Journal of Theoretical and Applied Information Technology
20th January 2016. Vol.83. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OPERATING SYSTEM INTEGRITY CHECK FRAMEWORK ALGORITHM FOR THREAT POSED BY ROOTKITS

DAVID MUGENDI, PROF. WAWERU MWANGI (PHD) DR. MICHAEL KIMWELE

School of computing and information Technology
Jomo Kenyatta University of Agriculture and Technology
P.O. Box 62000-00200 Nairobi Kenya

E-mail: david.mugendi@yahoo.com, dnthuni@yahoo.com

ABSTRACT

Kernel mode rootkits, KMRs have indeed gained considerable success as far as blackhat society is concerned raising much alarm to systems and system defenders. The danger posed by these rootkits has to some extent led to call for universal attention on the means to handle and deal with them. Rootkits have by far become more complicated and stealthy making it difficult to even detect their presence in the system using their susceptible methods. Bearing in mind the danger at hand posed by these rootkits to operating system and at large other computer systems, getting crucial information from already compromised system proves to be an uphill task. This thesis focused in addressing this problem. It focused on various techniques such as intelligent algorithm using neural networks technology to enable integrity checking for kernel mode rootkits. The research conducted also has described to some extent operating systems e.g. Linux kernel and some of the areas which are a common target by kernel rootkits. Virtualization technology was also introduced to enable readers understand some of the critical concepts. A number of requirements to be satisfied while addressing this issue have been outlined. A framework to implement the model has been set up to show how integrity check was achieved at the end of research.

Keywords: *Artificial Neural Network (ANN), Loadable kernel module (LKM), common object file format (COFF), Kernel mode rootkits, (KMR), probability mass functions (PMFs)*

1. INTRODUCTION

There is ultimatum danger posed by rootkits to computer systems in every day encounter. The overwhelming importance of remaining protected from the various threats presented by these rootkits such as cloaker cannot be underestimated. In particular, memory protection and integrity of the OS is of paramount significance hence the need to be more vigilant. Rootkits have been known to cause havoc and even force many organizations lose vital information rendering them obsolete. With this in mind, the call for this research work will play an important role in helping counter threat of malware in systems. Applications of ANN to computer systems integrity are a growing area of interest. Connection weights of links between neurons, and information are processed in parallel.

2. BACKGROUND OF THE STUDY

In order to surreptitiously control a compromised computer, an intruder typically installs software that tries to conceal malicious code. This software is commonly referred to as a rootkit. A rootkit hides itself and some malicious payload from the operating system, users and intrusion detection tools. The techniques utilized by rootkits to avoid detection have evolved over the years. Older rootkits modified system files and were easily detected by tools that checked for file integrity (Kim and Spafford, 1993) or rootkit signatures. To avoid being detected by such tools, rootkit designers resorted to more complex techniques such as modifying boot sectors (King and Chen, 2003) and manipulating the in-memory image of the kernel.

These rootkits are susceptible to detection by tools that check kernel code and data for alteration, (Perrigi, 2005). Rootkits that modify the system BIOS or device firmware ((al, Rootkits that modify system BIOS, 2004)) can also be detected by integrity checking tools. More recently, virtualization technology has been studied as yet another means to conceal rootkits (Hill et all , 2000). These rootkits remain hidden by running the host OS in a virtual machine environment. To counter the threat from these Virtual Machine Based Rootkits (VMBRs), researchers have

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

detailed approaches to detect if code is executing inside a virtual machine, (Kauer, 2007). Is this the end of the line for rootkit evolution? We believe that other hardware features can still be exploited to conceal rootkits.

Shadow (2005), exploits the existence of separate instruction and data address translation buffers to hide itself. While Shadow Walker exhibits some weaknesses that allow it to be detected by existing approaches, we aim to show that it is possible to construct a rootkit that exploits changes to hardware state for more effective concealment. Studying the construction of such a rootkit fuels the proactive design and deployment of new countermeasures. Similar approaches have been used in the past by other researchers, (Kim and Spafford, 1993).

