CONSTRUCTING SECURE WEB APPLICATIONS WITH PROPER DATA VALIDATIONS

Krishna Reddy[1], Prasanth Kumar[2]
[1,2]M.Tech Scholar
Department of Information Security
Manipal University
Jaipur, India.

Abstract: ***With the advent of World Wide Web, information sharing through internet increased drastically. So web applications security is today's most significant battlefield between attackers and resources of web service. It is likely to remain so for the foreseeable future. By considering recent attacks it has been found that major attacks in Web Applications have been carried out even when system having most significant network level security. Poor input validation mechanisms that using in Web Applications shall causes to launching vulnerable web applications, which easy to exploit easy in future stages. Critical Web Application Vulnerabilities like Cross Site Scripting (XSS) and Injections (SQL, PHP, LDAP, SSL, XML, Command, and Code) are happen because of base level Validations, and it is enough to update system in unauthorized way.***

Keywords: Security, Validation, Vulnerability, XSS, Injection.

I. INTRODUCTION

Constructing secure application is very difficult, in terms of complexity. More over there is no measures for security, providing security means to keep avoiding attacking patterns. When industry moving towards electronic communication, web service palace major role for information interchange. Mainly web applications serves public information, up to some extend attack patterns having less impact in web service, when service starts to store and transfer confidential in information through internet, attack patterns are involving in between normal communication, such activities spoils user or service present states. In initial stages security handles with antivirus, then in network level, now security threats more in application level, due to lack of secure code [1]. So vulnerability is defined as weakness in system or future of system that males easy to exploit. Vulnerability might be existing at the host, network or application levels.

Many application especially web based application faces risks and those applications will cause to violate policies that are maintained in application. Web applications works in the principle "web server accepting user request and process it, again gives proper acknowledgments" this process enough to implement communication channel. When organization moves to automate business with web service via internet attack surfaces comes in front, due to lack of security auditing. Application level attacks place interesting role in web applications, which causes to financial lose, and creates serious reputation. Majorly application level attacks happen because of input validations [2,13], digital information that requesting by user in application surface, service have to configure to process user requests in proper way, to keep out attack surfaces. In application level vulnerability identification is absolutely difficult; data validation is the only way to keep application secure from application attacks. Cross Site Scripting and Injection vulnerabilities are commonly found in complex business applications, unfortunately commercial applications also not so far from application level attacks.

II. DATA VALIDATION VULNERABILITIES IN WEB APPLICATIONS

Most common web application security threat is failure to properly validate input coming from user requests before using it. "All Input is Evil" data coming from external entity to server should never be trusted, treat data coming from external entity can be tampered by attacker [3]. Basically complex and commercial applications often have a large number of entry points; some cases it difficult to provide validation because of complexity issues. These data validation vulnerabilities leads

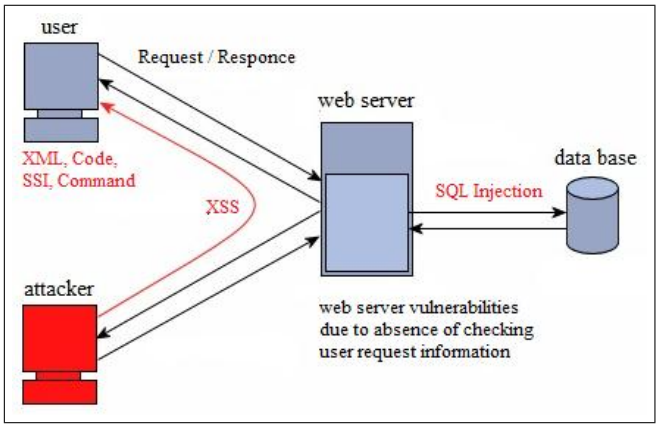

Figure 1: Data Validation Vulnerabilities

to form Cross Site Scripting and Injection flaws.

A. Cross Site Scripting (XSS) Vulnerability

Cross Site Scripting, abbreviated as XSS, is most common application level attack, which hackers can use to slip into vulnerable web application; such attack can directly take place at victim's browser that used across web. XSS attacks are caused by failure in the web application to properly validate user input [4]. Web applications provide private session with help of cookies, such secure cookies can be stealing with help of XSS attack. Also XSS attacks can able to redirect users into malicious pages, inject scripts into client side any scripting languages (VB, JAVA, PHP scripts) to execute on victim's browser. Script can able to inject in URI or may be in response header, depends on attack (Stored, Reflected, and DOM), attack location can be decided. Traditionally XSS used to update private settings by attackers, and able to launch shell, hijacking user state, and spread malicious worm over web [14].

