This is a preprint of the paper presented at the 1st International Conference on Information Technology, May 2008 in Gdansk, Poland and published in the IEEE Xplore Digital Library
DOI: 10.1109/INFTECH.2008.4621669

Modern Approaches to File System Integrity Checking

Jerzy Kaczmarek, Michal R. Wrobel
Gdansk University of Technology
Faculty of Electronics, Telecommunications and Informatics
jkacz@eti.pg.gda.pl, wrobel@eti.pg.gda.pl

Abstract. *One of the means to detect intruder's activity in system is to trace all unauthorized changes in file system. Programs which fulfill this functionality are called file integrity checkers. This paper concerns modern approach to file system integrity checking. It reviews architecture of popular systems that are widely used in production environment as well as scientific projects, which not only detect intruders but also take actions to stop their activity. The concept and architecture of ICAR System (Integrity Checking And Restoring System), which we are developing, will be presented. The ICAR System not only covers functionality of integrity checkers but also automatically restores files, which were modified by the intruder. ICAR has been designed as kernel module of the operating system and it uses read-only devices to store data. The article can prove useful to the operating systems users, that are interested in securing their data and system configuration.*

1. Introduction

Nowadays more and more computer systems are connected with the Internet network. Many intrusion attacks on computer systems have been observed. Therefore securing operating systems is very important. There are a lot of systems designed to protect computers against unauthorized access. However, because of the vulnerability of programs and human errors, there are no methods, that will completely secure computer systems. One of the means for detecting intruder's activity in the system is tracing all unauthorized changes in the file-system. Programs which fulfill this function are called file integrity checkers.

File integrity checkers are computer programs, that secure operating systems by monitoring changes in critical files [1]. Their aim is to protect system data as well as files of important applications. File checkers use the pattern of each protected file to determine its modification. Those patters should be generated right after operating system installation or upgrade.

The efficiency of the file integrity checkers depends on the checksum generation method. It is significant that in case of changing even one bit in the file, the method would generate a different pattern. It must not be possible to create, in a reasonable computational time, a file content based on a known pattern. Method shouldn't be also time-consuming even for large files.

Most used methods of pattern generation are cryptographic hash functions. They are a mathematical function which take strings (messages) of any length as an input and return so called hash values. An important property of the hash function is that the output is a fixed-size string [2]. The hash function is not a one-to-one function – it takes any length strings and returns fixed-size output, mostly shorter then the input. As a consequence for two or more distinct stings, the function can produce identical hash. Such situation is called a hash collision. For file integrity checkers it is important that it is not possible, in reasonable time, to find the message that has a specific hash value. Such situation is called a preimage attack.

In file integrity checkers the hash function takes file contents as an input, and fixed-size hash is called digital fingerprint. File integrity checkers most-commonly use the *MD5* (*Message-Digest algorithm 5*) or *SHA1* (*Secure Hash Algorithm*) cryptographic hash function. In 2004 the preimage attack on *MD5* and in 2005 on *SHA-1* was presented, they both allow to find collisions efficiently [3,4]. Therefore it is recommended now to use stronger cryptographic hash function such as *SHA-224*, *SHA-256*, *SHA-384* or *SHA-512*. However producing hash values with those more complex functions, takes more time especially on older computers.

File integrity checkers can be run like all other computer programs in user space. Such programs are usually executed periodically and allow detection of any changes performed by the intruders in a file-system. However they cannot prevent attack – file modifications are not detected automatically. To provide constant files monitoring it is necessary to implement the mechanism in the kernel space. This

allows to perform more complex protection procedures such as blocking access to attacked files.

2. User space file integrity checkers

User-space file integrity checkers are run as ordinary applications. Usually file checking is done periodically. The program is launched by specific scheduling services such as *Cron* in Linux or *Task Scheduler* in Windows. File integrity checking can also by executed manually by the system administrator, when there is a suspicion of break-in into the system.

