**Communications and Network, 2016, 8, 170-178**
Published Online August 2016 in SciRes. http://www.scirp.org/journal/cn
http://dx.doi.org/10.4236/cn.2016.83017

# Web Threats Detection and Prevention Framework

**Osama M. Rababah[1], Ahmad K. Al Hwaitat[2], Saher Al Manaseer[2], Hussam N. Fakhouri[2], Rula Halaseh[1]**

[1]Department of Business Information Technology, The University of Jordan, Amman, Jordan
[2]Department of Computer Science, The University of Jordan, Amman, Jordan
Email: o.Rababah@ju.edu.jo, alaseeljo@yahoo.com, saher@ju.edu.jo, hu_ss_am@yahoo.com, rula.alhalaseh@gmail.com

Received 29 February 2016; accepted 1 August 2016; published 4 August 2016



## Abstract

**The rapid advancement in technology and the increased number of web applications with very short turnaround time caused an increased need for protection from vulnerabilities that grew due to decision makers overlooking the need to be protected from attackers or software developers lacking the skills and experience in writing secure code. Structured Query Language (SQL) Injection, cross-site scripting (XSS), Distributed Denial of service (DDos) and suspicious user behaviour are some of the common types of vulnerabilities in web applications by which the attacker can disclose the web application sensitive information such as credit card numbers and other confidential information. This paper proposes a framework for the detection and prevention of web threats (WTDPF) which is based on preventing the attacker from gaining access to confidential data by studying his behavior during the action of attack and taking preventive measures to reduce the risks of the attack and as well reduce the consequences of such malicious action. The framework consists of phases which begin with the input checking phase, signature based action component phase, alert and response phases. Additionally, the framework has a logging functionality to store and keep track of any action taking place and as well preserving information about the attacker IP address, date and time of the attack, type of the attack, and the mechanism the attacker used. Moreover, we provide experimental results for different kinds of attacks, and we illustrate the success of the proposed framework for dealing with and preventing malicious actions.**

## Keywords

**SQL Injection, XSS, DDoS Attack, Suspicious User Behavior, Web Applications**

**How to cite this paper:** Rababah, O.M., Al Hwaitat, A.K., Al Manaseer, S., Fakhouri, H.N. and Halaseh, R. (2016) Web Threats Detection and Prevention Framework. *Communications and Network*, **8**, 170-178.
http://dx.doi.org/10.4236/cn.2016.83017

## 1. Introduction

The companies growing dependence on the use of web applications in their daily work came along with the massive development of the internet and the web applications where the web became the main link that connected all users all over the world as well as the place where data about the internet users were stored in databases [1]. With that advancement in technology, many security threats have arisen on daily basis [2] as the databases contain sensitive and private data about users such as credit card numbers, passwords, and money transaction information which if exposed can cause great deal of financial loss and damage to companies as well as losing the user trust and disrupting their daily operations and for that reason the security of information is a primary concern for all website and company owners. Another example regards the firewall method that deals with port 80 without checking the payload packet information, while security on application layer was overlooked and now became a necessity to reduce and mitigate malicious attacks [3].

The increased number of attack success stories [4] increased the need to find the causes for such increase and find new methods to mitigate, detect and prevent such attacks and also secure databases and private data provided by the users. Furthermore, we need to educate the developers and the company owners and decision makers of the importance of taking protective measures during the development of web applications.

This paper aims to find a solution for the challenges and limitations we face to protect confidential information and to improve the methods of prevention and detection of web attack attempts through the proposed framework. We also intend to achieve better interactivity and performance to protect web applications from the malicious users and also prevent those users from injecting malicious web content using the vulnerabilities that developers have overlooked.

In Section 2 of this paper, we talk about related work and previous research in this area. Section 3 is a demonstration of the phases in the framework we suggest. Section 4 contains the experimental results and an evaluation of these results. Section 5 compares the framework we suggest with related work. Finally Section 6 contains the conclusion of this paper.

