Information and Knowledge Management www.iiste.org
ISSN 2224-5758 (Paper) ISSN 2224-896X (Online)
Vol.8, No.1, 2018

Enhancing the Performance of Intrusion Detection System by Minimizing the False Alarm Detection Using Fuzzy Logic

Khaled Batiha* Mohammed O. Alshroqee
Computer science departments, Al al-Bayt University, Jordan

Abstract
According to the information technology and regarding to the revolutions of the computer worlds, this world has got important information and files that have to be secured from different types of attacks that corrupt and distort them. Thus, many algorithms have turned up to increase the level of security and to detect all types of such attacks. Furthermore, many algorithms such as Message Digest algorithm 5 (MD5) and Secure Hash Algorithm 1 (SHA-1) tend to detect whether the file is attacked, corrupt and distorted or not. In addition, there should be more algorithms to detect the range of harm which the files are exposed to in order to make sure we can use these files after they have been affected by such attacks. To be clear, MD5 and SHA-1 consider the file corrupt once it is attacked; regardless the rate of change .Therefore, the aim of this paper is to use an algorithm that allows certain rate of change according to the user, which is SSdeep algorithm. Meanwhile, it gives the rates of change depending on the importance of each file. Moreover, each rate of change determines whether we can make use of the file or not. I made assumption in creating four folders, each contains multiple files with minimum predefined allowed of similarity. Then graphical user interface is created to utilize the SSdeep algorithm and to permit user to define the allowed similarity on each folder or file depending on impotency of it. After applying the algorithm, I got results showing the benefits of such algorithm to make use of these attacked or modified files.
Keywords: Intrusion Detection System, false alarm, fuzzy logic, computer security

1. Introduction

As computers are becoming increasingly used by labor, security issues have posed a big problem within organizations. Firewalls, anti-virus software, password control are amongst the common steps that people take towards protecting their systems. However, these preventive measures are not perfect. Firewalls are vulnerable; they may be improperly configured or may not be able to prevent new types of attacks. Anti-virus software works only if the virus matches its signature. Passwords can be stolen and therefore, systems can be easily hacked into. Hackers can change the system on initial access and manipulate it so that their future access will not be detected. In these situations, Intrusion Detection Systems (IDS) come into play [Mallery, 2008].

Intrusion Detection System (IDS) meant to determine whether the file or the packet is intruded or not. Depending upon the place of anomaly, the IDS can be classified into Network-based IDS or Host-based IDS [Scarfone, 2007].

The present study focused on HIDS. Therefore, whether the IDS is HIDS or NIDS, it should inform the administrator by any means that suspicious activities had happened and should trigger an alarm.

The so-called "False Alarm Detection" is a method that specializes in detecting anomalies in computer systems and networks, which is mainly dependent on fuzzy logic and artificial intelligence. The main purpose is to differentiate between normal behaviors and anomalous ones. What makes gaps and weaknesses is the false alarm rate mainly measured and counted by the false positives of normal behaviors [Pokrywka, 2008].

To clarify the idea, some anti-virus programs deal with programs and data as viruses which are directly stopped. This, in role, gives false alarms. For example, the so-called "Kaspersky" anti-virus program deals with the program "Net Support" as a virus which is directly stopped and cannot be installed. Because they are not viruses, we conclude that what happens is the so-called "False Alarm".

Many researchers have done researches in the field of computer and its network, and they have talked and discussed the threats that threaten them without talking about what is supposed to do in order to recognize the consequences and the damage by attacks and hackers. Therefore, this research dedicated to measure these attacks' damages. Regarding this matter, the Intrusion Detection System is used to detect these threats in systems. Therefore, in using the software, we developed the techniques we have created. Thus, the user can recognize the consequences, the damages, and the attacks that happened. Consequently, this research shows in details the problems, types of attacks, used algorithms and approaches to detect these illegal threats and its impact.

When changes happens to some critical files, the IDS alerts the system or the administrator that these files intruded but without specifying the size of the damage and the effect of damage, which may increase the false alarm. As a result, we need an algorithm that can investigate the size of the damage and its effects. In so doing, the damage cannot be considered harmful and does not need to fire an alarm about it.

