

Running head: THE DETECTION OF UNKNOWN LINUX ATTACKS

A New Mechanism to Detect Unknown Linux/IOS Attacks

Student's Name

Institutional Affiliation

Copyright © 2008 AcademicWritersBureau.com. All Rights Reserved.

If you need an original copy of this writing feel free to contact us at admin@academicwritersbureau.com

Introduction

Generating Payloads using MetaSploit and MSF Venom

MSF Venom is a command line tool that will help the testing team to generate stand alone payloads to execute on vulnerable machines and systems to remotely access the systems. MSF Venom is a combination of MSF Payload and MSF encode which support various options in testing system vulnerabilities. Some of these options availed by the MSF Venom include payload use which describes how to use custom payloads for the tests, list the various payloads standard options, outline the module type which may include payloads, encoders, determine the length size on the payload, output format, list the available formats (Almgren, 2000).

The MSF Venom also requires the penetration testers to determine the type of encoder to use, the system architecture, the platform where the payload will operate, the maximum size of the resulting payload, maximum size of the encoded payload, list of characters to avoid during the tests, the number of times to encode the payload, specify a custom executable file to be used as template and preservation of template behavior and payload injection. Below is a snapshot of the MSF Venom various options available for the penetration testing phase.

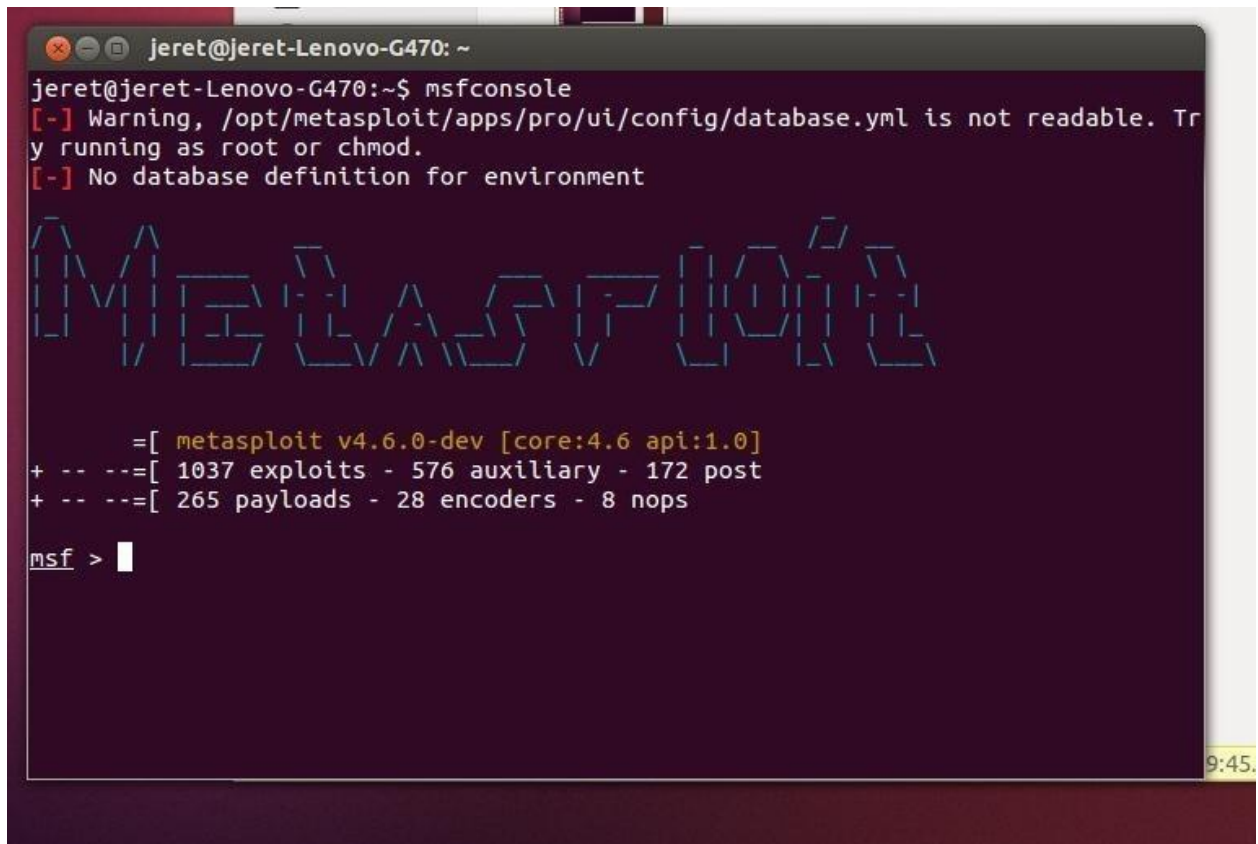
```

root@0SA-Kali-09:/# msfvenom
No options
MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /opt/metasploit/apps/pro/msf3/msfvenom [options] <var=val>

Options:
  -p, --payload <payload>      Payload to use. Specify a '-' or stdin to use custom payloads
  --payload-options             List the payload's standard options
  -l, --list [type]            List a module type. Options are: payloads, encoders, nops, all
  -n, --nopsled <length>      Prepend a nopsled of [length] size on to the payload
  -f, --format <format>        Output format (use --help-formats for a list)
  --help-formats                List available formats
  -e, --encoder <encoder>      The encoder to use
  -a, --arch <arch>            The architecture to use
  --platform <platform>        The platform of the payload
  -s, --space <length>         The maximum size of the resulting payload
  --encoder-space <length>     The maximum size of the encoded payload (defaults to the -s value)
  -b, --bad-chars <list>       The list of characters to avoid example: '\x00\xff'
  -i, --iterations <count>     The number of times to encode the payload
  -c, --add-code <path>        Specify an additional win32 shellcode file to include
  -x, --template <path>        Specify a custom executable file to use as a template
  -k, --keep                    Preserve the template behavior and inject the payload as a new thread
  -o, --out <path>             Save the payload
  -v, --var-name <name>        Specify a custom variable name to use for certain output formats
  --smallest                    Generate the smallest possible payload
  -h, --help                    Show this message
root@0SA-Kali-09:/#

