Volume: 5 Issue: X Month of publication: October 2017
DOI: http://doi.org/10.22214/ijraset.2017.10056

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor:6.887
Volume 5 Issue X, October 2017- Available at www.ijraset.com

Vulnerability Signatures with Application Classification & Performance Enhancement for IDS/IPS

Arpita D. Patel[1], Mr. Sunil Sharma[2]
[1]*M.Tech Scholar, Department of Electronics & Communication Engineering, Pacific University, Udaipur*
[2]*Assistant Professor, Department of Electronics & Communication Engineering Pacific University, Udaipur*

Abstract: There has been great amount of work done in the field of network intrusion detection over the past 20-30 years. With networks getting faster and with the increasing dependence on the Internet both at the personal and commercial level, intrusion detection becomes a challenging process. The challenge here is not only to be able to actively monitor large numbers of systems but also to be able to react quickly to different events. Intrusion Detection Systems (IDSs) aim at detecting intrusions that is actions that attempt to compromise the confidentiality, integrity and availability of computer resources. An intrusion Detection System (IDS) is a passive system which relies on the system administrator to take action when an attack is detected. The latency between an attack detection and corrective action taken by the administrator is usually high and therefore, by the time the administrator notices an attack and takes an action, the damage is already done. This necessitates the need for an Intrusion Prevention System (IPS) which can not only detect attacks but can also actively responds to them. Intrusion Prevention is a pre-emptive approach to system security which is used to identify potential threats and respond to them swiftly. In real environments IDSs have been observed to trigger busts of alerts. Most of them are false positives, i.e., alerts not related to security incidents.
Evaluation of Vulnerabilities with vulnerability signatures would provide a clear picture of all hosts on the network, the service that they provide and also information on the known vulnerabilities. This information would help the administrator in application classification as well as in configuring IDS and can also be used to assign priority to an alert.
In this paper, I describe the design and implementation of Intrusion detection, Intrusion Prevention and evaluation of vulnerabilities with vulnerability signatures for IDS. IDS is built on open source software, Snort, for signature based detection. A new version of Snort, Snort inline has been released which has intrusion prevention capability. I have tried to classify the applications in order to enhance the performance of the IPS by manipulating the Snort signatures such that least amount of false positives or false negatives are observed. The aim of creating vulnerability signatures is to prevent the vulnerabilities of devices monitored by IDS/IPS at regular intervals and to use this information to assign priority to alerts generated by Snort. A vulnerability signature matches all exploits of a given vulnerability, even polymorphic or metamorphic variants. In addition, this paper contains a detailed explanation of design and implementation of work done on load and performance testing of the IDS/IPS device by manipulating signatures with proper results.
Keywords: snort; intrusion detection system (IDS); intrusion prevention system (IPS); vulnerability signatures; application classification logs; rule; snort inline; false positive; packets; exploit.

I. INTRODUCTION

In today's world, where computer networks are essential for an organization's success, it is necessary to take care of organization's sensitive data and resources from intrusions. Since their gain in popularity, Intrusion Detection Systems (IDS) have begun to be used frequently as one component of an effective layered security model for an organization. Today they are used in many places both inside and outside security perimeters and in many different ways. An intrusion can be defined as any activity that violates the three main goals of network security, videlicet – confidentiality, integrity and availability of a system. Understandably,a protected system or network is only as secure as its defenses are strong. Many precautionary measures such as user authentication, tight access control mechanisms, or firewalls are employed by an organization to protect it from intrusions. These preventive measures do not provide complete security because they are incapable of detecting attacks from sulky employees and severe network attacks which exploit the weakness in application programs through the attack surface. Therefore, Intrusion Detection Systems have come into existence as a second line of defense.