## 2. Related Work

In this section we talk about related work done to detect and prevent various web threats. Firstly we refer to Duraisamy's paper [5] where he suggests to filter JavaScript in server side in web applications to protect against information leakage from the user's environment and protect against XSS which is difficult to detect and prevent in web applications and attackers use it to execute malicious code in the user's browser. This server side solution offers to mitigate and protect against XSS by decreasing number of communication alert prompts. The server here also acts like a web proxy to keep away XSS attacks. We also find Thopate's paper [6] where the focus is mainly on detecting and preventing XSS and SQL injection and also we see solutions to preventing such attacks, and they also provide a filtration method to mitigate SQL injection.

Mahapatra [7] proposes a new technique to keep Java web applications secure from XSS attacks by developing a framework that uses the pattern matching approach which can be used on any existing java web application with no modification on the source code. Mahapatra's framework is also composed of a request/response analyzer model and a modifier model.

Pratik [8] suggests a system to verify all kinds of SQL injection attacks and as well as preventing stored procedure attacks and XSS attacks. The developed system recorded and analyzed all input strings responsible for the query implementation.

Tang [9] presents an approach to indicate the meaning of XSS attacks with URL analysis as the base of the approach. The base of the technique suggests that valid JavaScript syntax tree can be produced from a part of the URL. The mechanism analyzes fundamental technical challenges and their executions. A sum of 13.000 URLs which contains XSS exploits helped in doing an evaluation as the XSS exploits were gathered from XSSed.com, and 800 URLs which were collected from websites with a social platform. The basics of this study are: the method to produce a correct JavaScript syntax tree and measure its depth; if it becomes more than a user-defined threshold, the URL is considered doubtful. The second is the immunity of URLs through analyzing their structure.

Balasundaram [1] develops a mechanism to detect and prevent SQL injection using both static and dynamic analysis. In the stage of static analysis, the prevention method is displayed in a level of three phases: viperous text detector, field constraint validation, and ultimate Structured Query Language Validation. In the stage of

runtime validation, the data which the user inputs is checked (*i.e.* validated) with all the mentioned steps, and the result shows if the information is safe or not. Balasundaram [1] mentions two more methods for the purpose of the detection and prevention of SQL injection attacks, a web service technique in [10] and an ASCII based string matching approach in [11].

Roy [12] presents another method for preventing SQL injection attacks through using URL filters, which legitimize web forms input. Using only one filter to legitimize the input makes this approach more competent and Flexible.

In paper [13] the author presents a Double guard Intrusion Detection System (IDS) that models user sessions' network behavior across both the front-end web server and the back-end database which enables us to conquer threats better than any other independent IDS can detect. The double guard IDS was implemented using apache web server, PHP and MySQL and lightweight visualization.

**Manmadhan [14]** presents a method to stop SQL injection attacks on web applications using SQLI flow (wild characters & taking advantage of commands made by SQL) where in the detector of tame-card and stop tame-card attacker transact-SQL statements are directly prevented from inputting invalid data. The technique checks the user input with legal correct database does not have an impact on database and afterwards Service Oriented Authentication technique takes place to access the allowed input. By doing so, we improve the server side's performance.

**A dynamic query structure validation technique in Lambert [15]'s paper where he** suggests validating before implementing the intended SQL query structure. The technique is performed by checking the semantics of the query. The techniques explores SQL injection by operating a good query from the last SQL query operated by the application and the user's input and then comparing with the semantic of safe and the SQL queries. The aim here is to stop stored procedure attacks which make it difficult to get query structure before implementation.

Garg [16] mentions a new algorithmic attempt which aims to detect SQL injection attacks and avoid them early. The technique consists of the following steps: collection data set which is meant for training, and also a cheat sheet for the analysis, then data set must be trained for the interrogation, generation number three of the patterns and keys, fourth is that each parameter gets a full analysis then the proposed model is implemented on the training set of data, the final step is to fetch the results and interpretation data.

