Journal of Graphic Era University
Vol. 6, Issue 2, 183-196, 2018
ISSN: 0975-1416 (Print), 2456-4281 (Online)

Analyzing and Defending Web Application Vulnerabilities through Proposed Security Model in Cloud Computing

Prabhat Bisht[1*], Devesh Pant[2], Manmohan Singh Rauthan[3]

[1,2]Uttarakhand Technical University Dehradun, India
[3]Department of Computer Science and Engineering
H. N. B. Garhwal University, Srinagar (Garhwal), India
[*]Corresponding author: prabhat.bisht@gov.in

(Received August 23, 2017; Accepted May 15, 2018)

Abstract
Security of web applications from attackers is one of a challenging task in cloud computing infrastructure. Unsecure source code is one of a top reason for cyber-attacks, due to which valuable data like username, password, credit card information or even personal information related to aadhar enabled biometric system, can be compromised. Most of the vulnerabilities in web application source code is related to Open Web Application Security Project (OWASP), these vulnerabilities are SQL, NoSQL, LDAP Injection, Broken Authentication, Sensitive data exposure, XML external entities, broken access control, security misconfiguration, Cross site scripting (XSS), Insecure deserialization and insufficient monitoring and logging etc. Vulnerable web applications are the hot spot for hackers. According to Symantec's Internet Security Threat Report published in July 2017, more than 2 lakh attacks against websites occur each day and more than 76% websites hosted in cloud contain un-patched vulnerabilities. This paper proposes a new innovative conceptual security tool name as SECUREWEB. This tool will detect vulnerabilities in web application source code and automatically patch detected vulnerabilities and return secure source code free from any identified vulnerabilities. This tool works on the concept of proxy based source code analyzer SECUREEYE model for detecting OWASP Top 10 vulnerabilities and SECURESOLUTION model for auto patching of detected vulnerabilities.

Keywords- Vulnerabilities; Security; Open Web Application Security Project (OWASP); Hypertext Transfer Protocol (HTTP); Threats; Cloud; Virtual Machine (VM).

1. Introduction
After digital India initiative most of the critical information like biometric information, credit card information, PAN card information, digitally signed document through eSign technology and many more information are stored in cloud based virtual machines through different web applications. Data security or information leakage becomes one of the biggest challenges over internet. Attackers are always ready to execute attack vectors so that to find out the loopholes in web or mobile applications and if web applications source code have OWASP (2015) vulnerabilities then such applications can be hacked. This paper aims to present deep analysis on detection and defending various OWASP vulnerabilities in web application source code and proposes a conceptual tool (SECUREWEB).

SECUREWEB works in two models SECUREYE and SECURESOLUTION.

SECUREEYE: source code analyzer works on HTTP request and HTTP response interceptor model while SECURESOLUTION: works on sanitization of input parameters to prevent SQL injection, Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF) and Phishing attacks.

2. Related Work

Most of the research work has been carried out to find the vulnerabilities in web application source code. Top 10 vulnerabilities in source code are related to SQL injection, Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF), Session hijacking, Phishing. These are the most common vulnerabilities which can be exploited by attackers in order to gain access to web server or web applications. For gaining access to web server or web applications attackers intercept HTTP request parameters through proxy based tool and inject malicious data in POST request variables. Malicious data when processed by web server results unexpected information, this information later exploited by attacker in order to gain server access.

To avoid this OWASP which is a free and open software security community identified Top 10 vulnerabilities. Common vulnerabilities are:

- Injection: Injection flaws such as SQL injection, NoSQL injection, LDAP injection occurs when malicious code is injected in SQL query string through attackers. Attackers intercept data passing from client web browser to server through proxy based interceptor tool like Burpsuit and manipulate data in between request and response, manipulated data is processed by web server leading to SQL, NoSQL, LDAP injection flaws.
- Broken Authentication: Mismanagement of username, password and session handling is often implemented incorrectly. Attackers compromise session key, unencrypted password or unencrypted critical information like credit card information, PAN card details to exploit server cluster and web application deployed on it.
- Sensitive data exposure: In most of the web application sensitive data like personal information, credit card information, and biometric data is not properly protected. Hacker can steal and manipulated weakly protected data which causes credit card fraud, identity theft. Sensitive data must be protected through data encryption mechanism.
- XML external entities: Poorly configured XML files if uploaded from client browser to server and if that XML file get manipulated in between Request and Response by hacker can result data loss and data violation. Hacker can execute remote code, internal port scanning, internal file access and even DOS attack through XML files to server.
- Security Misconfiguration: Misconfiguration of web server services like apache, tomcat, misconfiguration of firewall rule through open ports, incomplete or adhoc configuration of firewall, misconfiguration of HTTP header in web application source code etc. are basic causes of information security breach.

