File Integrity Monitor Scheduling Based on File Security Level Classification

Conference Paper *in* Communications in Computer and Information Science · June 2011

DOI: 10.1007/978-3-642-22191-0_16 · Source: DBLP

CITATIONS
3

READS
1,813

4 authors:

Zul Hilmi Abdullah
INTI International University
31 PUBLICATIONS **59** CITATIONS
SEE PROFILE

Nur Izura Udzir
Universiti Putra Malaysia
222 PUBLICATIONS **2,079** CITATIONS
SEE PROFILE

Ramlan Mahmod
Universiti Putra Malaysia
209 PUBLICATIONS **1,502** CITATIONS
SEE PROFILE

Khairulmizam Samsudin
Universiti Putra Malaysia
128 PUBLICATIONS **1,391** CITATIONS
SEE PROFILE

File Integrity Monitor Scheduling Based on File Security Level Classification

Zul Hilmi Abdullah[1], Nur Izura Udzir[1],
Ramlan Mahmod[1], and Khairulmizam Samsudin[2]

[1] Faculty of Computer Science and Information Technology,
Universiti Putra Malaysia
43400 Serdang, Selangor
[2] Faculty of Engineering,
Universiti Putra Malaysia
43400 Serdang, Selangor

Abstract. Integrity of operating system components must be carefully handled in order to optimize the system security. Attackers always attempt to alter or modify these related components to achieve their goals. System files are common targets by the attackers. File integrity monitoring tools are widely used to detect any malicious modification to these critical files. Two methods, off-line and on-line file integrity monitoring have their own disadvantages. This paper proposes an enhancement to the scheduling algorithm of the current file integrity monitoring approach by combining the off-line and on-line monitoring approach with dynamic inspection scheduling by performing file classification technique. Files are divided based on their security level group and integrity monitoring schedule is defined based on related groups. The initial testing result shows that our system is effective in on-line detection of file modification.

Keywords: Operating System Security, Files Integrity, Monitoring Schedule, File Security Classification, Malicious Modification, HIDS.

1 Introduction

File integrity monitoring (FIM) is one of the security components that can be implemented in host environment. As a part of host based intrusion detection (HIDS) components, FIM should play a big role in detecting any malicious modification either from authorized or unauthorized users on their contents, access control, privileges, group and other properties. The main goal of related integrity checking or monitoring tools is to notify system administrators if any changed, deleted or added files detected [8]. File integrity checkers or monitors measure the current checksum or hash values of the monitored files with their original value.

In general, FIM can be divided into two categories, off-line and on-line monitoring scheme [7]. File system monitoring tools were originally used on their own before becoming a part of the intrusion detection system (IDS) when it is

J.M. Zain et al. (Eds.): ICSECS 2011, Part II, CCIS 180, pp. 177–189, 2011.

integrated with other components such as system logs monitoring, rootkits detection, and registry monitoring. System files as a core of the operating systems, contains information of the users, application, system configuration and authorization as well as program execution files [8]. Malicious modification of system file may cause disruption of the services or worse if it is used as a tool to attack other systems.

Recent solution of file integrity monitoring focusing on the on-line or real-time checking to enhance the capabilities of malicious modification detection. However, performance downgrade is a big issue in real time checking making it impractical for real world deployment. On the other side, higher cost of investment is required to deploy a new technology of integrity verification for the system such as hardware based protection mechanism using the Trusted Platform Module (TPM) which not only require TPM chips embedded on the computer hardware but also require an additional software to make it efficient.

The main target of the HIDS is to protect the operating system environment from intruders and unintended alteration or modification by authorized users. As one of the critical part in the operating system environment, the integrity of the system files must be put as high priority. However, to monitor all those system files in real-time is very difficult task and very costly especially for multi host and operating systems environment.

In this paper, we propose a software based file integrity monitoring by dynamically checking related files based on their sensitivity or security requirement. Sensitive files refer to the files which, if missing or improperly modified can cause unintended result to the system services and operation [23]. Classification of the sensitive and less sensitive files is used to determine the scheduling of the integrity monitoring of those files.