## 3. Web Threats Detection and Prevention Framework Phases

**WTDPF Initial Phase**

The initial phase consists of many steps that have to be performed before data reaches the Rule based filter and action taken based on the signature component. The first step in this phase is to analyze the user request which is done by analyzing the data received from the user based on analyzing the HTTP-HTTPS Interceptor and HTTP head analysis, WTDPF captures all traffic directed to the web application, the packet payload will be logged and analyzed to detect and prevent the possible attack.

**The Second Phase**

The WTDPF role starts whenever the user submits Input to the Web-app page, the request will be sent from the client browser to the server over HTTP-HTTPS protocol. The WTDPF will then intercept and extract the HTTP request using the Input checker component, and then analyze the header type and the content of the payload contained such as (User Agent-Submitted Data-Requested parameters) and logs the payload with the user's info along with the date, time and type of the request as a first Step.

The next step is to extract the payload of the request to be compared with the set of security rules and signatures assigned to the signature-database and then determine whether it is a normal and acceptable input or malicious input which will be determined with different actions later.

If the user's behavior is accepted the website will continue normally with no interruption and all of the rest of their actions will be logged as a feedback in the logging report and the response to the user will be sanitized and any errors that may occur from a Web-app Bug or Internal server error will be eliminated.

As for the other scenario where one of the users tries to execute a malicious content or start any of the following types of attacks:

- Injection based attacks (SQL injection, XSS, command injection).
- Denial of Service attacks (Dos Attack, Dos Attacks by Exploits, DDos Attack).

- Other suspicious behavior (Brute Forcing, Fuzzy processing, Directory enumeration, number of visitors and their origins).

If the attack is identified in the set of rules and the Payload matched one of the attacks signatures in the signature-database then the security mechanisms using the decision making component of either the acceptable input path will be followed or the bad malicious input path which will trigger the alert phase to start.

**The Alert Phase**

The Alert Phase starts after the decision making component if there is any malicious input from the analyzed user sent request.

**The Response Phase**

The aim of this phase is to make the WTDPF reliable and effective system, by preventing the attacking from the attacker through taking a fast response either automatic or manual. To keep the web application working in a safe and secure environment.

The Response Phase has two phases, the first phase is the alert phase where the Attack info will be logged including:

- The Attacker's IP.
- Country.
- The time and date of the attack.
- Number of attempts.
- Type of Attack.

The user will receive a message to phone number administrator also an Email Notification which will include the attack info mentioned before and the permission to Take counter measures such as Displaying Warning message—Blocking the attacker IP Address—the range of IP's in the estimated region—flush the session that the attacker is using. After 120 seconds without the users response those actions will be taken automatically and the output will be changed for the attacker as a sanitization mechanism to compress any error caused by the attacker's actions. The sanitization mechanism is used instead of rejecting or blocking the malicious input to the web-app. and it changes the malicious Input/output into an acceptable format.

## 4. Evaluation and Experimental Results

### 4.1. Testing the Framework

We will experiment every component of the framework using different type of user HTTP requests**.** In WTDPF Testing section we will experiment the framework's ability to analyze the attacker in order to measure the requests effectiveness of the WTDPF input checker component which we will test by providing user input samples and the checker will check if the user sent HTTP requests that contains any method of attack or whether the user sent normal input. The evaluation is divided into three parts. The first section will check normal user inputs and the method the input checker deals with such user input. The first part of testing the framework involves samples that show possible types of normal user input and shows the result of the WTDPF during processing and analyzing sent requests.

Testing the WTDPF with attacks SQL injection, XSS.

#### 4.1.1. SQL Injection Attack

To test the framework with the SQL injection attack a list of possible forms of the attack has been experimented.

#### 4.1.2. Experiment One SQL Injection "Command or Query"

The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization. An example of SQL injection is modifying the "id" parameter value by the attacker in the browser to send: (or "1" = 1) in the SQL query will change the meaning of both queries to return all the records from the accounts table. Amore dangerous attack could modify data or even invoke stored procedures. To test sql injection query on an example website we perform the query as illustrated in **Figure 1**.