The file integrity checker uses digital fingerprints to determine whether a file was changed or not. Before deploying such security system it is required to create a database with hash values of every protected file. It is important to secure this database. If the intruder managed to modify the database, the whole security system would become unreliable. Most of the file checkers store database on read-only devices or use cryptographic functions to protect it.

During run-time the file checker produces digital fingerprint of protected files and compares them with their patterns stored in database. When the fingerprint differs from the pattern an alarm procedure is initialized. What actions are carried out depends on the capabilities and configuration of the particular system. Usually the system administrator is alerted, the file is moved to quarantine or even the whole operating system is halted. Of course the system can perform more than one of these operations [5].

User space programs are only able to verify if the protected files had been modified between the checking operations. In that period of time the intruder can replace system files with back-doors or trojans. Such prepared programs could be executed by an authorized system user, which would result in loosing crucial data, such as confidential information, password etc. User space file integrity checkers do not protect the file system, they only allow to detect if and which files had been changed.

The first widely used file integrity checker was *Tripwire*, which was created in 1992 on Purdue University. Since that it has been regarded as the exemplary integrity checker and many other programs are based on its ideas.

By using *Tripwire* the administrator can monitor changes in the file system, such as creating, removing and modifying files. It can monitor not only contents of the file but also meta data like file attributes, hard-links or permissions [6].

During the file checking process *Tripwire* produces hash strings for all protected files based on selection-mask which are set in configuration files. Then these values are compared with digital fingerprints stored in secure database. As a result the administrator receives a list with file names that had been added, deleted or modified.

At the beginning *Tripwire* was distributed for free as an academic project. However in 1996 authors started to sell a commercial version of the program. Since then a few open source projects based on *Tripwire* ideas, were developed. The most known programs are *AFICK*, *YAFIC*, *integrit* and *AIDE*. All of them work in a similar way, the only difference between them is the performance of issues, portability and some technical details.

Another variant of the user space integrity checkers are programs that monitor file systems of all computers in local network. They are called non-resident checkers. The most popular programs of this kind are *Radmind* [7], *Osiris* and *SAMHAIN* [8].

3. Kernel space file protections systems

User space integrity checkers perform only periodical files control. Therefore they do not protect the operating system from running files prepared by the intruder. More advanced protection systems run at kernel level. It allows to check files in real-time, i.e. during reading or executing operations. Such kernel space systems can also take more sophisticated actions like denying access to compromised files.

These mechanisms are placed in one of the lowest levels of the operating system. It is said that every security component is not more trustworthy than the component upon which it depends [9]. Hence integration file integrity checker with operating system kernel is much more reliable then implementing the same mechanism in the user space.

The protection systems implemented at kernel level are not ordinary programs. They are usually kernel extensions or modifications. Such implementation allows to verify the file integrity on every access request. After receiving request to open or execute a file the operating system starts the verification procedure. It produces and compares the file's digital fingerprint with its pattern stored in the database. If the file was verified positively, access is granted. Otherwise, when the file was modified or replaced, the administrator is alerted and other protection steps can be taken. Because kernel space systems can block access to compromised files in real time they are sometimes called files protections systems to differ them from user space file integrity checkers, which can only find modified files.

Nowadays there is no mature implementation of the file integrity checker in kernel space. However two products should be distinguished. *SOFFIC* was developed on FRGS University (*Universidade Federal*

do Rio Grande do Sul) in Porto Alegre, Brazil. Second was created on Stony Brook University in USA and it is called *I³FS*.

In the *SOFFIC* project (*Secure On-the-Fly File Integrity Checker*) file integrity checking is included in operating system kernel by modifying kernel source files. It is distributed as kernel patch files. The *SOFFIC* intercepts two POSIX system calls that are responsible for reading and executing files. The list of protected files is stored on the main file system. This kernel space checker uses public-key cryptography to secure database and cache memory to store information about already verified files [10].

