2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP) 978-1-7281-6708-4/20/$31.00 ©2020 IEEE 10.1109/ISEA-ISAP49340.2020.235004

A Traffic Monitoring and Policy Enforcement Framework for HTTP

1st Muraleedharan N
Centre for Development of Advanced Computing (C-DAC)
Bangalore, India
murali@cdac.in

2nd Anna Thomas
Centre for Development of Advanced Computing (C-DAC)
Bangalore, India
anna@cdac.in

3rd Indu S
Centre for Development of Advanced Computing (C-DAC)
Bangalore, India
indu@cdac.in

4th B S Bindhumadhava
Centre for Development of Advanced Computing (C-DAC)
Bangalore, India
bindhu@cdac.in

***Abstract*—Due to the accessibility and popularity of Internet, web based applications are commonly used for providing different services to the users. At the same time, the simplicity to conduct attacks and the availability of several attack tools have made web applications the most common target for attackers. Hence monitoring and analysis of web applications require special attention. In this paper, we describe a policy enforcement and web attack detection framework for HTTP protocol. The proposed framework can monitor and analyze HTTP traffic to detect injection, misconfiguration and directory traversal attacks. Moreover, this framework can be used to enforce web application access policies involving content type, URL and device level access.**

***Index Terms*—Web Application attack, SQL injection, Cross site Scripting, URL filter, Web access policy.**

I. Introduction

Internet has been used as a popular and powerful medium to connect, communicate and collaborate beyond any geographical boundaries. The popularity of Internet introduces different web based applications and services which can be accessed through Internet. Services such as e-commerce, finance, transport, health, education, and gaming are some of the examples of web enabled services that can be accessed through Internet. However, the adversaries are also using this same Internet as a medium for carrying out their malicious activities.
The web applications usually follow a three tier client server architecture which consists of three different layers or tiers involving web clients, web servers and databases. Most of the web based attacks target the flaws present in these components to compromise the applications, databases, system resources and network. Due to the absolute possibility of multifaceted target, the adversaries consider web applications as the most vulnerable and easy target for many of the attacks.
The web applications can be accessed by a web client like a web browser using the Uniform Resource Locater (URL). However, automated scanning tools can also send several requests to the server using the URL which may result in information leak. The information revealed can be the server application details, configuration settings and machine details. Since the web traffic is usually allowed to any network without any firewall level filter, attackers use web traffic for sneaking malicious code and content to the target network. Malicious software like virus, worms and Trojan use web traffic as a medium to enter, damage or disrupt the targeted system or network.
The web applications access various content types such as text, image, audio and video. Hence, the web based applications need to be monitored and analyzed to enforce the policies related to content level access. Presently, the increased use of Over The Top (OTT) media services introduce a challenge to monitor, analyze and regulate the content access. In this context, content access by the users need to be monitored and regulated as per the organization policy.

This paper presents a web application policy enforcement and attack detection framework that can identify and analyze the web based attacks and policy violations.
The novelty of this paper is the use of bit level signatures generated from packets for HTTP traffic identification and derivation of a comprehensive framework for web based attack detection and policy enforcement.
The proposed framework has the following advantages compared to the other existing web attack detection systems.

- Port independent HTTP application detection using bit level signatures.
- Detection of web attacks initiated by browser and other non browser clients such as web scanner and injection tools.
- Identification of the tools which initiated the attack traffic.
- Detection and notification of web application policy violations including content type and user agent detection.

This paper is organized as follows. Section II gives the overview of web based attacks and detection approaches.

The authors would like to thank Ministry of Electronics and Information Technology (MeitY), Government of India for the funding and supports.

Details of the proposed framework are explained in Section III. Test-setup used and the results obtained are shown in Section IV. The conclusion and future works are described in Section V.