```

This image above shows us the ruby language MSF Venom



```
jeret@jeret-Lenovo-G470: ~  
jeret@jeret-Lenovo-G470:~$ msfconsole  
[-] Warning, /opt/metasploit/apps/pro/ui/config/database.yml is not readable. Try running as root or chmod.  
[-] No database definition for environment  
  
Metasploit  
  
=[ metasploit v4.6.0-dev [core:4.6 api:1.0]  
+ -- --=[ 1037 exploits - 576 auxiliary - 172 post  
+ -- --=[ 265 payloads - 28 encoders - 8 nops  
  
msf > |
```

The screenshot shows a terminal window with a dark background. At the top, the window title is 'jeret@jeret-Lenovo-G470: ~'. The user has entered 'msfconsole' at the prompt. The output shows a warning about a database.yml file and a message about no database definition. Below this is the 'Metasploit' logo in a stylized, blocky font. Further down, the version information is displayed: '=[metasploit v4.6.0-dev [core:4.6 api:1.0'. This is followed by two lines of statistics: '+ -- --=[1037 exploits - 576 auxiliary - 172 post' and '+ -- --=[265 payloads - 28 encoders - 8 nops'. The prompt 'msf >' is shown at the bottom, with a cursor. In the bottom right corner of the terminal window, the time '9:45' is visible.

This is a language used to explain metasploit program. It can use any language is the default language.

This is a language used to explain metasploit program. It can use any language is the default language.

It is important to safeguard and protect the system from any cyber based threat when developing and running computer systems and applications. A sensitivity analysis is a complex study of uncertainties in the output of a computer system or application. Sensitivity analysis involves testing various assumptions and tests to determine the level of uncertainty and the impact of a variable on the system output (Denning, 2012).

