International Journal on New Computer Architectures and Their Applications (IJNCAA) 1(3): 766-779
The Society of Digital Information and Wireless Communications, 2011 (ISSN: 2220-9085)

Towards a Dynamic File Integrity Monitor through a Security 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, Malaysia.
zulhilmi_abd@yahoo.com, izura@fsktm.upm.edu.my, ramlan@fsktm.upm.edu.my
[2]Faculty of Engineering, Universiti Putra Malaysia,
43400 Serdang, Selangor, Malaysia
kmbs@eng.upm.edu.my

ABSTRACT

File is a component of a computer system that has importance value of its own, either in terms of availability, integrity, confidentiality and functionality to a system and application. If unintended changes happen on the related file, it may affect the security of related computer system. File integrity monitor (FIM) tools is widely used to minimize the file security risk. This paper proposed dynamic schedule for FIM. This paper presents a dynamic scheduling for FIM by combining on-line and off-line monitoring based on related files security requirement. 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

File Integrity, HIDS, File Security Classification, Dynamic Scheduling, Operating System.

1 INTRODUCTION

File is important element in computer system that used for input and output for most application [1]. In the operating system environments, file system is a most important component that must be protected in order to maintain the integrity and availability of their services. Ensuring the integrity of files on the computer system is crucial task nowadays due to huge number of instruction and data.

File integrity monitor (FIM) can be used to optimize the file security. In addition, other related tools known as file integrity verification, file integrity tools and file integrity checking. Generally, these tools have in common which serves to ensure the integrity of the files involved. Other integrity tools such as kernel [2-5], application [6, 7] and memory [8, 9] integrity tools also targeted to ensure the integrity of operating system.

FIM is one of the security tools that can be implemented in host environment as part of host based intrusion detection system (HIDS). FIM play a big role in monitoring the integrity of the files in the event of any changes to the files on their content, access control, privilege, group and other properties either by authorized or unauthorized users. The main goal of related integrity tools is to notify system administrator if any changed, deleted, or added files detected on the monitored system [10]. Basically, file integrity tools measure the current checksum or hash value of the monitored

files with their original value to detect any changes in file content.

In general, FIM can be divided into two categories in term of monitoring scheme, off-line and on-line scheme [11]. Off-line scheme refers to the FIM which monitors the integrity of the related files from time to time according to user setting. While on-line monitoring of FIM-related files in real time as each file is modified.

The recent solutions are more focused on on-line or real-time monitoring to enhance detection capabilities of malicious modification [11-13]. 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 additional software to make it efficient.

The main goal of the FIM is to ensure the integrity of file in operating system environment from intruders and also 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 [14].

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 and performance evaluation of our work. This paper ended with discussion and conclusion in Section 5.

2 RELATED WORKS

In operating system environment, every component such as instruction, device drivers and other data is saved in files. There are huge numbers 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 used by the attackers to attack the files in the operating system environment, 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 [10] 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 [16], AIDE [17], and Osiris [18] 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 schedule by classifying related files to certain groups and the inspection frequency will vary between the groups of files. Thus, from that approach, FIM can maintain its effectiveness with a more acceptable performance overhead to the system.

Wu et al. presents BinInt [19] as a new security model for binaries that prevents unauthorized binaries being executed. Although this work efficiently protects the binary, data files that more frequently changed cannot be covered. We are concern about this issue, so although we focus on system files integrity, our technique also can be implemented on the data files.

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. In addition, real-time application needs some scheduling guarantee from operating system to make it works [20] and load balancer also needed for minimize latency in multiple host environment.

As example, I3FS [21] 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 [22]. VMI was first introduced in Livewire [23] and then applied by the other tools like intrusion detection in HyperSpector [24] and malware analysis in Ether [25].

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 [26] which does not require a baseline database. NOPFIT [13] 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 [12]. 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 require 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. The other security tools also implement a centralized management for their tools, such as anti-malware [27] and firewalls [28], 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. [29] 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 [30], and OSSEC [31] 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 be detected as soon as the event occurs as fast detection can be vital to prevent further damage.

```
<syscheck>

<directoriesrealtime=”yes”
check_all=”yes”> /WINDOWS/system32
</directories>

<frequency>79200</frequency>
<!-- Directories to check -->
    <directories
check_all="yes">/WINDOWS</directorie
s>

</syscheck>
```

Figure 1. Example of system integrity checking configuration

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 Classification based FIM

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 need.