II. Related Work

Open Web Application Security Project(OWASP) [1] is an open community that creates awareness about application security by identifying some of the most critical web application risks. The OWASP also provides a top 10 attack list known as the 'OWASP Top 10' which outlines the ten most critical security risks. As per OWASP top 10 application security risk 2017, the top attacks affecting web application are Injection attacks, Broken authentication, sensitive data exposure, XML external entities, broken access control, security misconfiguration, cross site scripting, insecure de-serialization using component with known vulnerabilities and insufficient logging and monitoring [2].

There are several reasons for the existence of web application attacks. For instance, the injection attacks are present because of the improper validation of un-trusted inputs. Injection attacks are found in multiple components used in a web application such as Structured Query Language (SQL), LDAP, XPATH, OS command etc.

A survey of web application vulnerability attacks are presented in [3]. Many attempts have been made with the purpose of web application attack detection but most studies have only focused on SQL injection [4]–[8] and cross site scripting (XSS) [9]. Babiker et al. [10] investigated the techniques used in the detection and forensics of web application attacks. Defense mechanisms against code injection attacks such as SQL injection, cross site scripting are described in [11].

Attackers use malicious URLs to sent various attacks including spam, phishing and malware. Different approaches have been introduced to detect malicious URLs from the request [12], [13]. A recent review of the literature on this topic found that machine and deep learning techniques are used for web attack detection [8], [14].

Presently, multiple approaches are available to address individual attack on web applications. For instance, methods for detecting injection attacks on web application are already available. However, to the best of our knowledge, a comprehensive framework for web attack detection and policy enforcement to mitigate these attacks is missing. Our paper tries to address this gap by proposing a traffic monitoring and policy enforcement framework for HTTP. .

III. Web attack and policy violation detection framework

The architecture of the proposed web policy enforcement and attack detection framework is shown in Fig. 1. As depicted in the figure, this framework is divided into two major components namely the HTTP detection engine and the HTTP analyzer. The HTTP detection engine takes network packets as the input and classifies the HTTP traffic from other network applications. The detailed analysis of HTTP traffic for identifying the web attacks and policy violations are carried out by the HTTP analyzer module. The details of these components are described in the following sections.

A. HTTP Detection Engine

The classification of the HTTP traffic from other application layer protocols is the first step of our web traffic analysis framework. The traditional approach to application classification is based on application ports, packet payload and statistical methods [15]. Port based traffic identification is not relevant in the current scenario as many applications do not use predefined ports [16]. Statistical techniques that rely on the properties of the captured packets like size of the packet, their inter-arrival time and the arrival order to classify the network traffic are also not considered to be highly reliable [17].

Recent techniques for network traffic classification are based on deep packet inspection which analyzes the entire packet payload to search for any known application signatures. Even though the technique is highly reliable and accurate when compared to other methods, various drawbacks are reported for this method also. As this method involves processing the entire packet content, it requires more CPU and memory resources as compared to port or statistical based methods which examines only packet headers. This method also affects the network user's privacy as the examined payload may contain sensitive user data. Capturing the entire packet payload also requires more disk space usage.

A light weight traffic classification approach that examines only the first four bytes of packet payload observed in each direction is explained in [15]. This approach requires less processing power to classify the network traffic. As it analyzes only the first four bytes of packet payload, it also addresses the user privacy concerns.

A DPI based bit-level application signature generation using invariant bits of application flows are presented in [18]. Bit signatures are light weight for signature matching as well as storage efficient. The paper discusses a technique where usually the first 40 bits of the flow is used for signature generation and the signature bits are then run length encoded to reduce size. Though short signatures are efficient for processing, this may increase the chances of collision and cross signature matching when more number of applications are considered. Signature similarity detection using a variant of Hamming distance is also suggested in the paper. In such cases, the length of signatures is increased for a subset of protocols to avoid overlaps.

