NetReconner: An Innovative Method to Intrusion Detection using Regular Expressions

Radhika Gupte[1], Pranav Mundhe[2], Kiran Tonpe[3], Shruti Agrawal[4]

[1]B.E. Information Technology, Dept. of Information Technology, Vidyalankar Institute of Technology, Maharashtra, India
[2] B.E. Information Technology, Dept. of Information Technology, Vidyalankar Institute of Technology, Maharashtra, India
[3]B.E. Information Technology, Dept. of Information Technology, Vidyalankar Institute of Technology, Maharashtra, India
[4]Professor, Dept. of Information Technology, Vidyalankar Institute of Technology, Maharashtra, India

---

Abstract - *Intrusion detection is an essential part of Information Security, which plays a vital role in securing a network or a system from attacks on the confidentiality, integrity and availability of the system. An Intrusion Detection System (IDS) is a tool used to monitor the network with the purpose of detecting an attack on the system, which then reports the network administrator about the potential damage, so as to take appropriate measures. Network Intrusion Detection System (NIDS), a category of IDS, focuses on detection of network attacks and aims at securing the entire network instead of securing a single host. This paper describes complete working of NetReconner system. NetReconner is a Regular Expression (RegEx) based NIDS that captures packets from the network and compares the set of RegEx with each line of the captured packets. This system monitors the network continuously using a bash script. It uses a tool called as tcpdump (a command line tool in Linux). The output of tcpdump (the captured packets) is stored in a text file. The bash script, at regular intervals, also calls the detection engine which is programmed in Python3. The detection engine uses a set Regular Expressions and compares them with each line in the text file containing the captured packet. NetReconner also provides a facility for the administrator to add new RegEx for newly discovered network attacks to the existing set of RegEx. This paper elaborates the mechanism of NetReconner and discusses the results obtained using it.*

***Key Words*: Intrusion Detection System (IDS); Malicious packets; Network attacks; Network security; Packet capture; Regular Expressions (RegEx); Signatures.**

1. INTRODUCTION

In the current era, larger volumes of data go through and over the Internet every day. This makes the Internet a potential pathway to intrude a network. Malicious users take advantage of this pathway and carry out network attacks on a target system or network. This can cause loss of huge amounts of data or information and can prove disastrous when it comes to sensitive Government information, data related to financial sector and other such critical information. To avoid such circumstances, network security has become a necessity in most organizations and institutions. The most common way attackers harm computer systems, is by using malicious software, called malwares. Malwares are a piece of software designed for performing unethical activity to gain access to another individual's resource or to spy on them. Malwares can also disrupt the functioning of the complete system. According to the statistics report from Kaspersky's Security Bulletin 2016 [1], 31.9% of user computers were subjected to at least one Malware-class web attack over the year. Unauthorized users in a network pose a threat to the system where they perform network attacks to compromise the system or the services provided by it over the network. These attacks are data packets containing malicious codes in their payloads. These malicious codes can be detected using regular expressions. A regular expression (RegEx) is a sequence of characters used to search a specific pattern in a text file or for input validation under different applications. NetReconner is an IDS that makes use of RegEx for pattern matching with the captured packets' data looking for a possible attack. As stated in Guide to Intrusion Detection and Prevention Systems (IDPS) [2], an Intrusion Detection System (IDS) is a software that automates the intrusion detection where the administrator monitors events occurring in a computer system or network and analyses them for signs of possible incidents that may qualify as violations or threats of violations of computer security policies.

1.1 Detection Process

Capturing of packets from over the network is done using a simple Linux command 'tcpdump'. Tcpdump can also be referred to as a tool for scanning the network for capturing and analyzing data packets and has multiple real-time applications. The tcpdump command is triggered by a bash script at fixed intervals of time to enable continuous packet capture. The packets captured are stored in and appended into a text file on which the Detection Engine carries out the pattern matching algorithm. The Detection Engine is also triggered by the same bash script that triggers tcpdump command causing minimal down time of the Engine and enabling batch processing of the packets captured meanwhile. If a line in the text file matches with a particular RegEx from the existing set, an alert is generated identifying the attack that has been occurred. NetReconner also provides facility to the network administrator to add new RegEx to the existing set, manually through a user interface.

