Smart Offline Application Intrusion Monitoring System

Pranav John Issac

Abstract— Hackers and malware affect applications by changing the crucial files to extract critical data such as credit card information, bank passwords, etc. from users without user's knowledge. This is done by changing critical files of the applications such as chrome by the Trojan or backdoors. Existing systems like antivirus, anti-malwares and prevention systems does not detect new attack since the database have to be updated using attack patterns that are generated by the antivirus vendors after analyzing the attack. The antivirus vendors only analyses popular virus. Such localized attacks are undetected by antivirus vendors. The proposed system detects these unauthorized changes to the application files in real-time using a smart file signature that is generated from md5, modification time and name. The proposed system monitors applications regularly and reports any unauthorized changes to file structure, informing user of threat in Real time instead of waiting for antivirus update and also detect corrupt files.

Index Terms— File Intrusion Monitoring System, Offline Intrusion detection, SHA, MD5, Realtime File protection from corruption, hacker prevention , realtime application blocking.

1 Introduction

The existing system of antivirus scan uses general attack and known signature for application intrusion detection. The disadvantage with this are that hackers' attacks are unique and undetectable to Antivirus and existing systems does not monitor any behavioural changes in file system. The usual antivirus contains a global file signature database with unique file signature for most common virus. In certain cases, when a new threat is detected the antivirus checks only for the global database thus leaving the local ones unnoticed. But these changes that go unnoticed can affect the critical files and can extract the data stealthily. Thus, we could say that antivirus cannot protect the files and applications completely.

The changes that went unnoticed and noticed with the existing system can be detected by the proposed system; any unauthorized changes to critical files are notified and the respective application is blocked and thus prevents hacking or data leak. The advantages of our proposed system are: Reporting of corrupted files and continuous monitoring of file systems. The system creates a unique smart file signature that could even identify the slightest change made to the critical files of the application. The smart file signatures generated are stored into a database and are compared each time the application gets started and periodically without affecting the system performance. In case of any mismatch found the user will be notified and the corresponding applicaiton would be blocked from further execution. A daemon module present would always be there running in the background, scanning and reporting in realtime.

Proposed system detects above discussed unauthorized changes using a smart file signature usually generated from size, md5, modification time, access time and name. Unlike the existing system no Global database signature needed. Even the slightest change made to the path or content of the file can result in the mismatch of signature leading to blocking its further execution thus preventing the least possible way for a hacker to intrude.

This paper is divided into 5 chapters , Chaper 1 is the introduction , Chapter 2 deals with the design of the system, Chapter 3 explains the implementation details, Chapter 4 presents the results of implementation and Chapter 5 concludes the paper.

• *Pranav John Issac is currently researching in Application security and smart Hacking prevention, India, PH-+918129311475.*
E-mail: pranavjohnissac@gmail.com

2 Design of the intrusion monitoring system

2.1 Development Requirement.

It is generally accepted that majority of virus affects Windows users and most of the commercial users tends to use Windows operating system. Thus, the system is primarily focused for windows operating system. Due to nativity with windows the programming language that is recommended for the system is C# and the database for signature management is lightweight MySQL. The targeted .NET framework is 4.5.1. For a more platform independent version C# can be replaced with java, but it is recommended to use C# for Windows operating system.

2.2 System Requirement

Since the system access all critical files, highest administrative previleage is recommended. The recommended operating system is Windows 7 and above. The system should have atleast 1 GB RAM and a dual core processor with atleast 1GHZ clock speed. Internet is required only if application build updates.

2.3 Design

The system consists of a start up and a daemon module, the startup module starts as a Windows service and self scans the application itself and activates the daemon module which will continuously scan, detect and block intrusion and application.

Second one is the interface and configuration module which deals with the different operations that user requests, such as manual scan, application addition, scan interval setting, removal of application, regeneration of file signature, manually blocking application, pause/resume scan and perform build updates.

