an effective access control system that introduced controls that can prevent data from being exfiltrated via removable devices. Through logging, our system also provides accounting information associated with each transfer which has many benefits in a forensic investigation. Each aspect of the system is discussed in further detail in Section Three below.

The rest of the paper is organized as follows. Section two will discuss some of the previous and current work that has been done in the area of DE prevention. Section three outlines our approach to preventing DE via removable devices. Section four provides some preliminary results. Section four discusses our proposed future work in this area. Section five concludes the paper.

2 Related Work

Publicized efforts in the area of DE have been primarily focused on outbound network traffic analysis and the detection and mitigation of covert channels. Recently, the direction of the research in this area has shifted to application level exfiltration detection. To date, little work has been published that addresses exfiltration at the physical host level.

According to [3], outbound traffic analysis is used primarily to prevent a computer system from being utilized as an attack launcher or intrusion relay. This type of analysis is concerned with the identification of system activity that contains attack signs that target the network layer. This technique is very similar to ours in that it uses host information to determine the source and function of malicious-looking outgoing payload. It differs in that it examines traffic at the network layer. Effective DE requires the examination of actual data content. At the network layer, the most effective form of this analysis is called Deep Packet Inspection (DPI). This method examines a network packet (i.e. header and data) for protocol compliance and other potentially nefarious content (e.g. malicious logic) as it passes an inspection point [4]. The primary benefit of this approach is that it can stop an infection that relies on the network to propagate an infection (i.e. computer worm). However, examining each packet non-passively as it traverses a network segment adds significant overhead to normal network throughput. This makes it an infeasible approach for preventing DE at the network layer. Also, it differs from our approach in that analysis is performed at the network layer, not on the host.

A covert channel is a way to communicate information in a manner that hides the fact that a communication channel is actually established at all. [2]. According to [2], there are two main classes of covert channels: timing and storage. A timing covert channel is a communication between two machines that consists of sending and receiving data using a specific timing pattern that bypasses usual intrusion detection techniques. An example of this covert method is based on the idea of exploiting time delays between transmitted packets (i.e. longer time delay = binary 1, shorter time delay = binary 0). A storage covert channel simply stores information in the unused field of a packet (e.g. TCP/IP header). In this research, both the timing and storage covert channels are methods of exfiltration found at the network and/or transport layer. Therefore, the detection/prevention of covert channels does not address exfiltration that occurs on the host themselves.

As stated in the previous section, there are known techniques for detecting covert channels in Layer 3 (network) and Layer 4 (transport) protocols. There is relatively little work in detection at Layer 7 (application) despite the existence of many layer 7 channels. [5,6]. In this paper, an application is presented (Glavlit) that can prevent unauthorized release of data from a protected network while allowing authorized information to pass unhindered. Their approach has two major processes: vetting and verification. In the vetting phase, an authority is designated to determine whether an object, as a contiguous data segment, is appropriate for external release. This is similar to our process of verifying that a file can be released to an individual in a given community of interest based on its distribution label. The verification phase simply ensures that an object was previously vetted before releasing it across a designated network boundary. This supports our procedure of authorizing the individual attempting a transfer to a removable device based on their classification level and community of interest. Our approach does not address exfiltration that may occur via the network, but focuses on exfiltration via removable devices.

3 Technical Design Overview

Our initial setup consisted of the following major steps: choosing a deployment platform, setting up user accounts, creating a shared folder, installing MySQL server, and creating database to store matrix information and file transfer logs. We selected Windows XP Professional with Service Pack 3 installed as the deployment platform for our analysis tool. The application operates as a Windows service and was configured to run with administrative privileges. Running the program with administrative privileges provides full purview over all of the accounts on the system and ensured that it would be solely managed by a system administrator. This is an ideal setup since some aspects of the application operate at the kernel level (i.e. file system monitoring) and required root privileges. To demonstrate our approach in a multi-user environment, we created eight user accounts. The user accounts corresponded to the account names in the user matrix (described in Table 1 below). We also set up a shared folder that each user on the system had read/write access to. This shared folder was used to simulate a network share drive that can be simultaneously accessed. To store the information contained in the user and community of interest matrices, we used a MySQL database. This database was also used to log all (i.e. successful and unsuccessful) attempts to transfer a monitored file to a removable device. From a forensics standpoint, this would allow an investigator to issue queries against the log data very easily during the course of an investigation. The database consisted of three tables: user_matrix, coi_matrix, and transfer_info. The user_matrix and coi_matrix contained information about each user and community of interest details, respectively. The transfer_matrix contained information about each transfer that occurred on a monitored system.