- Cross Site Scripting: Cross site scripting occurs when data from client to server is not properly validated or not properly escaped through html entities. Attackers can execute JavaScript by hacking HTTP request $_POST variable data and inject malicious code on it. Data if not properly validated and escaped before sending it to web server for processing can cause cross site scripting attack.

Research is going on regarding developing a mechanism to find out vulnerabilities in web application source code. In past, Fonseca et al. (2014) had developed Vulnerability and attack injection tool for an evaluation of web security mechanism using fault injection technique. Djuric (2013) had developed a SQL injection vulnerability detection tool using black box approach. Through this approach analyse HTTP request and response from the web server Nguyen-Tuong et al. (2005) proposed a fully automated approach for securely hardening web applications. It checks for tainted data and dangerous content came from untrustworthy sources. Huang et al. (2017) proposed a tool VULSCAN, this scanner automatically generates test data for revealing vulnerabilities in source code. Kankhare and Manjrekar (2016) proposed a cloud based system to sense security vulnerabilities of web application in open source private cloud IAAS.

3. Statement of Problem

Today when all the valuable information like personal information, credit card information, biometric related information are available in cloud through different web applications hosted in virtual machines over cloud then security of critical data over internet is a matter of concern. For keeping web applications free from cyber attacks and hackers, web application should be free from OWASP Top 10 vulnerabilities because attackers find out loopholes in web application source code and manipulate the code by intercepting the request and inserting malicious code in the application as a result web application behaves differently as they intended by the coders. In this paper what we tried to find out the most common vulnerabilities any web application can have through the proposed security model source code analyzer tool SECUEEYE (Model 1) and correspondingly proposed a solution for making PHP based web application free from vulnerabilities through the proposed patching model SECURESOLUTION (Model 2).

3.1 Methodology Adopted for Detection and Prevention of Vulnerabilities in Source Code

In this paper, a vulnerabilities analysis tool SECUREWEB is proposed which works on two models, SECUREEYE (Model 1) and SECURESOLUTION (Model 2)**,** for making web applications secure from cyber threats.

Model 1 is based on proxy based interceptor mechanism, which is used to analyze the current security mechanism of application by injecting possible vulnerabilities in the source code.

Model 2 provides solution for preventing web application from attacks by automatically patching top possible indentified vulnerabilities.

Model 1 Intercept HTTP request parameters through proposed proxy based source code analyzer tool SECUREEYE. It manipulates request parameters $_POST variables by injecting malicious code and then forwards manipulated values/variables to server, which are processed by the server and server sends back the processed information to client as HTTP response. HTTP Response is then analyzed for vulnerabilities and it is found that most of the web applications are attacked from cyber-threats because of the following vulnerabilities.

- Blind SQL injection, SQL Injection, MongoDB NoSQL Injection, LDAP injection.
- Cross Site Scripting XSS.
- Link Injection which facilitates cross site request forgery CSRF.
- Phishing through Frames.
- Weak Cryptographic Hash for encryption of valuable data like credit card information, biometric information, password etc.
- System information is leaked externally through use of in-built functions in programming languages.
- Misconfiguration and mismanagement of session related issues.

4. Proposed Architecture of Secure Web Vulnerabilities Analysis Tool

Attackers are always ready to inject attack vector on vulnerable source code in order to gain access over server. Attacks on web server and web application in cloud based virtual machines are possible from n number of hosts concurrently, so it is mandatory to detect attacks efficiently and defend affectively to reduce its impact on web applications. Successful penetration of malicious data can lead to great damage of the web application or even web server or application database leading to financial loss. This purposed tool works through two models.