The third module is the signature generation module which calculates the unique SMART signature for each of the application to be monitored using a combination of size, md5, modification time, access time and name

The algorithm for signature generation of an application is as follows can be described as follows:

1. First critical file list of the selected application is generated.
2. For a file, the MD5 is generated
3. The MD5, last modified time, path , size, etc are converted to string and concatenated.
4. SHA256 for the string is calculated. This is the unique signature
5. The whole process is repeated for each of the files.

Generated signatures are stored to a smart signature database.

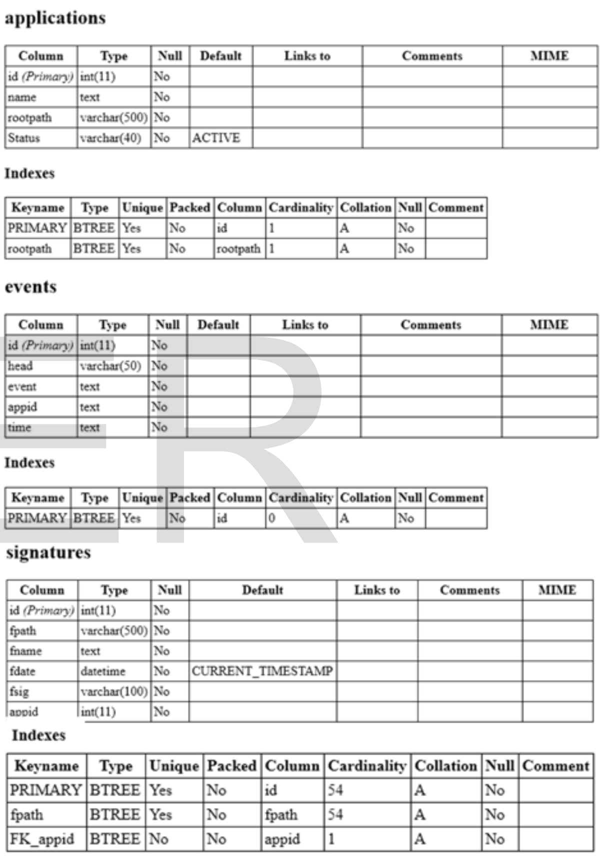
Finally, the last module is scanning and detection module which scans the files for any changes made to its critical files (which include .exe, .ini, .dll, .inf ,.xml ,pma files) of the application. The algorithm is as follows

1. For an application the critical files currently in the app folder is fetched.
2. The stored signature for the application is fetched from DB.
3. If the number of files mismatch, the application is blocked.
4. Else each file's signature is generated and matchedwith that in the smart signature database. If these signatures do not match then the application is blocked by adding the app to blocked app list .
5. After blocking an alert notification is sent to the user indicating that the critical file has undergone some changes and the current system application enters "blocked" status.

All the actions are tracked. Once all these steps are completed for one application the next application can be added for the scan.

The blocking module is a daemon thread which continuously watch currently running application. If a blocked application is being executed, it is blocked.

Figure 1. shows the the database structure,



applications

Column	Type	Null	Default	Links to	Comments	MIME
id *(Primary)*	int(11)	No				
name	text	No				
rootpath	varchar(500)	No				
Status	varchar(40)	No	ACTIVE			

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	1	A	No	
rootpath	BTREE	Yes	No	rootpath	1	A	No	

events

Column	Type	Null	Default	Links to	Comments	MIME
id *(Primary)*	int(11)	No				
head	varchar(50)	No				
event	text	No				
appid	text	No				
time	text	No				

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	0	A	No	

signatures

Column	Type	Null	Default	Links to	Comments	MIME
id *(Primary)*	int(11)	No				
fpath	varchar(500)	No				
fname	text	No				
fdate	datetime	No	CURRENT_TIMESTAMP			
fsig	varchar(100)	No				
appid	int(11)	No				

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	54	A	No	
fpath	BTREE	Yes	No	fpath	54	A	No	
FK_appid	BTREE	No	No	appid	1	A	No	

Fig. 1. Database structure for the application

3 Implementation in Windows

There are 5 modules namely, Startup Module, Daemon module, Interface and configuration module, Signature Generation Module, Scanning and detection modules. Each are explained below