2. LITERATURE SURVEY

Through this section, we summarize some of the existing research work on intrusion detection systems.

In January 2016, Akash Garg et al. [3], has proposed a system to improve performance of an existing intrusion detection system. They have used a dataset to detect attacks using Snort. Snort is a well-known network intrusion detection system that audits network packets and compares them with attack signatures. The complete performance analysis of Snort is presented in this paper. Snort identifies attacks even when the entire packet is not available to it. On the other hand, there is a drawback in this system that Snort generates false alerts or false alarms in considerable amounts.

In May 2017, Abdullah H Almutairi1 et al. [4], has proposed a solution to the problem of huge database size. Signature based intrusion detection systems often suffer from large number of signature patterns stored in database. The idea of frequent-signature database is used to decrease the database size, also resulting in improving the performance of the system. This generates an alert very quickly after a signature is triggered. In addition, the concept of parallel processing is used with two small databases. This will minimize the detection time. When the packet goes through a network, it gets parsed by the databases for matching. The scanning should be done on a separate machine. The proposed system has four components: detection engine, small databases to store the most frequently used signatures, updating agent, and a complementary database that stores the remaining signatures. Detection engine captures an incoming packet, then decodes it and extracts its features which could also be considered as the packet signature. This is compared with the signature database. If the database contains a match, the engine will trigger an alert by displaying it. It also logs the incident, disables the connection by forbidding all succeeding packets from coming from that source. The second component is the small databases containing the most frequent attack signatures. The size of these databases is smaller, compared to the complementary database and this database can be adjusted according to network load or administrator decision. The third component is the updating agent, used to update the small databases with the most frequent signatures and remove those signatures, either that were not used over a time period or related to a malware which depends on a vulnerability that has been predefined. The fourth component is the complementary database containing the remaining signatures. This database is larger in size than the small databases and is used to detect infrequent attacks. This module updates the small database module over time. The proposed solution sets a priority for each added signature in database. These priorities are 'high', 'low' and 'medium'. These are like labels to the signatures. The reason for assigning a priority to a signature is to know if a specific signature is likely to occur in the network or not. High priority signatures are kept in the smaller database, while the complementary database stores low priority signatures. Network administrator decides what to do with medium priority signatures.

In November 2012, Prof. D. P. Gaikwad1 et al. [5], has proposed Multithreaded design detection module. It is a single process that decides whether a captured packet is malicious or not. A single process works well when there is not much traffic in the network. However, if the network is flooded, it will slow down the detection speed or there is a possibility of missing a potential attack due to dropping of extra packets. To deal with this problem they used the multithreaded design. This design keeps a two-variable count. One variable is used to track the number of captured packets and another variable keeps count of threads created by the detection engine. The first variable can take some maximum value that a single thread can handle at a time for processing. When that value exceeds, a new thread will be created to handle further packets. The concept of an agent is used in their paper to describe the working of the system. Components of agents are a frequent-attack signature database, a detection engine and an updating module. In the first component, incoming packets are checked against attack signatures. This process is time consuming. To overcome this flaw, they have used cache mechanism for frequently triggered signatures, decreasing the response time. Another module is the detection module that takes a packet as an input and extracts its signature. This extracted signature is then compared with all the signatures in cached database to check for intrusion. If a match is found then that packet is labelled as a malicious packet. The work of the updating module is to keep the frequent-attack signature set up to date. A variable is associated with each attack. That variable is incremented whenever an attack signature is triggered. The concept of DIDS (Distributed Intrusion Detection System) is used here. A distributed IDS uses multiple detection modules over a large network. All these modules communicate with each other and send updates to a central server. This makes the task of network administrator and analysts easy and manageable. This allows them to update distributed servers using single central server.