Model 1-SECUREEYE is a proxy based HTTP request interceptor, which works in three stages:

a) **Monitoring Stage:** Intercept HTTP request.
   In this stage HTTP request is intercepted by attackers before request is send to web server for processing i.e. In between HTTP request and HTTP response.
b) **Injection Stage:** Injecting malicious code.
   In this stage attacker insert malicious data in request $_POST variables i.e. attacker manipulate $_POST variable data in between client and server i.e. inject vulnerable data as an attack vector for processing.
c) **Attack Stage*:*** Attack payload creation through penetration testing.

In this stage manipulated data is forwarded to server for processing and if Request $_POST variables are not properly escaped by html entities, can cause SQL injection and Cross site scripting attack.

Model 2- SECURESOLUTION which prevent web attacks by sanitizing $_POST variables.
a) Prevention Stage: Attack prevention by patching input variables.

The functionality of proposed conceptual tool (SECUREWEB) describing SECUREEYE and SECURESOLUTION is shown in Figure 1.

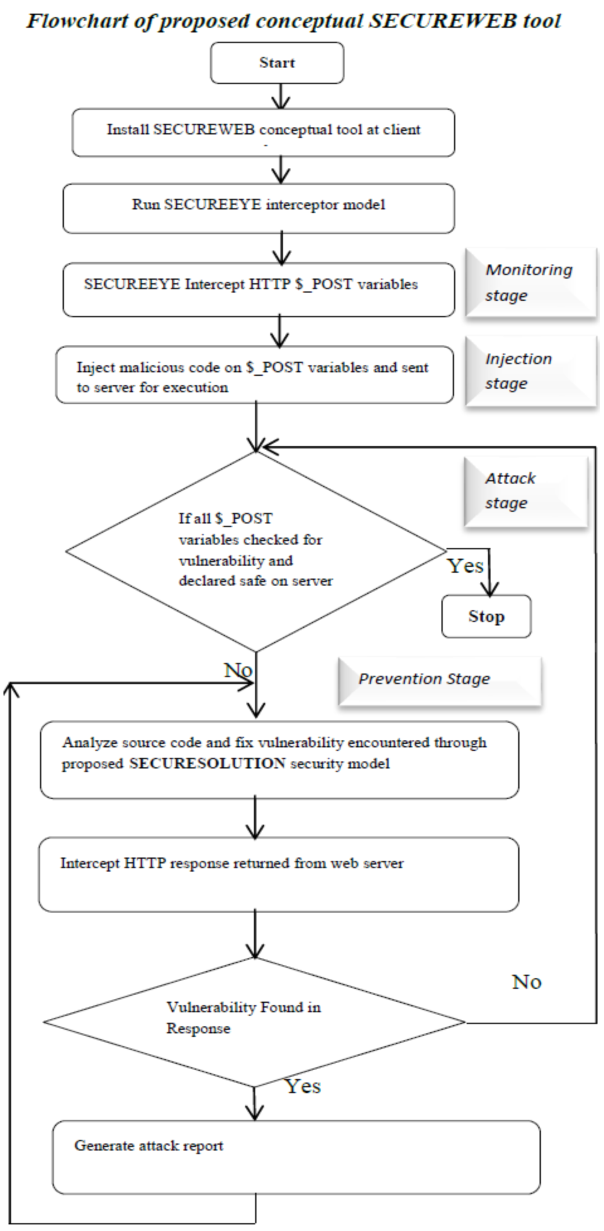

Flowchart of proposed conceptual SECUREWEB tool
Start
Install SECUREWEB conceptual tool at client
Run SECUREEYE interceptor model
SECUREEYE Intercept HTTP $_POST variables
Monitoring stage
Inject malicious code on $_POST variables and sent to server for execution
Injection stage
Attack stage
If all $_POST variables checked for vulnerability and declared safe on server
Yes
Stop
No
Prevention Stage
Analyze source code and fix vulnerability encountered through proposed SECURESOLUTION security model
Intercept HTTP response returned from web server
No
Vulnerability Found in Response
Yes
Generate attack report