The proposed framework uses a bit-coding approach for HTTP traffic detection. HTTP traffic is detected by capturing the first 40 bits of the TCP packet payload and comparing it with generated signature pattern. Experiments conducted as part of the study [18] shows that, the first 40 bits can accurately detect the application traffic. For example, the first 40 bits of the TCP payload for HTTP GET request is shown in Fig. 2. Table I shows the first 40 bit patterns of HTTP requests and response header. The first column of the table shows the HTTP method. The pattern shown in the second column of the table

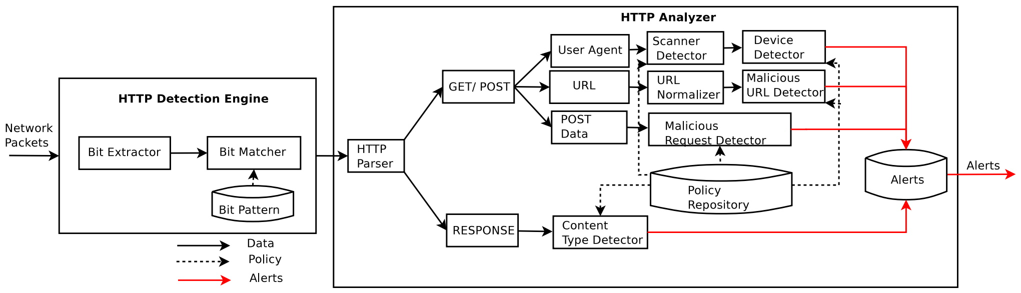
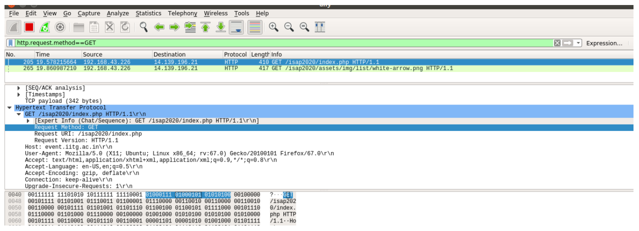


Fig. 1: Architecture of the web attack and policy violation detection framework.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

```
http.request.method==GET
No.  Time          Source          Destination    Protocol Length Info
 205 19.578215664  192.168.43.226  14.139.196.21  HTTP     410 GET /isap2020/index.php HTTP/1.1
 265 19.860987210  192.168.43.226  14.139.196.21  HTTP     417 GET /isap2020/assets/img/list/white-arrow.png HTTP/1.1

  [SEQ/ACK analysis]
  [Timestamps]
  TCP payload (342 bytes)
Hypertext Transfer Protocol
  GET /isap2020/index.php HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /isap2020/index.php HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /isap2020/index.php
    Request Version: HTTP/1.1
  Host: event.iitg.ac.in\r\n
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:67.0) Gecko/20100101 Firefox/67.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
0040  00111111 11101010 10111111 11110001 01000111 01000101 01010100 00100000   ?···GET
0048  00101111 01101001 01110011 01100001 01110000 00110010 00110000 00110010   /isap202
0050  00110000 00101111 01101001 01101110 01100100 01100101 01111000 00101110   0/index.
0058  01110000 01101000 01110000 00100000 01001000 01010100 01010100 01010000   php HTTP
0060  00101111 00110001 00101110 00110001 00001101 00001010 01001000 01101111   /1.1··Ho
```

Fig. 2: Bit level details of HTTP GET method.

shows the derived pattern from the first five characters of HTTP method. For example, in case of HTTP POST method, the pattern includes five characters namely ’P’, ’O’, ’S’, ’T’ and ’ ’. This patter when converted into hexadecimal will result in 0x50, 0x4f, 0x53, 0x54 and 0x20. The bit level representation of the pattern is shown in the last column of the table.