HTTP request is sent to server. The framework will take that request and read the payload for the received packet in order to check it using the input checker component. In this experiment the framework check found that the URL contain the ("1" = 1) characters and it matches the entry in the database signature. And this mean that HTTP request contains an illegal signature which indicates that there is a possible attack for the website and

```
http://securitysystemframework.com/finde
x.php?main=Contact&Lang=1' or '1'='1
```

**Figure 1.** SQL injection query.

here the framework will store the illegal attack, it will store the number of attack attempts, the IP Address, date and time and country of the attacker in the log file.

We use a tool named burp suite to view the submitted data as we see in **Figure 2**.

Sending the malicious signatures in the HTTP request will be stored in the framework's log file.

Repeating the SQL injection attack twice the user will trigger the alert component and response component will send a message to the administrator's mobile notifying him of possible attack in order to take the suitable action for this attack.

If administrator decides to not take an action during 120 second then the framework will take an action to block the IP address of the attacker and ban it from accessing the website. In order to control the attack and keep the website working through preventing the attacker from attacking the database that can be accessed through SQL injection. Another point of view the framework also sends an email to the administrator with information about the attack and the action performed to prevent the attack. As shown in **Figure 3**.

#### 4.1.3. XSS Attack

Testing the framework with XSS attack which occurs simply when the web application accepts a malicious java code which can lead the attacker to tamper with the web application's behavior and do multiple tasks harming the users such as (Session Hijacking—Defacing—Various types of client side attacks).

#### 4.1.4. Malicious Code

To test the framework with malicious code XSS is performed as part of a client side attack and session hijacking attack through a sample page body form shown in **Figure 4**, using burpsuite we can find HTTP header with post value.

Also the request is sent to the server. The framework will then take the HTTP request and read the payload for the received packet in order to check it using the input checker component. The role of the input checker component here is to test the HTTP. The framework will store the illegal malicious code in the WTDPF log as shown in **Figure 5** and will also store the number of attacks performed, the IP address, date and time and country of the attacker in the log file.

In order to take proper action for this attack, the responsible component will respond by sending a message to the administrator to inform him of the possible attack. The message here will be sent to the administrator after the first trial to attack the website and will take an automatic reaction to block the IP address of the attacker and ban it from accessing the website as shown in **Figure 3**.

If the administrators don't take an action during the first 120 second after sending the message then the framework will also take action as mentioned earlier. Another point of view the framework will also send an email to the administrator with information about the attack and the action taken to prevent the attack.

## 5. Comparing WTDPF with Related Work

Many researches are on methods detecting vulnerabilities and techniques to illustrate the common attacking methods such as SQL injection, XSS, DDoS and suspicious user behavior. Some of these methods and researches are mentioned in the related work section. The proposed framework will not be compared to prevent malicious software and ready tools because they hide their technique in a concept known by the black box to deal with different attack methods. Here the framework will be compared with other researches that describe methods for detection and prevention methods for SQL injection, XSS, DDoS, and suspicious user behavior.

The comparison is based on the following features: type of attack that is detected and prevented, runtime detection and prevention, implementation language, action and response type, message alert speed, method of communication with the administrator, blocking attacker, saving log file and history of attack, data saved for the attacker and determining the attack strength (**Table 1**).