This study is important to determine the robustness of the system by testing uncertainties, increase understanding of the application or system to identify vulnerabilities and strengths of

the system or application. Sensitivity analysis will also help the system and application development team to identify uncertainties and vulnerabilities thus help the development team to reduce these uncertainties, errors and vulnerabilities. It also helps in identifying parts of the model input which does not affect the output, thus help in simplifying the system model (Feng, 2004).

The study will also help to enhance communication between the development team and decision makers to exchange ideas and make recommendations to improve the system before it is released to the public. The study also identifies input factors and impacts on system output to determine the optimal measurements and sensitive parameters within the system which can influence the final system (Forrest, 2010).

The purpose of this research paper is to test a system for uncertainties and errors to reduce vulnerabilities that may expose the system to breach through cyber threats. Some of the attacks to be tested include unknown and known attacks. Known threats and attacks are basically attacks well identified by signatures on anti-virus and intrusion detection system engines and domain reputation blacklists. Unknown attacks on the other hand are new with no recorded attack signature and thus pose a more serious threat since their mode of operation is unknown (Ghosh, 2007).

In light of this, the paper will examine how to utilize the false positive and false negative concepts to test for sensitivity analysis. A false positive also called a false alarm is a test result that indicates that a given condition has been fulfilled. This is a type I error which checks a single condition e.g. the presence of code errors in the system with the results either true or false.

The false negative on the other hand is a test which indicates that the results in the condition being tested failed yet the actual result is successful. This is a type II error where a single condition is tested with the result either being positive or negative (Ilgun, 2011).

One of the applications to be used in this paper is the MSF Venom application and Metasploit tools to conduct penetration tests on the target system or application. The MSF Venom is a command line tool that will help the system developers to generate stand alone payloads on vulnerable systems to remotely access the system. Metasploit on the other hand is a testing tool that enables application penetration testers to circumvent anti-virus solutions that enables the testing team to generate payloads, test network segmentation and increase performance and productivity of the system by reporting system features and enabling automatic updates. This paper will examine penetration tests to be conducted to test for vulnerabilities in the systems, identify types of errors and attacks and recommend ways to improve the productivity of the system and optimize productivity (Javitz, 2010).

Known and Unknown Attacks

Known cyber attacks are malicious attacks on computer systems and applications which are already known to the penetration testers and whose signature has already been identified. Known attacks begin as unknown and become known only when they have been identified as malicious through automatic or manually in vendor labs and a signature assigned to them. Most of the current dynamic analysis solutions use sophisticated metrics to detect if a generated payload is malicious with predictive algorithms applied to characterize the payload activity (Klein, 2005).

The unknown attack is just a new attack without a signature. The difference between a known and unknown attack is a difference of time, since with time the unknown becomes known. The attackers file will simply be recognized by search engines as malicious thus creating the need to identify a payload that is unknown to the search engine. This will involve purchasing of available search engines and running malicious files until a hacker finds a means to circumvent the search engines and make the malicious file undetectable (Ko, 2011).

The design of secure systems that can withstand cyber attacks is not only hard but very complex. In the past designers used to construct locks that are very strong through placing strong mechanisms of security around their systems. In the recent past designers have changed and are now constructing systems that are still in operation in spite of continued attacks through leveraging on techniques that are well developed through fault tolerance as well as dependability (Kruegel, 2003).

Meta-sploit

The exploit concept is usually broken into several elements. It first gets connected to vulnerability. Upon getting connected the vulnerable code gets the payload data. The vulnerability is then exploited by the payload and a shell kind of component is created. Depending on the kind of conveyed payload the component of the shell could be reconnected to the handler that is usually generated before delivering the payload. Upon establishing the connection the shell can receive commands to extract information from the targeted machine. It could also make an extension of its functionality through the delivery of the code (Lane, 2010).