The second project, *I³FS* (*An In-Kernel Integrity Checker and Intrusion Detection File System*), is more advanced. The system uses the idea of a stackable file system to integrate files monitoring with the operating system. It allows to overlay a few file systems and as a result to form a single file system. *I³FS* uses *FiST* (*Stackable File System Language and Templates*) language to implement file checking over every file system which is used in the operating system. As well as *SOFFIC*, *I³FS* also control every read and execute operation. It uses modified *Berkeley Database* to keep information about protected files, their fingerprints and some statistics. It has an advanced cache system, which stores not only results of previous operations, but also the hashes of memory pages. That solution allows to keep information about all files smaller then 5MB [11].

Within these two systems *I³FS* seems to by more mature. Its architecture, based on stackable layers is more flexible and easier to implement than solutions based on system calls interception.

4. File integrity checking and restoring system

The file integrity checkers, which were described in the previous chapters, detect unauthorized file modifications. Modern systems not only alert system administrations, but also can deny access to compromised applications. However there are no methods, that guarantee continuous and secure work on the system which has been successfully attacked by the intruder.

In the course of the research conducted by the authors a new file protection model was created. It allows not only to monitor file system but also to automatically restore files which where modified by an unauthorized user. This system was called *ICAR* (*Integrity Checking and Restoring*). In our approach the file integrity checking idea is extended through a mechanism that automatically restores modified files from their hard copy stored on read-only devices.

In our model read-only devices, such as CD-ROM or flash memories, are used to store key components of the *ICAR* system, such as copies of protected files and the file's digital fingerprints database. Because write access is blocked on the hardware level it is not possible to modify this data by an intruder. As a proof-of-concept for the *ICAR* the Linux LiveCD distribution called *cdlinux.pl*. has been created. The CD-ROM disc contains system files, which are used to boot operating system. These files are also used as safe copies of files stored in RAM memory, which could be restored whenever *ICAR* detects unauthorized file modifications.

Our project of the *ICAR* system consists of a kernel module, digital fingerprints database and administration utilities. The kernel module is responsible for monitoring and restoring protected files. Before deploying *ICAR* it is essential to produce digital fingerprints for all protected files. The database that contains information about files with their hashes has to be stored on read-only device. Additional utilities allow to create database and read-only image of operating system files.

The *ICAR's* kernel module is the main component of our file checking and restoring security system. Using *Linux Security Module* (*LSM*) to monitor every attempt to open or execute a file the *ICAR* checks if the file was modified. In case of an unauthorized modification the module not only alerts the system administration and blocks access, but also restores the file from a copy stored on read-only device.

The flowchart for *ICAR's* file checking and restoring is shown on figure 1.

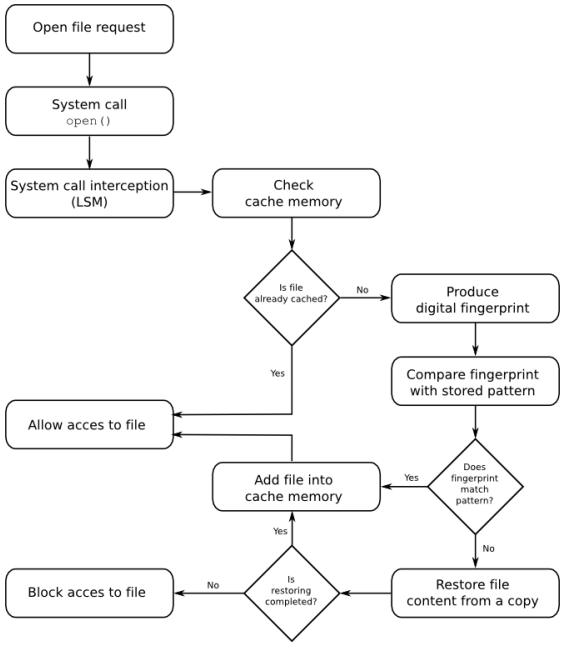

Figure 1. ICAR's file checking and restoring

On request to open a file, Linux kernel executes our security module. At the beginning the module checks if the file was included to protection. If the database contains information about this file its digital fingerprint is obtained. Then the module checks in the cache memory if the file was recently opened. If it is found, the process of computing hash is omitted and file access is granted. Otherwise hash value is produced and it is compared with the value taken from secure database. If the verification is positive the file name is added to the cache memory.