Cloaker is an extremely stealthy proof-of-concept rootkit that exploits ARM processor features to avoid discovery. ARM processors power 90% of mobile handsets shipped today and five billion ARM processors have already been consumed by the mobile device market, (Kim and Spafford, 1993). With four billion mobile phone subscriptions expected by 2009 and an increasing deployment of ARM-based phones, it is essential that the threat posed by rootkits such as Cloaker to be carefully evaluated. In contrast, the number of worldwide PC users is expected to be a little over one billion in 2009, reaching the two billion mark only by 2015 (Chen et al, 2006. However, to the best of our knowledge, Cloaker represents the first exploration of an ARM processor specific rootkit. In this paper, we have demonstrated that most existing techniques used for rootkit detection are ineffective against Cloaker. Cloaker does not leave any detectable trace in the file system and is thus invisible to typical intrusion detection tools that scan file systems. Similar to VMBRs, Cloaker does not modify OS code or data. Therefore, it cannot be detected by integrity checks of the host OS. Cloaker tweaks hardware registers and settings in a manner that allows it to execute without interfering with the existing OS. More specifically, Cloaker exploits an ARM processor setting that allows the interrupt vector to be located in an alternate region of memory. It also exploits hardware functionality that allows several entries in the address translation cache to be locked down and not be automatically removed using the typical approaches used by operating systems to flush it.

3. MALWARE

Malware refers to malicious software which is used to disrupt the normal functioning or operation of a computer while gathering information through unauthorized access to computer systems. There are different forms in which a malware can present itself to the computer such as other software, scripts, code as well as active content. Different examples of computer systems malware exist and may include the following; adware, worms, scareware, horses, spyware, viruses, Trojan among others. From previous statistics done in 2011, most of these active malware which posed a greater threat included Trojan. Sometimes malware can be found in programs supplied by legit companies but mostly they are downloaded from websites which appear attractive to users but with a hidden agenda of gathering information on marketing statistics. A good example is the Sony rootkit, a Trojan which is embedded on CDs sold by the company to prevent unauthorized copying of the discs sold Sony.

Rise of widespread broadband internet access malware has been much often used for all beneficial reasons. For example, since 2003 majority of these software have been designed to control user computers and gather information about companies and other personal information of interest. A good example is the zombie computers used to send spam emails to host contraband data such as child pornography or engage in distributed DOS [denial of service] attacked as a form of extortion. There is greater risk posed by all kinds of malware software such as ransomware which demands payment of some sort to reverse the process or damage caused to the computer system. Cryptolocker encrypt files securely and only decrypt themselves on payment of substantial amount of money which directly implies losses to the company at question.

4. PROLIFERATION

Results from various publications released highlight how the use of malicious code has become commonplace. For instance in 2008 the results from Symantec suggested that the release of unwanted programs had by far exceeded that of legit software applications. This is by far a much outcry as the use of spam emails and World Wide Web has become more rampant fostering more desire of encapsulating attacks in much unwarranted manner.

Characteristics of malware in a system

Presence of malware in a system can be identified by some characteristics which they exhibit in a computer. This however may not be an easy task

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

since malware are highly deceptive and conceal their presence to avoid being detected. However, we have highlighted some of the various properties attributed to malware as follows;

Loadable kernel module (LKM); due to advancement in malware and rootkit detection in the recent past, malware developers have developed more technical mechanisms of hiding their presence in the system. Loadable kernel module are the most highly deceptive malware as they are loaded during the boot process thus providing a window for them to be loaded to the system without detection. Most anti-virus tools use signatures extracted from rootkit bodies (e.g. file or network packet). In addition, there are heuristic and behavioral patterns based on certain activities typical for rootkits (e.g. an aggressive use of a certain network port on the computer).

Network sniffers; the nature of malware in a system exhibited by this feature is to gather information about the system and its environment stealthily and relay back some of this information to the attacker. Such is highly commonplace in networked systems where the attacker eavesdrop data packet and modifies them in a manner which the user isn't able to notice.

Log editors; this property of malware enables them to hide the presence of the attacker and give him the privilege to perform backdoor activities without the user realizing the presence of the attacker. Such is critically dangerous as the attacker is able to modify, edit and even delete files from the system.

Backdoor; such a property enables the attacker to regain access to the system and perform file modification as well as sniffing on password and other credential details which gives them access to the system at any given time without the knowledge of system administrators.

Auditing programs; this used to exist in the old system such as ps/netstat for UNIX. Such are able to record important information about a system and relay information to the attacker. A good example is keeping track of users' passwords and user name which the attacker uses to access such a system from time to time.