The rest of the paper is organized as follows: Section 2 discusses related works and compares our proposed techniques with these works. In Section 3, we describe our proposed system focusing on file security classification algorithm and FIM scheduling and how it differs with previous FIM. In Section 4, we quantify the initial implementation result of our algorithm in detecting file modification. This paper ended with discussion and conclusion in Section 5.

2 Related Work

In operating system environment, every component such as instruction, device drivers and other data is saved in files. There are huge number of files contained in modern operating system environment. Most of the time, files become a main target by the attackers to compromised the operating systems. The attack can be performed by modifying or altering the existence files, deletion, addition, and hide the related files. Many techniques can be implemented by the attackers to attack the files in the operating system environment and make file protection become a vital task. Implementation of FIM and other related system security tools is needed for that purpose.

As part of the HIDS functions, file integrity monitoring can be classified as off-line and on-line integrity monitoring. In the next section we discuss the off-line and on-line FIM followed by the multi platform FIM.

2.1 Off-Line File Integrity Monitoring

Tripwire [8] is a well known file integrity monitoring tool that motivates other researchers to develop more powerful FIM tools. Tripwire works based on four process, *init*, *check*, *update* and *test*. Comparing the current hash values of the files with the baseline values are the main principle of the FIM tools like Tripwire. However, relying on the baseline database require more maintenance cost due to more frequent system updates or patches [15]. In addition, off-line FIM needs to be scheduled in order to check the integrity of related files and most of the time can cause delay in detection of the modification. Samhain [19], AIDE [16], and Osiris [20] use the same approach too, so they also inherit almost the same issues as Tripwire.

Inspection frequency and the modification detection effectiveness is the main issue in the off-line FIM. In order to maintain the effectiveness of the FIM, high frequency inspection is needed at the cost of system performance, and vice versa. We overcome this issue by proposing a dynamic inspection scheduling by classifying related files to certain groups and the inspection frequency will vary between the group of files. Thus, from that approach, FIM can maintain its effectiveness with a more acceptable performance overhead to the system.

2.2 On-Line File Integrity Monitoring

On-line FIM is proposed to overcome the delay detection in off-line FIM approach by monitoring the security event involving system files in real-time. However, in order to work in real-time, it requires access of low level (kernel) activities which require kernel modification. When kernel modification is involved, the solution is kernel and platform-dependent, and therefore incompatible with other kernels and platforms.

As example, I3FS [12] proposed a real-time checking mechanism using system call interception and working in the kernel mode. However this work also requires some modification in protected machine's kernel. In addition, whole checksum monitoring in real time affected more performance degradation. I3FS offers a policy setup and update for customizing the frequency of integrity check. However it needs the system administrator to manually set up and update the file policy.

There are various on-line FIM and other security tools using the virtual machine introspection (VMI) technique to monitor and analyze a virtual machine state from the hypervisor level [13]. VMI was first introduced in Livewire [4] and then applied by the other tools like intrusion detection in HyperSpector [10] and malware analysis in Ether [3].

On the other side, virtualization based file integrity tools (FIT) has been proposed by XenFIT [15] to overcome the privileged issue on the previous user mode

FIT. XenFIT works by intercepting system call in monitored virtual machine (MVM) and sent to the privileged virtual machine (PVM). However, XenFIT requires a hardware virtualization support and only can fit with the Xen virtual machine, not other virtualization software. Another Xen based FIT is XenRIM [14] which does not require a baseline database. NOPFIT [9] also utilized the virtualization technology for their FIT using undefined opcode exception as a new debugging technique. However, all those real-time FIT only works on the Linux based OS.

Another on-line FIM, VRFPS uses *blktap* library in Xen for their real time file protection tool [22]. This tool is also platform-dependent which only can be implemented in a Xen hypervisor. An interesting part in this tool is their file categorization approach to define which file requires protection and vice versa. We try to enhance their idea by doing the file classification to determine the scheduling process of file monitoring. VRFPS work on Linux environment in real time implementation but we implement our algorithm in Windows environment by combining on-line and off-line integrity monitoring. Combining the on-line and off-line integrity monitoring is to maintain the effectiveness of the FIM and to reduce the performance overhead.