TABLE I: HTTP methods and initial 40 bits bit pattern

Method	Pattern	Bit level Pattern
GET	GET /	0100 0111 0100 0101 0101 0100 0010 0000 0010 1111
POST	’POST ’	0101 0000 0100 1111 0101 0011 0101 0100 0010 0000
HEAD	HEAD	0100 1000 0100 0101 0100 0001 0100 0100 0010 0000
PUT	PUT /	0101 0000 0101 0101 0101 0100 0010 0000 0010 1111
TRACE	TRACE	0101 0100 0101 0010 0100 0001 0100 0011 0100 0101
CONNECT	CONNE	0100 0011 0100 1111 0100 1110 0100 0011 0100 0101
OPTIONS	OPTIO	0100 1111 0101 0000 0101 0100 0100 1001 0100 1111
PATCH	PATCH	0101 0000 0100 0001 0101 0100 0100 0011 0100 1000
DELETE	DELET	0100 0100 0100 0101 0100 1100 0100 0101 0101 0100
HTTP/1.1	HTTP/	0100 1000 0101 0100 0101 0100 0101 0000 0010 1111

1) Bit Extractor: The ’Bit Extractor’ (BE) component of HTTP detection engine, extracts out the first 40 bits from the TCP payload. It is assumed that there is no error induced in the channel during the packet transfer. All the received Ethernet packets are collected and decoded to extract the first 40 bit values. The BE component decodes the packets as per the protocol RFCs.
The pattern used to classify the HTTP traffic for analysis are shown in Table II. As we have used only HTTP GET, POST request methods and HTTP/1.1 response for further analysis, the bit values from these methods are considered for pattern generation. To derive the pattern, bit positions that are changing the values are considered as don’t care and it is indicated as ’*’ in the pattern.

TABLE II: HTTP traffic classification pattern

Method	Bit level Pattern
GET	0100 0111 0100 0101 0101 0100 0010 0000 0010 1111
POST	0101 0000 0100 1111 0101 0011 0101 0100 0010 0000
HTTP/1.1	0100 1000 0101 0100 0101 0100 0101 0000 0010 1111
Derived Pattern	**010* **** 010* *1** 0101 0*** 0*** 0*00 0010 ******

2) Bit Matcher: The ’Bit Matcher’ (BM) component takes the first 40 bits of TCP payload from the BE and compares them with the bit patterns defined in the bit pattern repository. If it exactly matches with the pattern defined in the repository, then it is considered as HTTP protocol. Further parameters such as source, destination IP addresses, port numbers and protocol field of the packet is used to group the remaining packets in that HTTP session.

B. HTTP Analyzer

The HTTP Analyzer module inspects all the detected HTTP packets for any malicious content or application vulnerability. The module identifies a packet as malicious using various packet header attributes, possible attack patterns and black-listed malicious URLs database.
The HTTP parser engine which lies between the HTTP Detection and Analyzer module parses the HTTP packets identified by the detection module to obtain the various HTTP header attributes. The attributes identified include User-Agent, HTTP URL, Content-Type and the HTTP POST payload. The HTTP Analyzer module later uses these attribute values for it’s packet filtering functioning. The steps followed by the HTTP Analyzer module are described in the Algorithm 1. As the algorithm describes, the HTTP Analyzer takes the identified HTTP packets and extracts the header values for filtering. For HTTP POST requests, the request body is checked for attack patterns present in the pattern repository. The module also inspects the header ’User Agent’ value of all HTTP GET and POST requests to identify the client who initiated the connection. The module even blocks any blacklisted or malicious HTTP URL access.

1) Application Vulnerability Detection: Every HTTP packet is examined for application vulnerabilities like injection patterns, security misconfiguration and cross site scripting which are listed as the top 10 OWASP web attacks.