XSS attack patterns described as follows:

Input -> Output = Cross Site Scripting

Consider following scripts to inject:

<script>window.location='https://example.com/doc/hack='+document.cookie;</script>

Executing above script able to steal user cookie and send it to attacker addressed location.

<x href=http://example.com/>xss</x>

Above code can able to redirect victim location to attacker site.

<iframe src=javascript:alert(abc)>

Injecting script irrespective of client or server side scripting languages, scripts can directly executing in victim's machine.

<body onload=alert(page locked)>

This exploitation causes to mask front end of user actual response pattern, happened when page body loading in users local application.

Like this XSS can able to launch bad script into user surface, when user active in malicious pages impact is not measurable.

B. Command Injection Vulnerability

OS Command injection which trying to inject a command through HTTP request to application. This technique used via web interface in order to execute operating system commands on web server which is not properly sanitized [5]. Such a vulnerability can able to upload malicious scripts to service, and possible to get passwords from service.

OS Command injection pattern describes as follows:

Input -> OS Command = OS Command injection

Consider following URL request:

http://abcd/cgi-bin/udata.u1?doc=user.txt

Modified URL:

http://abcd/cgi-bin/udata.u1?doc=/bin/ls|

Where modified URL executes /bin/ls.

Executing OS Commands in URL can able to update server state. In some cases command patterns able to execute along with user requests, or maybe input fields along with user application. Command might be anything if those command lines got execute permissions in service, then each command will processed in server. It is essential to prohibit executing operating system commands, on service to prevent command injection.

C. XML Injection Vulnerability

When XML parser fail to make appropriate data validation, XML injection vulnerability arise in service [6]. Such services have possible to accept any malicious information along with actual user requests.

Attack pattern can be defined as follows.

Input –> XML document = XML injection

In XML document there is own specialty for Angular parenthesis > and <, Comment tag <!–/–>, Ampersand &, CDATA begin and end tags <![CDATA[/]]>.

Each XML document shell form with help of above tags, in such case poorly configured service vulnerable to injection.

Consider following code:

<html>
<![CDATA[<]]>script<![CDATA[>]]>alert('123')<![CDATA[<]]>/script<![CDATA[>]]>
</html>

Successful execution of above code will launch the script **"<script>alert('123')</script>"** which is XSS vulnerable.

D. Code Injection Vulnerability

Malicious code possible to inject into an application data that will be later executed by web server. Code submitted along with input request, that process by service as dynamic code or as in an include file [7]. This exploitation mainly causes to lose of availability.

Code injection pattern describes as follows:

Input –> Malicious code = Code injection

Consider following malicious URL to request service:

http://www.abc.com/sys.php?pin=http://www.xyz.com /user/is.j pg?&cmd=name%20-x

Launching this malicious URL is accepted as a parameter for the above PHP page, which will later use the value in an include file. Particularly above issue arises because of length validations and URL ordering. Service have to take care about user entry points, because this exploitation is not only in URL so service have to check all application entry points.

E. Server Side Includes (SSI) Injection Vulnerability

SSI have ability to add piece of dynamic code inside static HTML page, without having full-fledged connection with client or server side scripting languages [8]. This exploitation allows an attacker to inject code into HTML pages or possible to perform remote code execution. Pages hosted in web server will arises SSI vulnerability. So SSI surely arises security threats when service available in public environment.

Let see following code in HTML pages.

<!–#exec cmd="ls" –>
<!–#include virtual="/etc/passwd" –>

This allows to execute system command, if system command will execute in web server, then there is no place for security.

F. XPATH Injection Vulnerability

XPATH applies to data that stored in XML format, this vulnerability can be possible to bypass authentication or able to gain access in an unauthorized manner to manage system, this impact is not leave for single page, it can able to corrupt whole document [9].

Consider following query for authentication purpose:

string(//user[username/text()='uname' and password/text()= 'pass']/account/text())

XPATH vulnerable application can able to pass query as

string(//user[username/text()=' ' or '1' = '1' and password/ text()= ' ' or '1' = '1']/account/text())

So this vulnerability can bypass existing security controls. So XPATH injection attacks can be much more adaptable and ubiquitous.

G. SQL Injection Vulnerability

SQL injection used to attack data driven application in which malicious SQL query inserted to bypass authentication by passing true values to service. SQL injection is not a database vulnerability poorly configured server will always process malicious requests and send sensitive information to malicious users.

A successful SQL injection exploitation can able to assign administrative permissions mainly for accessing database files, such as insert, delete and update permissions. In some cases failed attempts also caused for SQL injections. Gaining access to manage sensitive information by malicious users then there is no security mechanism. SQL injection mainly occurs because of parameters used in service, and this vulnerability can able to do extracting data from database, execution of remote commands, privilege escalation [10]. Query which we are requesting to database, server accepts parameters from user and make it as query then send it to database, in this process poorly configured service causes SQL injection attack.

