Web Server Security and Survey on Web Application Security

Shaikh Bushra Almin,

Department of Information Technology,

PIIT, New Panvel.

University of Mumbai, India.

skbushra78691@gmail.com

Abstract— A web server is a computer host configured and connected to Internet, for serving the web pages on request. Information on the public web server is accessed by anyone and anywhere on the Internet. Since web servers are open to public access they can be subjected to attempts by hackers to compromise the server's security. Hackers can deface websites and steal data valuable data from systems. This can translate into significant loss of revenue if it is a financial institution or e-commerce site. In the case of corporate or government systems, loss of important data means launch of information espionages or information warfare on their sites. Apart from data loss or theft, web defacement can also result in significant damage to the image of company [1]. The fact that an attacker can strike remotely makes a Web server an appealing target. Understanding threats to Web server and being able to identify appropriate countermeasures permits to anticipate many attacks and thwart the ever-growing numbers of attackers [3]. This work begins by reviewing the most common threats that affect Web servers. It then uses this perspective to find certain countermeasures. A key concept of this work focuses on the survey of most prevailing attacks that occurs due to certain vulnerabilities present in the web technology or programming which are exploited by attackers and also presents general countermeasures. In addition, various methods to detect and prevent those attacks are discussed and highlighted the summary and comparative analysis of the approaches on the basis of different attacks that shows you how to improve Web server's security.

Keywords— *SQLIA (SQL Injection Attack), XSS (Cross Site Scripting), CSRF (Cross Site Request Forgery), OWASP (Open Web Application Security Project).*

*****

I. INTRODUCTION

A secure Web server provides a protected foundation for hosting Web applications, and Web server configuration plays a critical role in Web application's security. Badly configured virtual directories, a common mistake, can lead to unauthorized access. A forgotten share can provide a convenient back door, while an overlooked port can be an attacker's front door. Neglected user accounts can permit an attacker to slip by your defenses unnoticed.

The fact that an attacker can strike remotely makes a Web server an appealing target. Understanding threats to Web server and being able to identify appropriate countermeasures permits to anticipate many attacks and thwart the ever-growing numbers of attackers [3].

A. Threats to Web Server and Countermeasures

The main threats to a Web server are [3]:

- Profiling
- Denial of service
- Unauthorized access
- Arbitrary code execution
- Elevation of privileges
- Viruses, worms, and Trojan horses

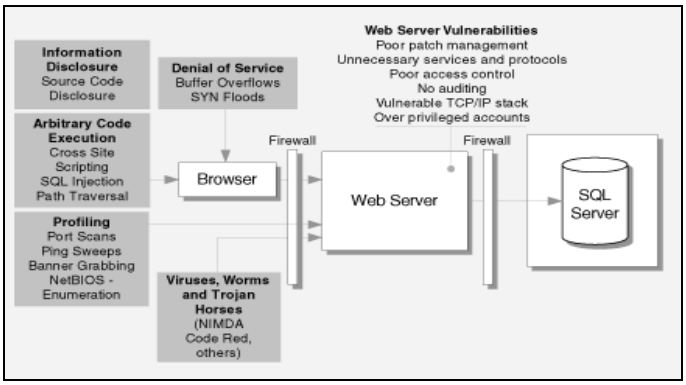

Fig. 1 Prominent web server threats

1) Profiling: Profiling, or host enumeration, is an exploratory process used to gather information about your Web site. An attacker uses this information to attack known weak points.

- **Vulnerabilities**

Common vulnerabilities that make your server susceptible to profiling include:

- Unnecessary protocols
- Open ports
- Web servers providing configuration information in banners

- **Attacks**

Common attacks used for profiling include:

- Port scans
- Ping sweeps
- NetBIOS and server message block (SMB) enumeration

- **Countermeasures**

Countermeasures include blocking all unnecessary ports, blocking Internet Control Message Protocol (ICMP) traffic, and disabling unnecessary protocols such as NetBIOS and SMB.

2) Denial of Service: Denial of service attacks occur when your server is overwhelmed by service requests. The threat is that Web server will be too overwhelmed to respond to legitimate client requests.

- **Vulnerabilities**

Vulnerabilities that increase the opportunities for denial of service include:

- Weak TCP/IP stack configuration
- Unpatched servers

- **Attacks**

Common denial of service attacks include:

- Network-level SYN floods
- Buffer overflows
- Flooding the Web server with requests from distributed locations

- **Countermeasures**

Countermeasures include hardening the TCP/IP stack and consistently applying the latest software patches and updates to system software.