Figure 1. SECUREWEB tool

4.1 OWASP Vulnerabilities Analysis through SECUREEYE Proposed Model 1

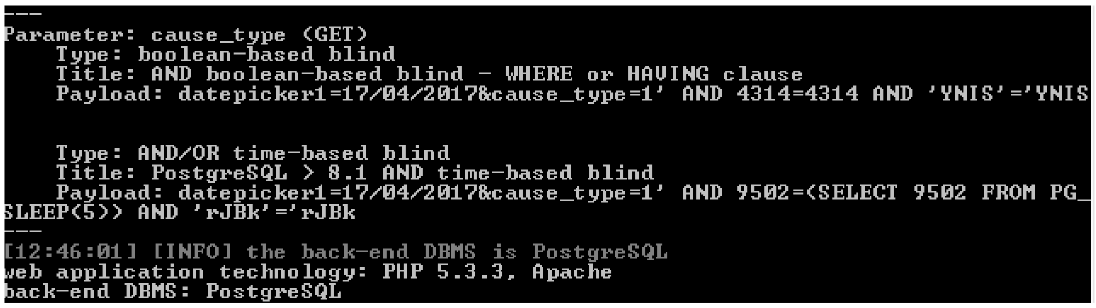
4.1.1 Boolean-Based Blind SQL Injection

Step 1: Open the following URL
"**http://127.0.0.1/test.php?datepicker1=17/04/2017&cause_type=1**"

Intercept the request through SECUREWEB-SECUREEYE proxy based proposed tool and sends it to web server for processing and then analyse the response returned for vulnerabilities as shown in the snapshot below:

test.php [HTML input form]
```
<form name="frm" action ="test_action.php">
<input type ="text" name="datepicker" value="<?php echo $date_picker;?>">
<input type ="text" name="cause_type" value ="<? php echo $cause_type ;?>">
<input type="button" name="submit">
</form>
```

Step 2: Intercept request through SECUREEYE and enter the following payload, cause_type =**1'%20or%20'1'%3d'1(1' or '1'='1)** and it is seen that the content length is **10622,** now if it is changed as cause_type =**1'%20or%20'1'%3d'2(1' or '1'='2)** then content length changes from **10622** to **3672** as shown below in Figure 2.


```
---
Parameter: cause_type (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: datepicker1=17/04/2017&cause_type=1' AND 4314=4314 AND 'YNIS'='YNIS


    Type: AND/OR time-based blind
    Title: PostgreSQL > 8.1 AND time-based blind
    Payload: datepicker1=17/04/2017&cause_type=1' AND 9502=(SELECT 9502 FROM PG_
SLEEP(5)) AND 'rJBk'='rJBk
---
[12:46:01] [INFO] the back-end DBMS is PostgreSQL
web application technology: PHP 5.3.3, Apache
back-end DBMS: PostgreSQL
```

Figure 2. Boolean based blink SQL injection

This confirms that application is vulnerable to Boolean-based Blind SQL Injection.

4.1.2 Blind SQL Injection Attack Scenario

The HTTP request was intercepted through SECUREWEB-SECUREEYE proposed tool and malicious data was inserted on user request, as a result response gets manipulated, which indicates the existence of blind SQL vulnerability, because the tool succeeded in appending malicious values to parameter and was executed by the server.

HTTP Request from browser
```
<form name="frm" action="new_page.php"><input type ="text" name="book" value="<? php echo book;
?>" />
<input type="submit" name="submit1" value="add"></form>
```

Journal of Graphic Era University
Vol. 6, Issue 2, 183-196, 2018
ISSN: 0975-1416 (Print), 2456-4281 (Online)

HTTP Response from server
```
CurYear=2017&curMonth=08&curDay=13&book=1234%27+and+%27f%27%3D%27f%27+--
+&submit1=add
```

This vulnerability can be removed by proper sanitization of hazardous character on user input. The process of data sanitization is explained in proposed SECURESOLUTION security model.

4.1.3 SQL Injection Attack Scenario

Request and response test was performed on PHP source code as written below, request was successfully intercepted in between client and server and malicious data was successfully inserted on user request, as a result response gets manipulated.