The aim of Intrusion Detection System (IDS) is to detect illegal and unusual use of system resources by unauthorized individuals by monitoring network traffic and audit data. The techniques employed by an Intrusion Detection System fall into two broad categories: Signature-Based detection and Anomaly-Based detection. A signature-based IDS uses a signature database for detecting malicious activities. Each signature represents a pattern of activity which corresponds to a known attack. The signature-based IDS examine ongoing traffic, activity, transactions, or behavior and try to find a match with these known patterns of predefined attacks. The strength of these systems lies in their signature database and therefore the database needs to be continuously updated to incorporate information about new attacks.

An anomaly-based IDS constructs a profile that represents normal usage and then uses current behavior data to detect deviations from the profile to recognize possible attack attempts. The profile can be constructed using various data mining and machine learning techniques and should be updated at regular intervals. The merit of this approach over signature-based IDS is that is able to detect novel attacks. However, it is non-trivial to define what constitutes a ‘normal’ behavior and therefore systems based on this approach have a high tendency to generate false alarms.

Intrusion Prevention is a preemptive approach to system security which is used to identify potential threats and respond to them swiftly. Like an IDS, Intrusion Prevention System (IPS) monitors network traffic or audit data. Other than that, IPS also has the ability to take immediate action, based on a set of rules specified by the network administrator. For example, IPS might drop a packet that it determines to be malicious and block all further traffic from that IP address. Legitimate traffic, meanwhile, is forwarded to the recipient with no apparent disruption or delay of service.

Intrusion Prevention System (IPS) can be classified into two main categories: Host-Based IPS and Network-Based IPS. A host-based IPS is installed on the system to be protected. It works in co-ordination with the operating system kernel to block abnormal application or user behavior. For example, it may monitor system calls or APIs invoked by applications in order to detect attacks. A host-based IPS requires tight integration with the operating system which implies that future operating system upgrade might cause problems.

A network-based IPS, also known as Inline IPS or Gateway IPS, is developed to monitor a single host or an entire network segment. It analyzes the incoming network traffic for malicious activities. It may drop the malicious packets, reset the network session, or block traffic from particular hosts depending on user-specified policy. It should work in inline-mode, that is, incoming packets should pass through it before reaching the target application. Therefore, it is essential that its impact on overall network performance is minimal. It should also have high detection accuracy because it is an active system and inaccurate detection would lead to loss of legitimate traffic.

IDS generate thousands of alerts per day. Therefore, it becomes critical to prioritize the alerts so that the administrator can focus on major threats. This is usually done through vulnerability assessment and then application classification. In this scheme, information about the systems to be protected is maintained so that the attacks to which the system is known to be vulnerable are given higher priority.

A. Vulnerability

Basically, vulnerability is the weakness that one has in it like a body, a device, a system etc. Threat or malicious activity to enter into a computer system and affect the overall business impact either by damaging the system or corrupting the information inside the system.

B. Types of vulnerability

In general there are two types of vulnerabilities viz - known & unknown (zero day).

C. Known vulnerability

Known vulnerabilities are those that are found & reported. Every known vulnerability has a unique id.Common vulnerability enumeration (CVE) is a dictionary of publicly known information security vulnerabilities and exposures. CVE id is unique identity of vulnerability is a specific number that is assigned by CVE to each and every known vulnerability that exists.

D. Zero day attack

A zero-day attack or threat is a computer threat that tries to exploit computer application vulnerabilities that are unknown to others or the software developer. The following are some points to identify a zero day. The developer creates software containing a (unknown) vulnerability. The attacker finds the vulnerability before the developer does, hence zero day. The attacker writes and distributes an exploit while the vulnerability is not known to the developer. The developer becomes aware of the vulnerability and starts developing to fix it.

E. Solution to the vulnerabilities
Overall, two types of solutions can be thought of:

F. Vendor patches
Vendor patches are the updates or the new releases that are essential so as to work with the ongoing computer security environment. For example, to add some extra functionality to the application or on identifying vulnerability in a system, the developer creates patches

G. Intrusion prevention systems (IPS):
IPS is network security appliances that monitor n/w and/or system activities for malicious activity. The main function of IPS is to identify malicious activity, log information attempt to block/stop activity, and report activity. Advantage of using IPS is it can be centrally managed and downtime is negligible.

