

Approaches to Integrated Malware Detection and Avoidance

Amon Ott
Hamburg University
Faculty for Informatics
lott@informatik.uni-hamburg.de

Simone Fischer- Hübner
Stockholm University / KTH
Department of Computer and Systems
Sciences (DSV)
(on leave from Hamburg University,
Faculty for Informatics)
simone@dsv.su.se

Morton Swimmer
IBM T.J. Watson Research
Center / Yorktown Heights, New
York, USA
swimmer@us.ibm.com

Abstract

An implementation of a mandatory virus protection policy is described. This policy is implemented within an implementation of the Generalized Framework for Access Control (GFAC) on the Linux operating system. This approach is similar, but more secure, than that implemented in IBM AntiVirus. The implementation is extended to include socket accesses in order to prevent malicious code entering the system via network connections. Finally, the challenges that are posed by application-level code (e.g. so-called Macro viruses) are discussed.

Keywords

GFAC, Access Control, Linux, Malware, Virus, Socket, Spoofing, Security, Privacy, Virus Scanner, Java, Visual Basic for Applications (VBA), ActiveX

1. Introduction

In our networked society, malicious code (malware) is a serious threat to system integrity, security and privacy. The major threats are viruses, which can import malicious code into local workstations from the Internet, Trojan horses, which can exploit features or weaknesses of Internet services. Macro viruses (viruses written in interpreted code hosted by applications) are a significant risk because they are platform independent, easier to write than 'traditional' file viruses and reside in what one usually considers 'data files', which get exchanged far more frequently than executable files. Besides, the user's privacy is especially endangered by technology that uses downloaded code, such as ActiveX controls or Java. Malicious downloaded programs can scan the end user's hard disk or network through security holes for important information and then smuggle the data to the outside world using the computer's network connection.

Special security mechanisms are needed to deal with malicious code threats, and are also required by legislation: Art. 17 of the EU-Directive on Data Protection, for example, requires the implementation of security measures to protect personal data against accidental or unlawful destruction or accidental loss, alteration, unauthorized disclosure or access.

In [Fischer-Hübner 1997], [Fischer-Hübner/Ott 1998], the implementation of a formal privacy policy according to the Generalized Framework for Access Control (GFAC, see [Abrams et al. 1990]) approach in the Linux system has been presented. In this paper, we will show how we have implemented an on-access virus protection policy within the Generalized Framework for Access Control similar to the methodology incorporated in IBM AntiVirus, except that it is implemented in the kernel and is therefore better protected from manipulation. The model is then extended to include scanning and denying access to data, if appropriate, from a given origin of network connections by controlling UNIX-type

sockets. We implemented both of these previous methods in an experimental Linux installation on top of an implementation of GFAC. Various problems occur when using these techniques on macro viruses, therefore we include a discussion of implementing access control in the application layer.

2. Implementation of a kernel-based on-access scanner using the GFAC approach

In a project at Hamburg University, a formal privacy policy, the Bell LaPadula policy and two supporting policies (Security Information Modification-, Functional Control-Policies) have been implemented according to the GFAC concept in the Linux operating system (see [Fischer-Hübner 1997], [Fischer-Hübner et al. 1998], [Ott 1998]). GFAC is a framework for expressing and integrating multiple policy components. GFAC makes it feasible to configure (and extend) a system with security policies chosen from a set of options provided by a vendor with the confidence that the resulting system's security policies will be properly enforced (see [Abrams et al. 1990], [LaPadula 1995]).