After the setup of the “user community” and database, we internally assigned each user a classification level and to a community of interest and stored this information in the user matrix. Though not shown in table 1, it is important to note that the community of interest is hierarchically arranged. More specifically, a user is allowed to be in more than one community of interest over a period of time, but not at the

same time. If this were allowed in this model, it could potentially compromise the integrity of a particular community of interest. Within the community itself, users can access all documents at their clearance level as well as those below. This makes the model hierarchical within a community of interest, but not across disparate communities of interest. The user matrix contained three pieces of information about each user: user label (or username), user classification level, and a community of interest identifier. This information was stored in the database created in the initial setup. For our purposes, the user classification level and community of interest information remained the same throughout testing. However, in practice both user classification levels and communities of interest can change based on number of factors (e.g. job responsibility or company change). Discussion of these factors is outside of the scope of this paper. A sample user matrix of four users is shown below in Table 1.

Table 1. Sample user matrix

User Label	User Classification Level	COI
User1	A	2

The community of interest matrix maps the community of interest number to an actual description of the community using the hierarchical model mentioned in the previous section. This mapping was used for logging purposes only and was also stored in the database created during the initial setup. Table 2 details a sample community of interest matrix corresponding to the user matrix above.

Table 2. Community of interest matrix

Community of Interest	Of Description
2	Hydro Particles
3	Nano Technology

The classification of a file was done upon creation. Users will have the option to reclassify a file upon modification. In order to determine when a file was created or modified, we monitored the file system for these specific change events. Once one of these events occurred, we prompted the user to either classify the file upon creation or reclassify (if necessary) upon modification. The created file was automatically assigned to the community of interest(s) to which the user currently belongs. The assumption was that a user will create files that contain information related to one or more of their communities of interest.

The monitoring process began as soon as users logged into the system (i.e. operating system booted up). The program was invoked upon the creation of files of the type that are being monitored. Our research consisted of monitoring Microsoft Word Documents. It could easily be expanded to monitor files of different types, which will be addressed in our future work. It is important to note that this part of the application must be run in kernel mode, since access to specific operating system functions are necessary to determine when a file is created and/or modified.

To facilitate DE prevention it was necessary to define a scheme to classify data (i.e. in our case MS-Word documents) based on the information that they contain. Our classification scheme is comprised of two pieces of information: a distribution level and a community of interest. A distribution level (or statement) is a notation that marks a document according to its security classification to ensure it is circulated only among the authorized recipients. A community of interest is simply a group of individuals that have a "need to know" the same body of information. For our purposes, we use letters to designate a distribution level and numbers to designate a community of interest – with 'A' being the highest distribution level and '1' being the most restrictive community of interest. For instance, a document with a tag 'C1' would be open to an individual who possesses a clearance level 'C' and is a member of community of interest '1'.

There are three methods we evaluated for tagging the file with a distribution level and assigning it to a community of interest: Extended file attributes, Alternate Data Streams, the extensible markup language (XML). Extended file attributes are file system features that enable a user to associate computer files with metadata that is not interpreted by the file system [7]. An Alternate Data Stream (ADS) is a relatively unknown compatibility feature of the Windows NT File System. An ADS provides the ability to fork file data into exiting files without affecting their functionality or size. ADS capabilities were originally conceived to allow for compatibility with the Macintosh Hierarchical file system (HFS); where file information is sometimes forked into separate programs [8]. XML can be used for multiple purposes including storing configuration information, file attributes, or any other type of metadata.

Both ADS's and extended attributes are usually limited in size to a value significantly smaller than the maximum file size. ADS's, for instance, are restricted to a size of 4 Mb. They are both very simple to use. Extended attributes are similar in structure to a hash table entry, containing a key and a value pair. ADSs are simply metadata that can be attached to a file on the file system using a colon after the file's name. For example, the following command can be used to create an ADS containing the phrase "Hello World" attached to a file called "test.txt" (echo "Hello World" > test.txt:hidden). This command demonstrates how to view the contents of an ADS from the command line (more <test.txt:hidden).

Lastly, since both extended attributes and ADSs are considered to be properties associated with a file, whenever a file is deleted, they are deleted along with the file. This also allows us to check this information without actually opening each file, making it very efficient.