H. Why IPS should be given more priority compared to vendor patches?
IPS gives:
1) Proactive solution.
2) Saves bandwidth.
3) Better performance.

I. Platform for IPS:
Open source tools are available that can work as ids as well as IPS. Some of them are snort, suricata, bro and as such. These tools are openly available with enumerable features to be configured as detection and prevention systems. Out of them I have selected snort as a tool to develop our intrusion detection system (IDS). Also I have made use of snort inline, a new version of snort to build my intrusion prevention system (IPS).snort can perform several simple tasks; however if its architecture is understood, snort makes a lot more sense.

J. Snort
Snort is an open source intrusion detection system (IDS) written by martin roesch. It was bought by the commercial company source fire which was bought itself by firewall giant checkpoint in 2005. Like tcpdump, snort uses the libpcap library to capture packets. Snort is a signature-based IDS, and uses rules to check for errant packets in network. A rule is a set of requirements that would trigger an alert. For example, one snort rule to check for peer-to-peer file sharing services checks for the get string not connecting to a service running on port 80. If a packet matches that rule, that packet creates an alert. Once an alert is triggered, the alert can go a magnitude of places, such as a log file or to a database.

K. Snort
It is a libpcap-based packet sniffer and logger that can be used as a lightweight network intrusion detection system (NIDS). It features rules based logging to perform content pattern matching and detect a variety of attacks. The detection engine is programmed using a simple language that describes per packet tests and actions. Ease of use simplifies and expedites the development of new exploit detection rules.
Snort performs protocol analysis, content searching, and content matching. The program can also be used to detect probes or attacks, including, but not limited to, operating system fingerprinting attempts, common gateway interface, buffer overflows, server message block probes, and stealth port scans.

L. Architecture of snort
Snort has several important components, most of which take plug-ins to customize your snort implementation. These components include pre-processors and alert plug-ins, which enable snort to manipulate a packet to make the contents more manageable by the detection engine, and the alert system, which can send its output through various methods. Author M Roesch [14] presented by Snortcan perform several simple tasks; however, if you understand the architecture, snort makes a lot more sense. Snort consists of four basic components

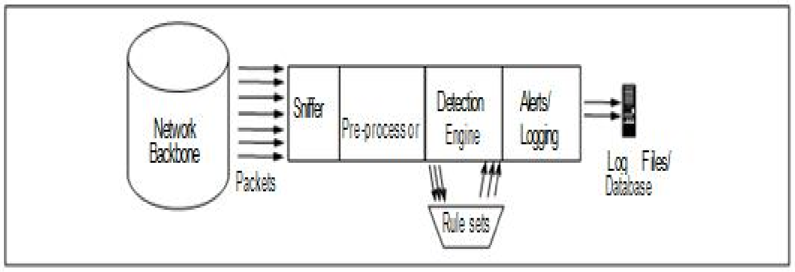


Fig: snort architecture

1) *Decoder:*it fits captured packets into data structures and identifies link level protocols. Then it takes the next level, decodes ip, and tcp/udp to get information about port addresses.
2) *Pre-processors:*packet re-assembly and tcp stream re-assembly.
3) *Detection engine:* applies rule to packets.
4) *Logging and alerting system:*Generates logs and alert messages.
5) *Output modules:* -process alerts and logs and prepares the final report.

M. The content keyword

One important feature of snort is its ability to find a data pattern inside a packet. The pattern may be presented in the form of an ASCII string or as binary data in the form of hexadecimal characters.

N. The offset keyword

The offset keyword is used in combination with the content keyword. Using this keyword, one can start your search at a certain offset from the start of the data part of the packet.

O. The depth keyword

The depth keyword is also used in combination with the content keyword to specify an upper limit to the pattern matching. Using the depth keyword, you can specify an offset from the start of the data part.