Algorithm 1 HTTP Analyzer Algorithm

```
Input: HTTP packets identified by HTTP detection engine
Output: Alerts generated
 1: HTTP parser extracts required header values
 2: if (HTTP POST) and
    (HTTP POST Body has attack patterns) then
 3:   Generate alerts
 4:   Invoke Packet Drop/Connection Reset
 5: end if
 6: if (HTTP GET) or (HTTP POST) then
 7:   Inspect 'User-Agent' value in HTTP packet
 8:   if (User-Agent$\neq$browser) then
 9:     Generate alerts
10:     Invoke Packet Drop/Connection Reset based on
        defined policies
11:   end if
12:   if (HTTP URL is present) then
13:     nHTTPURL = Normalize the HTTP URL
14:     if (nHTTPURL == blacklisted URL) or
        (nHTTPURL has attack patterns) then
15:       Generate alerts
16:       Invoke Packet Drop/Connection Reset
17:     end if
18:   end if
19: end if
```

The HTTP URL value present in the request header of both GET and POST requests is inspected for any application vulnerability. Similarly, every HTTP POST request is also inspected for the presence of any application vulnerability in the HTTP POST body. In both cases, the system looks for the presence of any attack patterns and generates appropriate alerts.
Injection attacks are the most common attack targeted on database driven web applications where the attacker sends an HTTP request with patterns involving SQL database queries. These queries if executed can result in grave consequences like database update, deletion or unauthorized access.
For instance, in most of the websites the user is prompted to enter user name and password in a login form. Suppose, an attacker enters patterns like say, user1 for username and ' or '1'='1 for password. This can bypass password verification as the resultant back end query might look like SELECT * FROM USERS WHERE name='user1' AND password=''or '1'='1'. Here, '1'='1' is an always true condition which allows the attacker to successfully login to the system. The system identifies such attacks involving SQL queries by searching for the presence of any SQL query patterns in the POST request body.
Some of the web applications are prone to security risks due to misconfiguration. For instance, a misconfigured web server can result in directory listings. This can expose many sensitive details of the server and the attacker can target his future attacks based on this leaked information. The system can identify such attacks by patterns like ../../ or $\%2e\%2e\%2f$ pattern in HTTP request URL. In the same way cross site scripting attacks can be detected by the presence of $<script>$ tags in HTTP request.

2) Device Detection: The HTTP request header contains an attribute named User-Agent. The User-Agent value reveals details regarding the client who is trying to access the server. For example, the User-Agent value when a person tries to access the Internet from a Firefox web browser on an Ubuntu machine will be `Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:70.0) Gecko/20100101 Firefox/70.0`. Other than web browsers there are several types of scanning tools that can inject packets to the server. An example for one such tool is sqlmap, an open source penetration testing tool that can reveal sql injection flaws in a server. The 'User-Agent' value in that case will be `sqlmap/1.2.4#stable (http://sqlmap.org)`. Moreover, there are many On The Top(OTT) media devices like Chromecast, Smart TVs and Amazon Fire Stick which provide media services over Internet. These OTT devices can also be detected by various methods. The system uses a predefined set of User-Agent strings to detect the clients initiating the HTTP traffic.

3) Malicious HTTP URL Detection: The HTTP request URL is another attribute which is present in every HTTP request packet header. A URL (Uniform Resource Locator) specifies the unique address which can identify or locate different resources in the Internet. The system uses a list of malicious HTTP URLs to identify the malicious HTTP packets which are later dropped by the system with alerts.
The HTTP URLs need to be initially normalized before applying any blacklisted URL filtering mechanism [19]. For example, the HTTP URLs are made protocol agnostic by removing 'http://' or 'https://' from the URL as part of HTTP normalization. These steps are done in order to standardize all the similar URLs to a canonical form which in turn improves the URL filtering efficiency. The HTTP URLs are also compared against a predefined set of attack patterns to identify any OWASP application vulnerability.

4) Content Access Policy Management: Every organization has it's own set of policies which controls the traffic in their organization network. The type of content that can be accessed within a network can be controlled by using content type filtering mechanisms. Every HTTP response header contains a header field named 'Content-Type' which defines the type of data contained in the response which can be inspected to identify the content and filter traffic as per defined policies.

IV. Test setup and Results

The details of test setup, tools used and results obtained are explained below.