In this paper, we propose a new checking scheduling technique that dynamically updates 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 systems 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 is often targeted by attackers is a challenging task due to the various techniques that can be used to compromise the systems.

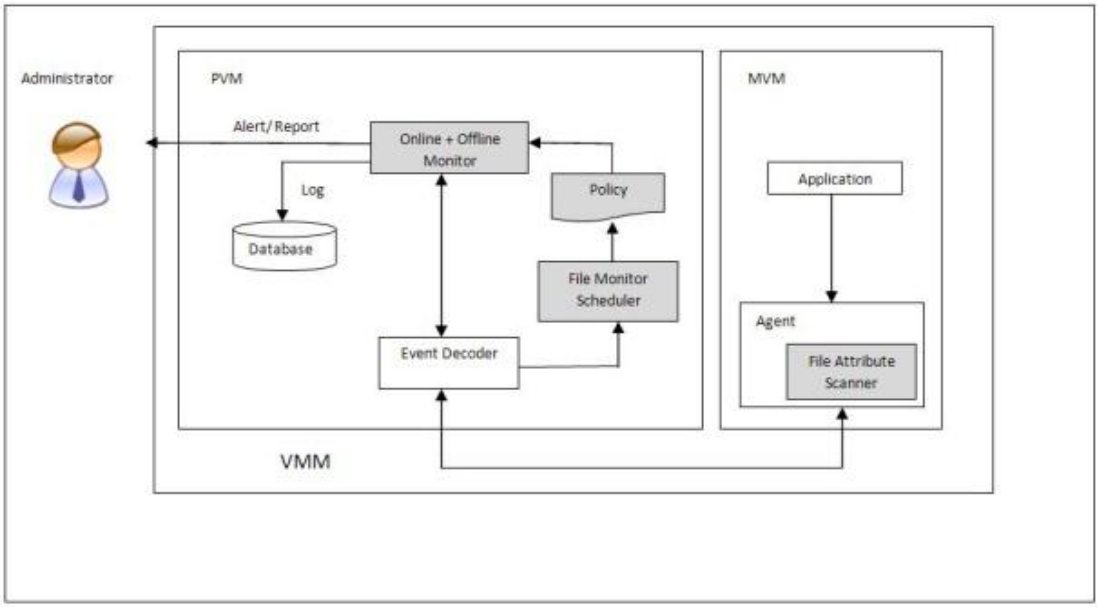

Figure 2. The DFIM architecture.

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.

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 types 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 package 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 communicate 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 OSSEC structure, so we also use the check parameter in Table 1 same as OSSEC check parameter [31].

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.

```
<files realtime="yes"
check_all="yes">Shigh</files>

<frequency>36000</frequency>

  <files check_all="yes">Smed</files>

<ignore>slow</ignore>
<alert_new_files>yes</alert_new_files>
<auto_ignore>yes</auto_ignore>
```

Figure 3. Classification based FIM monitoring policy

Monitoring Engine. 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

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. [11] 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 weighted value for file i, f_i shows the file i access frequency, and they describing the significance of the directory contain the file i with d_i. They measure the files and directory weighted on the Linux environment which w_i represent the importance of the files. The variables, α and β, relate to the proportion of the frequency and the significance of directory.

Microsoft offers File Classification Infrastructure (FCI) in their Windows Server 2008 R2 to assist users in managing their files [32]. 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 [12] to describe the security level of related files. In this paper, we also divide our file security

level into three classes, *high*, *medium* or *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 [33]. 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.