Consequently, the present study gives the user the control with the range of trueness of produced alarms that detecting intrusions.

This research comes up with findings and solutions that are supposed to be followed when our computers and networks are exposed to face malicious attacks and hackers who tend to distort our main and important files.

Besides, it show we can utilize the proposed algorithms to detect the anomalies in these files and take re-use of some of the attacked files.

2. Related Work

Zadeh [Zadeh, 1965] started with the concept of fuzzy set theory, which was mean to focus on the vagueness for dealing with it in several cases in the world. The function called membership explains the values of universe that lie between (0, 1). Each value has got an indication. The (0) value indicates that it is not a member in the fuzzy set, whereas the (1) value indicates that it is a member in the fuzzy set. So, the other values remain within this range.

Kornblum [Kornblum, 2006] found Context Triggered Piecewise Hashing (abbreviated CTPH) that divides an input dependent on its context. It was basically dependent on a spam detection algorithm of Tridgell. Since then several researches had been published which checked this method in details. For example, improvements related to efficiency and security had been proposed by F.Breitinger in "Performance Issues about Context-Triggered Piecewise Hashing", whereas a security analysis had shown that this method cannot resist an active opponent regarding whitelisting and blacklisting.

Vassil Roussev [Vassil Roussev, 2011] proposes a comparison between SDhash and SSdeep which demonstrates that an "approach significantly outperforms in terms of recall and accuracy in all tested scenarios and insists on active and scalable behavior".

Despite the importance of the above list of sources, this study presents a different perspective. As mentioned earlier, the previous studies highlight and detect the damage on the network only. However, this study detects the damage on the host. In fact, it tends to propose an algorithm that measures the size of the damage and the effects of the damage when changes influence some critical files. Moreover, it comes up with solutions to re-use some of the attacked files.

3. Combining the Hash Algorithms

The rolling hash is used when the current piecewise hashing programs used fixed offsets to conclude when to begin and to stop the traditional hash algorithm. Yet, when the rolling hash's output is specific, the traditional hash is triggered. While processing the input file, one must compute not only, the traditional hash for the file but also, the rolling hash in order to record the value of the traditional hash in the CTPH signature, paving the way for the traditional hash to rest. As a result, any change in the input is recorded and seen in localized changes only in the CTPH signature, maintaining the majority of the CTPH signature. Therefore, the modified file is associated with CTPH signature of known files.

3.1Approximate matching

Using the SHA-1 or MD5 only gives two simple answers (yes or no), so two matched files might match or might not. However, according to SSdeep algorithm, it gives a probable answer within the interval of numbers (0-1) once two files are compared. Here we have got two types of scores as the following:
Confidence score that indicates a low score when there is a small amount of similar content in the two files.
High score when the ratio of similarity of content is high [Roussev, 2013].

3.2 Custom Score

We proposed another type of score called custom score. It is mainly dependent on the importance of certain files according to the user point of view. This allows the user the ability to choose the files he wants to be secured and how much he allows a percentage of changes.

To be clear on this matter, we suggested that the user has important files which kept in a folder with minimum 95 percent of importance, and the less important files are kept in a folder that has got minimum 85 percent of importance. The files which have got minimum 75 percent of importance are also saved in a third folder, and the less important files are saved in a folder that has got minimum 65 percent of importance. Moreover, the custom score comes in the interval (0, 1) because (0) means that the file is totally different, whereas (1) indicates that the file is totally identical. What comes in between is dependent on the user's custom score.

Empirical tests stated that more than 65% of similarity leads to the recognition of the same files as they seem to be identical [Chyssler, 2004].

At last, we conclude that the files are classified into many percentages of importance according to the user's desire. This also shows if the input file can be used or distorted.