2.3 Multi Platform File Integrity Monitoring

Developments in information technology and telecommunications led to higher demand for on-line services in various fields of work. Those services requires related servers on various platforms to be securely managed to ensure their trustworthiness to their clients. Distributed and ubiquitous environment require simple tools that can manage security for multi platform servers including the file integrity checking. There are a number of HIDS proposed to cater this need.

Centralized management of the file integrity monitoring is the main concern of those tools, and we take it as the fundamental features for our system and we focus more on the checking scheduling concern on the multi platform host. As other security tools have also implemented centralized management of their tools, such as anti-malware [18] and firewalls [2], FIM as part of HIDS also needs that kind of approaches to ensure the ease of administration and maintenance. We hope our classification algorithm and scheduling technique can also be applied to the other related systems.

Another issue to the FIM like Tripwire is the implementation on the monitored system which can be easily compromised if the attackers gain the administrator privilege. Wurster et al. [21] proposed a framework to avoid root abuse of file system privileges by restricts the system control during the installing and removing the application. Restricting the control is to avoid the unintended modification to the other files that not related to the installed or removed application. Samhain [19], and OSSEC [1] comes with centralized management of the FIT component in their host based intrusion detection system which allow multiple monitored systems to be managed more effectively. Monitoring the integrity of files and registry keys by scanning the system periodically is a common practice of the OSSEC. However, the challenge is to ensure the modification of related files can

```
<syscheck>

  <!-- Real-time checking setting -->
  <directoriesrealtime="yes" check_all="yes"> /WINDOWS/system32
  </directories>

  <!-- Frequency of Offline checking setting -->
  <frequency>79200</frequency>

  <!-- Directories to check -->
  <directories check_all="yes">/WINDOWS</directories>

</syscheck>
```

Fig. 1. Example of system integrity check configuration

be detected as soon as the event occurs as fast detection can be vital to prevent further damage.

OSSEC has features to customize rules and frequency of file integrity checking as shown in Figure 1. However it needs manual intervention by the system administrator. This practice becomes impractical in distributed and multi platform environment as well as cloud computing due to the large number of servers that should be managed. Therefore we try to implement multi platform FIM on the virtualized environment by customizing the scanning schedule with our techniques. Allowing other functions work as normal, we focus the file integrity monitoring features to enhance the inspection capabilities by scheduling it based on related files security requirements on related monitored virtual machines.

3 Proposed System

We found that most of the on-line and off-line FIM offer a policy setting features for the system administrator to update their monitoring setting based on the current requirement. However it can be a daunting task to the system administrator to define the appropriate security level for their system files especially those involving large data center. Therefore, a proper and automated security level classification of the file, especially system files, is required to fulfill this needs.

In this paper, we propose a new checking scheduling technique that dynamically update the file integrity monitoring schedule based on the current system requirement. This can be achieved by collecting information of related files such as their read/write frequency, owners, group, access control and other related attributes that can weight their security level. For initial phase, we only focus on the files owner and permission in our security classification.

Inspired by various services offered by modern operating systems, and multi services environments such as email services, web services, internet banking and others, the criticality of the integrity protection of those system is very crucial.Whether they run on a specific physical machine or in virtual environment, the integrity of their operating system files must be put in high priority to ensure the user's trust on their services.

Centralized security monitoring is required to ensure the attack detection is still effective even though the monitored host has already been compromised. Windows comes with their own security tools such as Windows File Protection (WPC), Windows Resource Protection (WRP) and many more. However most of the tools rely on the privileged access of the administrator. If an attacker gains the administrator privileges, all modifications to the system files or other resources will look like a legal operation. So here where the centralize security monitoring is needed, when the critical resources are modified, the security administrator will be alerted although it is modified by local host administrator.

Identifying the most critical file that are often targeted by attackers is a challenging task due to the various techniques that can be used to compromise the systems. Based on the observation that specific attack techniques can be implemented to specific types of operating system services, we try to enhance the file integrity monitoring schedule by looking at the file security level for the specific host. It may vary from the other host and it can result dissimilarity type of scheduling but it is more accurate and resource-friendly since it fits on the specific needs.

3.1 System Architecture

The architecture of our proposed system is shown in Figure 2. The shaded area depicts the components that we have implemented. We develop our model based on the multi platform HIDS.