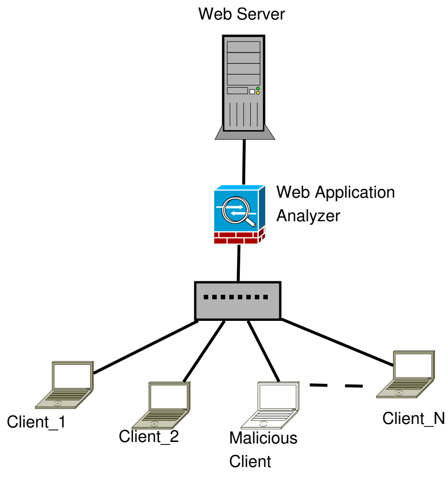


Fig. 3: Test setup used.

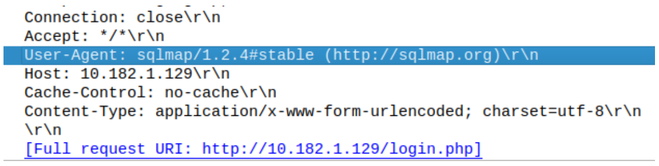
A. *Test setup*

The test-setup used for evaluating the functionality of the web application analyzer is shown in Fig. 3. As shown in the figure, a web server and client machines are connected through a switch. Upon requests, these client machines can access the web pages from the server. All requests to the web server are verified by the web application analyzer which then forwards only the genuine packets to the server. One of the client machines is installed and configured with web application scanners, misconfiguration detection and SQL injection tools. The malicious client machine is installed with Kali Linux operating system. The tools are used by the malicious client to generate malicious requests to the web server. Details of the tools and their configurations are described next.

1) Tools Used: The scanning tools used and their descriptions are summarized in Table III. The test-setup uses an Apache web server, Apache/2.4.37 which is installed in an Ubuntu 18.04.3, Kernel version Linux 4.15.0-66-generic machine with default configuration and web pages. The malicious client machine is running Kali Linux operating system version 2019.3 with kernel version 5.2.0.

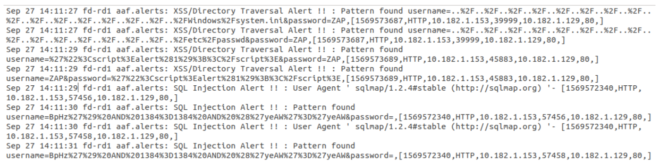
TABLE III: Tools used and Patterns used for identification

Tool used	Tool Usage	Default User-Agent String
sqlmap 1.2.4	SQL Injection, Database finger printing, enumeration	sqlmap/1.2.4#stable (http://sqlmap.org)
OWASP ZAP	Web application scanning, Misconfiguration detection, Information Disclosure, Command Execution	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
nikto 2.1.5	Web application scanning, Misconfiguration detection, Information Disclosure, Command Execution	Mozilla/5.00 (Nikto/2.1.5) (Evasions:None) (Test:getinfo)



```
Connection: close\r\n
Accept: */*\r\n
User-Agent: sqlmap/1.2.4#stable (http://sqlmap.org)\r\n
Host: 10.182.1.129\r\n
Cache-Control: no-cache\r\n
Content-Type: application/x-www-form-urlencoded; charset=utf-8\r\n
\r\n
[Full request URI: http://10.182.1.129/login.php]
```