4. Methodology

The attacks have different impacts on the files, some of these attacks are considered innocuous depending on the data types. Also, we have set an assumption stating that determining the allowed rate of change belongs to the user's request of choosing the type of file and the location of it, and the files are compared by File Finger Print

(Hash). Some algorithms such as SHA-1 or MD5 consider that any attack on the data affects it and not useful. But it cannot determine the size of the damage on these data, whereas the SSdeep algorithm can determine the size of damage on these data. Depending on the above assumption, some data are very important. Thus, if the attack happened over the allowed rate, they are considered useless or unbeneficial. On the other hand, if some of the files are attacked within the allowed rate, they can be used and considered safe. Therefore, the figure (1) demonstrates our assumption, depending on the location and the extension of file. In so doing, we can determine that the alarm, produced by the attack, is true or not.

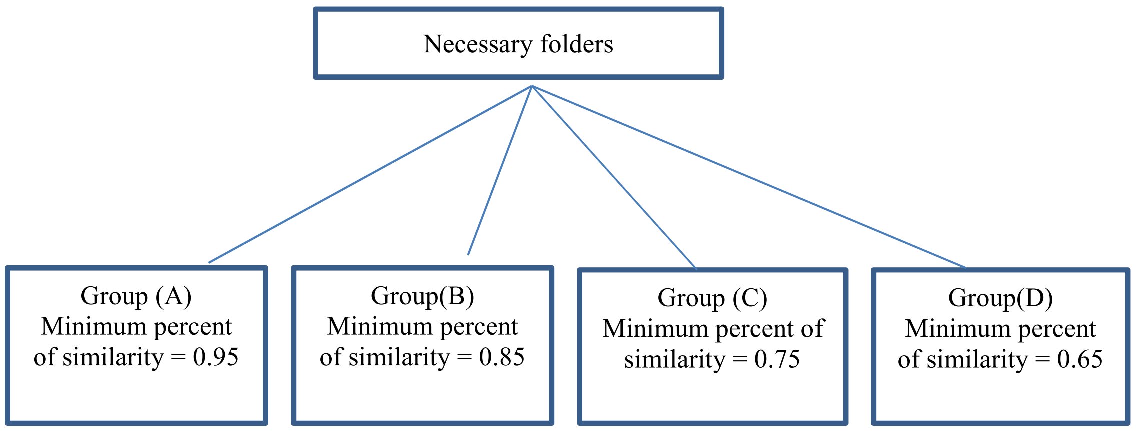
Necessary folders
Group (A) Minimum percent of similarity = 0.95
Group(B) Minimum percent of similarity = 0.85
Group (C) Minimum percent of similarity = 0.75
Group(D) Minimum percent of similarity = 0.65

Figure 1: Minimum percent of similarity for each folder

The folder that is called “Necessary folders” is classified in to four main folders according to minimum percent of allowed similarity, and the following folders show this classification:
Folder "A"- is the most important one that includes sensitive files.
Folder "B"- is less important than folder "A", which includes files, which can be exposed to few changes.
Folder "C"-includes files, which can face more changes than folder "B".
Folder "D"- includes files that can face many changes in comparison with the previous folders.

In addition to this, we allowed the user to set the importance of minimum of similarity according to the type of the file (extension) in order to increase the security.

Here, the proposed assumption of the minimum of similarity depends on the user himself and his job. For example, the most important files for secretary are MS-word files. Therefore, the minimum of allowed similarity on these files is very high and these files are critical, while the pdf files are less important with less minimum of similarity. However, according to a programmer, the most important files are the DLL files and the minimum allowed of similarity is very high, while the MS-Excel files are less important with fewer minimums allowed of similarity.

As we can see in Figure (2), we proposed generally the minimum allowed of similarity of some files, but the user can set his own minimum allowed percent of similarity depending on his job and the location he puts the files in to. Also, we made assumption states that the default percentage of any other files is 65 per cent.

TXT Minimum percent of similarity is 0.67.

PDF Minimum percent of similarity is 0.75.

XLS Minimum percent of similarity is 0.80.

DOCX Minimum percent of similarity is 0.90.