Osaghae [6] also proposed a similar model that decreases the size of the main signature database. The model consists of components such as a detection engine, a family malware reduction buffer database engine, a scalable malware signature database and a temporary database called the reduction buffer database. The working of these modules is similar to the other modules explained before.

3. PROPOSED SYSTEM

In order to detect malicious activity in a network, there is a need to monitor the network continuously. To achieve this, capturing the packets flowing on the line is necessary. This is done using TCPDUMP, a packet capture tool which is available in all Linux and Unix based operating systems. It is often used to help troubleshoot network issues and is also a security tool. It is the most powerful and versatile tool that includes many options and filters by using the 'libpcap' library to do this task. After capturing all data packets, the result is stored in a simple flat file or a text file. The amount of data packets in flat files depends on the activity performed on the web. Then, the detection engine parses all the data packets using Regular Expressions which are stored in flat files based on attacks. Regular expression is a library in Python programming language.

Regular expression library or RE specifies a set of strings that matches it. The functions in this module are used to check if a particular string matches with a given regular expression or not. This flat file is used by Detection Engine for parsing the data packets to generate alerts and log suspicious activities.

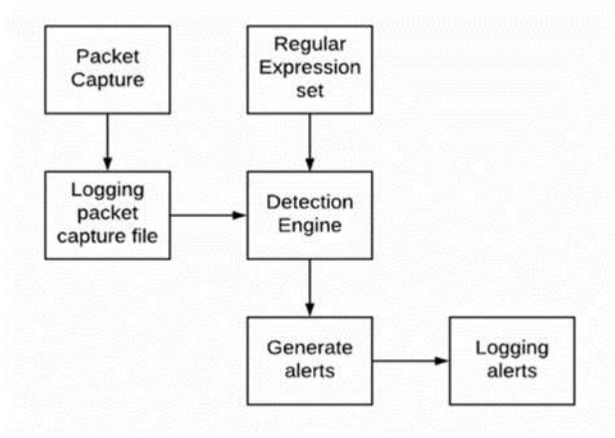


Fig 3.1: Block diagram of NetReconner

Detailed explanation about modules in Fig 3.1 is given below.

1. Packet capture Module: TCPDUMP command/tool is used for capturing data like header and payload in this module. It captures all major protocol packets like TCP, UDP and DNS requests. Following command is used for packet capture:

```
tcpdump -nnvvSs 1564 -A
```

2. Rules or RegEx set: A set of rules, written in regular expressions that a detection engine uses to find typical intrusive activity. These rules are stored in a specific repository and are fetched to compare with captured data.

3. Detection engine: It consists of a Python script which is used to check if a particular string matches with any of the given regular expressions. Parsing through all data and then applying Regular Expression set on each packet takes place in the Detection Engine.

4. IMPLEMENTATION AND RESULTS

4.1 Implementation

The first module deals with packet capturing and generating alerts for designated attacks carried out in a network. Here the first module logs network traffic in a flat file in order to find malicious packets in it. Detection engine continuously parses through, the logged packets so that the system remains active for all the time without allowing any infected packets to escape. Now the generated alerts are appended in different files based on the attack type. This separates out the bad traffic so that the network administrator can analyze the generated alerts. Detailed packet information can tell a lot about the attacker. This is how the system can backtrack an attacker.

In the second module, a graphical user interface (GUI) is rendered. This is done for appending a new RegEx in a particular file which is not included before. This module is administrator triggered. If an attack is done using a different approach for which the RegEx is not present in the existing files, then the administrator can add a new RegEx for that particular attack in a particular file mentioned by the administrator. These RegEx files are simultaneously accessed by both modules. Hence, there is a possibility of deadlock in the operating system. This problem is solved by creating a temporary file for appending RegEx entered by the administrator and then adding that RegEx to the main file when module one execution is completed the next first time in a cycle. Thus, there is no need to halt the detection engine for alert generation. This makes the system real-time.