HTTP Request from browser
```
<form name=”frm” action=”new_page.php”>
<input type =”text” name=”username” value=”<? php echo $username ;?>”/>
 <input type =”text” name=”password” value=”<? php echo $password ;?>”/>
<input type=”button” name=”submit1” value=”add”>
< /form>
```

HTTP Response (new_page.php)
```
<?php
$username =$_POST[‘username’];
$password =$_POST[‘password’];
$sql=”select loginid from user_master where username=’$username’ and password =’$password’”;
$db->execute($sql);
?>
```

Most of the time programmers do not write parameterized SQL queries as shown in above script, resulting vulnerable SQL Injection. A simple example of an SQL Injection payload can be achieved by setting the password field to password’ OR 1=1. This will result SQL query something as shown below.

```
SELECT loginid FROM user_master WHERE username=’$username’ AND password=’password’
OR1=1’
```

This query if gets executed in database server can result in authentication bypass. In the event of authentication bypass, the application will most likely enters the attacker into the database and results the login-id from users_master in the query result.

This vulnerability can be removed by writing parameterized queries. Tricks to write parameterized queries in PHP code is explained in proposed SECURESOLUTION security model.

4.1.4 MongoDB NoSQL Injection Attack Scenario

The request and response test was performed on PHP source code as written below, request was successfully intercepted in between through SECUREWEB-SECUREEYE conceptual tool and had successfully inserted special characters { }' " \ ; in request input variable, as a result response gets manipulated, which lead to trigger a database error within MongoDB. For example within MongoDB, if a string containing any of the following special characters ({ }' " \ ;) it will trigger the database error.

HTTP Request from browser

```
<form name=”frm” action=”new_page.php”><input type =”text” name=”book” value=”<?php echo
$book;?>”/>  //$book variable is not sanitized
<input type=”button “name=”submit1”value=”add”><form>
```

The request was intercepted in between and special characters ({ }' " \) were inserted in request parameters which results MongoDB SQL vulnerability in source code;

HTTP request intercepted

```
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET
CLR 2.0.50727; .NET4.0C; .NET4.0E; .NET CLR 3.5.30729; .NET CLR 3.0.30729)
Content-Length: 500
curYear=2017&curMonth=08&curDay=13&book=2","$2836":"&submit1=add
```

HTTP Response showing MongoDB NoSQL injection

```
Failed: SQLSTATE[22P02]: Invalid text representation: ERROR: invalid input syntax for integer:
"2","$query":{},"A":""curYear=2017&curMonth=08&curDay=13&book=2","query":"
```

MongoDB NoSQL injection is possible only when the hacker succeeds in injecting MongoDB commands and those commands gets executed on database server. This issue can also be patched by proper sanitization of hazardous character on user input described in SECURESOLUTION security model.

4.1.5 Cross Site Scripting Attack Scenario

Request and response test performed on PHP source code seems to indicate XSS vulnerability, because if a piece of code is written like this then hacker can easily append script in the response, and that script gets executed on page loads in the user’s browser, leading to XSS vulnerability.

HTTP Request from browser
```
<form name=”frm” action=”new_page.php”><input type =”text” name=”book” value=”<?php echo $book;?>”/>
<input type=”button” name=”submit1” value=”add”><form>
```

HTTP request intercepted and embedded javascript
```
Content-Length: 147
curYear=2017&curMonth=03&curDay=13&book=09"/>
<script>alert(1243)</script>
```

HTTP Response from server
```
<input type="text"  name="book" value="09"/>
<script>alert(1243)</script>" onFocus="SetBg(this)" onBlur ="UnSetBg(this)">
<input type="button"  name="button" value="Go" size="20" onClick="return submitForm2();"/>
```

JavaScript is successfully embedded in source code. This vulnerability in source code can be patched by proper sanitization of hazardous character on user input, proposed in SECURESOLUTION security model.

4.1.6 Link Injection (Facilitates Cross Site Request Forgery CSRF) Attack Scenario

Request and response test performed on PHP source code through SECUREEYE tool, intercepted HTTP request variables and manipulated HTTP request variable by injecting link, request got executed seems to indicate CSRF vulnerability because HTTP response contained a link to the file. Because of that it is possible to steal or manipulate customers session and cookies allowing the hacker to view or alter user records. This vulnerbility can be patched by sanitization of input parameters as described in SECURESOLUTION model.