In offering the flexibility of the delivery of various kinds of shell payloads the exploit elements tends to employ the generic interface that is offered by the payload element. The various executions of the payload interface offer optional shell payloads that are likely to be distributed to the vulnerability. The discovery of vulnerabilities and the development of exploits is a challenging task that calls for various mindsets as well as motivation. As much as fizzling tools are present in Metasploit frameworks in the discovery of vulnerabilities there is a lot of time in which the Metasploit will be employed in exploiting known vulnerabilities which is its primary function (Lee, 2000).

It is not a must for vulnerabilities to be present in the network and given that networks have become very secure administrators and even users that are not trained are usually the targeted vulnerable element. Upon the identification of vulnerable users, the delivery of exploits could be done through phishing attacks or in person through using a recruited insider. All metamodel elements are perceived as facades. For instance the connect elements for Metasploit a generic framework has API calls being offered to make connections with almost all renown services on OS (Liberty, 2002).

Payload elements offer the exploit framework with codes that could be implemented upon the success of an exploit. However, handlers do not function as payloads but for reasons of execution they are part of the payload hierarchy of class. To find a wide detailed control of targeted networks there is a need to employ payloads that are capable of spawning server kind of elements on targeted machines. In addition their capability needs to be able to be extended whenever this is needed (Liljenstam, 2003).

To obtain this kind of functionality there is a need to incorporate Protocol as well as the attacker side command as well as management elements. A number of these elements can be observed in the Metasploit framework. The UI façade elements offer interfaces that permeate the attacker to manage how a number of payloads as well as handlers could be attached to the exploit elements before its launch. These façade elements include managers, controllers, proxies, views, command processors, and commands. Data store elements have interfaces that permeate the attacker to store an attack's configurations details (Lindqvist, 2001).

Type I, And Type II Error

The Type 1 error is also referred to as the false positive this is so because it tends to reject the null hypothesis even when it is true. It essentially does accept an optional hypothesis when outcomes are subjected to chance. It tends to happen when observing disparities where there is none. The Type II error is also referred to as the false positive. This is so because it tends to reject the null hypothesis when the optional hypothesis is deemed as nature's true state. It is an error where one fails to accept an optional hypothesis because of being inadequate in power. It happens when individuals fail to observe disparities even though there could be one (Mahoney, 2002).

The testing of hypothesis is the process of testing if disparities in two sampled distributions could be explained by random chance or not. If conclusions can be made that there is a meaningful manner in which two distributions vary there must be adequate precaution to perceive that the disparities are not by random chance. Type I errors do not allow unwarranted hypothesis consequently individuals take precaution in reducing occurrence chances.

Conventionally attempts are made to set Type I errors as 0.05 or 0.01 where there is only one or five chances out of 100 of the occurrence of a phenomena as regards the degree of significance. However, it is not guaranteed that 0.05 or 0.01 is adequately rare there is thus a need to cautiously choose significant levels. Multiple testing which is the likely increment in Type I errors happens whenever there is a repeated use of statistical errors (Paxson, 2008).

New Mechanism That Detects Unknown Attacks to Make Them Known

It is not possible to construct an incursion tolerance system that can survive for a meaningful quantity of time without dealing with unknown attacks challenges as well as restricted failover resources. It is prudent to harden one's system so as to appreciate the work factor of the adversary. However, the time necessary for identification of emerging vulnerabilities in any system as well as the development of its exploit could be somehow large (Almgren, 2000).

To an opponent that is determined this could be about time as well as money. Upon the development of an attack it requires limited time to be executed. In most cases there is limited time for the creation of simple attack variants. In case the threat environment of an intrusion tolerant system entails adversaries that are well resourced the system should be able to deal with a lot of attacks that are unknown (Denning, 2012).