Flow chart for NetReconner system is shown below:

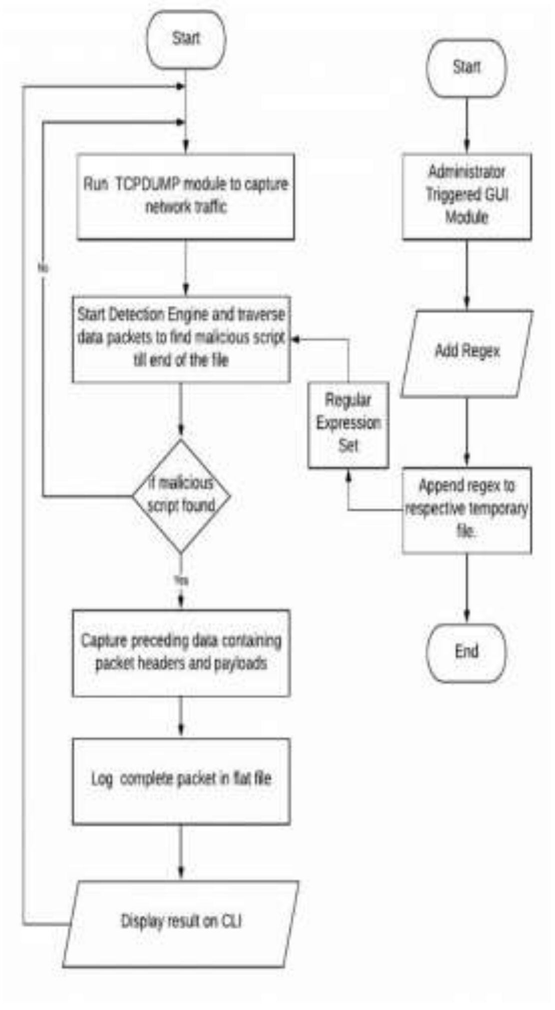
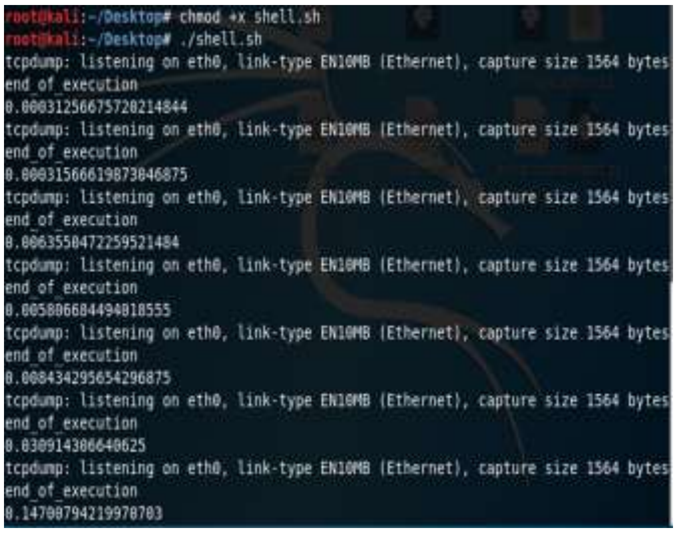

Start
Run TCPDUMP module to capture network traffic
Start Detection Engine and traverse data packets to find malicious script till end of the file
If malicious script found
No
Yes
Capture preceding data containing packet headers and payloads
Log complete packet in flat file
Display result on CLI
Regular Expression Set
Start
Administrator Triggered GUI Module
Add Regex
Append regex to respective temporary file.
End

Fig 4.1: Flowchart for working of NetReconner

As shown in Fig 4.1, NetReconner comprises of two modules. Each module acts independent of each other without being interrupted by another one.