5. ALGORITHMIC METHOD

1) NEURAL NETWORK

There are various real life applications of neural networks which instigated the idea of bringing into board their usability. One good example of this is data processing including filtering, clustering, blind source separation and compression. Citing from the nature of malware once their get access to the system they undergo various phases before reaching their execution stage. Through these stages it's possible to distinctively detect irregularities in files or documents as may be depicted by changes in file extensions. As a result, understanding the powerful role played by neural networks in pattern recognition opens the avenue to adopt this great feature and employ it in providing the solution to the question at hand. The following is the lifecycle of a malware in a system;

Dormant phase; once they pass various authentication procedures, malware remains dormant in the host system for some time then proceeds to the propagation phase.

Propagation phase; during this stage, the malware does replicate to avoid detection and also increase chances of attacking the system.

Triggering phase; it can be triggered by presence or absence of a certain file or record a particular day of week, a particular user running the application.

Execution/damage phase; during execution stage malware Performs a disguised malicious function, e.g. destroy or alter data, change file permissions, trick user into typing password, copy or transmit data over covert channels

2) Number Of Layers And Neurons

The system implementation has been carried out in 2 phases. The first phase performs the feature selection procedure by taking input from the files which make up the training data. Phase 2 performs the training of the neural network and then classifies a file as either legitimate or virus infected.

Phase1

In Phase 1, the Training Data, which comprises of two types of executable files-legitimate and virus infected, is given as input to the Feature Extractor. The Feature Extractor takes a one feature (PE Structure field) at a time from all files present in the Training Data, and sends it to the Fisher Score Implementer, Eric Filiol et al, 2006. This in turn evaluates the most optimum or relevant features, using Fisher Score technique. It assigns a rank to each feature which will be stored in the Database. Finally, the Feature Selector selects the 'M' most relevant features based on their rank. It then provides them as input to Phase 2

Journal of Theoretical and Applied Information Technology
20th January 2016. Vol.83. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

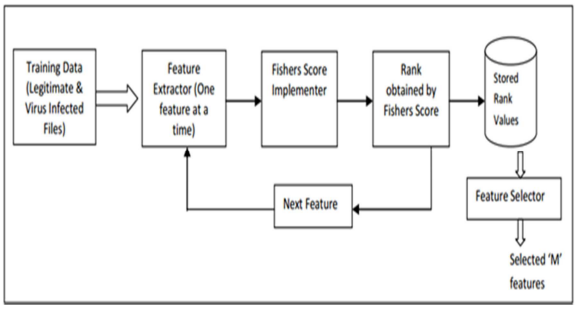

Figure 1; Phase 1 Feature Selection

Phase 2

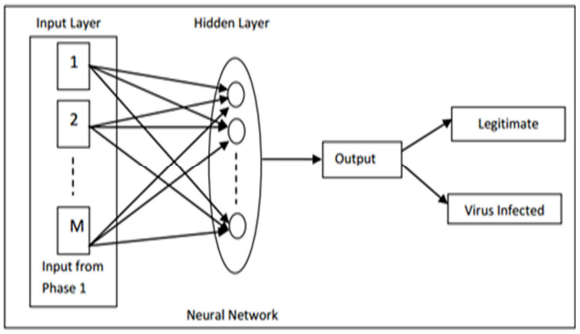

Figure 2; Training And Classification Of .Exe File

The Input from Phase 1, which comprises of the 'M' most relevant features, is given as input to the neural network, Isabelle Guyon, 2003. Using these features, the neural network trains itself, and then whenever a file is given as input, it will classify it as either a legitimate or a virus infected file.

3) Training The Algorithm

To distinguish legitimate data from virus infected as the neural network algorithm has been trained using supervised learning where the network is trained by providing it with input values and matching output patterns. These input-output pairs can be provided by an external teacher or by a system. This means presence of malware is detected as the algorithm classifies inputs into either legitimate or malware. After the classification, we have shown what data is malware and which isn't using the confusion matrix.

6. MODELING FILE ATTRIBUTES AS DISCRETE RANDOM VARIABLES

We have modeled each file attribute as a discrete random variable by mapping each plausible value of an attribute to an integer outcome. Using the benign and malware datasets, for each random variable we have computed histograms of each outcome. By normalizing these histograms, we obtained the probability mass functions (PMFs) of the attribute random variables. The outcomes to integer mappings for some representative random variables are enumerated in Table 1 below.

The first three attributes in Table 1 are checked for their presence only, that is whether value of these attributes is present (non-zero) or absent (zero).

The file content section is the byte distribution. Much emphasize is in *Table 3* which lists all the file attributes that are used for classification in this section. Using this mapping and our dataset, future studies can compare accuracy of their results with the technique used in this work.