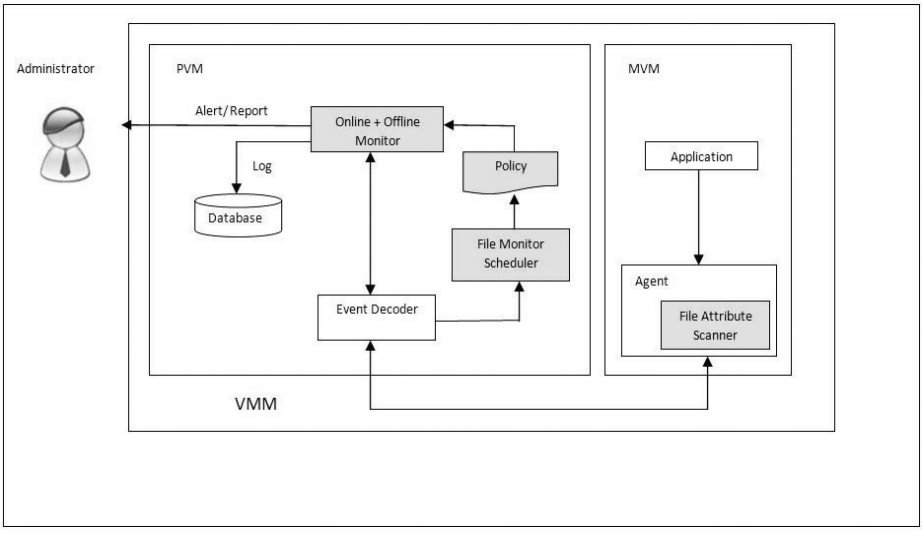

Fig. 2. Proposed FIM scheduling architecture

File Attribute Scanner (FAS). We collect file attributes to manipulate their information for our analysis and scheduler. Determining the specific group of files that require more frequent integrity inspection is a difficult task due to the various type of services offered by the operating systems. We assume that the system file structure is quite similar to various Windows based operating system. The security level of related group of files is the result of the combination between the file owner's rights and file permissions.

File attributes scanner (FAS) is locate in the agent packages that is deployed in MVM. In the FAS, files are scanned for the first time after our system installation on the MVM to create the baseline database. The baseline database of the files is stored in the PVM. In this process, the initial scheduler is created and added to the file monitor scheduler (FMS), which will overwrite the default policy. The monitoring engine will check the related files based on the defined policy. Then, if any changes occur in related files owner and permission, the FAS will update the classification and scheduler database.

We highlighted the FAS because it is what we have added in the previous agent's components. Another agent component is the file integrity monitor (FIM) that runs as the daemon process. FIM monitors the changes of the file content using the MD5 and SHA-1 checksum as well as changes in file ownership and permission. Event forwarding is part of the agent component which notifies the server for any event regarding file modification. Agent and server communicates via encrypted traffic.

Table 1. FIM check parameter

Check Parameter	Function
check_sum	Check files integrity using MD5/SHA1
check_size	Check changes of files size
check_perm	Check changes of files permission
check_group	Check changes of files group ownership
check_own	Check changes of files ownership

We implement our algorithm based on the OSSEC structure, hence, we also use the same check parameter suppose to (in Table 1) as OSSEC [6].

File Monitor Scheduler. File monitor scheduler (FMS) is one of our contributions in this paper. FMS collects file information from FAS in MVM via the event decoder to perform the file monitoring schedule based on the classification criteria. FMS has its own temporary database which contains groups of file names captured from FAS. The file groups will be updated if any changes occur in MVM captured by FAS. FMS will generate the FIM schedule and overwrite the default configuration file in the monitor engine. The monitoring engine will check related files based on the policy setting.

Policy. In default configuration, there are many built-in policy files which can be customized based on user requirements. In our case, we leave other policies as

default configuration, but we add new policy enhancement on the FIM frequency. Our FIM policy relies on the file security level classification which is based on file ownership and permission captured on MVM. We offer dynamic policy updates based on our FMS result. The frequency of the policy update is very low due to infrequent changes in the file security level.