3) Unauthorized Access: Unauthorized access occurs when a user without correct permissions gains access to restricted information or performs a restricted operation.

- **Vulnerabilities**

Common vulnerabilities that lead to unauthorized access include:

- Weak IIS Web access controls including Web permissions
- Weak NTFS permissions

- **Countermeasures**

Countermeasures include using secure Web permissions, NTFS permissions, and .NET Framework access control mechanisms including URL authorization.

4) Arbitrary Code Execution: Code execution attacks occur when an attacker runs malicious code on your server either to compromise server resources or to mount additional attacks against downstream systems.

- **Vulnerabilities**

Vulnerabilities that can lead to malicious code execution include:

- Weak IIS configuration
- Unpatched servers

- **Attacks**

Common code execution attacks include:

- Path traversal
- Buffer overflow leading to code injection

- **Countermeasures**

Countermeasures include configuring IIS to reject URLs with "../" to prevent path traversal, locking down system commands and utilities with restrictive access control lists (ACLs), and installing new patches and updates.

5) Elevation of Privileges: Elevation of privilege attacks occur when an attacker runs code by using a privileged process account.

- **Vulnerabilities**

Common vulnerabilities that make your Web server susceptible to elevation of privilege attacks include:

- Over-privileged process accounts
- Over-privileged service accounts

- **Countermeasures**

Countermeasures include running processes using least privileged accounts and using least privileged service and user accounts.

6) Viruses, Worms, and Trojan Horses: Malicious code comes in several varieties, including:

- **Viruses**. Programs that are designed to perform malicious acts and cause disruption to an operating system or applications.
- **Worms**. Programs that are self-replicating and self-sustaining.
- **Trojan horses**. Programs that appear to be useful but that actually do damage.

- **Vulnerabilities**

Common vulnerabilities that make you susceptible to viruses, worms, and Trojan horses include:

- Unpatched servers
- Running unnecessary services

- **Countermeasures**

Countermeasures include the prompt application of the latest software patches, disabling unused functionality such as unused ISAPI filters and extensions, and running processes with least privileged accounts to reduce the scope of damage in the event of a compromise [3].

II. OWASP TOP 10 WEB SECURITY THREATS

To keep pace, OWASP periodically update the OWASP Top 10. In this 2013 release, they made the following changes: [2]

OWASP Top 10 – 2010 (Previous)	OWASP Top 10 – 2013 (New)
A1 – Injection	A1 – Injection
A3 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A6 – Security Misconfiguration	A5 – Security Misconfiguration
A7 – Insecure Cryptographic Storage – Merged with A9 →	A6 – Sensitive Data Exposure
A8 – Failure to Restrict URL Access – Broadened into →	A7 – Missing Function Level Access Control
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<buried in A6: Security Misconfiguration>	A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards	A10 – Unvalidated Redirects and Forwards
A9 – Insufficient Transport Layer Protection	Merged with 2010-A7 into new 2013-A6

Fig. 2 OWASP Top 10 Web Application Security Threats

In this survey, we are focusing only on 3 most prevalent attacks which are occuring most frequently.

III. SQLIA

Injection flaws, such as SQL, injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization [2]. An example of simple tautology type of SQLIA is shown in figure 2, which will result in displaying all the records in the database irrespective of wrong username and password because the condition 0=0 always evaluates to true.

```
String SQLQuery ="SELECT Username, Password
    FROM users WHERE Username=" or 0=0"
      "' AND Password=" or 0=0" "'";
```

Fig. 3 Example of Tautology SQLIA

- Some of the most commonly followed prevention mechanism for SQLIA are as follows [5]:

- **Use prepared statements**
- **Perform Input Validation**
- **Escape all user supplied input**
- **Enforce least privilege**
- **Use stored procedures**

Apart from these, there are few research oriented techniques that have proved to be successful in preventing SQLIA to a greater extent. They are as follows:

- Access Control Mechanism [6]
- Network Vulnerability Scanner [7]
- Encryption [8]

Access control mechanism presents a technique, which will be used for the detection and prevention from SQL Injection. The parameterized cursor is used to implement the concept. The user session information will be passed as a parameter to cursor. If the user is an authorized user then the cursor will fetch the desired tuples else will fail to execute [6].