DLL Minimum percent of similarityis 0.95

Figure 2: Minimum per cent of similarity depend on file type

Consequently, the comparison is made according to the type of the file (extension) and where it exists, based on the highest rate, as demonstrated in the following equation:
α: file acceptable percentage (extension for file).
β: folder acceptable percentage.
μ: result percentage.

$$\mu = \left\{ \begin{matrix} \alpha \geq \beta & \alpha \\ \text{otherwise} \beta & \end{matrix} \right\}$$ μ: is it the highest percentage

Stages of our proposed work:

a) Preprocessing Stage:

Our system deals with a huge set of files that cannot process them shortly so we do pre-processing for these files and calculate the hash for each one. Then, we store them in the data base to be used later.

In this stage, the system scans all the selected files by the user and passes them to "SSdeep algorithm" in order to produce a hash code which consists of alphabets and symbols for each file. After that, it stores their hash codes, sizes, and paths in the data base as shown in the figure 3.

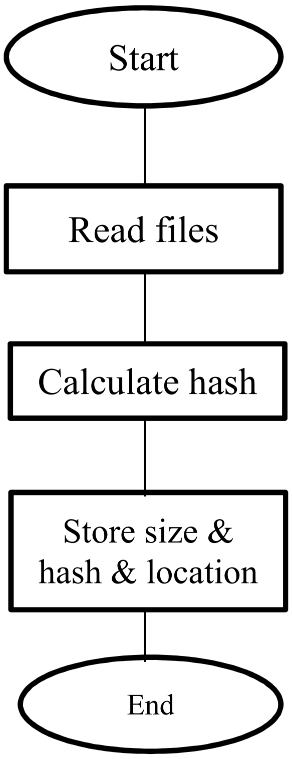

Figure 3: Preprocessing stage

b) Processing stage:

In this stage, we tend to divide this process into the following steps:

I. Input file**:**Here we choose the file we need to check whether it is attacked or not.

II. Calculating the file hash (using "SSdeep algorithm"): The system passes the file to SSdeep algorithm to produce a hash code that consists of alphabetical characters and symbols.

III. Hash existence check:

The system searches in the database for the hash code and compare it to the input file hash code. If the hash code of input file is similar to the current file hash code, we make sure that the file is not attacked because there is, at least, one file with the same size and hash, so no alarm is on this file. Otherwise, it calculates the file size and stores it in (T). Then (T) equals " file size *0.35". Afterwards, it creates a file probability range that tends to add T to the input file size and subtract T from the input file size. In addition, it does a small procedure that creates a range of files (zero or more files), based on the target file size. After that, it starts searching in the result list (zero or more files). If it doesn't find any file within the specified files size range (+T,-T), the system will send an alarm to the user. But if it finds files, it calculates the percentage of similarity depending on extension and location.

If it has, at least, one file in the database which has an acceptable percentage of similarity; it will alert the user that the file has been changed without any alarm. Otherwise, it will send an alarm to the user because the file has been changed over the predefined allowed percentage of similarity as shown in figure 4.