4.2 Results

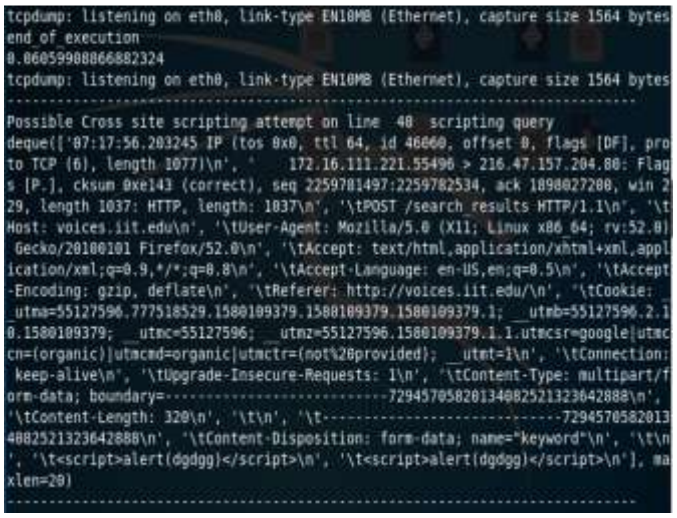
Tests were conducted to check the performance of NetReconner's Detection Engine. During the test, the Detection Engine was deployed on an Ubuntu machine equipped with Intel Core i3 CPU, 64-bit OS and 8 GB RAM. To launch the attacks, Kali Linux was used with inbuilt tools like Nmap, Xerxes and Hping3. Following are some of the snapshots taken while recording the results.



```
root@kali:~/Desktop# chmod +x shell.sh
root@kali:~/Desktop# ./shell.sh
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1564 bytes
end_of_execution
0.00031256675720214844
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1564 bytes
end_of_execution
0.00031566619873046875
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1564 bytes
end_of_execution
0.0063550472259521484
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1564 bytes
end_of_execution
0.005806684494018555
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1564 bytes
end_of_execution
0.008434295654296875
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1564 bytes
end_of_execution
0.030914306640625
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1564 bytes
end_of_execution
0.14700794219970703
```

Fig 4.2.1: System before attack detection

Fig 4.2.1 shows that the packets are captured using tcpdump command and detection engine starts parsing through all the captured packets. The "end_of_execution" shows that the detection engine has parsed all packets. Till this instant no attack has been detected and the execution time remains almost constant as a result of the same.



```
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1564 bytes
end_of_execution
0.06059908866882324
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 1564 bytes
-------------------------------------------------------------------------------
Possible Cross site scripting attempt on line  48  scripting query
deque(['07:17:56.203245 IP (tos 0x0, ttl 64, id 46060, offset 0, flags [DF], pro
to TCP (6), length 1077)\n', '    172.16.111.221.55496 > 216.47.157.204.80: Flag
s [P.], cksum 0xe143 (correct), seq 2259781497:2259782534, ack 1898027200, win 2
29, length 1037: HTTP, length: 1037\n', '\tPOST /search_results HTTP/1.1\n', '\t
Host: voices.iit.edu\n', '\tUser-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
 Gecko/20100101 Firefox/52.0\n', '\tAccept: text/html,application/xhtml+xml,appl
ication/xml;q=0.9,*/*;q=0.8\n', '\tAccept-Language: en-US,en;q=0.5\n', '\tAccept
-Encoding: gzip, deflate\n', '\tReferer: http://voices.iit.edu/\n', '\tCookie: _
_utma=55127596.777518529.1580109379.1580109379.1580109379.1; __utmb=55127596.2.1
0.1580109379; __utmc=55127596; __utmz=55127596.1580109379.1.1.utmcsr=google|utmc
cn=(organic)|utmcmd=organic|utmctr=(not%20provided); __utmt=1\n', '\tConnection:
 keep-alive\n', '\tUpgrade-Insecure-Requests: 1\n', '\tContent-Type: multipart/f
orm-data; boundary=---------------------------72945705820134082521323642888\n',
'\tContent-Length: 320\n', '\t\n', '\t-----------------------------7294570582013
4082521323642888\n', '\tContent-Disposition: form-data; name="keyword"\n', '\t\n
', '\t<script>alert(dgdgg)</script>\n', '\t<script>alert(dgdgg)</script>\n'], ma
xlen=20)
-------------------------------------------------------------------------------
```

Fig 4.2.2: System after attack detection

Fig 4.2.2 shows that Cross-site scripting attack is detected and the entire packet is captured and stored in a flat file. The captured packet has many attributes, such as Source IP address, Destination IP address, Host (the site on which the attack has been performed), User-agent (Browser used for performing the activity), Content (the content that the attacker had put in the input field of the web page).