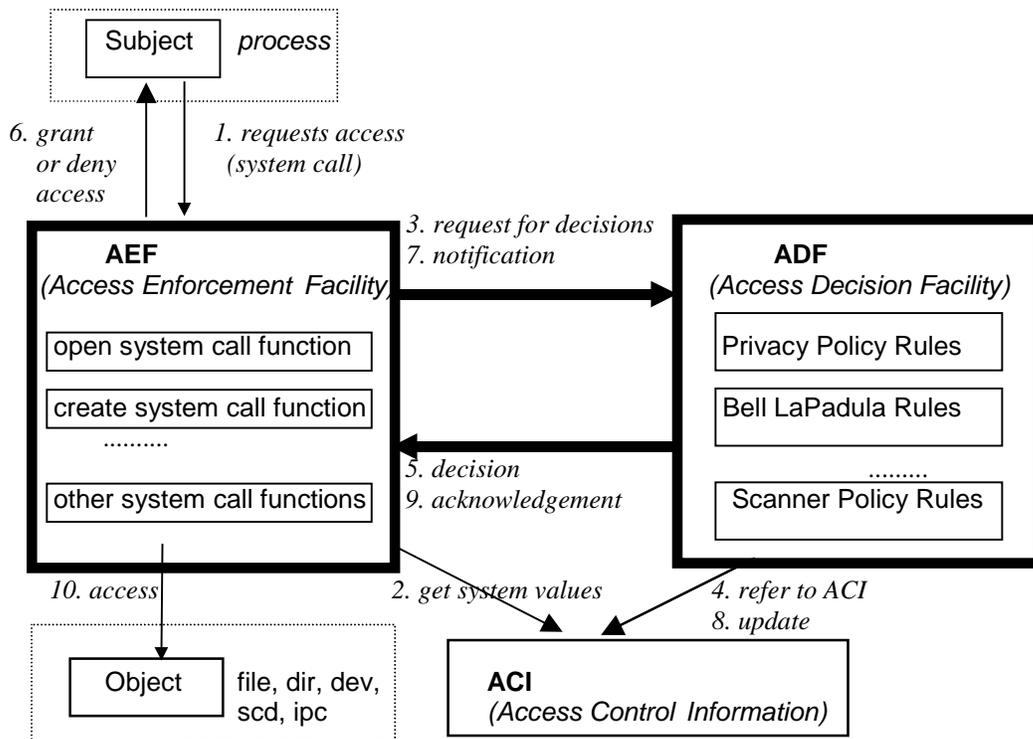


Figure 1: Implementation of the GFAC concept including scanner policy rules

According to the GFAC approach, the Trusted Computing Base (TCB) consists of an access enforcement facility (AEF) and an access decision facility (ADF). ADF enforces the system's mandatory security policies and a metapolicy to decide whether processes' requests satisfy those security policies. AEF uses the ADF-decisions to implement the operations of security-relevant system call functions.

In our Linux GFAC implementation (that we have called RSBAC: "Rule-Set Based Access Control"), the access control system of the system kernel is divided into the AEF and ADF components and the ACI-module which administers Access Control Information (ACI, e.g. security attributes). Figure 1 shows the interactions between the system

components. For each security-relevant system call (if a subject wants to access an object) AEF sends a decision request to ADF. Parameters of the decision request are the request type, describing the desired type of functionality, the identification of the calling process and possibly the id of the target of access (a subject or an object). ADF evaluates its security policies by using the policy rules for the request type and the ACI needed for these rules. It then evaluates its metapolicy, which uses the decisions of the different security policies to finally decide about the process' request. AEF then enforces the decision, by either performing the system call functionality or returning an error to the calling process. In the first case, after successful execution, ADF is notified, so that the attributes can be set accordingly. Finally, control is returned to the process.

2.1. Integration of on-access scanner policy rules component

We have implemented an On-Access Scanner using the GFAC concept by integrating a scanner policy rules component to ADF and adding further access control information to the ACI administration module. Although GFAC was originally designed as a framework for expressing and integrating policy components of formal security models (usually, state machine-type models), we show that it can also be used to integrate heuristic policies. Virus scanners are inherently heuristic, as there is no known method for always determining the presence of a virus in a file. Currently, all known antiviruses are only able to detect previously known viruses reliably. Additional ACI and policy rules needed for the integration and implementation of the scanner policy component, are described in the following sections.

Access control information:

For each file, a new security attribute MS-scan-result is used. Possible values of the scan-result attribute are a version number, "rejected" or "unscanned". The scan-result attribute of a file has the value of a version number of a scanner, if the file was scanned with this scanner version and no infection was found. If the file was scanned and a virus infection was detected, the file attribute is set to "rejected". The file attribute is set to "unscanned," if the file has not been scanned so far, or if the file has been modified by the last access to it. A new attribute MS-trusted is introduced. The attribute MS-trusted is set for the antivirus or backup programs and processes that are executing those programs. Furthermore, the ACI-module administrates a database of virus patterns/signatures, which also has a version number attached to it.