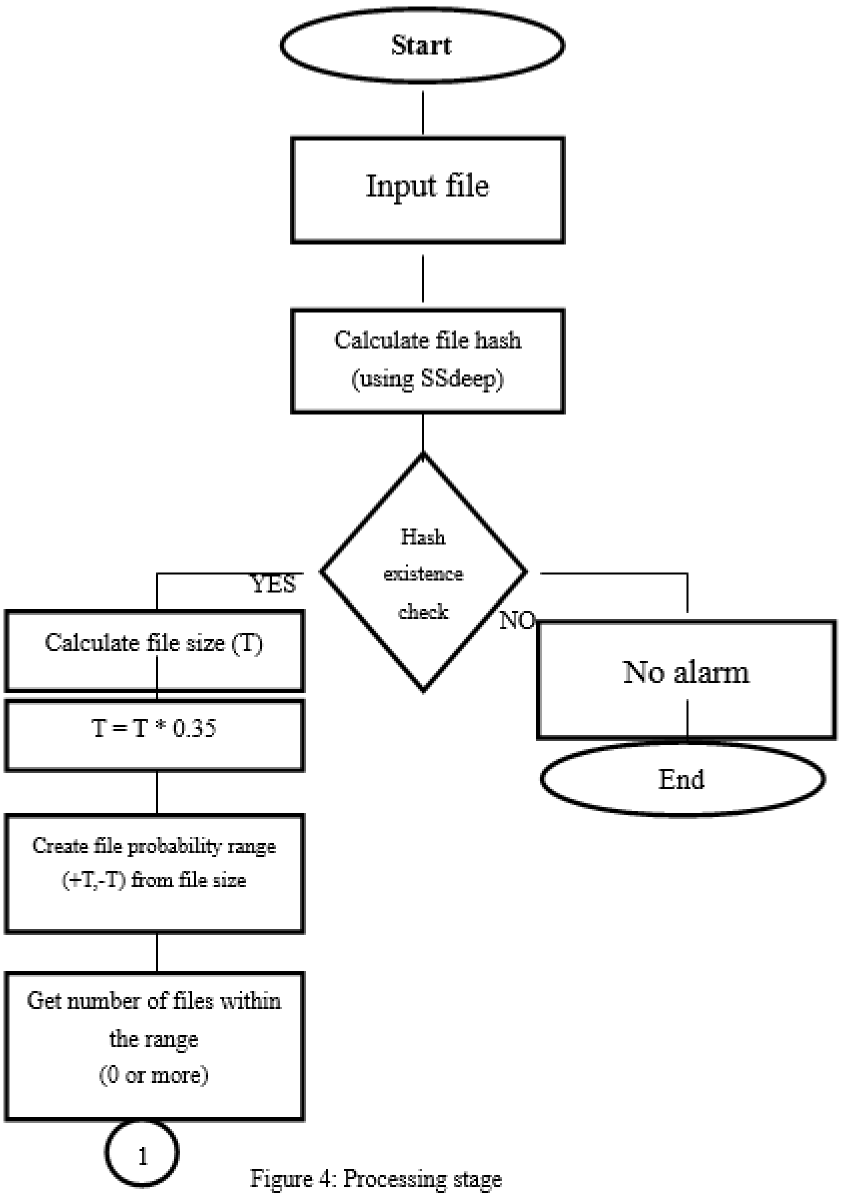
Start
Input file
Calculate file hash (using SSdeep)
Hash existence check
YES
NO
Calculate file size (T)
T = T * 0.35
Create file probability range (+T,-T) from file size
Get number of files within the range (0 or more)
1
No alarm
End