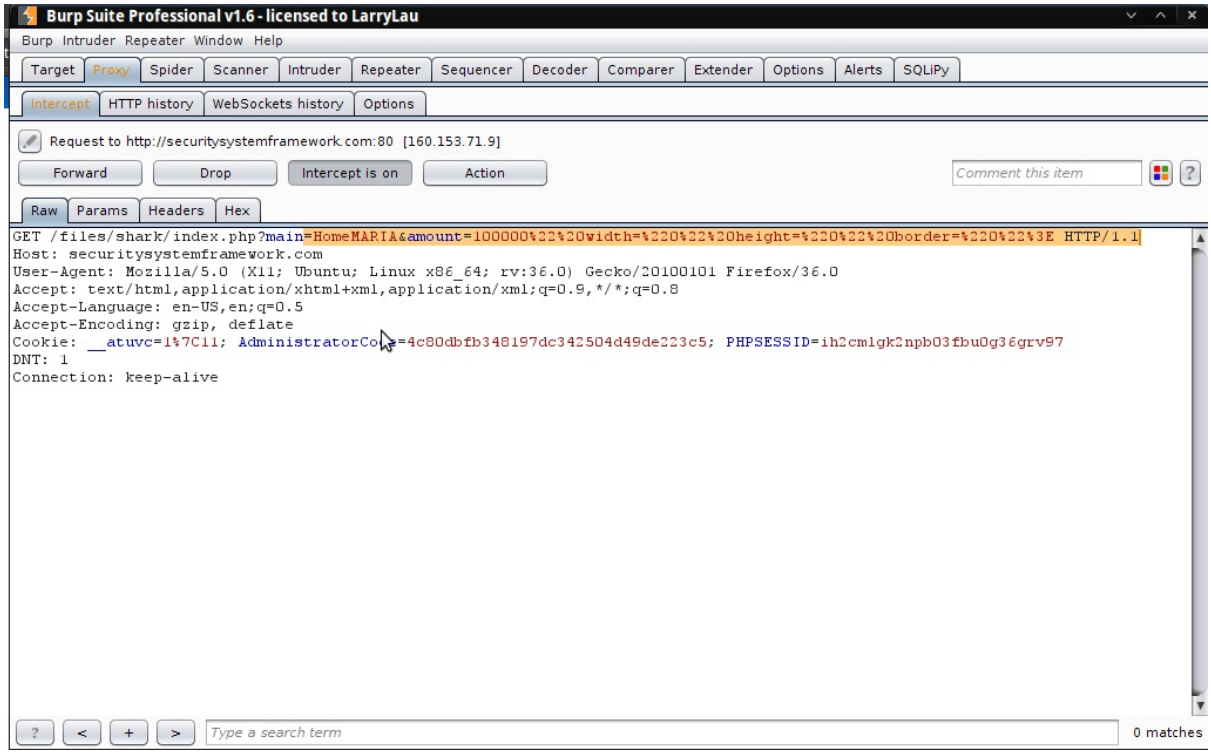


**Figure 2.** Burpsuite HTTP analysis.

**Table 1.** Comparison between the framework other researches.

| Research / Feature | Tzvi Chumash, 2009 [17] | Praveen Kumar, 2013 [2] | Balasundaram and Ramaraj, 2011 [1] | Zhi'hua Tang *et al.*, 2012 [9] | NAVALE *et al.* , 2014 [18] | WTDPF |
|---|---|---|---|---|---|---|
| **Type of attack that is detected and prevented** | SQL injection | SQL injection | SQL injection | XSS | DDos | SQL injection, XSS, DDos, suspicious user behavior |
| **Runtime detection and prevention** | Yes | Yes | No | yes | Yes | Yes |
| **Implementation language** | C/C++ | Microsoft .Net framework 3.5 & Visual Studio 2008 | Php | JavaScript | php, Hadoop framework | Php |
| **Action and response type** | Automatically | Automatically | No action | Automatically | Automatically | Manual/Automatic |
| **Message Alert speed** | Instant | Instant | No alert Message | No alert Message | No alert Message | Instant |
| **Communication with administrator** | Web based | Web based | Not available | Web based | Web based | SMS message/Email/web based |
| **Blocking attacker** | No | No | No | no | No | Yes |
| **Saving log file &attack history** | Yes | Yes | Yes | yes | No | Yes |
| **Data saved for the attacker** | Yes | Yes | Yes | yes | No | Yes |
| **Determine the attack strength** | No | Yes | No | no | No | Yes |