On-access scanner policy rules:

The scanner policy rules are invoked at each execute or open decision request sent to ADF by AEF.

Infections by file viruses are caused by the execution of an infected file, whereas reading an infected document causes infections by macro viruses. Thus, each request from AEF to ADF to execute or to read a file should only be positively decided by ADF, if the file has first been scanned with the latest scanner version and if no infections were found.

Hence, if the access request type is execute, read-open or read-write-open, then the decision rule will be as follows:

If the file has not been scanned so far with the latest scanner version (i.e., the scan-attribute value is "unscanned" or is a version number that is less than the version

number of the virus pattern database), then the file will be scanned for possible infections. In this case, the access decision will only be positive, if no infection is found. The MS-scan-result attribute will then be reset to the current version number if the access decision is positive, or else to “rejected”.

If the MS-scan-result attribute value is equal to the current version number, the access decision is positive.

If the MS-scan-result attribute value is “rejected”, the access decision is only positive, if the current process has the attribute MS-trusted set. Thus, only trusted programs (e.g. antivirus tools) can be used to access “rejected” files.

For the modifying access request types write-open, append-open and truncate, no decision rules are defined. However, for these and for read-write-open, the scan-attribute value is reset to “unscanned”, because it might have been infected or disinfected.

The integration of the scanner policy rules to the access decision facility (ADF) has some impact on the overall system performance. If a file is to be read, the scanner policy rules are only invoked on the read- (or read-write) open request, and then do not have to be invoked for each single read access. Moreover, upon execute-, read-open and read-write open requests, files have only to be scanned, if they were modified or if a new scanner version was installed (that is, if the virus pattern database was extended and has received a new version number). Problematic are complex files containing both data and code, which may be scanned more often than necessary. We will come back to this problem later.

2.2. Integration of socket-level on-access scanner rules

The on-access malware scanner introduced in the last section provides a reliable and tamper-proof protection against known malware in executable files. Increasingly however, malware is brought into the system by a network connection, often executed without ever being saved to a file.

Currently, scanning traffic at the firewall, in order to protect the whole local network against malware, is becoming popular. This way protection for all local hosts relies on knowledge and control of all connections to the local network. Every additional connection must be protected by additional mechanisms, and every proxy must have a reliable scanning mechanism, increasing the workload on the firewall.

Recently, we have taken another approach, where each host protects itself with only one instance of a malware scanner in the system kernel.

Additional scanner policy rules, that are also part of the ADF component of the GFAC-system, control all read accesses to network connection sockets. If malware is detected in the data stream further read access is denied and a new error code "malware-detected" is returned. Depending on the configuration, the connection can be closed by the kernel, or a trusted process can allow the whole stream to be delivered anyway.

Individual protection on every host can reduce the impact of malware that gets injected locally into a network avoiding the firewall. It can be a second level of trust after the firewall. Still, there remains the same problem with application level encryption as firewalls generally have, because only plaintext malware can be detected.

Implementation of a socket-level malware scanner in a Linux kernel

All network communications in a Linux system are based on sockets, which are the only targets in this section. Sockets are general implementations of protocols, like IP, TCP, UDP and ICMP, on ISO-OSI levels 3 and 4. Since data streams are to be checked, UDP and TCP must be controlled. Other protocols, e.g. SPX/IPX, might be addressed later.

When a TCP socket connection has been established, the two processes on both ends can send and receive data in a bidirectional stream. UDP sends data unidirectionally without a connection, making it more difficult to distinguish between different sending processes.

The reading of data is done in blocks of varying sizes via the usual system read calls. The general socket read function calls the read function assigned to the level 4-protocol. The function return value is the number of bytes read in this cycle, or a negative error code. These values are returned to the calling process.

Our scanning mechanism, which is currently being implemented, is invoked by a request sent to ADF for the access type READ and for target IPC (Inter-Process Communication) in the TCP and UDP read functions, giving socket id, length and memory position of the current stream segment, and type of protocol in an attribute parameter.

If a malware was found and results in *"not_granted"*, the original return value, the segment length, is changed into an error code *"malware-detected"*.

After scanning a data segment, the scanner state is stored as a security attribute of IPC-type objects in the ACI-module, including the necessary amount of data read for stepping back in the scanning algorithm, a scan-result flag and the size of the last processed segment.