Table 1; Possible Values For Attribute Random Variables

Attribute	Possible outcomes	Condition
Timestamp	2	Value is zero OR non-zero
File content	3	Indeterminable,≤4>4
File size	5	≤500KB,≤1000KB,≤2000KB,≤3000KB, ≤4000KB,>4000KB
Symbol attribute	3	{Wininet,ws2_32,wsock32,raspi}

7. RESULTS AND ANALYSIS

In this section we have showed and compared results and performance of neural network algorithm based on various attributes to show the improvement it brings forth as well as the overall performance of the algorithm in conclusion.

To determine various attributes of malware from our data sets, we examined two kinds of data sets. We referred to these two as source data where one is the zoo population which represents malware corrupted data from vxheaven delfi rootkit which incorporates a combination of worms, Trojans, viruses, spyware, and network intrusion related files (password dumpers, covert channels, etc. collected while responding to confirmed network intrusion events). All files in this population are Win32 files with target platforms of MS Windows Server 2008, MS Windows XP, and MS Windows 7. We called this the "known bad" set, henceforth referred to as the zoo set. The other small sample set or the known good set which we shall refer to as the control set Win32 files extracted from a number of MS Windows XP, MS Windows 7, and MS Windows Server 2008 systems. The reason for the smaller size of the control set is largely because there is little variance in PE32 files from system to system for common application and operating system files. Additionally, Copywrite issues prohibit the collection and sharing of executables from valid applications.

Journal of Theoretical and Applied Information Technology
20th January 2016. Vol.83. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Attributes

The approach adopted in the analysis of these data sets involved extracting several attributes and comparing the two so as to establish any changes which might be exhibited. To do this, we used data mining rules to extract specifics which guided in drawing our conclusion from a series of attributes. These attributes were primarily extracted from the file's PE32 headers (Microsoft Corporation, 2010), (Pietrek, 1994) with the addition of a few whole file attributes such as file entropy and file size. For completeness purposes, we picked four major attributes for analysis with our model as they yielded much significance to our case study. Other attributes such as sizeOfHeader attribute, SectionAddress attributes did not provide significance detection rates.

Timestamp; examining date of file creation which is automatically set during the file creation compile time can be interfered with by the malware. This can be modified using specialized tools indicating a malicious or abnormal behavior, [ligh, Adair 2011]. We check for the field within our two data sets and establish changes in dates using a specified rule. For example we select for all years >2012 thus detecting anomaly in data. This is demonstrated this using area graph below;

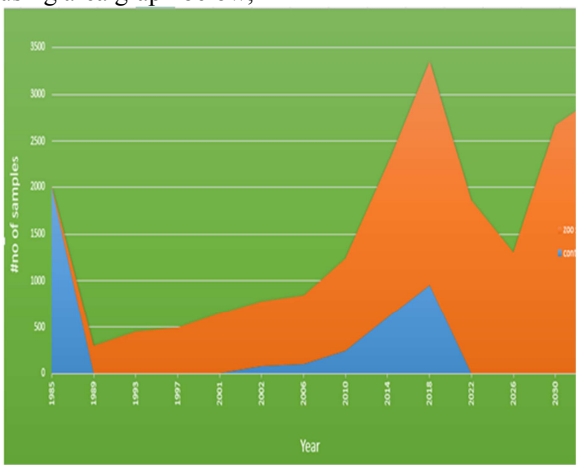

Figure 3; Distribution Of Samples By Year Of Timedatestamp

We can notice various deviations when we look at the above figure comparing how the zoo set datasets compare to the control set. A noticeable point is where the zoo set greatly deviates from the control set giving a sharp spike. One practical use of this approach could be to apply this method to a smaller more targeted subset of samples as an aid in discovering attacker TimeZone, Country of Origin, and periods of activity.

File size; we compared the range in the two data sets in terms of file size and establish if there is any anomaly. Normally each process in the file is allotted some reasonable resources to execute it. If the size of a given file grows without know how of resource allocator of the operating system then it becomes obvious that it has been tampered with from a malicious source. A certain known range is compared for both the zoo data set and the control data set and see if there is any deviation from the norm.