Monitoring Engine (On-line and Off-line Monitor). Monitoring engine plays a key function for our system. It communicates with the event decoder in order to obtain file information from MVM and pass instructions to the agent in MVM. File information is needed in the monitoring process either in real time or periodic checking based on the policy setting (Figure 3). The monitoring engine should send instructions to the agent in MVM when it needs current file information to compare with the baseline databases especially for the off-line monitoring process.

```
<!-- Online checking for High security class files -->
<files realtime="yes" check_all="yes">Shigh</files>

<!-- Offline checking for Medium security class files -->
<frequency>36000</frequency>
<files check_all="yes">Smed</files>

<!-- Ignore checking the Low security class files --
<ignore>slow</ignore>
```

Fig. 3. Classification based FIM monitoring policy

3.2 File Classification Algorithm

In operating system environment, system files can be vulnerable to malicious modifications especially when attackers obtain administrator privileges. Therefore system file is the major concern in the FIM. However there are other files that should also be protected especially when related systems provide critical services to each other, such as web hosting, on-line banking, military related system, and medical related systems. It is quite subjective to define which files are more critical than others since every system provide different services.

In addition, huge number of files in the operating system environment is another challenge to the FIM in order to effectively monitor all those file without sacrificing the system performance. Hence, for that reason, we propose a file classification algorithm that can help FIM and other security tools to define the security requirements of related files.

Hai Jin et al [7] classified the files based on their security level weight as follows:

$$w_i = \alpha * f_i + \beta_i * d_i \ (\alpha + \beta = 1).$$

They represent the w_i as the weighted value for file i, f_i shows the file i access frequency, and they describe the significance of the directory containing the file i with d_i. They measure the files and directory weighted on the Linux environment where w_i represent the sensitivity level of the files. The variables, α and β, relate to the proportion of the frequency and the significance of the directory.

Microsoft offers File Classification Infrastructure (FCI) in their Windows Server 2008 R2 to assist users in managing their files [11]. FCI targets the business data files rather than system files. In other words, the files classification is based on the business impact and involves a more complex algorithm. Here we focus on the security impact on the systems and start with a simpler algorithm. In VRFPS file categorization, they classified the files the in Linux system into three types: *Read-only* files, *Log-on-write* files and *Write-free* files [22] to describe the security level of related files. In this paper, we also divide our file security level into three classes, *high*, *medium* and *low* security levels.

In this initial stage, we use the simple approach based on **user's right** and **object's permission** combination to define the file security level. However we exclude the user and group domains in this work as we are focusing more on the local files in MVM. User's rights refer to files owner that belong to a specific group that have specific privileges or action that they can or cannot perform. The files as objects that the user or group has permission or not to perform any operation to their content or properties [5]. For example, Ali as user, and a member of the Administrator group is permitted to modify the `system.ini` files contents. We define our files security level as follows:

- *High* security files: The files belong to Administrator. Other user groups have limited access to these files. Most of the system file type is in this group. This group of files requires on-line integrity checking.
- *Medium* security files: The files belong to Administrator group but other user groups also have permissions to read and write to these files. This group of file does not need on-line integrity monitoring but requires periodic monitoring, e.g. once a day.
- *Low* security files: The files are owned by users other than the Administrator group. This group of files can be ignored for integrity monitoring to reduce the system performance overhead during the monitoring process.

The goal of file security classification algorithm in Windows-based operating system is to dynamically schedule the integrity monitoring of those files. Different security levels of files need different monitoring schedules and this approach can optimize the FIM tool effectiveness and system performance as well. Moreover, the result of the file security classification provides information to the system administrator about the security needs of the related files.