Figure 4: Processing stage

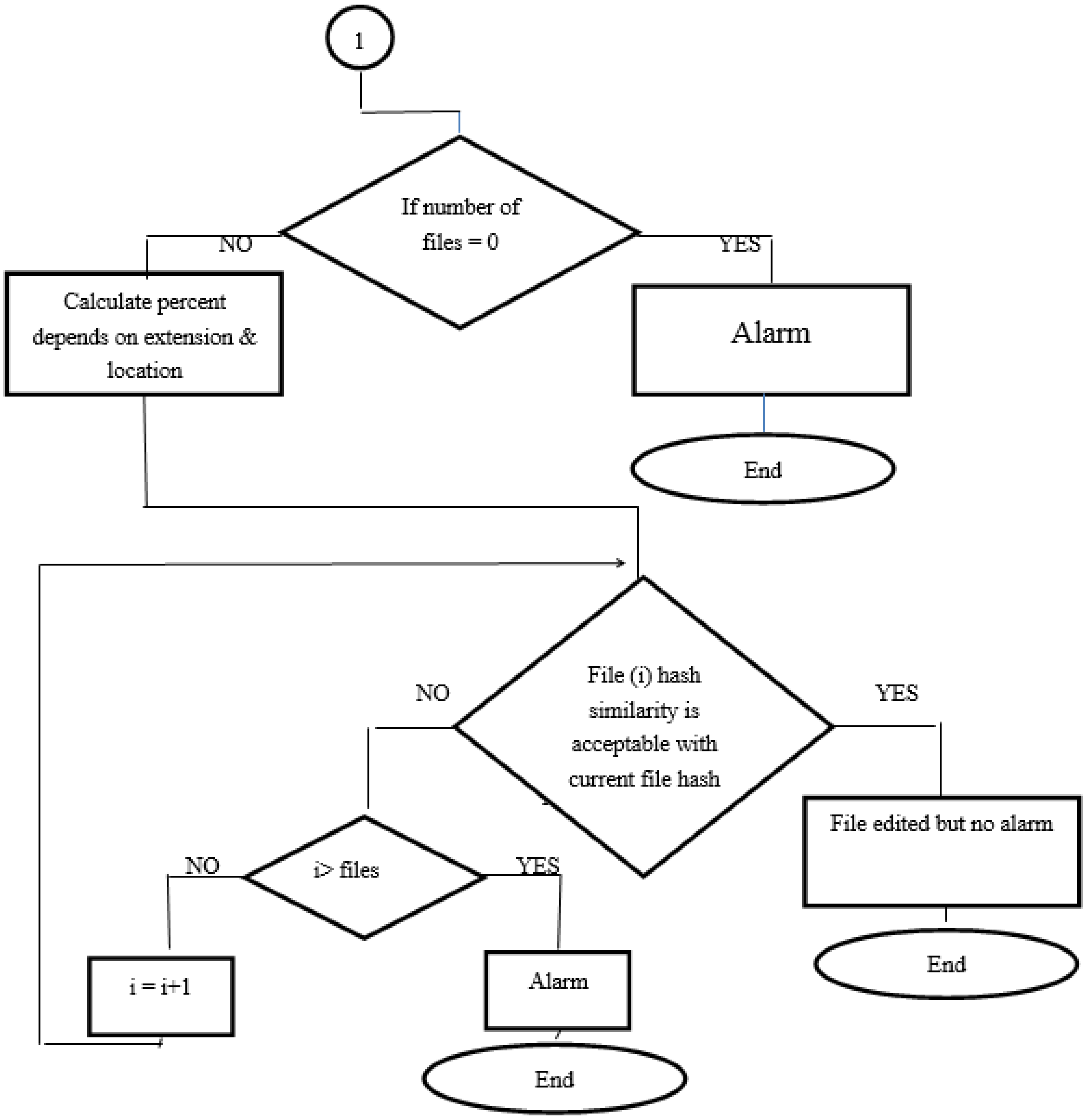
1
If number of files = 0
NO
YES
Calculate percent depends on extension & location
Alarm
End
File (i) hash similarity is acceptable with current file hash
NO
YES
File edited but no alarm
End
i> files
NO
YES
i = i+1
Alarm
End

Figure 4: Processing stage (Cont.)

5. Results

Assuming we have four folders each has minimum percent of similarity as shown in table 4.

Table 4: Folder priority

Folder name	Score
A (Critical)	>= 0.95
B (Very high)	>= 0.85 and <0.95
C (High)	>= 0.75 and <0.85
D (Low)	>= 0.65 and <0.75

Here five experiments are made, in experiment one we took twenty five files and we let the system go through the preprocessing stage to store the size and the hashes, then we change six files which have been intruded, then we checked the whole files with MD5 algorithm to check whether these files have been hacked or not, and how many files have been hacked. As a result, six files have been hacked. Then when we move the files to folder (A) and check them with SSdeep algorithm, the system finds five files have been hacked. After that, we move them to folder (B) and check them again with SSdeep algorithm. As a result, four files have been hacked. Then we repeat that with folders (C and D) and the results are similar to the ones shown in table 5 and figure 5. In the second experiment, we increased the number of files to one hundred, and change fifteen files. Then we checked them with MD5 algorithm and then with SSdeep algorithm while being in folder (A) then folder (B) until folder (D) and the results are similar to the ones as shown in table 5. Then, we repeat the whole procedures with the third experiment until five, taking into consideration, changing the total number of files and the number of the changed files as shown in table 5 and figure 6.