SQL injection vulnerability attack pattern describes as follows.

Input -> SQL query = SQL Injection

Consider following SQL query :

SELECT * FROM CSE WHERE USERNAME = '$USERNAME' AND PASSWORD = '$PASSWORD'

Suppose if we pass request as

SELECT * FROM CSE WHERE USERNAME = '1' OR '1' = '1' AND PASSWORD = '1' OR '1' = '1'

This request in vulnerable application carried as follows;

http://www.abc.com/index.php?username=1'%20or%20'1'%20=%20'1&password=1'%20or%20'1'%20=%20'1

This condition is always true because of true values, where vulnerable system has authenticated without username and password. So SQL injection vulnerability applications can easily bypass authentication, by processing true values, along with requested information.

III. IMPACT OF DATA VALIDATION VULNERABILITIES

Security is nonfunctional issues for service, when security threats arises system survivability is difficult in public service.

a) **XSS :** Redirect to other service, may be malicious system. Requesting malicious payloads along with service. Upload malicious code along with any one of images, scripting languages, URL. Directly under take system events. Steal user's sensitive information from service, and able to send it to malicious user address. Uploading shell to system to process unnecessary commands. Injecting logger into system to store user activities [11]. Able to change system, user settings, sending unnecessary information to system users, requesting for modifying authentication mechanisms. Disturbing business work flow path. Importing malicious scripts to system, sending encoded form information. Injecting scripts under images to server. Mainly XSS can able to launch other type of injection patterns.

b) **COMMAND :** Up loading malicious scripts via service requests, able to execute operating system commands. Creating executing permission without proper authorization, undertaking system activities, able to update system. Possible to add new system users, change permissions on existing users and transferring information from service.

c) **XML :** Injecting unwanted or malicious script to service, which process along with request. Uploading malformed tags to server, which executed dynamically on server.

d) **CODE :** Sending malicious code along with user request headers. Mainly it depends on code injecting on service which can able to store in server and possible to run malicious code in later stages. It can able to change parameter list to process code in later stages. Which leads to loss of confidentiality integrity and availability.

e) **SSI :** Irrespective of client side or server side scripting language injecting command lines on web pages, and able to give execute permission in web page itself.

f) **XPATH :** Able to gain access on service, elevating privilege if data stored in XML files.

g) **SQL :** Bypassing authentication by requesting true values along with requesting parameters for accessing user sessions. Attempting to retrieve database details, such as table names, number of table rows and columns. Attempting to pass command lines along with service fields, and those command lines can able to execute at processing time [12]. Injecting error statements, sending null values, uploading encoding values, SQL injection mainly used for detecting database schema and able to retrieve data from database. And able to get access permissions, possible to privilege escalation. Major amount of data can leak from service only with SQL injection.

IV. ISSUE REMEDIATIONS

Depends on security threat, data validations have to maintain in service. Mainly in this category there must be strict validations in server side, even better to prefer sanitize. In case of complexity case better to go for sanitization [13].

a) **XSS :** Use application program interface, to handle user inputs. In complex applications there is many entry points, for user requests, keep on validating input information (string length, special characters, and script tags). Then go for sanitization.

b) **COMMAND :** Use application program to filter command activities, because there is possibility to execute runtime commands. Prevent to process static commands. Always go for predefined list of instructions.

c) **XML :** Use least privilege access, go for secure patch management, and check network status after system updates, to check number of requests.

d) **CODE :** Go for input validation, selective input inclusion and exclusion on input data. Escaping dangerous characters, by secure functions implemented in sever side languages. Implement input and output encoding. Put restrictions on dynamic code execution.

e) **SSI :** Keep privilege for write and executing permissions on system, execute predefined command lines only, and avoid static command execution. Sanitization is preferable, prevent dynamic changes in system.

f) **XPATH :** Use web application firewall. Keep avoiding changes in existing system, if changes is necessary go for secure policies, authorization. Don't allow all users to be run all queries in system. Implement accessing limits on data by using Role Based Access Control (RBAC) technique.

g) **SQL :** Primarily go for input validations; such as string lengths, special characters, later go for sanitize user input. Use parameterized queries, to generate dynamic SQL queries. Go for database functions and stored procedures. Always prefer predefined function executions. Put limit access on service, to prevent brute force. Consider restrictions on strings and all inputs, string lengths. Use least privilege access. Implement secure cache management.