Linux servers

Linux has become a popular choice for operating systems in the server environment. The granularity as well as suppleness of settings, security, high performance as well as reliability are

its merits compared to other systems. Given infrastructural limitations numerous services are hosted on one server thus realizing the challenge of how the Linux server could be protected. The Linux server protection practice cannot be a onetime ordeal rather it is a permanent mechanism that endures so long as the server is being used. The objective is to increase security and quickly detect challenges (Feng, 2004).

Upon installation of the Linux server it is critical that unnecessary services are disabled. Unnecessary packages should also be removed. The biggest threat to Linux packages stems from unsafe software packages employed in the execution of a number of commands which come from remote locations. They include the R commands such as; *rsh*, *rexec* and *rlogin*. These software packages tend to transmit commands, user names and passwords through networks in the absence of previous recording. Upon intercepting the traffic the attacker can see the information which could pose a critical security challenge. It is thus required that these packages are deleted from the Linux server and employ protocols which make use of encrypted communication for instance the SSH.

The administrator of the server should make cautious considerations on whether it is critical for a number of protocols for instance the File Transfer Protocol and Telnet to possess the client as well as the server support. It is important that files from isolated FTP servers are downloaded. The running of an FTP server that is not being used is a critical threat to Linux servers (Forrest, 2010).

The administrator of Linux servers must institute Linux operating system which offers secured transfer of files to enhance the package updating as well as the application of pertinent

patches. All data that goes through these networks must be encrypted by the administrator where feasible. The administrators of systems have been using the SSH as well as the Telnet protocols in their remote access of the Linux server. However, Telnet is now absolute because it never encrypts data that it exchanges with servers and this includes the credentials of users. Any person in this network between the computer administrator as well as the communicating server is capable of intercepting packets and possessing user credentials thus getting complete control of the server. The SSH protocol is thus preferred over the Telnet protocol for these reasons.

The SSH protocol provides communication protection in the server through provision of asymmetric fundamental cryptography. The communication between SSH servers and clients tends to be encrypted and inexplicable to third parties that could intercept exchanged packets. The authentication of the SSH server could be attained through the encrypted transmission of user credentials. This could also be avoided through using manually created asymmetric keys where there is no necessity of employing passwords (Ghosh, 2007).

In case the authentication is done by keys that are generated manually without the transfer of passwords the authentication key should be I the device that can access the Linux server. The employment of user credentials is critical for administrators to log in to the server where distinct computers are employed for isolated log in to the server. For communication and coordinated authentication between two Linux servers it is critical that manually created asymmetric keys are employed. This is so because there are permanent parties to this communication thus avoiding the requirement of writing passwords on the files.

Secure Channels of communication

If there is a need for the transfer of files between Linux servers and remote machines it should be done via secure channels of communication. The FTP protocol is very popular for file transfers. However, it faces similar security risks just as the Telnet protocol. More secure options such as SCP, FTPS, or SFTP are thus recommended. The Secure File Transfer Protocol (SFTP) tends to establish secure and encrypted communication using remote devices so as to enhance the access to, transfer as well as management of files. This protocol also employs an SSH tunnel in the manipulation of files in isolated servers (Ilgun, 2011).

The File Transfer Protocol Secure (FTPS) stems from the FTP founded on Transport Layer Security (TLS) as well as the Secure Sockets Layer (SSL) which enhance the secure transportation of data. FTPS is very much the same as the Hypertext Transfer Protocol Secure (HTTPS) protocol and demands a digital certificate on servers. Given that other parties to the communication should trust the mounted server digital certificate this could realize challenges in the execution of the entire solution. If the capability of the transfer is employed for the system needs as well as for purposes of server administration it is required that the SFTP protocol is employed given that it provides a lot of file management alternatives (Javitz, 2010).

In case files are transmitted from the Linux server as a service for its end users it is recommended to employ the FTPS protocol since it enhances the authentication of the server by use of digital certificates. FTPS tends to employ the UDP or TCP 990 as well as 989 ports to establish connections and transfer files. It is thus critical for these ports to remain open on Linux servers. The secure copy protocol (SCP) which employs SSH protocol guarantees the encryption as well as authentication of data that is encrypted. This process is the same as that used by SFTP

protocol. However, this one involves more capabilities than just transferring files. SCP employs the port 22 TCP to guarantee secure transfer of files.