P. The dsize keyword

The dsize keyword is used to find the length of the data part of a packet. Many attacks use buffer overflow vulnerabilities by sending large size packets.

Q. The flags keyword

The flags keyword is used to find out which flag bits are set inside the tcp header of a packet. Each flag can be used as an argument to flags keyword in snort rules.

R. The msg keyword

The msg keyword in the rule options is used to add a text string to logs and alerts. You can add a message inside double quotations after this keyword. The msg keyword is a common and useful keyword and is part of most of the rules.

S. The nocase keyword

The nocase keyword is used in combination with the content keyword. It has no arguments. Its only purpose is to make a case insensitive search of a pattern within the data part of a packet.

T. The flow keyword

The flow keyword is used to apply a rule on tcp sessions to packets flowing in a particular direction. You can use options with the keyword to determine direction.

U. The pcre keyword

The pcre keyword allows rules to be written using Perl compatible regular expressions.

V. Vulnerability signature creation:

Vulnerabilities and exposures will always be a part of the systems, as will the groups that find and share information about commercial and open source software errors. The ability to apply all known security fixes and patches offers a robust way to keep an organization's software infrastructure healthy. The common-names-integration and cross-referencing abilities now emerging in vulnerability and exposure tools, web sites, and databases make it possible to deal with security-relevant mistakes and improve systems' security.

Snort uses its own rules for creating custom signatures to detect the threat as well as vulnerability and it is known as vulnerability signature. I define a vulnerability signature as a set of symbolic predicates based on the protocol semantic information. This form of vulnerability signatures can express most known vulnerability conditions precisely. Based on the principle of optimizing common cases, my design mainly speeds up the matching speed of symbolic predicate signatures.

To recover the protocol semantic information, I need to parse the input. In addition, a protocol state machine (a.k.a., vulnerability state machine in) is required for adjusting the protocol states when sending/receiving different protocol data units (pdus).

1) Step 1: vulnerability signatures for the applications

a) Generic application:-Those apps that are used in general and on regular basis in web/world in an organization. (Antivirus, browser, etc.)

b) Most recently updated vulnerabilities:-Vulnerabilities are available on internet and new ones are added consecutively as they are found. Exploits are readily available in most of the cases to perform an ideal attack using the vulnerability.

2) Step 2: application classification

a) Example:-considering one organization.

b) Organization has n number use.

c) All are use different application.

d) It is very difficult task for administrator or network controller how to tune IPS?

e) For that solution first list out total application run on network.

f) If any vulnerability found related to those application than create signature for that and properly tune the IPS.

3) Step 3: behavioral study of application

Before making the vulnerability signature and classifying apps and all, one needs to know the working scenario of that apps as well as the extent of harm the vulnerability can perform. Few steps include:

a) Download application.

b) Install application.

c) Understand the normal working of application.

d) Capture the normal traffic packets.

4) Step 4 exploit search

a) By exploits I mean that a code an advantage through which I can perform an attack using that vulnerability.

b) Search the exploits on internet. If not found try to develop such code.

c) Build an environment that satisfies the necessary and sufficient conditions to run the exploits.