HTTP Request from browser
```
Content-Disposition: form-data; name="case_no""><IMG SRC="/WF_XSRF1698.html">
```

HTTP Response from server
```
<td width="54" align="left">
<input type="text"  maxlength="7"  size="8" name="case_no"value=" ">
<IMG SRC="/WF_XSRF1698.html">" onFocus="SetBg(this)" onBlur="UnSetBg(this)">
</td>
```

4.1.7 Phising Through Frames Attack Scenario

Intercepted request and successfully injected HTML frames in request parameters, response result seems to indicate phising vulnerability through frames because test response contained an iframe/frame to URL. This vulnerability can be patched through proper sanitization of hazardous characters into input paramters.

HTTP Request from browser

```
curYear=2017&curMonth=03&curDay=13&book=1234%27%22%3E%3Ciframe+id%3D1198+src%3Dhttp
%3A%2F%2Fdemo.testfire.net%2Fphishing.html%3E&
```

HTTP Response from server

```
<input type="text"  name="book" value="1234'"><iframe id=1198
src=http://demo.testfire.net/phishing.html>" onFocus="SetBg(this)" onBlur="UnSetBg(this)
```

4.1.8 Weak Cryptographic Hash Attack Scenario

It is found that weak cryptographic hashes cannot guarantee data integrity and should not be used in web application source code.

```
$key = sha1(microtime());
d = md5_gg(d, a, b, c, x[i+ 2], 9 , -51403784);
return hex_md5("abc").toLowerCase() == "900150983cd24fb0d6963f7d28e17f72";
return rstr2any(rstr_hmac_md5(str2rstr_utf8(k), str2rstr_utf8(d)), e)
return binl2rstr(binl_md5(opad.concat(hash), 512 + 128));
```

There should be strong 256 bit encryption policy.

4.1.9 System Information Leak External

While auditing source code it was found that most of the time coder use in-built system functions in program such as use of error_reporting which reveled important information about web server and error logs.

```
error_reporting(E_ALL);
print "<h4>Testing adodb_date and adodb_mktime. version=".ADODB_DATE_VERSION.'
PHP='.PHP_VERSION."</h4>";
```

Use of system in-built functions should be strictly prohibited into the source code.

4.1.10 Session Identifier

It is always a great threat to steal or manipulate customer session and cookies if session identifier remains same before and after login, so it is mandatory to change or reset session identifier after login and logout.

```
Content-Type: application/x-www-form-urlencoded Cookie: PHPSESSID=61nidlbc1tqk1rnoir4o34n2c2
```

4.2 Defending Source Code Vulnerabilities through SECURESOLUTION Security Model 2

***Case Study* 1**

Since web applications are hosted in cloud infrastructure and these are accessible 24*7 by different type of users through different type of devices. Suppose all the users are genuine, who access web applications, surf applications and move back.

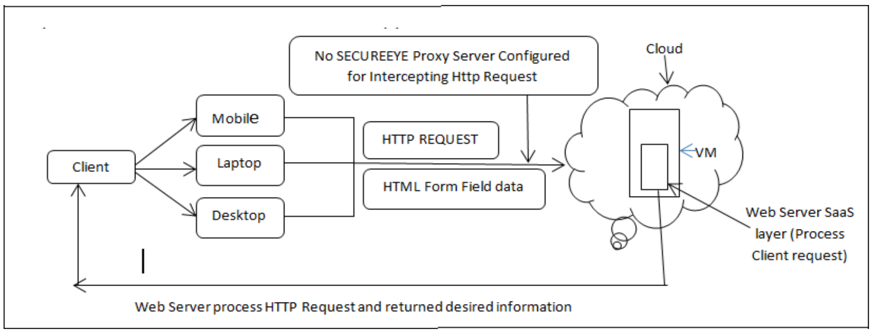

No SECUREEYE Proxy Server Configured for Intercepting Http Request
Cloud
Mobile
Client
Laptop
Desktop
HTTP REQUEST
HTML Form Field data
VM
Web Server SaaS layer (Process Client request)
Web Server process HTTP Request and returned desired information

Figure 3. Request with no attack performed

In this case, users receive expected result corresponding to the input data parameters. Since users are genuine so there is no threat to the web application even if source code is vulnerable as shown in Figure 3.