V. ANALYSIS

Web applications are mainly deals with request and response to process and manage information with webserver. In terms of business point of view, there must be place to security, if not possible to change private state turn to public. Result to loose integrity, confidentiality, availability. For this web server have to make sense before it process, which digital information that server receives from user requests. Use secure coding strategies from application designing phase onwards. User input never be trusted it contains anything before processing request data have to validate. Methods and functions used to host pages in web server causes to security threats.

In order to complete application developers basically follows software development life cycle, which analyze whole functional architecture of application except security risks, this SDLC mainly manipulate by the security unaware team. To overcome these risk patterns implement secure software development life cycle (SSDLC), which consider only secure methods and functions.

Data validations $\not\subset$ web server $\Rightarrow$ vulnerable web applications

Data validations $\subset$ web server $\Rightarrow$ secure web applications

Mainly security threats will occur because of functions and methods used in application, so major risks will active in the development stage itself. It is difficult to change methods and functionality of application when attack happens, so prefer only secure application development strategies, and consider security as functional issue in each develop and deployment stages, to provide secure communication.

VI. DEPLOYMENT STRATEGIES

Web applications are hosted rapidly for business mobility, to providing service throughout web. When applications are becoming complex and transaction oriented, application have to consider secure communication, to continue survivability of application in the web. For that start application development in secure manner from beginning stage itself. Test the security strategies from development stage only, so security awareness is essential to development team, then only secure software development life cycle (SSDLC) will fulfill. Once application is deployed irrespective of attacks keep on update the system. And go for log analysis, to prevent malicious users. Periodically check the network status of system, to analyze request patterns. Then check for load and stress on system to prevent unexpected system crashes [14]. Prefer only secure patch management strategies. Take system backup to local machine, never link old backups to service.

VII. CONCLUSION

In this paper, we present data validation vulnerabilities and describing attack patterns of each attack, and we proposed precautions to maintain in web application development, deployment and maintaining strategies, to prevent attack happening from service entry points. To provide secure application, it is difficult to wait until attack take place, keep on avoiding attack patterns with help of data validations and input sanitization is only the best way to introduce secure web applications.

Future evaluation of work shall focus on evaluating the secure web application development strategies, to provide reliable and secure communication, having lesser complexities and more reliable services, which prevent internal and external attack patterns on a system.

References

[1] Gary McGraw and John Viega. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley Pub Co.

[2] Dafydd Stuttard, Marcus Pinto. *The Web ApplicationâĂŹs Handbook - Discovering and Exploiting Security Flaws*. Wiley, 2008.

[3] Mike Howard and David LeBlanc. *Writing Secure Code*. Microsoft Press.

[4] Sivakumar K and Garg K. Constructing a common Cross site scripting vulnerabilities enumeration (CXE) using CWE and CVE. *LNCS 4812, eds. P McDaniel and SK Gupta - Springer*, pages 277–291, 2007.

[5] Gary McGraw and Greg Hoglund. *Exploiting Software: How to Break Code*. Addison-Wesley Pub Co.

[6] Rosa, T.M., Santin A.O., and Malucelli A. Mitigating XML Injection 0-Day Attacks through Strategy-Based Detection Systems. *Security and Privacy, IEEE, 10.1109/MSP.2012.83*, pages 46–53, 2012.

[7] Saxena P., Akhawe D., Hanna S., Feng Mao, McCamant S., and Song D. A Symbolic Execution Framework for JavaScript. *Security and Privacy (SP), IEEE*, pages 513–528, 2010.

[8] Sverre Huseby. *Sverre Huseby*. AJohn Wiley and Sons.

[9] Asmawi A., Affendey L.S., Udzir N.I., and Mahmod R. Model-based system architecture for preventing XPath injection in database-centric web services environment. *Computing and Convergence Technology (ICCCT)*, pages 621–625, 2012.

[10] Johari, R. ; Sharma, P. A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection. *Communication Systems and Network Technologies (CSNT), 10.1109/CSNT.2012.104*, pages 453–458, 2012.

[11] Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager, and Seth Fogie. *Cross Site Scripting Attacks: XSS Exploits and Defense*. Syngress, 2007.

[12] Joel Scambray, Mike Shema, Caleb Sima. *Hacking Exposed Web Applications*. McGraw-Hill, 2006.

[13] Atashzar, H. ; Torkaman, A. ; Bahrololum, M. ; Tadayon, M.H. A survey on web application vulnerabilities and countermeasures. *Computer Sciences and Convergence Information Technology (ICCIT)*, pages 647–652, 2011.

[14] James S. Tiller. *The Ethical Hack: A Framework for Business Value Penetration Testing*. Auerbach.