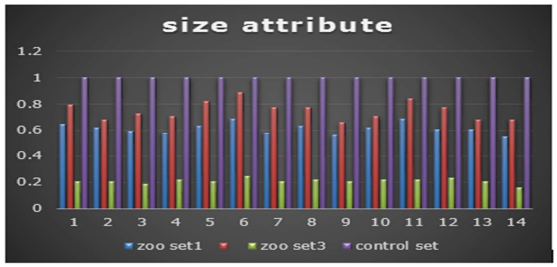

Figure 4; Size Attribute Chart

From the above figure 2 we can notice some anomalies on the size of data and how it various from one zoo set when compared with the control set. For example, *zoo set1* represents approximately 65% while the control set is at 100% giving a deviation of 35%. This is a big anomaly and thus we are able to establish interference of the file size. The malware will try to take some small size to avoid detection but with this kind of analysis we are able to establish the maliciousness in the data.

File content;

In PE32 files, sections divide the file content between code, data, resources, and various types of variable and configuration data. While there are a vast number of section types and section names possible, in practice most non-malicious PE32 files use a small number of sections. As shown in Figure 3 and Table 2 most of the control set samples has between 1 and 8 sections. When comparing this range against the zoo set two interesting deviations are noted. First, figure 3 shows a spike of zoo set samples where the NumberOfSections value is set to 7.

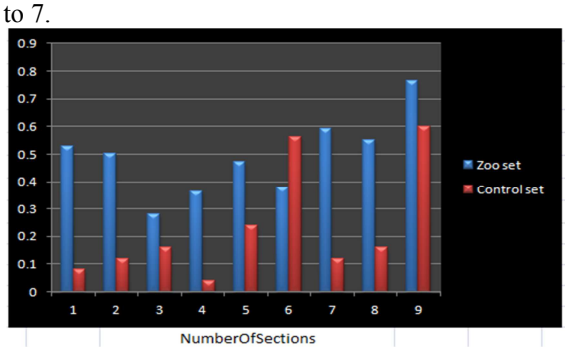

Figure 3; File Content Attribute Chart

Journal of Theoretical and Applied Information Technology
20th January 2016. Vol.83. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2; File *Content Deviation Between Two Samples*

Number of sections	Zoo set	Control set
1	0.5288	0.08
2	0.5026	0.12
3	0.2793	0.16
4	0.3618	0.04
5	0.4704	0.24
6	0.3785	0.56
7	0.5907	0.12
8	0.5504	0.16
9	0.7643	0.6
n…+1		

Symbol Attributes

The pointerToSymbol and NumberOf Symbol fields define the location and size of the COFF [common object file format] debugging information (Pietrek, 1994).
A value of zero specifies no debugging Information was included during compile time. In the Majority of cases the control samples should not contain debug information fields set to zero. The lack of debug information is driven by the deprecation of COFF [common object file format] debugging in favor of PDB [program database] files (Glaister, 2007) and the common practice of stripping debugging symbols for production files. Table 3 Shows the control set, indeed has a very small population (0.29%) of samples having a PointerToSymbolTable Value greater than zero. When examining the same field within the zoo set a seven-fold increase (2.06%) is observed in the occurrence of samples that have a non-zero value for this field. One could argue that this value alone would make a good detection rule (*PointerToSymbolTable>0*).

Table 3; Symbol Attributes Deviation

Rule	Zoo set	Control set
PtrToSymTable>0	2.06%	0.29%
PtrToSymTable>=Size	1.48%	0.03%
PtrToSymTable>=2xSize	1.46%	0.00%

Improved detection rate of the In fact the data shows a 76.59% deviation between the control set and the zoo set for this value of NumberOfSections. This deviation may be useful when considered in conjunction with other potential indicators considering this value alone as an indicator of malicious effect.
algorithm is demonstrated using a confusion matrix to show how the classification of these files is done once all the relevant attributes as discussed above are selected. The confusion matrix is an indicator of what percentages of the samples used have been classified as either benign or bad shown in the figure below;

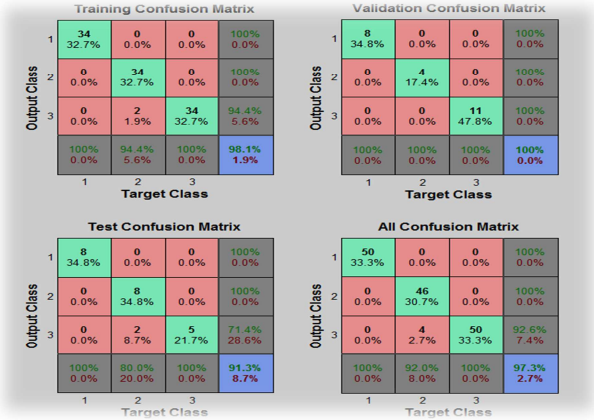

Figure 6; Confusion Matrix

A confusion matrix classifies dataset as legit or malware, based on uniformity of timestamp, file size, file content, symbol attributes. The diagonal cells show the number of cases that were correctly classified, and the off-diagonal cells show the misclassified cases [malware]. The blue cell in the bottom right shows the total percent of correctly classified cases (in green) and the total percent of misclassified cases (in red). The results show very good recognition.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