When the file name is not present in the cache memory and the computed hash is not equal to the hash from the database the rescue procedure is initialized. Aside from writing system logs and alerting the administrator the *ICAR* system tries to restore the file from the read-only copy. If the file copy is found on a backup device, its content is moved to the root file system. The compromised file can also be stored in quarantine directory for further investigation.

Finally the *ICAR* module returns flow control to Linux kernel. If the checked file was not modified or was successfully restored from the copy system it allows to open or execute file. Otherwise access is denied.

The results of performed tests on prototype system indicated efficiency of that protection mechanism. Including the *ICAR* system in the operating system entails some computing overhead. However users should not notice a slowdown on modern computers.

5. Conclusion

Protection of operating systems is nowadays one of the most important aspects of computer science. The number of intruders, trojans, backdoors, botnets is significant. They often cause a lot of problems including serious financial loss. There are many researches that are trying to solve that problem. One of the most efficient, is to deny unauthorized users to modify files. There are a few programs that allow file integrity checking, from simple methods working in user-space to advanced solutions integrated with operating system kernel. Our proposal, the *ICAR* system, extend the file integrity checking idea through automatic file restoring from read-only devices.

It is obvious, that the data stored on read-only device cannot be modified in any way. Some modern researches assume that putting a whole operating system in read-only devices, such as EPROM memory, guarantee its security. There are first new projects which allow installing Linux operating system in BIOS. The most advanced are *coreboot* and *splashtop*. In our opinion this is the way that modern operating systems will follow.

6. References

[1] Bace R., Mell P., *Intrusion Detection Systems*, NIST Special Publications SP 800-31, 2001.

[2] Cormen T. H., Leiserson C. E., Rivest R. L., *Wprowadzenie do algorytmów*, WNT 2001, ISBN 83-204-2556-5

[3] X. Wang, H. Yu: *How to break MD5 and Other Hash Functions*, Advances in Cryptology – EURO CRYPT2005, volume 3494 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 2005.

[4] X. Wang, Y. L. Yin, and H. Yu. *Finding collisions in the full SHA-1*. In Advances in Cryptology – EURO CRYPT 2005, volume 3621 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 2005.

[5] Motara,YM and Irwin,BVW. *File Integrity Checkers: State of the Art and Best Practices*. Information Security South Africa (ISSA). Sandton Conference Centre, Johannesburg. July 2005.

[6] Kim G., Spafford E., *Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection*, Purdue Technical Report CSD-TR-94-012, February 1994.

[7] Craig W., McNeal P.,. Radmind: *The Integration of Filesystem Integrity Checking with File System Management*. In Proceedings of the 17th USENIX Large Installation System Administration Conference (LISA 2003) , October 2003.

[8] Chuvakin A., *Ups and Downs of UNIX/Linux Host-Based Security Solutions*, login: The Magazine of USENIX and SAGE, 28(2), April 2003.

[9] Hans Hedbom, Stefan Lindskog, and Erland Jonsson. *Risks and dangers of security extensions*. In Proceedings of Security and Control of IT in Society-II (IFIP SCITS-II), pages 231Â–248, Bratislava, Slovakia, June 15-16 2001.

[10] Serafim V., Weber R., *The SOFFIC Project*, http://www.inf.ufrgs.br/~gseg/projetos/the_soffic_project.pdf

[11] Patil S., Kashyap A., Sivathanu G., Zadok E., I3fs: An In-Kernel Integrity Checker and Intrusion Detection File System, Proceedings of the 18th USENIX Large Installation System Administration Conference (LISA 2004)