d) Run the exploit and capture the traffic packets in attack condition

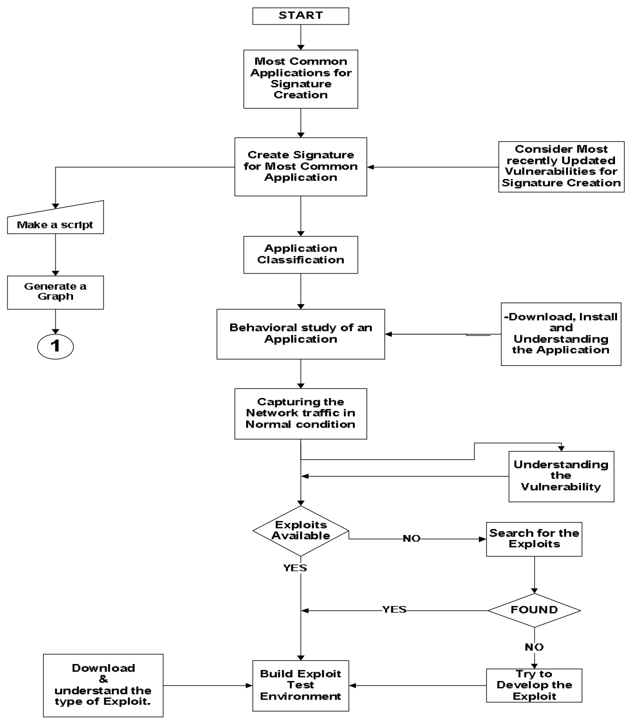

START
Most Common Applications for Signature Creation
Create Signature for Most Common Application
Consider Most recently Updated Vulnerabilities for Signature Creation
Make a script
Generate a Graph
1
Application Classification
Behavioral study of an Application
-Download, Install and Understanding the Application
Capturing the Network traffic in Normal condition
Understanding the Vulnerability
Exploits Available
NO
Search for the Exploits
YES
YES
FOUND
NO
Download & understand the type of Exploit.
Build Exploit Test Environment
Try to Develop the Exploit

Figure: work flow logic (vulnerability signature creation before the exploit)

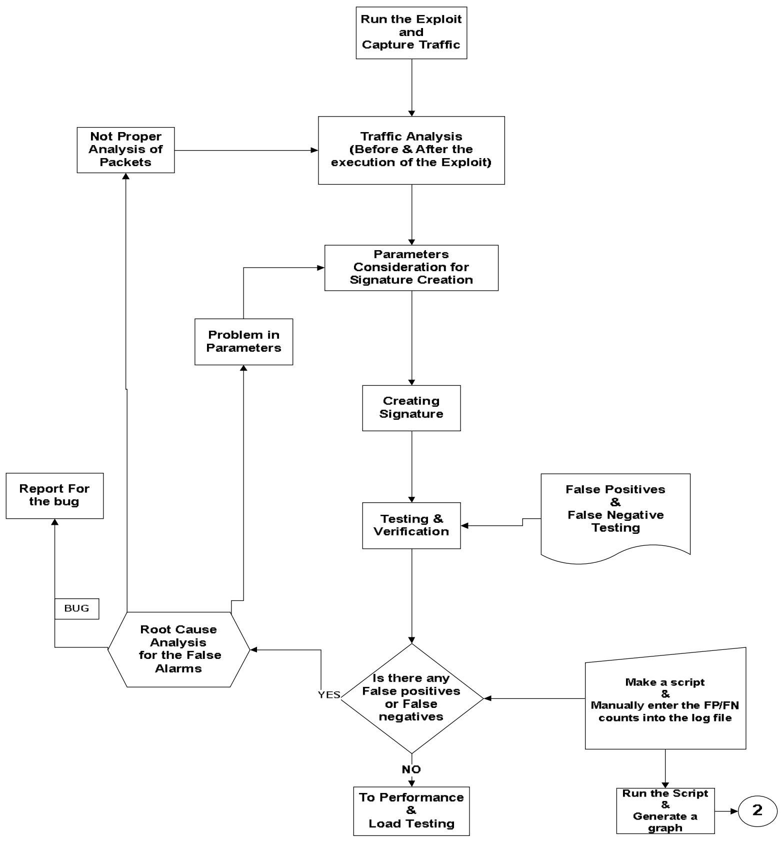
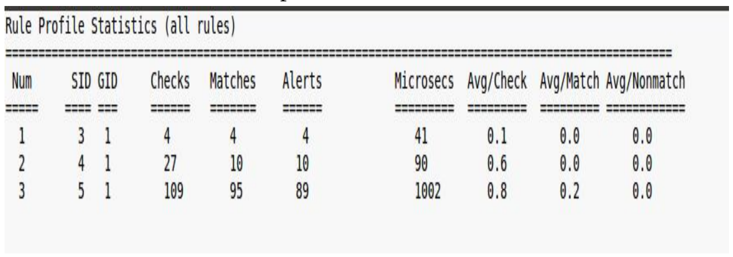