Fig. 4: Wireshark console showing User Agent string for sqlmap.



```
Sep 27 14:11:27 fd-rd1 aaf.alerts: XSS/Directory Traversal Alert !! : Pattern found username=..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..
%2F..%2F..%2F..%2F..%2F..%2F..%2FWindows%2Fsystem.ini&password=ZAP,[1569573687,HTTP,10.182.1.153,39999,10.182.1.129,80,]
Sep 27 14:11:27 fd-rd1 aaf.alerts: XSS/Directory Traversal Alert !! : Pattern found username=..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..
%2F..%2F..%2F..%2F..%2F..%2F..%2Fetc%2Fpasswd&password=ZAP,[1569573687,HTTP,10.182.1.153,39999,10.182.1.129,80,]
Sep 27 14:11:29 fd-rd1 aaf.alerts: XSS/Directory Traversal Alert !! : Pattern found
username=%27%22%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E&password=ZAP,[1569573689,HTTP,10.182.1.153,45883,10.182.1.129,80,]
Sep 27 14:11:29 fd-rd1 aaf.alerts: XSS/Directory Traversal Alert !! : Pattern found
username=ZAP&password=%27%22%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E,[1569573689,HTTP,10.182.1.153,45883,10.182.1.129,80,]
Sep 27 14:11:29 fd-rd1 aaf.alerts: SQL Injection Alert !! : User Agent ' sqlmap/1.2.4#stable (http://sqlmap.org) '- [1569572340,HTTP,
10.182.1.153,57456,10.182.1.129,80,]
Sep 27 14:11:30 fd-rd1 aaf.alerts: SQL Injection Alert !! : Pattern found
username=BpHz%27%29%20AND%201384%3D1384%20AND%20%28%27yeAW%27%3D%27yeAW&password=,[1569572340,HTTP,10.182.1.153,57456,10.182.1.129,80,]
Sep 27 14:11:30 fd-rd1 aaf.alerts: SQL Injection Alert !! : User Agent ' sqlmap/1.2.4#stable (http://sqlmap.org) '- [1569572340,HTTP,
10.182.1.153,57458,10.182.1.129,80,]
Sep 27 14:11:31 fd-rd1 aaf.alerts: SQL Injection Alert !! : Pattern found
username=BpHz%27%29%20AND%201384%3D1384%20AND%20%28%27yeAW%27%3D%27yeAW&password=,[1569572340,HTTP,10.182.1.153,57458,10.182.1.129,80,]
```

Fig. 5: Application vulnerability Detection Alerts.

B. *Results*

The web application scanning tools like nikto [20], Zed Application Proxy (ZAP) [21] and sqlmap [22] were used to launch application vulnerability attacks on the server. Nikto is an web application scanning tool which is used to identify the directory indexing, default installation files, vulnerable scripts and other issues. ZAP is another web application security scanner tool. The URL of the web server was given as input to the web application scanning tools to analyze different vulnerabilities. To scan the SQL injection vulnerability, we used the sqlmap tool.
During the SQL injection attack scan carried-out using sqlmap tool, the system detected default user agent of sqlmap tool. The screen shot of the 'User-Agent' string of sqlmap tool is shown in Fig.4. By including this string in the blacklisted user agent list, the SQL injection attack using sqlmap tool can be identified.
As the tools like sqlmap are used to identify the SQL injection vulnerability, such requests contain this user agent string which can be monitored to identify a malicious user. However, many tools like nikto provides a feature which can alter the User-Agent value to older version browsers while crafting the packets. In such cases, the malicious requests can be identified only by comparing with the attack patterns.
A snippet of alerts generated by the proposed system is shown in Fig.5. From the alerts, we can observe that, the system is able to identify directory traversal, cross site scripting and SQL injection attacks with the help of attack patterns and user agent strings.

V. Conclusion and Future Works

This paper presents a frame work for policy enforcement and attack detection for web applications. The proposed model is focused to detect injection, misconfiguration and directory traversal attacks. Enforcement of policy on web access are also discussed as part of the framework. Policy related to the website access using URL, content and client application level access can be monitor and enforce using the proposed framework. Results obtained from the test-setup shows that, the proposed system is able to detect different web attacks with the attack and tool details. As part of future activities, we are planning to carryout exhaustive tests using different attack tools and benchmark the system's detection capabilities. Moreover, we would like to extend this work for HTTPS based web application monitoring, analysis and policy enforcement.