**Figure 3.** Illustration of the message sent to the administrator.

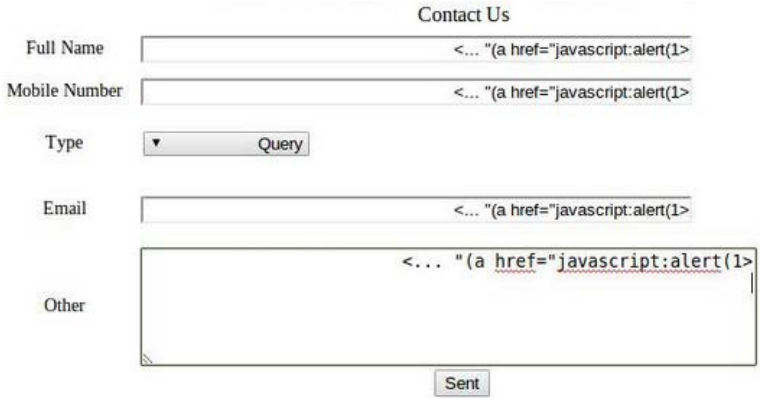


**Figure 4.** XSS attack using malicious code.

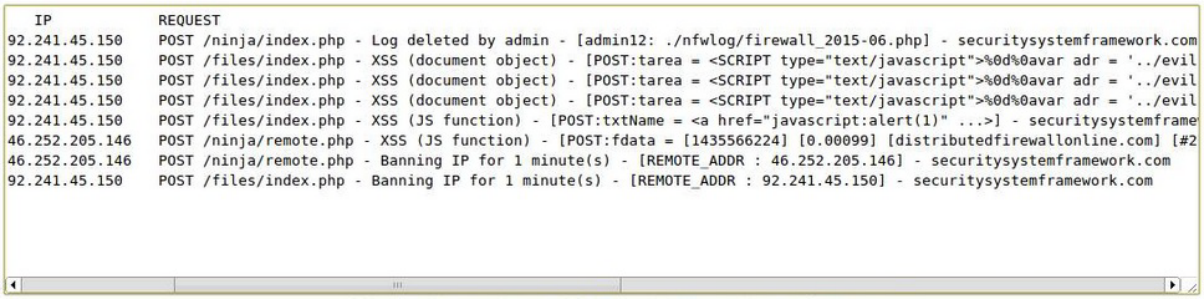


**Figure 5.** Logging process.

The comparison results shows that the proposed framework has some advanced features that does not exist in other types of frameworks such as the instant alert message that is sent to the user's phone. The framework has two types of alert methods email and SMS. The proposed framework also covers many types of common attack methods such as SQL injection, XSS, DDos, suspicious user behavior. Only one research of the compared methods which is done by Khoch are [19] mentions a feature of determining the attack strength as well as covering many types of attacks unlike other papers that covered only one type of attacks and also mentioning the blocking feature. Our framework covers and combines all those three precautious and preventative features and measures in [19] which helps in preventing the malicious action from taking place and protects the website accordingly.

## 6. Conclusions

This research presented a new framework for the detection and prevention of common attack methods: SQL injection, XSS, DDos and user suspicious behavior. Aside from being compatible with shared hosting accounts and ability to implement it for any web application, the framework was able to detect and prevent the attack while the attacker was performing the malicious action. The framework checked the used input, analyzed the HTTP request and then took action either by automatically banning the IP address of the attacker or manually by sending a message to the administrator informing him about the possible attack and by sanitizing the requests before the attacked reaching the website sensitive data and providing a real time detection during the attack.

The framework's components and phases were explained to show how the components interacted with each other to prevent and detect the attacks. Moreover, the framework consisted of four main phases that started with the input checking phase, then the signature based action component, and in the third phase was the alert phase and finally the response phase.

The testing of the framework was experimented using different types of attacks SQL injection, DDos and XSS and the results showed that the proposed framework was efficient in detecting and preventing the attacks through banning the IP address of the attacker automatically or through sending a message to the administrator with information about the attacker at the runtime process. The sent message through the alert and response component also allowed the administrator to perform an action manually to stop the attacker from going further with the attack action.

The framework also stored a log file about the attack with the type of the attack, the IP address of the attacker and date and time of attack. Furthermore, the framework classified the type of attack if it was strong or moderate attack.