Run the Exploit and Capture Traffic
Not Proper Analysis of Packets
Traffic Analysis (Before & After the execution of the Exploit)
Parameters Consideration for Signature Creation
Problem in Parameters
Creating Signature
Report For the bug
Testing & Verification
False Positives & False Negative Testing
BUG
Root Cause Analysis for the False Alarms
YES
Is there any False positives or False negatives
Make a script & Manually enter the FP/FN counts into the log file
NO
To Performance & Load Testing
Run the Script & Generate a graph
2

Figure: work flow logic (vulnerability signature creation after the exploit)

5) *Step 5: analyze the captured traffic*
a) Normal traffic analysis (as in step3).
b) Analysis of traffic after executing the exploit.
6) *Step 6: creating the signature*
a) Considering parameter for signature creation.
7) *Step 7: testing & verification*
a) It includes testing of signatures mainly against the false positive and false negative.
b) *False positive:* a false positive exists if a signature that is created to detect a particular threat or malicious activity generates alerts based on normal traffic.
c) *False negative:* if a signature bypasses and unable to detect any threat or malicious form of attack, and misses the attack is termed as false negative for that threat.
d) *Signature profiling:*how much time a rule takes for profiling in network traffic.
e) *Num: this* column reflects this rule's position number in regard to how bad the rule performs.
f) *id, gid, and rev***:** The snort id, generator id, and revision of the rule. This is shown to help us identify the rule in question in our rule-set.
g) *Checks:* the number of times rule options were checked after the fast pattern match process.
h) *Matches:*the number of times all rule options match, therefore traffic matching the rule has been found.
i) *Alert:* number of time rules generates an alert.
j) *Microsecs:*total time taken processing this rule againstthe network traffic.
k) *Avg/check:* average time taken to check each packet against the rule.

Rule profile statistic table



```
Rule Profile Statistics (all rules)
==========================================================================================
 Num      SID GID      Checks   Matches    Alerts          Microsecs  Avg/Check  Avg/Match Avg/Nonmatch
 =====    ==== ===     ======   =======    ======          =========  =========  ========= ============
   1        3   1         4        4          4               41        0.1        0.0        0.0
   2        4   1        27       10         10               90        0.6        0.0        0.0
   3        5   1       109       95         89             1002        0.8        0.2        0.0
```

```
Root@arpita-pegatron:/etc/snort_inline/script# ./fp.sh
1 false positives count is 5
1 false negatives count is 3
2 false positives count is 2
2 false negatives count is 2
  % total    % received % Xferd Average speed   Time     Time    Time current
                          Dload   upload   total   spent    left     speed
100 11871    0 11871    0     0          6015      0         --:--:-- 0:00:01 --:--:--   703
```

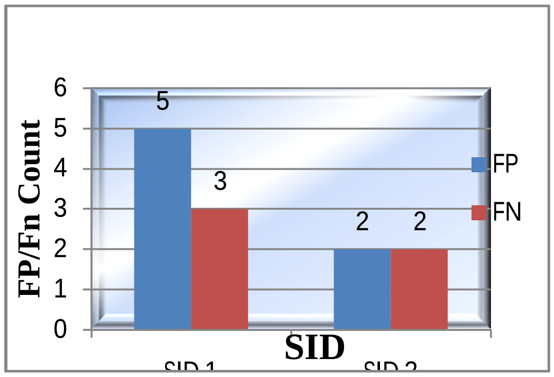


Fig: Output to FP/FN Test

```
root@arpita-Pegatron:/etc/snort_inline/script# ./fp.sh
1 false positives count is 2
1 false negatives count is 2
2 false positives count is 1
2 false negatives count is 0
% Total    % Received % Xferd Average Speed   Time    Time     Time Current
                                 Dload   Upload  Total   Spent    Left  Speed
100 12627    0 12627    0    0     6672      0      --:--:--  0:00:01 --:--:--  7765
```