As explained above, different actions can take place in the access decision code on malware detection, depending on the requesting process. While TCP connections are blocked by the kernel and usually closed by the calling process, actions to be taken for UDP streams are more problematic, because of the uncertainty of the sender's identity.

For TCP, another process ACI attribute *ms-sock-trusted-tcp* is used, which can have three values:

"not_trusted": no way to get blocked data, the stream segment buffer is cleared

"active": error code is returned, but process can get original return value by extra call

"full": process has unlimited access to data

The same security attribute is used for programs and is inherited by every process executing the program. The current UDP solution implements the same mechanism, but uses the attribute *ms-sock-trusted-udp* instead.

Later, as another option, sender tables might be added to block only malicious senders, but these tables would still be susceptible to address spoofing.

The impact of socket-level scanning on network performance has not yet been measured, but will mostly be determined by the scanning algorithm and by the number of cycles to be performed. The basic decision request itself should not be noticeable on

standard Pentium PCs working with 10 Mbit Ethernet, as our tests for other RSBAC components show.

A major limitation of the socket-level malware scanner approach is that it cannot detect application-level encrypted malware. Thus, another approach is to incorporate more knowledge of the application environment into the access control, and we will discuss a few approaches in the next section.

3. Difficulties with application level executable code

Whereas the previous virus prevention methods work well for executable code executed directly by the operating system, application level code doesn't fit as well into this model. While using specific on-access scanner rules within the GFAC implemented under the operating system does protect against application level viruses, but it is inefficient. Scanning raw data received from sockets has the disadvantage of being a harder problem for a scanner that relies on parsing the entire file rather than scanning for byte strings. A problem that both methods share is that the scanner will not find viruses if the code it is scanning has been encrypted.

Let us look at an example. Microsoft Word for Windows documents of versions 7 and above use a dialect of BASIC called Visual Basic for Applications (VBA). All objects in this type of file are stored in a directory-like structure, including the text and the executable code (see figure 2.) In our example the actual text is stored in the object called

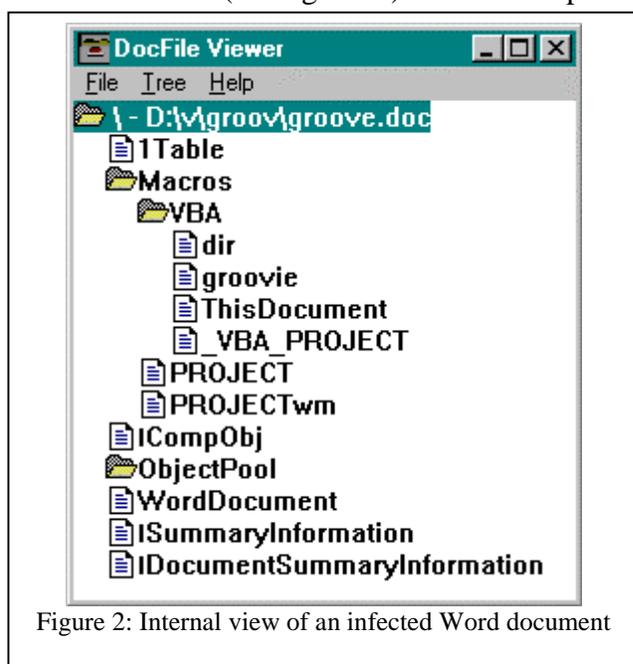


Figure 2: Internal view of an infected Word document

“WordDocument” and the virus itself is contained in the \Macros\VBA\ ‘directory’, specifically in the “groovie” object. The objects contain other data for internal use by Windows and Word.

If the user writes to the text part of the file s/he only writes to the WordDocument object and there is no reason to rescan the file at the next open, as the code in the \Macros\VBA\ directory has not changed. However, if the file is administered by a GFAC system that treats files as the only objects it controls, any write access to the text of the file will result in an unneeded rescan of the file at the next open.

If we try to scan this file at the socket level, we will not have the chance to parse the file into these separate objects. Experience with IBM AntiVirus has shown that it is less reliable to scan the file as a byte stream, instead of parsing the file and scanning only the relevant objects in the \Macros\VBA directory.

The encryption problem is similar. If, for instance, the ‘groovie’ object is encrypted, the scanner will not be able to scan it. However, the application is able to decrypt it and therefore a solution to this and the other problems may be to implement security at the application layer.