The Linux server must have extra space to store data gathered during operations. Administrators tend to employ isolated systems of files in the expansion of their server capacities. Linux servers are usually connected to remote servers for storing data and using a part of the file system. Remote file systems are accessed via networks and system administrators have similar experiences as if the remote file systems are encrypted in the Linux server. The transfer of data between servers and isolated file systems should be sufficiently secured so as not to compromise the transferred data. It is recommended that system administrators should employ the Secure Shell File System (SSHFS) which enhances the employment of isolated systems of files on Linux servers. SSHFS employs the SSH link for securely transferring data in networks. Fundamentally SSHFS employs the SFTP protocol which enhances file management, safe access as well as data transfer on isolated file systems (Klein, 2005).

The Linux sever can only be effectively protected through the administrator being well informed. This can be done of he has a consistent flow of reliable information on security trends that concern pertinent Linux distribution as well as installed packages of software. A lot of vulnerabilities have been found over time in Linux software packages and they could be employed by attackers in compromising servers. Given mailing lists that tackle security challenges in Linux distribution administrators are able to act promptly through timely updating of the system as well as elimination of vulnerabilities. It is thus critical to employ secure mailing

lists in Linux distributions. Founded on gathered information administrators could decide when the system can be updated as well as software packages that could be employed on the servers.

The *Yum* package manager has been recommended for *apt get* or Red Hat distributions for it guarantees the efficient installation as well as updating of software packages and pertinent dependencies. It also enhances effective deletion of software packages. Package managers tend to repeatedly check the a package's digital signature and avoid installation if such packages have invalid signatures. Administrators must employ the standard software repositories that have been suggested by vendors. Software packages could also be retrieved from several repositories but this calls for extra precautionary measures. It is prudent for administrators to verify the new versions of packages before updating them to assess their stability as well as any likely security challenges. Installations should only happen to software that is absent in official repositories. These packages could also be obtained from a number if repositories but with extra care. Unofficial software packages in a testing face should not be installed on production Linux servers (Ko, 2011).

Recording a list of packages that are installed as well as their past versions is believed to be the best practice. This is so because the administrator would be able to access the latest steady versions. If new packages are a threat to the server's stability the administrator could obtain the versions installed previously and reinstall them. Administrators could set the system to make an automatic update of software packages though this is not usually recommended. The best practice is to make a mutual verification of each update of packages.

The administrator could also put together the package managers to send intermittent notifications of emails where he can list all packages that could be server updated. Upon a review of these packages it is upon the administrator to make a decision on the separate update of each package. The interrupted sending of emails through the *Yum* package manager can tend to be configured in the configuration file (Kruegel, 2003).

It is critical to resume the *Yum* updated service upon making changes to the configured file. Administrators that employ Debian distributions are commended in the installation as well as the employment of the *apticron* package of software in order to guarantee routine checks in the update of packages as well as email notifications. In the configuration file for the *apticron* package software it is prudent to enter the email address where the notification will be sent. In case the administrator is adequately experienced in defining the packages required on the servers it is perceived a refined practice to come up with a local repository in such a network. The repository of known as well as stable package versions could be generated in one Linux server where other network servers could retrieve the packages. Such solutions demand detailed checks of all packages before their storage in a local repository.

Linux servers tend to multitask through offering several services in one instance. This leads to the server being economically utilized as well as optimum efficiency. The system administrator must however, continue observing security's golden rule. This ensures that the system is as secure as the highly venerable services in it. The perfect solution is obtained when each Linux server offers only one service to the final users (Lane, 2010).