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

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
```

Figure 4. File security classification algorithm based on files ownership and permission

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. In this paper, we create a single group of files for this level.

Next, we will improve it to create more additional groups in medium security level. The additional group will result different scheduling in periodically mechanism for different type of files in medium security level groups. Hence, distribution of scheduling can minimize

the usage of computer resources and reduce performance bottleneck.

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 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) [34] implementation in Windows based OS which allow the file owner to determine and change access permission to user or group.

Table 2 indicates the comparison between our works 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 market 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 monitoring policy.

Table 2. Comparison with previous FIM tools

	Tripwire	XenFIT	OSSEC	DFIM
Multi Platform	No	Yes	Yes	Yes
Frequency Check	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. We measure the performance of the system with the classification algorithm run. More comprehensive study will be carried out in future to enhance the file security classification algorithm for better result.

4 Testing and Result

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).

Figure 5. Detection of files 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 measured the performance overhead of our system to be compared to the native system. Initial testing carried out on the client side running Windows XP SP3 with minimal software installed. We will carry out the further experiment on the server side as well as other operating systems platform on the client side environment.

In this stage, we measure time to be completed by the system (1) to build a software package and (2) to create an archive for the software package using archive manager (WinRAR). Time taken to complete both workloads are measured by activate related function and compared against the native system (without DFIM function). Testing was carried out ten times for each workload in each function activated. The average time taken was recorded in Table 3.

Table 3. DFIM Performance Evaluation

Workload	System	Time Completed	Overhead
Build	Base	24.2s	-
	Initial Classification	25.4s	4.9%
	DFIM + Off-line	24.6s	1.7%
	DFIM + On-line	25.0s	3.3%
	DFIM Dual	25.2s	4.1%
Archive	Base	35.1s	-
	Initial Classification	36.7s	4.5%
	DFIM + Off-line	35.7s	1.4%
	DFIM + On-line	36.1s	2.8%
	DFIM Dual	36.4s	3.7%

Table 3 shows the performance evaluation of DFIM tools when related function in DFIM activated. Both workloads recorded maximum time to complete during the initial classification activated. In the initial classification process, more information needed in order to create a list of file security group. Scanning the file attributes result the highest overhead on the system. Computer resources are shared with this classification process, make time to complete for both operation take longer.

Minimum time to complete for both workloads is during the off-line

monitoring process activated. In this process only files that are scheduled, monitored within certain period of time. Process does not involve acquisition of real-time information resulting only minimum use of resources. In such cases, the computer resources can be shared by other workloads (testing workload).

Overhead of the on-line monitoring, 3.3% for *build* and 2.8% of *archive* shows that it still in acceptable level. Next, activation of whole system function by combining the off-line and on-line monitoring result higher overhead (4.1% and 3.7%) but it also still practical to implement. It not just a dual scheme monitoring process, the dynamic classification and scheduling update as add-on function should be considered.

5 Conclusions

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. DFIM provides automated mechanism to update the file security level if any changes involved. The changes of security level will result the update of monitoring schedule.

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 implemented due to the large number of systems owned by organizations to ensure security updates and patches can perform in a more manageable manner.

6 REFERENCES

1. Stallings, W., Operating Systems: Internals and Design Principles. Sixth ed. 2008: Prentice Hall Press Upper Saddle River, NJ, USA.
2. Nick L. Petroni, J., et al., An architecture for specification-based detection of semantic integrity violations in kernel dynamic data, in Proceedings of the 15th conference on USENIX Security Symposium - Volume 15. 2006, USENIX Association: Vancouver, B.C., Canada.
3. Xu, M., et al., Towards a VMM-based usage control framework for OS kernel integrity protection, in Proceedings of the 12th ACM symposium on Access control models and technologies. 2007, ACM: Sophia Antipolis, France.
4. Xuan, C., J. Copeland, and R. Beyah, Shepherding Loadable Kernel Modules through On-demand Emulation, in Detection of Intrusions and Malware, and Vulnerability Assessment. 2009. p. 48-67.