Table 5: Total alarms according to the number of changed files

Experiment number	Total files	Number of changed files	Total alarms of MD5 in all folders	Total assumed alarms in folder "A"	Total assumed alarms in folder "B"	Total assumed alarms in folder "C"	Total assumed alarms in folder "D"
1	25	6	6	5	4	3	2
2	100	15	15	8	5	4	2
3	200	25	25	12	10	7	4
4	300	40	40	20	17	16	13
5	500	60	60	31	28	20	16

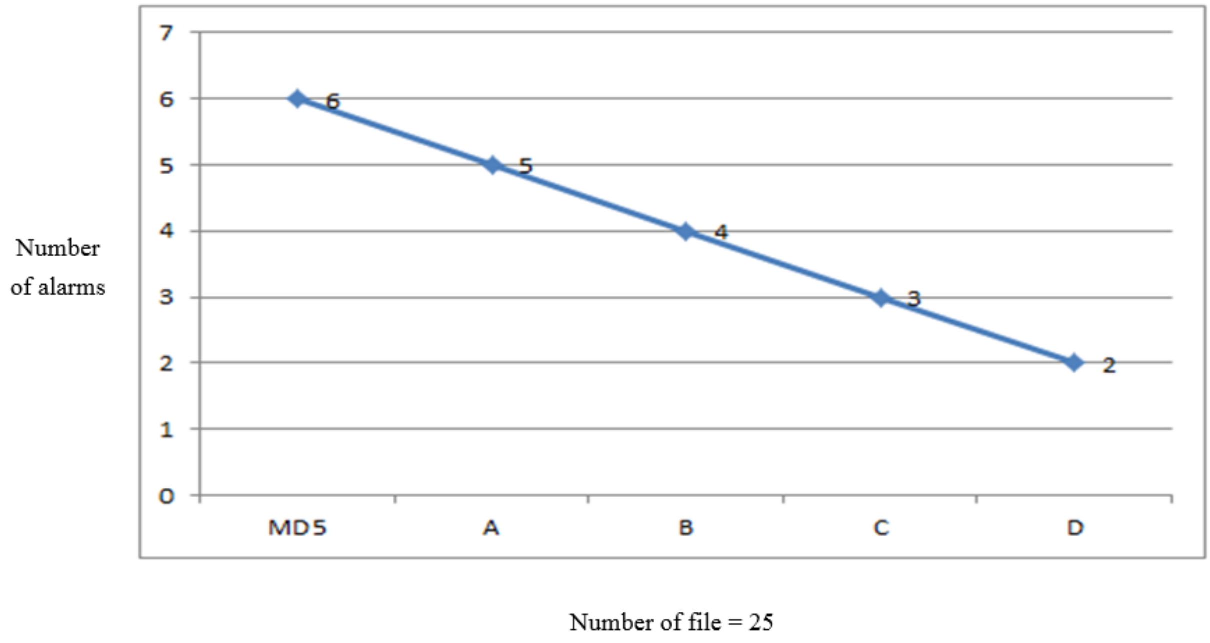

Figure 5: Number of alarms when number of file = 25

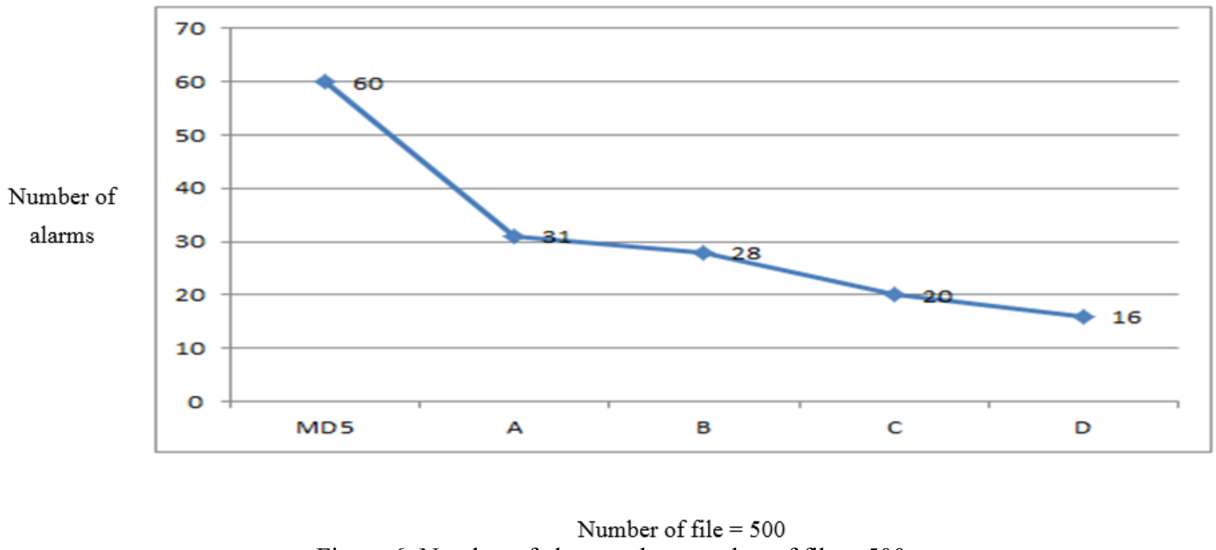

Figure 6: Number of alarms when number of file = 500

In addition, we can see in table 6 all the accepted files in folder (A) will be also accepted in the other folders (B, C and D). Accordingly, all the files accepted in folder (B) will be accepted in folders (C and D) but they will not be accepted in folder (A). While the files accepted in folder (C) will be accepted in folder (D) , but they will not be accepted in folders (A and B) . Finally, all the files accepted in folder (D), will not be accepted in Folders (B, C and D), Depending on the folder minimum of similarity.

Table 6: results for each folder

Location / μ	**A**	**B**	**C**	**D**
Critical	Acceptance	Acceptance	Acceptance	Acceptance
Very high	Avoidance	Acceptance	Acceptance	Acceptance
High	Avoidance	Avoidance	Acceptance	Acceptance
Low	Avoidance	Avoidance	Avoidance	Acceptance

6. Conclusion and future work

We can notice from the results, in the previous chapter, that as we increased the number of total files and the number of changed files, the MD5 algorithm will always give the same number of changed files, whereas the detected number of changed files will decrease, depending on minimum allowed percentage of similarity. As we can see I developed a simple graphical user interface for a program utilizing the SSdeep algorithm to allow a user manually to check if his files have been attacked or not with allowable setting for the minimum allowed percent of similarity for the files types or its locations. As a result the user can re-use some of the attacked files depends on the minimum of similarity.