There are a number of tools that could be employed to enhance security. They curb unauthorized access to malicious attempts as well as services that could conciliate the Linux server. Linux has several inherent security tools for instance *SELinux* and *Netfilter*. Netfilter is a function that usually intercepts as well as processes packets of networks. This could be found in about all Linux systems that have 2.4 and above versions of kernel.

A New Mechanism to Detect Unknown Linux/IOS Attacks

Web founded vulnerabilities provide considerable portions of computer networks security exposures. To guarantee the detection of known attacks that are found in the web misused detection systems tend to be equipped with a lot of signatures. However, it is not easy to be upfront with day to day disclosure of vulnerabilities that are web related. Apart from that vulnerabilities could be introduced through installation specified web founded applications. This means that misuse detection systems should be assisted by anomaly detective applications (Lee, 2000).

This paper proposes an intrusion detective system that employs a distinct anomaly detection method in the detection of Unknown Linux/IOS Attacks. This system tends to assess various client queries which have a reference for a number of server programs and generates models for a wide range of distinct features for these questions. Some of these features include the serve side programs assess patterns as well as the values for each parameter in its invocation. Particularly the employment of application particular attributes of the invoked parameters permeates the system to undertake a concentrated assessment and generate a minimized number of false positives.

This system tends to automatically derive the parameter profiles that are linked to web applications for instance their length as well as their structure and the association between queries. This means that it is capable of being deployed in various applicable environments without the necessity of time consumed tuning or configuration. The anomaly detection application presented in this paper tends to accommodate in form of input web servers web files that have a conformity to the Common Log Format (CLF) which generate a score that is anomaly in nature for every web request (Liberty, 2002).

More concisely, the assessment methods employed by this tool capitalize on the specific HTTP structure queries that have parameters. Such queries access patterns as well as their parameters tend to be compared with instituted profiles which are particular to the program or any program active in nature that can be referenced. This strategy tends to enhance a highly concentrated assessment when it comes to the detection methods of generic anomaly that do not account for particular programs that are invoked.

Anomaly detection depends on models of users intended behaviors as well as applications and tends to interpret deviations from ordinary behavior as malicious activities evidence. The assumption is that patterns of attack tend to differ from ordinary behavior. Anomaly detection presumes that the disparity could be expressed both quantitatively and qualitatively. The proposed detection strategy evaluates HTTP requests in the same way they are logged by most ordinary web servers. The assessment concentrates on requests that employ parameters which pass values to active documents. More so the detection process input is made up of a set of URIs

that is ordered $U = \{u_1, u_2, \dots, u_m\}$ and extracted from effective GET requests; their return codes tend to be more or equal to 200 but should be less than 300 (Liljenstam, 2003).

The URI u_i is expressed in terms of elements of the desired resource path, an alternative component of path information (pinfoi), as well as an alternative string of query (q). The query string is employed in passing parameters to referenced resources where it is pointed out by leading “?” characters. A query string is made up of ordered lists of n pairs of attributes alongside their analogous values. In this case $q = (a_1, v_1), (a_2, v_2), \dots, (a_n, v_n)$ where $a_i \in A$, is a set of attributes while v_i is a string.

The assessment procedure concentrates on the relationship between values, programs and parameters. URIs lacking query strings are considered irrelevant and thus expunged from U . Apart from the URIs there is the partition of U into U_r subsets. Consequently, all referenced programs r are assigned to corresponding queries U_r . The process of anomaly detection employs several distinct models in a set of various models to point out anomalous entries. A model can be said to be a set of mechanisms employed in assessing specific query features. This feature could be associated with single query characters (Lindqvist, 2001).

In light of the approximated query feature distribution of length with l and r_2 parameters it is up to the detection phase to evaluate a parameter anomaly whose length is l . This is done through calculating the length l distance from μ the length distribution mean value. This can be expressed using the Chebyshev inequality. Only strings whose length exceeds μ are perceived as malicious. This is mirrored in the probability calculation since the strings upper bound is longer compared to the mean and thus relevant. The categorization of intrusion detection strategies

into misuse and anomaly based tends to be orthogonal in regard to the specific method employed to attribute malicious or ordinary attacks. In some cases signatures are employed in the specification of users projected behavior (Mahoney, 2002).