***Case Study* 2**

Since all the users who are accessing web portal cannot be genuine, few can be attackers, who can try to hack application in cloud. So it is the responsibility of web application owner to make sure web application must be free from all types of vulnerabilities as discussed in this paper. To identify vulnerabilities in source code a proxy based source code analyzer tool named SECUREEYE is proposed in this paper, which intercepts HTTP request and modify HTTP request data to check response in client's web browser. If the source code is not secure, then response will throw SQL, CSRF, PHISING, CSRF and many more vulnerabilities in client's browser. In this case all users are not genuine and hence they can intercept HTTP request as a result output varies from user to user as shown below in Figure 4.

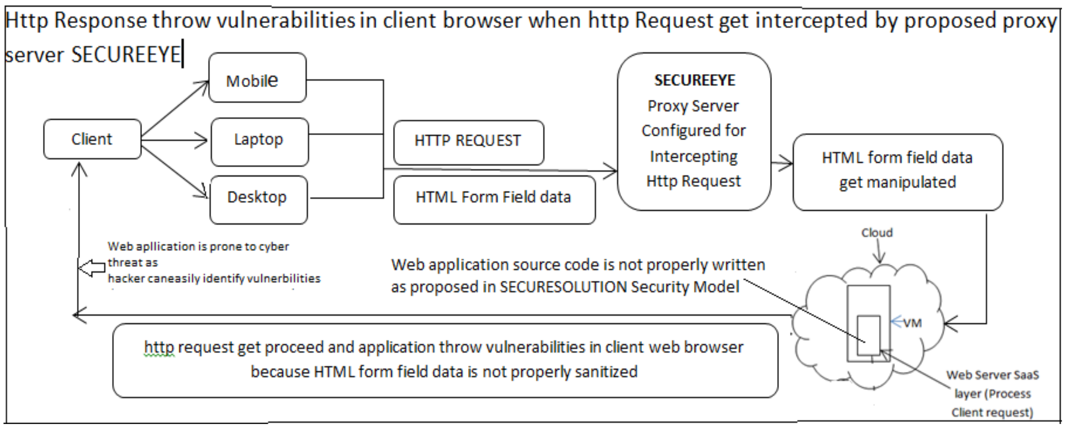
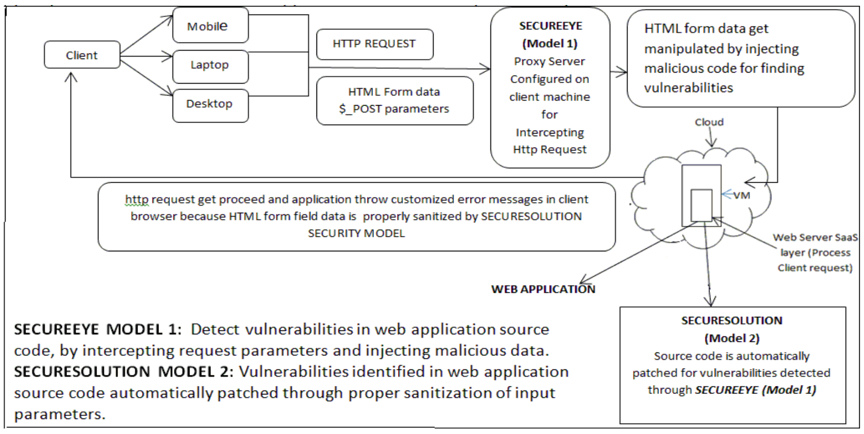

Http Response throw vulnerabilities in client browser when http Request get intercepted by proposed proxy server SECUREEYE
Client
Mobile
Laptop
Desktop
HTTP REQUEST
HTML Form Field data
SECUREEYE Proxy Server Configured for Intercepting Http Request
HTML form field data get manipulated
Cloud
VM
Web Server SaaS layer (Process Client request)
Web aplication is prone to cyber threat as hacker caneasily identify vulnerbilities
Web application source code is not properly written as proposed in SECURESOLUTION Security Model
http request get proceed and application throw vulnerabilities in client web browser because HTML form field data is not properly sanitized

Figure 4. Request with payload attacks

Case Study 3

Suppose users are not genuine but source code is properly secured and input request parameters are properly sanitized as proposed in SECURESOLUTION security model, then what will happen.