7. CONCLUSION

The above malware attributes have been discussed and compared with some benign to identify if there are any disparities and consequently be able to detect presence of malicious activities in a system. These factors are leading reasons that detection based on file attributes provides an excellent supplement to the normal Anti-Virus detection regiment and have a potential for a much longer life span than traditional detection signatures. This will eventually help in keeping our system safe and thus prevent any form of data modification from external sources. This is the ultimate objective of this research to help improve on the integrity of our system by ensuring that any anomalies in systems are detected. The use of neural network facilitated achieving the core objective the research of improving the integrity of systems through the ability to detect changes made within a kernel through pattern recognition providing secure and isolated environments for execution of untrusted code. One of its main advantages is that it provides the detection of mismatch in system sequence thus upholding integrity of the data.

8. FUTURE WORK

Analyzing the ability to discover changes; the ability to discover any changes made to a monitored system, needs to be further explored. This can only be done through thorough analysis and testing of the capabilities of the integrity checker.

Looking at the possibilities of new virtualization technologies; AMD and Intel will within near future incorporate support for virtualization into their processors. Incorporating these into the framework will probably allow a higher degree of isolation.

Improving integrity checking technologies; The use of integrity checking is expensive with regards to system resources. Therefore, developing more efficient integrity checkers is important.

Analyzing the isolation and transparency capabilities of the framework; The ability to provide isolated and transparent environments is critical to the success of the detection system. An analysis revealing the systems weaknesses should be conducted to allow further improvements of the system.

Extending the suggested model; currently, the model focuses on the use of integrity checking as its main detection mechanism. However, the model is open for extensions through the implementation of detection modules. Therefore, developing such detection modules is important to improve the model. These modules may contain a number of functionalities, looking at and analyzing the existing tools would provide basis functionalities to be incorporated into such modules.

REFRENCES:

[1] Spafford, E.H.: "The Internet Worm Program: An Analysis" Tech. Report CSD-TR-823. Department of Computer Science, Purdue University (1988).

[2] kephart J.O Arnold, WC:"Automatic extraction of computer virus signatures" in 4th virus Bulletin International Conference, 1996

[3] Lo, R.W., Levitt, K.N., Olsson, R.A: MCF: "a malicious code filter". Compute. Secure. **14**(6), 541–566 (1995)

[4] A.Rajavelu, M. T Musavi and M.V Shirvaikar, A Neural Network Approach to Character Recognition" Neural Networks, Vol. 2. Pp. 387-393, 1989.

[5] B CJohnson, D. (2013). "Publication on Malware and Antivirus Systems for Linux operating system".

[6] Garfinkel, T. and Rosenblum, M., (2003), "A virtual machine introspection based architecture for intrusion detection, In *Procedings of the Net- work and Distributed Systems Security Symposium*, pages 191–206, 2003.

[7] Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, (2005). Pioneer: "Verifying Code Integrity and Enforcing Untampered Code Execution on Legacy Systems", In *Proceedings of the twentieth ACM Symposium on Operating Systems Principles*, pages 1–16, New York, NY, USA,. ACM

[8] [Skou2003] E. Skoudis, "Fighting malicious Code", Prentice-Hall. ISBN-0131014056, 2003

[9] Sullivan, D., (2009) "*The Shortcut Guide to Prioritizing Security Spending",* USA: Real-time Publishers

[10] [BC2002] D. Bovet; M. Cesati, "Understanding the Linux Kernel", 2nd Edition. O`Reilly,ISBN-0596002130, 2002.

[11] J.B. Ekanayake and N. Jenkins, "A Three-Level Advanced Static VAR Compensator", *IEEE Transactions on Power Systems*, Vol. 11, No. 1, January 1996, pp. 540-545.

[12] A.Rajavelu, M. T Musavi and M.V Shirvaikar, A Neural Network Approach to Character Recognition" Neural Networks, Vol. 2. Pp. 387-393, 1989