Using such hashing, the examiners will be able to associate the previous lost files. In this way, the examiners, in investigating the homologous but not the identical files, find easily relevant materials in other investigations.

Regarding the matter of computer world and its revolution, this conclusion indicates the mentioned improvements and solutions discovered and found to detect the unallowable access done by several types of attackers and hackers. As a result, the attacked data do not seem to be useless, but the rate of change shows that the data can be reused or not. Therefore, I think by inventing new hashes or modifying the existed hashing algorithms, we can determine the exact portion of hacked file or increase the number of restored files.

References

Chyssler, Tobias, Stefan Burschka, Michael Semling, and Tomas Lingvall, KalleBurbeck, 2004. "Alarm Reduction and Correlation in Intrusion Detection Systems". In Detection of Intrusions and Malware & Vulnerability Assessment". Ulrich Flegel, Michael Meier (Eds) Dept. of Computer and Information Science Linköping University, S-581 83 Linköping , Sweden.

Kornblum, J. 2006, "Identifying almost identical files using context triggered piecewise hashing," Digital Forensic Research Workshop (DFRWS), vol. 3S, pp. 91–97.

Mallery, John 2008. "Network Intrusion Detection". Security Technology & Design. 44 – 47.

Pokrywka, 2008 Rafał. Pokrywka http://link.springer.com/chapter/10.1007%2F978-3-540-69384-0_45. Last visited Jan 2015.

Roussev, Vassil 2013 Simson Garfinkel, Frank Breitinger, John Delaroderie, Barbara Guttman, John Kelsey, Jesse Kornblum, Mary Laamanen, Michael McCarrin, Clay Shields, Douglas White, John Tebbutt, and Joel Young. The NIST Definition of Approximate Matching.Technical report, National Institute of Standards and Technologies.

Roussev, Vassil, 2011. An evaluation of forensic similarity hashes. Digital Forensic Research Workshop, 8:34–41.

Zadeh, L A .1965. "Fuzzy Sets", Information and Control, Vol.8, pp. 338-353.