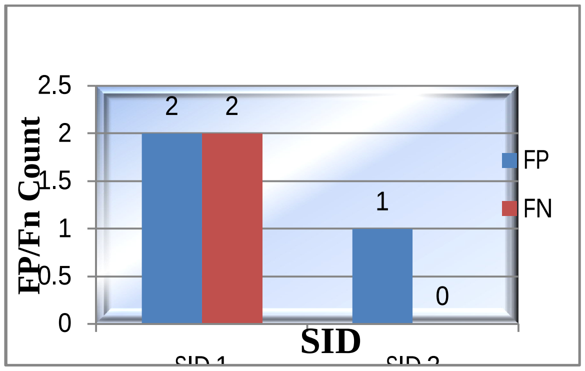


Fig: Output to FP/FN Test (after modification)

Output to signature profiling:

```
Root@arpita-Pegatron:/etc/snort_inline# ./arpita.sh
1 is taking 11149
 % Total    % Received % Xferd Average Speed   Time    Time     Time Current
                                 Dload  Upload  Total  Spent    Left    Speed
100 96830 9683          0     0   4200           0   --:--:-- 0:00:02 --:--:--    4689
```

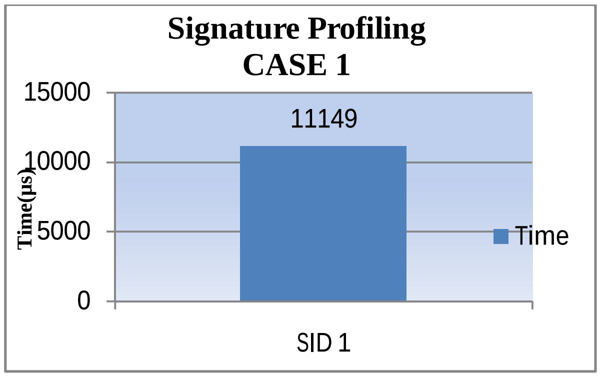


Fig: Result of Case 1

```
root@arpita-Pegatron:/etc/snort_inline# ./arpita.sh
2 is taking 207
 % Total    % Received % Xferd   Average Speed  Time    Time     Time Current
                                  Dload  Upload Total  Spent   Left     Speed
  100 9110    0 9110    0     0          3447      0     --:--:--   0:00:02   --:--:--     3886
```

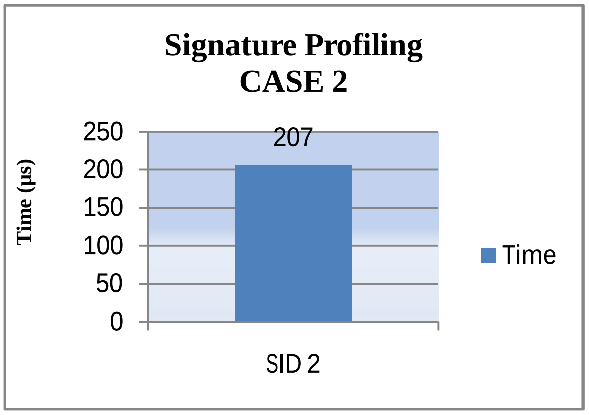


Fig: Result to case 2 (after modification)

```
root@arpita-Pegatron:/etc/snort_inline# ./arpita.sh
3 is taking 41
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
 1009514   0     9514    0     0    4785      0 --:--:-- 0:00:01 --:--:-- 5489
```