Figure 4 shows our initial file security classification algorithm. We need basic file information including file names and its directory (*fname*), group of file's owner (*fgrp*), and file permission (*fperm*) as input, together with existing FIM policy files. All specified files will be classified as high (*Shigh*), medium (*Smed*) or low (*Slow*) security level based on their ownership and permission. Files' security

```
Algorithm 1: File security classification algorithm
Input: File information (fname, fgrp, fperm),policy files
Output: Shigh, Smed, Smed

procedure FileSecurityClassification

  Shigh, Smed and Slow are empty
  read the default policy files
  append the specified file to Shigh, Smed and Slow
  get the file information (fname, fgrp and fperm)
  the total of files names (fnum)
  for (i=0; i < fnum; i++)
  {
       if ( (fgrp = Administrators && fperm = full control)
       && (fgrp != Administrators || SYSTEM  &&  fperm != modify || write))
            append fname to Shigh

       else if ((fgrp = Administrator || SYSTEM || Power Users
           && fperm = modify || write) && (fgrp != Administrator
           && fperm = write))
            append fname to Smed
       else
            append fname to Slow

  }
  end procedure
```

Fig. 4. File security classification algorithm based on file ownership and permission

level information will be appended to the files information list, so any changes on their ownership and permission will be update. Dynamic update of the security level is needed due to discretionary access control (DAC) [17] implementation in Windows based OS which allow the file owner to determine and change access permission to user or group.

Table 2 indicate the comparison between our work with other FIM tools. We call our work as a dynamic file integrity monitoring (DFIM). The main objective of our work is to produce file integrity monitor in multi-platform environment. Variety of operating system in the needs more effective and flexible approaches. Therefore, base on some drawbacks of current FIM tools, we use file security classification algorithm to provide dynamic update of checking policy.

Table 2. Comparison with previous FIM tools

	Tripwire	XenFIT	OSSEC	DFIM
Multi Platform	No	Yes	Yes	Yes
Checking Approach	Periodic	Runtime	Periodic + Run-time	Periodic + Run-time
Policy Configuration	Static	Static	Static	Dynamic
File Classification	No	No	No	Yes
Require Virtualization Extension Support	No	Yes	No	No

This is an initial work for file security classification in Windows environment and is not complete enough to secure the whole file in general. More comprehensive study will be carried out in future to enhance the file security classification algorithm for better result.

4 Experiment Environment

We tested our approach in the virtualized environment using Oracle Sun Virtualbox. Ubuntu 10 Server edition is installed as a management server or privileged virtual machine (PVM) for our FIM and Windows XP Service Pack 3 as a monitored virtual machine (MVM). We install HIDS for client server packages. The experiment environment is Intel Core2 Duo CPU E8400 with 3.0GHz, and 3GB memory.

We assume that the virtual machine monitor (VMM) provides strong isolation between PVM and MVM that fulfills the virtualization technology security requirement. Basically, our system does not require hardware-based virtualization support and it can be deployed on any CPU platform. However the algorithm can also be tested on other virtualization based FIT that relies on the hardware-based virtualization support such as XenFIT and XenRIM.

We tested our algorithm by doing some modification to the high security level files to measure the effectiveness of on-line FIM setting. We found that the modification can be detected immediately after the changes are made (Figure 5).

Fig. 5. Detection of file modification

We are carrying out more detail experiments to measure the effectiveness of on-line and off-line FIM in detecting the file modification. In addition we will measure the performance overhead of our system to be compared to the native system.

5 Conclusion

We propose a new FIM scheduling algorithm based on file security classification that can dynamically update FIM needs. Most current FIM focus on their real-time FIM for sensitive files and ignored the other files without periodic checking

their integrity. In addition, changes in file attributes are also ignored by most of FIM tools which can reduce their effectiveness. First, we try to simplify the different security groups for the files based on user's rights and object (file) permission combination. In Windows environment, DAC provides flexibility to the users to determine the permission setting of their resources. Changes to the object permission sometimes also require changes to their security requirement. Next, we will enhance the algorithm to develop more comprehensive classification of files security. Moreover, file security classification can be also used in other security tools to enhance their capabilities with acceptable performance overhead. Other platforms such as mobile and smart phone environments also can be a next focus in the file security classification in order to identify their security requirement. Lastly, centralized management of security tools should be implement due to the large number of systems owned by organizations to ensure security updates and patches can perform in a more manageable manner.