3.1. Place access control at the application level

One possible, and often implemented, solution is to place additional access control monitors in the application layer. This is done in Web browsers hosting Java as a scripting language and Word for Windows hosting VBA. The advantages are that the parsing is taken care of by the application, which also is aware of its own object model. The access control monitors can therefore be applied in a more optimal manner.

Immediately, one problem is evident: the security mechanisms are vulnerable to attack from within the application layer. The monitors must therefore be protected by the security kernel against modifying attacks. The Security Information Modification (SIM) – Policy, enforced by our GFAC-system, can be used to protect the monitors against unauthorized modifications. The SIM policy [LaPadula 1995] uses the ACI attribute *data-type* with two possible values, “*si*” and “*NIL*”, as well as *system roles* for the users. When a process requests access to data of type *si* in a mode that enables modification of the data, the SIM policy allows the access only if the system role of the owner of the process is “*security officer*”. Thus, monitors in the application environment should have the data-type attribute value *si*, and are thereby protected by the security kernel against manipulations.

As an aside we would like to mention the use of ‘certificates’ in security of applications. It has often been proposed that certificates should be issued as authentication for an applet, such as a Java applet, an ActiveX object or a VBA macro, should be used to assure the security of these applets. However, certificates do not solve anything by themselves. They raise the confidence level in the trustworthiness of the applet by identifying beyond reasonable doubt the origin of the applet. It doesn’t prevent the applet from containing malicious code implemented by the signer.

3.2. Require applications to use central access control in accessing files

A more secure design would require all applications to register all objects they define with the operating system and use operating system facilities to grant or deny access to thus designated objects. This would require the interpreter to be separate from the calling program and that the program's object model is exposed to the operating system. This avoids placing the access control monitors at the application level and keeps the security system centralized.

Word for Windows 7.0 could be redesigned to comply with such a model. Word itself exposes many of its objects and uses another set of objects belonging to VBA to execute code contained in its documents (which are accessed via another set of objects belonging to OLE 2.0.) Unfortunately for us, after access has been granted to the application and the document, no further access control can be applied to any of the other objects. If, however, the operating system were to control access to all objects defined by all applications, a mandatory access control policy could be implemented to prevent violations of security and privacy.

The same can be said about Java. Java implements far more stringent security measures than VBA has as it was designed for use over the Internet, where the intent of the service provider is not known. However, the enforcement of a security policy is left to the application in which runs Java, typically the Web browser. The type of security policy the web browser defines may not be in compatible with the user's or company's policy, e.g. it may prevent legitimate accesses of Java applets to the local file system in internal applications, but allow a Java applet download software to the local drive in non-

compliance with the company policy. Centralizing the access control would allow more effect security.

Further extensions to GFAC

The challenge to security is that each of the objects inside this file, may possess a different security requirement, but there is no way of enforcing the security of these objects without the help of the application.

As an example let us take a report that Fred is writing. The main document contains the introduction and some general information about his company. His analyst has compiled another document, which gets embedded into the main document. This document, however, contains sensitive payroll information. He might want to allow other employees to read the main document, but wants to prevent access to the sensitive data.

The problem, from a security point of view, is that the access control in the file system is not extended to the documents contained inside the file. The access attributes of the file are inherited by the documents, which is not necessarily appropriate. The user needs access to the entire file to access even only one of the documents contained within it. In particular, applications frequently store executable code in a document within the file, which in practice is executed outside the operating system's direct control.

A similar problem applies to many HTML documents. The document contains inlined references to other documents, as well as images and executable code. This code is often executed in a virtual machine, e.g., Java, or on the native machine, e.g., ActiveX.

Further problems arise when we try to enforce a security strict policy. The objects defined are often different in nature and there is no easy way to correlate object access. For example, in VBA, a program has access to the integrated editor/compiler and therefore program source code. Although any ordinary program has access to the file as a whole, it has no easy access path to the code, as it requires a complex file format parser requiring unpublished documentation. It would be necessary to control access to program code to prevent VBA viruses from becoming viable while allowing ordinary operations on documents.

A subject of future research will be to define a model for operating system – application interaction that will allow a strict policy to be enforced without losing the functionality users have come to expect.