3.1 Startup Module

The **start up** module gets automatically invoked when the system starts. This module is registered as a service in operating system. This is done by adding the SMART file intrusion application to the task scheduler of the windows operating system. The service is registered as follows:

"sc.exe create SmartIntrusionbinPath= \"" + AppDomain.CurrentDomain. BaseDirectory + "app.exe\" start=auto";

The above code can be explained as the windows contains a "sc" executable file. The filename to be added to the registry is given as an argument using the sc create statement followed by the binpath.
On executing it in the command prompt the application is added as a service to the OS.
This module performs self analysis of all the applications using scanning and detection module. It also regularly checks for updates. The module starts the database service and activate the daemon module. All the applications which are already added to the scanned list will be automatically scanned for errors and updates.
The module finally exits after the scan activating the interface module in background.
Figure 2. shows the data flow diagram for the above discussed matter.

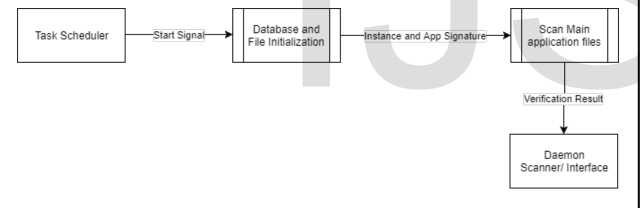

Task Scheduler
Start Signal
Database and File Initialization
Instance and App Signature
Scan Main application files
Verification Result
Daemon Scanner/ Interface

Fig 2. DFD for Startup Module

3.2 Daemon Module

The module is essentially a low priority daemon thread which performs 2 funtions which are blocking of blocked application from starting up and scanning monitored files for changes using scanning module. The module schedules the scanning process and shut down running application. This module can be paused using the interface module. The DFD for this module is given in Fig 3.

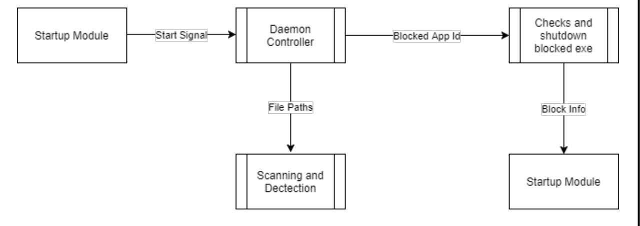


Fig 3. DFD for Daemon Module

3.3 Interface and Configuration Module

It's the one that handles the user communication part of the application. The operations are taken from the user to this module for further procedures or processing. From here the process takes lead to scanning and detection module and then to signature generation module from there.
The module provides interface for the following operations:

1. Changing settings
2. Manage Exclusions
3. Updating
4. Application addition
5. Pausing Scan.
6. Reset Database
7. Regeneration
8. Manually Perform scans.

Fig 4. shows the dataflow diagram of the module.

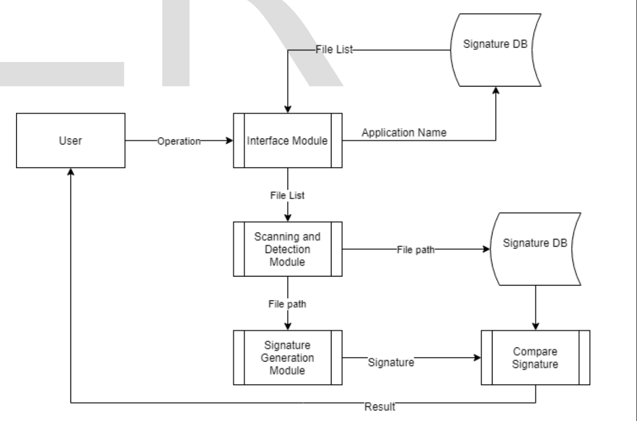


Fig 4. DFD for Interface and configuration Module

3.4 Signature Generation Module

The signature generation module is responsible for generating smart file signature based on the attributes such as path, modification time, md5 and name of the file. Generated smart signature is stored in a signature database along with its path. Further when an application is explicitly given for scanning, a signature for the current file is generated which is then compared with the signature in the database. If at all a mismatch is found after the comparison a notification is given to the user.