References

1. Ossec - open source host-based intrusion detection system, `http://www.ossec.net/`
2. Al-Shaer, E.S., Hamed, H.H.: Modeling and management of firewall policies. IEEE Transactions on Network and Service Management 1(1), 2 (2004)
3. Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: malware analysis via hardware virtualization extensions. In: CCS 2008: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 51–62. ACM, New York (2008)
4. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proc. Network and Distributed Systems Security Symposium, pp. 191–206 (2003)
5. Glenn, W.: Windows 2003/2000/xp security architecture overview in expert reference series of white papers. Expert reference series of white papers, Global Knowledge Network, Inc. (2005)
6. Hay, A., Cid, D., Bary, R., Northcutt, S.: System integrity check and rootkit detection. In: OSSEC Host-Based Intrusion Detection Guide, Syngress, Burlington, pp. 149–174 (2008)
7. Jin, H., Xiang, G., Zou, D., Zhao, F., Li, M., Yu, C.: A guest-transparent file integrity monitoring method in virtualization environment. Comput. Math. Appl. 60(2), 256–266 (2010)
8. Kim, G.H., Spafford, E.H.: The design and implementation of tripwire: a file system integrity checker. In: CCS 1994: Proceedings of the 2nd ACM Conference on Computer and communications security, pp. 18–29. ACM, New York (1994)
9. Kim, J., Kim, I., Eom, Y.I.: Nopfit: File system integrity tool for virtual machine using multi-byte nop injection. In: Computational Science and its Applications, International Conference, vol. 0, pp. 335–338 (2010)
10. Kourai, K., Chiba, S.: Hyperspector: virtual distributed monitoring environments for secure intrusion detection. In: VEE 2005: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments, pp. 197–207. ACM, New York (2005)
11. Microsoft. File classification infrastructure, technical white paper. Technical white paper (2009), `http://www.microsoft.com/windowsserver2008/en/us/fci.aspx`

12. Patil, S., Kashyap, A., Sivathanu, G., Zadok, E.: I3fs: An in-kernel integrity checker and intrusion detection file system. In: Proceedings of the 18th USENIX Conference on System Administration, pp. 67–78. USENIX Association, Berkeley (2004)
13. Pfoh, J., Schneider, C., Eckert, C.: A formal model for virtual machine introspection. In: VMSec 2009: Proceedings of the 1st ACM Workshop on Virtual Machine Security, pp. 1–10. ACM, New York (2009)
14. Quynh, N.A., Takefuji, Y.: A real-time integrity monitor for xen virtual machine. In: Proceedings of the International conference on Networking and Services, p. 90. IEEE Computer Society, Washington, DC, USA (2006)
15. Quynh, N.A., Takefuji, Y.: A novel approach for a file-system integrity monitor tool of xen virtual machine. In: ASIACCS 2007: Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security, pp. 194–202. ACM, New York (2007)
16. Rami, L., Marc, H., van den Berg Richard.: The aide manual, `http://www.cs.tut.fi/~rammer/aide/manual.html`
17. Russinovich, M.E., Solomon, D.A.: Microsoft Windows Internals. In: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000 (Pro-Developer), 4th edn. Microsoft Press, Redmond (2004)
18. Szymczyk, M.: Detecting botnets in computer networks using multi-agent technology. In: Fourth International Conference on Dependability of Computer Systems, DepCos-RELCOMEX 2009, June 30- July 2, pp. 192–201 (2009)
19. Wichmann, R.: The samhain file integrity / host-based intrusion detection system (2006), `http://www.la-samhna.de/samhain/`
20. Wotring, B., Potter, B., Ranum, M., Wichmann, R.: Host Integrity Monitoring Using Osiris and Samhain. Syngress Publishing (2005)
21. Wurster, G., van Oorschot, P.C.: A control point for reducing root abuse of file-system privileges. In: CCS 2010: Proceedings of the 17th ACM Conference on Computer and Communications Security, pp. 224–236. ACM, New York (2010)
22. Zhao, F., Jiang, Y., Xiang, G., Jin, H., Jiang, W.: Vrfps: A novel virtual machine-based real-time file protection system. In: ACIS International Conference on Software Engineering Research, Management and Applications, pp. 217–224 (2009)
23. Zhao, X., Borders, K., Prakash, A.: Towards protecting sensitive files in a compromised system. In: Proceedings of the Third IEEE International Security in Storage Workshop, pp. 21–28. IEEE Computer Society, Los Alamitos (2005)