## References

[1] Balasundaram, I. and Ramaraj, E. (2011) An Approach to Detect and Prevent SQL Injection Attacks in Database Using Web Service. *International Journal of Computer Science and Network Security*, **11**, 197-205.

[2] Kumar, P. (2013) The Multi-Tier Architecture for Developing Secure Website with Detection and Prevention of SQL-Injection Attacks. *International Journal of Computer Applications*, **62**, 30-36.

[3] Uddin, M., Rehman, A., Uddin, N., Memon, J., Alsaqour, R. and Kazi, S. (2013) Signature-Based Multi-Layer Distributed Intrusion Detection System Using Mobile Agents. *International Journal of Network Security*, **15**, 79-87.

[4] Darwish, F., *et al.* (2011) The Impact of the New Web 2.0 Technologies in Communication, Development, and Revolutions of Societies. *Journal of Advances in Information Technology*, **2**, No. 4.

[5] Duraisamy, A., *et al.* (2013) A Server Side Solution for Protection of Web Applications from Cross-Site Scripting Attacks. *International Journal of Innovative Technology and Exploring Engineering* (*IJITEE*), **2**, 130-137.

[6] Thopate, P., *et al.* (2014) Cross Site Scripting Attack Detection & Prevention System. *International Journal of Advanced Research in Computer Engineering & Technology* (*IJARCET*), **3**, 4035-4039.

[7] Mahapatra, R., *et al.* (2012) A Pattern Based Approach to Secure Web Applications from XSS Attacks. *International Journal of Computer Technology and Electronics Engineering* (*IJCTEE*), **2**, 196-203.

[8] Pratik, S. and Gheewala, J. (2014) Detection and Prevention of SQL Injection Attacks. *International Journal of Engineering Development and Research*, **2**, 2660-2666.

[9] Tang, Z., *et al.* (2012) Identifying Cross-Site Scripting Attacks Based on URL Analysis. *International Journal of Engineering and Manufacturing* (*IJEM*), **2**, 52-61.

[10] Balasundram, I. and Ramaraj, E. (2013) Prevention of SQL Injection Attacks by Using Service Oriented Authentication Technique. *International Journal of Modeling and Optimization*, **3**, 302-306.

[11] Balasundaram, I. and Ramaraj, E. (2011) An Efficient Technique for Detection and Prevention of SQL Injection Attack Using ASCII Based String Matching. *International Journal of Engineering Development and Research*, **2**, 2660-2666.

[12] .Roy, S., *et al.* (2012) A Novel Approach to Prevent SQL Injection Attack Using URL Filter. *International Journal of Innovation*, *Management and Technology*, **3**, 499-502.

[13] Ramesh, D. and Kumar, R. (2012) Double Guard Approach for Detecting Intrusions in Multitier Web Applications. *International Journal of Communication Network and Security*, **2**, 203-213.

[14] Manmadhan, S., *et al.* (2012) A Method of Detecting SQL Injection Attack to Secure Web Application. *International Journal of Distributed and Parallel Systems*, **3**, 1-6.

[15] Lambert, N. (2010) Use of Query Tokenization to Detect and Prevent SQL Injection Attacks. *Computer Science and*

*Information Technology* (*ICCSIT*), **2**, 438-440.

[16] Garg, S. and Narula, P. (2014) A Novel Approach and Implementation of Secured Algorithm against SQL Injections. *International Journal of Enterprise Computing and Business Systems*, **4**, 1-7.

[17] Chumash, T., *et al.* (2009) Detection and Prevention of Insider Threats in Database Driven Web Services. *IFIP International Federation for Information Processing*, **300**, 117-132. http://dx.doi.org/10.1007/978-3-642-02056-8_8

[18] Navale, G., *et al.* (2014) Detecting and Analyzing DDoS Attack Using Map Reduce in Hadoop. *International Journal of Industrial Electronics and Electrical Engineering*, **2**, 56-88.

[19] Khochare, K. and Dr. Meshram, B. (2012) Tool to Detect and Prevent Web Attacks. *International Journal of Advanced Research in Computer Engineering & Technology*, **1**, 375-378.