Conclusion

Unknown Linux/IOS Attacks must be dealt with using tools and techniques that comprise of the precision of signature founded exposure with anomaly founded intrusion flexibility detection system. This paper has proposed a novel strategy of performing anomaly detection of Unknown Linux/IOS Attacks. It employs as input HTTP queries that are made up of parameters. It is the pioneer detection anomaly system modified to detecting Unknown Linux/IOS Attacks. It capitalizes on application specified correlations between Linux server programs and parameters that are employed in their invocation. Ideally this application does not call for any installation particular configurations. It also learns its query attributes from the training of data. However, the degree of its sensitiveness with anomalous data could be configured through thresholds that can suit various site policies.

References

Almgren, M. H. Debar, M. Dacier, (2000), A lightweight tool for detecting web server attacks, in: Proceedings of the ISOC Symposium on Network and Distributed Systems Security, San Diego, CA,.

- Denning D.E., (2012), An intrusion detection model, *IEEE Transactions on Software Engineering* 13 (2) 222–232.
- Feng, H. J. Giffin, Y. Huang, S. Jha, W. Lee, B. Miller, (2004). Formalizing sensitivity in static analysis for intrusion detection, in: *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA,.
- Forrest, S. (2010), A sense of self for UNIX processes, in: *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, pp. 120–128.
- Ghosh, A.K. J. Wanken, F. Charron, (2007), Detecting anomalous and unknown intrusions against programs, in: *Proceedings of the Annual Computer Security Application Conference (ACSAC_98)*, Scottsdale, , pp. 259–267.
- Ilgun, R.A. Kemmerer, P.A. Porras, K. (2011) State transition analysis: a rule-based intrusion detection system, *IEEE Transactions on Software Engineering* 21 (3) 181–199.
- Javitz, A. Valdes H.S., (2010), The SRI IDES statistical anomaly detector, in: *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA,.
- Klein D., (2005), Defending against the wily surfer: Web-based attacks and defenses, in: *Proceedings of the USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, CA,.
- Ko, C. M. Ruschitzka, K. Levitt, (2011), Execution monitoring of security-critical programs in distributed systems: a specification-based approach, in: *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, pp. 175–187.

- Kruegel, C. D. Mutz, W.K. Robertson, F. Valeur, (2003), Bayesian event classification for intrusion detection, in: Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003), Las Vegas, NV,.
- Lane, T.C.E. Brodley, (2010), Temporal sequence learning and data reduction for anomaly detection, in: Proceedings of the ACM Conference on Computer and Communications Security, San Francisco, CA, ACM Press, New York, pp. 150–158.
- Lee, W. S. Stolfo, (2000), A framework for constructing features and models for intrusion detection systems, ACM Transactions on Information and System Security 3 (4) 227–261.
- Liberty, D. J. Hurwitz, (2002), Programming ASP.NET, O'Reilly, Sebastopol, CA.
- Liljenstam, M. D. Nicol, V. Berk, R. Gray, (2003), Simulating realistic network worm traffic for worm warning system design and testing, in: Proceedings of the ACM Workshop on Rapid Malcode, Washington, DC, pp. 24–33.
- Lindqvist, M. Almgren, U. (2001), Application-integrated data collection for security monitoring, in: Proceedings of Recent Advances in Intrusion Detection (RAID), Davis, CA, October 2001, LNCS, Springer, , pp. 22–36.
- Mahoney, P. Chan, M. (2002), Learning nonstationary models of normal network traffic for detecting novel attacks, in: Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining, Edmonton, Alberta, Canada, pp. 376–385.
- Paxson, V. (2008), Bro: a system for detecting network intruders in real-time, in: Proceedings of the 7th USENIX Security Symposium, San Antonio, TX.