ADS's initially proved to be the most appropriate choice for tagging files for the following. Firstly, the information that is present in an ADS is not accessible by either a directory listing command ('dir') or using Windows Explorer for viewing the metadata associated with files on a file system. If used in a multi-user environment, the classification level and community of interest label should not be easily viewable. This information is very sensitive and would compromise the integrity of the system if compromised. Secondly, ADSs have come to be used legitimately by a variety of programs, included native Windows operating system to store file information such as attributes and temporary storage [9]. This demonstrates that this particular capability of the OS was designed for these type of purposes (i.e. storing file attribute information), making it an ideal choice for storing our tags. Lastly, extended

attributes are not supported on the NT file system; which was the file system used for this research project. However, there were no appropriate APIs for working with ADSs programmatically. Therefore, we used XML to store the tag information associated with each file created. The file is loaded when the Windows Service starts and updated dynamically as files are created, modified/edited, and deleted. The file has two levels of tags: FileTags (to designate the type of XML file) and FileTypes (to list of different file types). The contents of a sample XML tag file (containing one file) are below:

```
<?xml version="1.0" encoding="UTF-8"?>
<FileTags>
 <FileTypes>
  <TEXT>
   <a.txt>
    <DistributionLevel>8</DistributionLevel>
    <CommunityOfInterest>1</CommunityOfInterest>
   </a.txt>
  </TEXT>
 </FileTypes>
</FileTags>
```

Fig. 2. Sample XML file (tagfile.xml)

In a forensic investigation, an investigator needs to be able to answer the basic questions of who, what, where, when and how about the crime that has occurred. Our tool provides these answers by recording information about each transfer to a removable device that occurs on a monitored system. Whenever a transfer occurred, we logged the following information: user performing the transfer, certain attributes of the file, the system and removable device involved in the transfer, and the time of the transfer.

When recording information about a file, it was important to take into account its distinctive features. Some of the information about a file that is useful to record include the: file name, size, hash value, and access tags. For the computer system involved, the system name (if available) and MAC address should also be logged. The name and type of removable device that the document is transferred are also relevant items to store. For our purposes, we recorded a subset of the aforementioned information. Each time a transfer operation took place this information was stored in the database created during the initial setup. A 'flat file' could have been used for logging; however, querying the data would have been less efficient than using a database. By design, databases are indexed for efficient information retrieval (i.e. with a primary key per table). For flat files, an index would have to be built separately (e.g. based on offsets or line number) in order to increase the efficiency of retrieving information.

As detailed above, our application functions as a Windows service and polls the file system for transfers to removable devices. We first began by initializing the variable that stores whether or not an attempted transfer was successful. Next, we obtained the username of the person attempting to perform the transfer.

We utilized this username to obtain the user's clearance level and community of interest. Then, we obtained the distribution level and community of the item being transferred (e.g. document). Finally, we determined whether to allow or disallow a transfer based on these simple rules. The first check was based on the clearance level

of the user and the distribution level of the document being transferred. If the user possessed a higher or equivalent clearance level than that of the document being transferred, the next check was performed. Otherwise (i.e. user clearance level was lower that the document distribution level) the transfer was not allowed. The second check was a co-requisite rule that must be met along with the rule previously mentioned in order for a transfer to be allowed. This particular rule checked to see if the user belonged to the community of interest that holds the right(s) to the information in the document being transferred.

T_SUCCESS is only set to true if the transfer was permitted. In either case, all information about the transfer was logged in the database including: user, file, system, device, time, transfer mechanism/success. Our algorithm can be enhanced to take into account more advanced access control policies. Pseudo code of algorithm below:

LEGEND	**ALGORITHM**
	(condition clause expanded for clarity)
USER_CL: User clearance level	0) Set T_SUCCESS = 0
	1) Read username
USER_COI: User community of interest	2) Get USER_CL (username)
	3) Get USER_COI (username)
	4) Read DOC_DL
DOC_DL: Document clearance level	5) Read DOC_COI
	6) If (USER_CL = = DOC_DL) && (USER_COI = = DOC_COI) allow transfer
DOC_COI: Document community of interest	T_SUCCESS = true
	7) If (USER_CL > DOC_DL) && (USER_COI = = DOC_COI) allow transfer
	T_SUCCESS = true
T_SUCCESS: Variable denoting whether a transfer was successful or not	8) If (USER_CL < DOC_DL) && (USER_COI = = DOC_COI) disallow transfer
	9) If (USER_CL > = DOC_DL) && (USER_COI != DOC_COI) disallow transfer
	10) Log {user, file, system, device, time, T_SUCCESS} in Database

4 Research Results

Based on the research design, we were able to show the results of querying the database based on the following: user, file name, and transfer success. Below are some *sample* results:

Query: select * from transfer_info where user = 'tprice'

User	File	System	Device	Time	Success
Tprice	b.txt	WILSON_COMPUTER	SanCruzer	14:28:32	0

Query: select * from transfer_info where file = 'b.txt'

User	File	System	Device	Time	Success
Tprice	b.txt	WILSON_COMPUTER	SanCruzer	14:28:32	0

Query: select * from transfer_info where success = 1

User	File	System	Device	Time	Success
natashanduane	a.txt	WILSON_COMPUTER	SanCruzer	14:15:17	1

As stated previously, more information can be stored about each attempted transfer to a removable device as desired by the system administrator and/or forensic investigator. However, the use of a database is essential to retrieving this information quickly. A useful extension to this work could be to provide an interface to eliminate the need to know the Structured Query Language (SQL) in order to retrieve results. Searches would be issued based on criteria provided by the user. Other potentially interesting queries are provided below could include transfers by a specific user with a certain time range.