4. Final remarks

In this paper we have shown how we have implemented an on-access virus protection policy within the Generalized Framework for Access control. This policy is similar to IBM AntiVirus's method of protecting the user from viruses, except that it is implemented in the kernel and is therefore better protected from manipulation.

We also extended the model to include scanning and denying access to data from an origin of network connections by controlling UNIX-type sockets. This is a useful extension to virus protection at the firewall, as has become popular in recent years. It is superior in some ways to the firewall approach in that it limits the spread of viruses within a corporation instead of just from the Internet to the Intranet. It is an important compliment to the on-access scanning policy that protects against viruses that are not saved to disk, but run directly from memory after loading.

We implemented both of these previous methods in an experimental Linux installation on top of an implementation of GFAC.

We also discussed the implementation of access control monitors in the application layer as well as forcing applications to export and rescind control of their objects to the operating system so that a policy can be enforced over all objects in the system and not just file system objects. This represents work in progress, but is necessary to reconcile the increasingly dangerous functionality in applications with security and privacy requirements demanded by law and corporate policy.

In addition to the three approaches to integrated malware detection and prevention, Intrusion Detection Expert System technology can be used to detect malware in real-time. One first concept for an expert system for virus detection (VIDES) was presented in [Brunnstein/Fischer-Hübner/Swimmer 1991]. This system used a priori rules based on general virus behavior to detect and prevent virus behavior.

Finally, our model relies ultimately on a virus scanner to serve as an 'oracle' to tell us if a given object contains a known virus. The quality and performance scanner becomes critical in this model. A System like the Computer Immune System [Kephart/Sorkin/Swimmer/White 1997][Kephart et al. 1992], currently being developed at IBM Research and deployed by Symantec is designed to react to new viruses far quicker than current systems can as they rely on limited human resources for analysis. A lot of research has gone into making automatic analysis of the highest standards and the system also includes parts of VIDES.

5. References:

- [Abrams et al. 1990] Marshall Abrams, K.Eggers, L.LaPadula, I.Olson, "A Generalized Framework for Access Control: An Informal Description", *Proceedings of the 13th National Computer Security Conference*, Washington, October 1990.
- [Brunnstein/Fischer-Hübner/Swimmer 1991] Klaus Brunnstein, Simone Fischer-Hübner, Morton Swimmer, "Concepts of an Expert System for Virus Detection", in: D.Lindsay, W.L.Price (Eds.): *Information Security – Proceedings of the IFIP/Sec '91-Conference*, Brighton/UK, May 1991, North Holland.
- [Fischer-Hübner 1997] Simone Fischer-Hübner, "A Formal Task-based Privacy Model and its Implementation: An updated Report", *Proceedings of the Second Nordic Workshop on Secure Computer Systems NORDSEC'97*, Helsinki, November 6-7, 1997.
- [Fischer-Hübner/Ott 1998] Simone Fischer-Hübner, Amon Ott, "From a Formal Privacy Model to its Implementation", *Proceedings of the 21st National Information System Security Conference*, Arlington, October 1998.
- [LaPadula 1995] Leonard LaPadula, "Rule-Set Modelling of Trusted Computer System", Essay 9 in: M.Abrams, S.Jajodia, H. Podell, "Information Security - An integrated Collection of Essays", IEEE Computer Society Press, 1995.
- [Ott 1998] Amon Ott, "Rule Set Based Access Control in Linux", <http://agn-www.informatik.uni-hamburg.de/people/1ott/rsbac/eng.htm>
- [Kephart/Sorkin/Swimmer/White 1997] Jeffrey O. Kephart, Gregory B. Sorkin, Morton Swimmer, Steve R. White, "Blueprint for a Computer Immune System", *Proceedings of the Virus Bulletin Conference 1997*, <http://www.av.ibm.com/InsideTheLab/Bookshelf/ScientificPapers/Kephart/VB97/>
- [Kephart 1994], Jeffrey O. Kephart, "A Biologically Inspired Immune System for Computers", *Artificial Life IV*, *Proceedings of the Forth International Workshop on the Synthesis and Simulation of Living Systems*, Rodney A. Brools and Pattie Maes, eds, Mass. 1994, <http://www.av.ibm.com/InsideTheLab/Bookshelf/ScientificPapers/Kephart/ALIFE4/alife4.distrib.html>