ACKNOWLEDGMENT

This work has been carried out as part of the project funded by Ministry of Electronics and Information Technology (MeitY), Government of India.

REFERENCES

[1] OWASP Top 10 Application Security Risks - 2017. [Online]. Available: https://www.owasp.org/index.php/Top_10-2017_Top_10

[2] OWASP Foundation: The free and open software security community. [Online]. Available: https://www.owasp.org/index.php/Main_Page

[3] O. B. Al-Khurafi and M. A. Al-Ahmad, "Survey of web application vulnerability attacks," in *2015 4th International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pp. 154–158, IEEE, 2015.

[4] H. Shahriar, S. North, and W.-C. Chen, "Client-side detection of sql injection attack," in *International Conference on Advanced Information Systems Engineering*, pp. 512–517, Springer, 2013.

[5] I. Balasundaram and E. Ramaraj, "An efficient technique for detection and prevention of sql injection attack using ascii based string matching," *Procedia Engineering*, vol. 30, pp. 183–190, 2012.

[6] D. Das, U. Sharma, and D. Bhattacharyya, "An approach to detection of sql injection attack based on dynamic query matching," *International Journal of Computer Applications*, vol. 1, no. 25, pp. 28–34, 2010.

[7] I. A. Elia, J. Fonseca, and M. Vieira, "Comparing sql injection detection tools using attack injection: an experimental study," in *2010 IEEE 21st International Symposium on Software Reliability Engineering*, pp. 289–298, IEEE, 2010.

[8] X. Xie, C. Ren, Y. Fu, J. Xu, and J. Guo, "Sql injection detection for web applications based on elastic-pooling cnn," *IEEE Access*, vol. 7, pp. 151475–151481, 2019.

[9] X. Guo, S. Jin, and Y. Zhang, "Xss vulnerability detection using optimized attack vector repertory," in *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 29–36, IEEE, 2015.

[10] M. Babiker, E. Karaarslan, and Y. Hoscan, "Web application attack detection and forensics: A survey," in *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, pp. 1–6, IEEE, 2018.

[11] D. Mitropoulos, P. Louridas, M. Polychronakis, and A. D. Keromytis, "Defending against web application attacks: approaches, challenges and implications," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 188–203, 2017.

[12] W. Yang, W. Zuo, and B. Cui, "Detecting malicious urls via a keyword-based convolutional gated-recurrent-unit neural network," *IEEE Access*, vol. 7, pp. 29891–29900, 2019.

[13] H. Choi, B. B. Zhu, and H. Lee, "Detecting malicious web links and identifying their attack types.," *WebApps*, vol. 11, no. 11, p. 218, 2011.

[14] Z. Tian, C. Luo, J. Qiu, X. Du, and M. Guizani, "A distributed deep learning system for web attack detection on edge devices," *IEEE Transactions on Industrial Informatics*, 2019.

[15] S. Alcock and R. Nelson, "Libprotoident: traffic classification using lightweight packet inspection," *WAND Network Research Group, Tech. Rep.*, 2012.

[16] M. Perényi, T. D. Dang, A. Gefferth, and S. Molnár, "Identification and analysis of peer-to-peer traffic," *Journal of Communications*, vol. 1, no. 7, pp. 36–46, 2006.

[17] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 5–16, 2007.

[18] N. Hubballi and M. Swarnkar, "*bitcoding*: Network traffic classification through encoded bit level signatures," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2334–2346, 2018.

[19] Enable the Blocking of URLs. [Online]. Available:https://docs.umbrella.com/deployment-umbrella/docs/custom-url-destination-list-how-to

[20] nikto web scanner. [Online]. Available: https://cirt.net/Nikto2

[21] OWASP Zed Attack Proxy Project. [Online]. Available: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

[22] sqlmap Automatic SQL injection and database takeover tool.[Online]. Available: http://sqlmap.org/