5 Suggestions for Further Research

Below are several suggestive areas for future research. Our research method can also be used to prevent exfiltration of other data types using a similar tagging/monitoring mechanism. Since the distribution labels were "applied" to each file using a separate XML file and not on the files themselves, we believe they can be applied to any type of data that a file system recognizes. Therefore, a viable extension of our work would be to allow a user to specify the data type(s) to monitor – in addition to the directory that is being monitored. This specification would serve as a mechanism that filters files based on their type unless the user requires all files to be monitored. A combination of factors including: magic numbers (byte signatures at the beginning of files that designate the file type), file size, and file extension could be used for file type verification. In extending our work to include the monitoring of other data types, we would be able to better assess how this may affect the performance of a machine across an entire file system. Since we only focused on one data type (i.e. MS WORD) for testing, we could not obtain a realistic estimate of how the monitoring process affected the file system. A natural extension of our work (after 5.1 is completed) would be to collect statistics of the file system performance over a variable period of time. Within this timeframe, the amount of file system usage would be varied (i.e. both up and down) to get accurate snapshots of its performance at different workload levels. We speculate that the performance of the file system would be directly proportional to the volume of work being done, rather than the monitoring process itself. In addition to removable media, there are a number of other ways that data can be exfiltrated from a computer system. Some of these include the use of email attachments, CD burning software, and even internal LAN transfers. These methods of moving data from one place to another are used frequently. Yet, exfiltration via these routes is rarely considered until it actually occurs. We plan to expand our file system monitoring capability to record the use of processes associated with these methods of data transfer. When one of the monitored processes is invoked, we would take record of the data that is being acted upon and block the process if the distribution rights are violated. As stated in the research results section, a user interface for retrieving log results would be a valuable resource for a forensic investigator. This type of interface would provide an investigator with options to search by date, time, user, transfer success, or device name. Prior knowledge of SQL would not be needed, thus making it very simple to obtain information during the course of a forensic investigation.

6 Conclusion

In this paper, we present a discretionary access control method that focuses on DE via removable devices. Using this approach showed that the unauthorized release of information can be prevented by using access control policies combined with file system monitoring. We also show that the data collected can be used as supplemental forensics evidence. The benefits of the approach include: its simplicity, utilization of built in OS features, and it addresses a domain of DE that is commonly used for legitimate purposes while adding minimal overhead to normal system operations.

References

1. OSF DataLossDB | Data Loss News, Statistics, and Research. Open Security Foundation (January 2, 2009), http://www.datalossdb.org
2. Richard, Clayton. Stopping Spam by Extrusion Detection. CEAS (July 30, 2004). Conference on Email and Anti-Spam (January 2, 2009), http://www.ceas.cc/papers-2004/172.pdf
3. Annarita, V.H. (Giani/Dartmouth College), Cybenko, G.V.(Berk/Dartmouth College): Data exfiltration and covert channels. SPIE Infrastructure Protection and Cyber Security I 6201 (2006) 620103. SPIE (May 10, 2006). SPIE (January 2, 2009), http://www.spie.org
4. Dubrawsky, I.: "Firewall Evolution - Deep Packet Inspection." SecurityFocus (July 29, 2003) (March 16, 2009), http://www.securityfocus.com/infocus/1716
5. Schear/University of California at San Diego, Nabil, Carmelo Kintana/University of California at San Diego, Qing Zhang/University of California at San Diego, and Amin Vahdat/University of California at San Diego. Glavlit: Preventing Exfiltration at Wire Speed. ACM - Hot Topics in Networks (2006). University of California at San Diego (January 2, 2009), http://www.cs.ucsd.edu/~vahdat/papers/glavlithotnet.pdf
6. Castro, S.: Covert Channel and Tunneling over the HTTP protocol Detection. Infosecwriters.com (November 2003) (January 29, 2009), http://www.infosecwriters.com/hhworld/cctde.html
7. Eckstein, K., Jahnke, M.: Data Hiding in Journaling File Systems. Digital Forensic Research Workshop (2005), http://www.dfrws.org (2005) (March 16, 2009), http://www.dfrws.org/2005/proceedings/eckstein_journal.pdf
8. Technical Note TN1150: HFS Plus Volume Format. Apple Developer Connection. 05 Mar (2004), Apple Incorporated (March 16, 2009), http://www.developer.apple.com/technotes/tn/tn1150.html
9. Parker, Don. Windows NTFS Alternate Data Streams. Security Focus (February16, 2005) (March 16, 2009), http://www.securityfocus.com/infocus/1822