5. Seshadri, A., et al., SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes, in Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles. 2007, ACM: Stevenson, Washington, USA.
6. Lifu, W. and P. Dasgupta. Kernel and Application Integrity Assurance: Ensuring Freedom from Rootkits and Malware in a Computer System. in Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on. 2007.
7. Li, N., Z. Mao, and H. Chen, Usable Mandatory Integrity Protection for Operating Systems, in Proceedings of the 2007 IEEE Symposium on Security and Privacy. 2007, IEEE Computer Society.
8. Dewan, P., et al., A hypervisor-based system for protecting software runtime memory and persistent storage, in Proceedings of the 2008 Spring simulation multiconference. 2008, The Society for Computer Simulation, International: Ottawa, Canada.
9. Akritidis, P., et al. Preventing Memory Error Exploits with WIT. in Security and Privacy, 2008. SP 2008. IEEE Symposium on. 2008.
10. Kim, G.H. and E.H. Spafford, The design and implementation of tripwire: a file system integrity checker, in Proceedings of the 2nd ACM Conference on Computer and communications security. 1994, ACM: Fairfax, Virginia, United States.
11. Jin, H., et al., A guest-transparent file integrity monitoring method in virtualization environment. Comput. Math. Appl., 2010. **60**(2): p. 256-266.
12. Feng, Z. VRFPS: A Novel Virtual Machine-Based Real-time File Protection System. in oftware Engineering Research, Management and Applications, ACIS International Conference. 2009: IEEE Computer Society.
13. Junghan, K. NOPFIT: File System Integrity Tool for Virtual Machine Using Multi-byte NOP Injection. in 2010 International Conference on Computational Science and Its Applications. 2010. Fukuoka, Japan.
14. Zhao, X., K. Borders, and A. Prakash, Towards Protecting Sensitive Files in a Compromised System, in Proceedings of the Third IEEE International Security in Storage Workshop. 2005, IEEE Computer Society.
15. Quynh, N.A. and Y. Takefuji, A novel approach for a file-system integrity monitor tool of Xen virtual machine, in Proceedings of the 2nd ACM symposium on Information, computer and communications security. 2007, ACM: Singapore.
16. http://www.la-samhna.de/samhain/. The SAMHAIN file integrity / host-based intrusion detection system. 2006 [cited 2010 November 2nd].
17. Lehti, R., M. Haber, and R.v.d. Ber. The AIDE manual. 20 May 2011 [cited 2011 July 10th]; Available from: http://aide.sourceforge.net/stable/manual.html.
18. Wotring, B. and B. Potter, Osiris, in Host Integrity Monitoring Using Osiris and Samhain. 2005, Syngress: Burlington. p. 141-239.
19. Wu, Y. and R.H.C. Yap, Towards a binary integrity system for windows, in Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. 2011, ACM: Hong Kong, China.
20. Lee, M., et al., Supporting soft real-time tasks in the xen hypervisor, in Proceedings of the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. 2010, ACM: Pittsburgh, Pennsylvania, USA.
21. Patil, S., et al., I3FS: An In-Kernel Integrity Checker and Intrusion Detection File System, in Proceedings of the 18th USENIX conference on System administration. 2004, USENIX Association: Atlanta, GA.
22. Pfoh, J., C. Schneider, and C. Eckert, A formal model for virtual machine introspection, in Proceedings of the 1st ACM workshop on Virtual machine security. 2009, ACM: Chicago, Illinois, USA.
23. Tal Garfinkel, M.R. A Virtual Machine Introspection Based Architecture for Intrusion Detection in Proc. Network and Distributed Systems Security Symposium. 2003.
24. Kourai, K. and S. Chiba, HyperSpector: virtual distributed monitoring environments for secure intrusion detection, in Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments. 2005, ACM: Chicago, IL, USA.
25. Dinaburg, A., et al., Ether: malware analysis via hardware virtualization extensions, in Proceedings of the 15th ACM conference on Computer and communications security. 2008, ACM: Alexandria, Virginia, USA.

26. Quynh, N.A. and Y. Takefuji, A Real-time Integrity Monitor for Xen Virtual Machine, in Proceedings of the International conference on Networking and Services. 2006, IEEE Computer Society.
27. Szymczyk, M., Detecting Botnets in Computer Networks Using Multi-agent Technology, in Proceedings of the 2009 Fourth International Conference on Dependability of Computer Systems. 2009, IEEE Computer Society.
28. Shaer, E. and H. Hamed, Modeling and management of firewall policies. IEEE Trans. Network and Service Management, 2004. **1**(1).
29. Wurster, G. and P.C.v. Oorschot, A control point for reducing root abuse of file-system privileges, in Proceedings of the 17th ACM conference on Computer and communications security. 2010, ACM: Chicago, Illinois, USA.
30. Wotring, B. and B. Potter, Samhain, in Host Integrity Monitoring Using Osiris and Samhain. 2005, Syngress: Burlington. p. 241-305.
31. Hay, A., et al., System Integrity Check and Rootkit Detection, in OSSEC Host-Based Intrusion Detection Guide. 2008, Syngress: Burlington. p. 149-174.
32. Microsoft. File classification infrastructure, technical white paper. 2009; Available from: http://www.microsoft.com/windowsserver2008/en/us/fci.aspx.
33. Weadock, G., Windows 2003/2000/xp security architecture overview. 2005, Global Knowledge Network, Inc.
34. Russinovich, M.E. and D.A. Solomon, Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000 (Pro-Developer). 2004: Microsoft Press Redmond, WA, USA.