The signature generation module is incorporated with functions to get the file path, calculate the hash function using Secure Hash Algorithm 1(SHA-1) and calculate the MD5. The process is described in the design section.

The DFD for the module is in Fig 5.

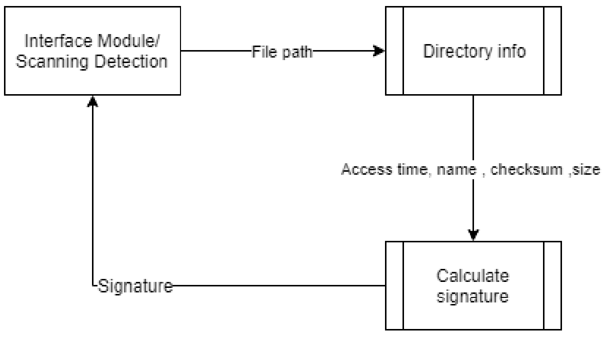

Interface Module/ Scanning Detection
File path
Directory info
Access time, name , checksum ,size
Calculate signature
Signature

Fig 5. DFD for Signature Generation Module

3.5 Scanning and detection

This module scan files for each application as invoked by daemon module. For each application, there are sub-files and critical files (.dll, .exe, .ini, .inf extension files). The smart signature generated for each sub-file is stored in database. The current signature is then compared with the smart signature stored in database. If they are not matched, the current application enters "blocked" status. Otherwise, it continues with the "active" status. The actions made after the comparison is stored in the database by a function HandleMalware. It also reports to the user if the signature mismatches. If the change is desired by the user then, it is considered as a volatile file and added to exclusion. If not, it uploads the signature to the server and blocks application from running. Get the next file to be scanned and repeat this process until daemon module stops.
Figure 6 shows working of scanning and detection module

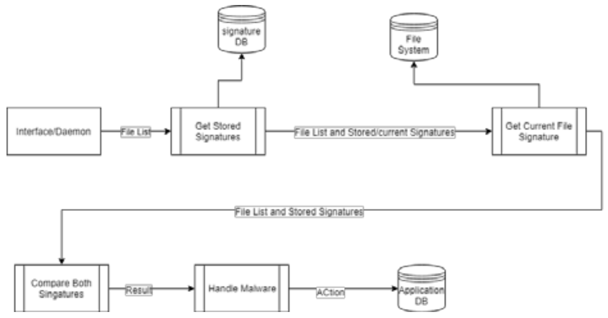


Fig 5. DFD for Scanning and detection module

4 Results

The proposed system was implemented using C# and .NET 4.5.2 and MariaDB. The application found out to be effective most of the time. The resource usage was considerably low. The only disadvantage observed was that the application file signatures had to be re generated automatically each time an application is updated by the vendor.

5 CONCLUSIONS

In conclusion the application is provides offline protection to files and application that are added for monitoring from hackers, malwares and corruption. Even though the implementation was targeted on windows based Operating systems, the same can be implemented for any operating system using most effective native programming languages. The advantage of the system over other existing prevention systems is that the proposed system operates offline and is targeted to specific application. The future scopes may include an online signature database that automatically ignores the genuine files after the vendor update.

References

[1] Alok Kumar Kasgar, "A Review Paper of Message Digest 5 (MD5)," International Journal of Modern Engineering & Management Research Volume 1, Issue 4, December 2013 ISSN: 2320-9984 (Online)

[2] R. Rivest. The MD5 Message-Digest Algorithm [rfc1321]

[3] Tao Xie and Dengguo Feng (30 May 2009). "How to Find Weak Input Differences for MD5 Collision Attack".

[4] Raaed K. Ibrahim, Ali SH. Hussain, Roula A. Kadhim "IMPLEMENTATION OF SECURE HASH ALGORITHM SHA-1 BY LABVIEW" IJCSMC, Vol. 4, Issue. 3, March 2015, pg.61 – 67

[5] X. &. L. G. Chan, Discussion of One Improved Hash Algorithm Based on MD5 and SHA1, San Francisco, USA: World Congress on Engineering and Computer Science (WCECS), 2007