Client
Mobile
Laptop
Desktop
HTTP REQUEST
HTML Form data $_POST parameters
SECUREEYE (Model 1) Proxy Server Configured on client machine for Intercepting Http Request
HTML form data get manipulated by injecting malicious code for finding vulnerabilities
Cloud
VM
Web Server SaaS layer (Process Client request)
http request get proceed and application throw customized error messages in client browser because HTML form field data is properly sanitized by SECURESOLUTION SECURITY MODEL
WEB APPLICATION
SECURESOLUTION (Model 2) Source code is automatically patched for vulnerabilities detected through SECUREEYE (Model 1)

SECUREEYE MODEL 1: Detect vulnerabilities in web application source code, by intercepting request parameters and injecting malicious data.
SECURESOLUTION MODEL 2: Vulnerabilities identified in web application source code automatically patched through proper sanitization of input parameters.

Figure 5. Defended attacks through SECURESOLUTION security model

As shown above in Figure 5 user tried to intercept HTTP request and injected malicious code, but application did not behaved wrongly and had not thrown any server and web vulnerabilities as a response to web browser. This is achieved by proper sanitization of input variables through SECURESOLUTION model, in this case only customized error messages shown in web browser.

Checklist for Prevention of Vulnerabilities in Source Code

- Implement captcha on all public pages, captcha must be alphanumeric with minimum 6 character.
- Proper validation in request and response parameters from client and server end. Special characters must be whitelisted.
- Use parameterized queries or stored procedure to avoid SQL injection, which occurs through inline SQL queries.
- There should be a strong application audit trail mechanism to check what activates are carried out in application by users.
- Store critical data like aadhar information, credit card information in encryption form.
- Password must be strongly hashed on client browser before sending to server for processing.
- Implementation of proper error handling modules in application.
- Use POST method to pass values from one page to another page.
- Implementation of random token base system that changes with every page request in application to avoid CSRF attacks.
- Implementation of proper session handling and session timeout mechanism in application.

5. Conclusion

Best practices such as protecting source code against SQL injection flaws, broken authentication and session management, insecure direct object references, security misconfiguration, insecure cryptographic storage, failure to restrict URL, filtering, session handling, CSRF protection, XSS and Phishing can prevent applications from cyber-attacks. This research paper proposes threat finding and threat defending conceptual tool named SECUREWEB, which is based on combination of two models. Model 1: SECUREEYE Proxy based source code analyzer tool for intercepting request and response for finding vulnerabilities in source code. Model 2: SECURESOLUTION security model automatically sanitizes input variables and make application secure from cyber-attacks.

Since, there is no such tool available right now which can perform both these tasks automatically, so developers manually patches vulnerabilities in source code, which is time consuming and not 100% efficient. So there is great need of one such tool like SECUREWEB.

References

Djuric, Z. (2013, September). A black-box testing tool for detecting SQL injection vulnerabilities. In Informatics and Applications (ICIA), 2013 Second International Conference on (pp. 216-221). IEEE.

Fonseca, J., Vieira, M., & Madeira, H. (2014). Evaluation of web security mechanisms using vulnerability & attack injection. IEEE Transactions on Dependable and Secure Computing, 11(5), 440-453.

Huang, H. C., Zhang, Z. K., Cheng, H. W., & Shieh, S. W. (2017). Web application security: threats, countermeasures, and pitfalls. Computer, 50(6), 81-85.

Kankhare, D. D., & Manjrekar, A. A. (2016, December). A cloud based system to sense security vulnerabilities of web application in open-source private cloud IAAS. In Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT), 016 International Conference on (pp. 252-255). IEEE.

Nguyen-Tuong, A., Guarnieri, S., Greene, D., Shirley, J., & Evans, D. (2005, May). Automatically hardening web applications using precise tainting. In IFIP International Information Security Conference (pp. 295-307). Springer, Boston, MA.

Open Web Application Security Project (OWASP) 2015, OWASP Top 10, Available https://www.owasp.org, last accessed 10-02-2018.