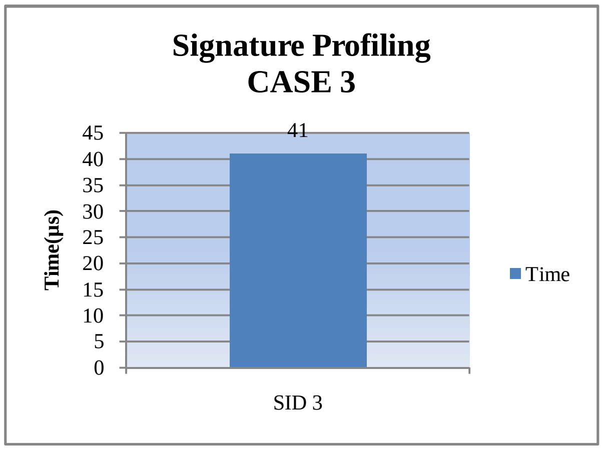


Fig: Result to case 3 (after modification)

II. CONCLUSION AND FUTURE WORK

I have shown that network-based vulnerability specific filter is feasible to implement, with low false positive rates, manageable scalability, and board applicability across protocols. I have shown that application can be classified and can be kept safe from vulnerability in an organizational network. I have snort configuration file and using its rule profile statistic developed a script that automatically shows output in from of graph and hence statistical graphs are observed. My methods are the practical approach for creating sound vulnerability signatures.

Further work include load and performance testing for the IPS device and creating a script for generating relatively for developing better vulnerability signatures and hence automatically generating graph for that analysis.

III. ACKNOWLEDGEMENTS

Iwould like to thanks Mr. Deepak Vyas &Mr. Sunil Sharma for giving an opportunity to work on this topic. Also thanks for their support and supervision during this work and the anonymous reviewers for their many helpful comments and suggestions while preparing this paper. This research was supported by the department of electronics and communication engineering, pacific university. Any options, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. "Towards automatic generation of vulnerability signatures," IEEE symposium on security and privacy, may 2006.
[2] [3] D. Brumley, C. Hartwig, Z. Liang, J. Newsome, D. Song, and H. Yin. "Towards automatically identifying trigger-based behavior in malware using symbolic execution and binary analysis". Technical report cmu-cs-07-105, CarnegieMellon university school of computer science, January 2007
[3] W. A. Arbaugh, W. L. Fithen, and J. Mchugh. Windows of vulnerability: a case study analysis. Ieee computer, 2000.
[4] G Vigna, W Robertson, Davide Balzarotti, "Testing network-based intrusion detection signatures using mutant exploits" reliable software group, ccs '04 proceedings of the 11th acm conference on computer and communications security, 2004 - dl.acm.or
[5] Byung-chul park. Won, Y.J, Myung-Sup Kim, Hong, J.W, "Towards automated application signature generation for traffic identification", dept. of computer. Sci. & eng.,postech, Pohang, network operations and management symposium, 2008. Noms 2008. Ieee ieeexplore.ieee.org
[6] James Newsome and Dawn song. "Dynamic taint analysis: automatic detection, analysis, and signature generation of exploit attacks on commodity software," network and distributed systems security symposium, Feb. 2005
[7] US-CERT. Vulnerability note vu#196945 – isc bind 8 contains buffer overflow in transaction signature (tsig) handling code. Http://www.kb.cert.org/vuls/id/ 196945
[8] H.J.Wang, C.Guo, D.Simon, and A.Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. In proc. of the 2004 acm sigcomm conference, august 2004.
[9] H.Kim, B.Karp. "autograph: toward automated, distributed worm signature detection," 13th usenix security symposium, 2004
[10] D. Brumley, Hao Wang, Jha, S., Dawn Song, "creating vulnerability signatures using weakest preconditions", computer security foundations symposium, 2007. Csf '07. 20th iee
[11] urbana-champaign, 16 September 2008
[12] M Sutton, F Nagle,A citation on emerging economic models for vulnerability research, cissp - workshop on the economics of information security, 2006
[13] M Roesch, Snort-lightweight intrusion detection for networks, proceedings of the 13th usenix conference, systems administration conference Seattle, Washington, USA, November 7–12, 1999.