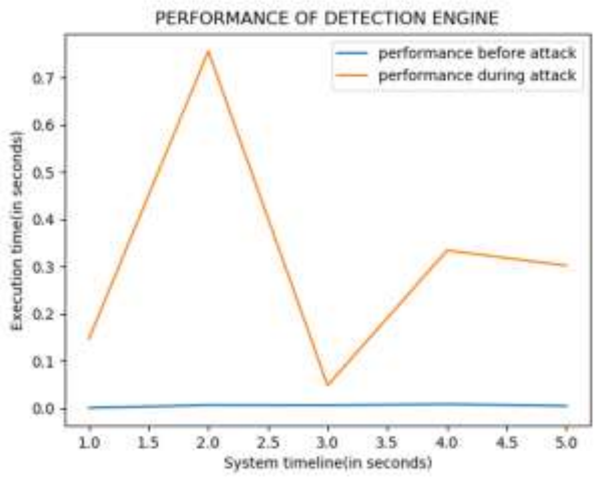


Fig 4.2.3: System performance

Fig 4.2.3 shows a graph of system timeline versus execution time. The blue line represents the performance of detection engine before attack, while the orange line represents performance during attack. From the time the detection engine starts till the time an attack is detected, the execution time remains almost constant. When an attack is detected, the magnitude of increase in execution time depends upon the type of the attack.

5. CONCLUSIONS

This paper focused on the proposed signature-based IDS, NetReconner, and elaborately explains the implementation of the Intrusion Detection System. There are quite a number of IDSs available in the market that require to be configured on the system to be deployed on. NetReconner, on the other hand, does not need to be configured. It only requires installation of a few files and the IDS is good to go. The detection engine is lightweight and uses simple flat files for storing the set of regular expressions. No relational database is used in the NetReconner which makes operations of the detection engine easier. Another factor which makes NetReconner advantageous is that the detection engine works on string operations making it less cumbersome for comparisons. Despite these advantages, problems still exist in today's Intrusion Detection Systems. RegEx generation is still a challenge being a manual process. Furthermore, processing of the packets depends on the system hardware. The existing IDS can be added with more features such as behavioral security and conversion to Intrusion Prevention System (IPS).

REFERENCES

[1] Kaspersky Security Bulletin OVERALL STATISTICS FOR 2016 Maria Garnaeva, Fedor Sinitsyn, Yury Namestnikov, Denis Makrushin, Alexander Liskin.

[2] Karen Scarfone[1], Peter Mell[2], Guide to Intrusion Detection and Prevention Systems (IDPS), Computer Security Division Information Technology Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899-8930.

[3] Akash Garg[1], Prachi Maheshwari[2], 'Performance Analysis of Snort-based Intrusion Detection System "3rd International Conference on Advanced Computing and Communication System", January 2016.

[4] Abdullah H Almutairi[1], Dr. Nabih T Abdelmajeed[2], "Innovative Signature Based Intrusion Detection System using Parallel Processing and Minimized Database' Institute of Electrical and Electronics Engineers", May 2017.

[5] Prof. D. P. Gaikwad[1], Pooja Polshettiwar[2], Priyanka Musale[3], Pooja Paranjape[4], Ashwini S. Pawar[5] "A Proposal for Implementation of Signature Based Intrusion Detection System Using Multithreading Technique" International Journal of Computational Engineering Research, November 2012.

[6] Osaghae EO[1]. Improved signature-based antivirus system. International Journal of Computer Science and Information Technology Research. 2015;3(4):250-254.