Network Vulnerability scanner are designed to penetrate the web applications against the security issues. They are the automated tools designed in such a way that they will perform the same attack as we do manually, the advantage of using Scanners is that they generate the automated report which shows what are the input points which are vulnerable [7].

In [8], the advantages of randomization are employed to prevent SQL injection attacks in web based applications. For a hacker to modify a database, details such as field and table names are required. So a solution to the above problem is proposed by preventing it using an encryption algorithm based on randomization. The random4 algorithm is based on randomization and is used to convert the input into a cipher text incorporating the concept of cryptographic salt. This algorithm forms the basis of the proposed approach.

IV. XSS

XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. [2]

- **Example Attack Scenario:**

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping: (String) page += "<input name='creditcard' type='TEXT' value="" + request.getParameter ("CC") + "'>"; The attacker modifies the 'CC' parameter in his browser to: '><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi? foo='+document.cookie</script>'.

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session. [2]

- **Types of XSS**

1) Stored XSS:

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-I XSS [15].

2) Reflected XSS:

Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web site. Reflected XSS is also sometimes referred to as Non-Persistent or Type-II XSS. [15]

Following are some recent techniques that are applied to detect/prevent XSS:

- Dynamic Cookies Rewriting[9]
- Comment Injection [10]
- Multi Agent Scanner [11]

With this [9] technique in place, the web proxy will automatically rewrite the value of the name attribute in the cookie with the randomized value before sending the cookie to the browser, so the browser will keep the randomized value in its database instead of the original value sent by the web server. The returned cookie from the browser will also be rewritten back to the original value at the web proxy before being forwarded to the web server. As the browser's database does not store the original values of the cookies, so even the XSS attacks can steal the cookies from the browser's database, the cookies cannot be used later to impersonate the users.

This [10] approach is based on the concept of injecting comment statements containing random tokens and features of legitimate JavaScript code. When a response page is generated, JavaScript code without or incorrect comment is considered as injected code. Moreover, the valid comments are checked for duplicity. Any presence of duplicate comments or a mismatch between expected code features and actually observed features represents JavaScript code as injected. A prototype tool is implemented that automatically injects JavaScript comments and deploy injected JavaScript code detector as a server side filter.

A novel multi-agent architecture allows for each one of those tasks to be carried out by a different type of agent. This design decision has been taken to allow each of the stages of the scanning process to be performed concurrently with the other stages. It also allows for the different sub- tasks of the scanning process to take place in a distributed and/or parallel way [11].

V. CSRF

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests to the vulnerable application thinks are legitimate requests from the victim. [2]

The differences between XSS and CSRF

Though CSRF seems similar to (XSS) at first, both are completely different attack vectors. Where XSS aims at inserting active code in an HTML document to either abuse client-side active scripting holes, or to send privileged information (e.g., authentication/session cookies) to an unknown evil website, CSRF aims to perform unwanted actions on a website where the victim has some prior relationship and authority.

Moreover, where XSS sought to steal your online trading cookies so an attacker could manipulate a victim's account, CSRF seeks to use the victims' cookies to force them to execute a trade without their knowledge or consent. While XSS attacks exploits the trust that a user has on the website, CSRF attacks exploit the trust that the website has in its user. [16]

- **Types of CSRF attacks**

1. Reflected CSRF attacks

In a reflected CSRF attack, the attacker uses a system outside the application to expose the victim to the exploit link or content. This can be done using a blog, an email message, an instant message, a message-board posting, [16].

2. Local/stored CSRF attacks

A stored/local CSRF attack is one where the attacker can use the application itself to provide the victim the exploit link, or other content which directs the victim's browser to perform attacker-controlled actions in the application. Examples include bulletin boards and social sites where users are allowed to post images with foreign URL sources.[16].

Most commonly used methods to prevent CSRF are as follows [17]:

- **Use of random tokens**
- **Use of POST in form rather than GET**
- **Limiting the lifetime of authentication cookies**
- **Damage limitation**
- **Force user to use your form**

CSRF attacks are also successfully prevented by applying following techniques:

- Shared secret token[12]
- Referer header[13]
- Origin header[13]
- Visibility and content checking[14]

A proxy based solution [12] uses a proxy that is placed on the server side between the web server and the target application. This proxy is able to inspect and modify client requests as well as the application's replies (output) to automatically and transparently extend applications with the previously sketched shared secret technique. In particular, the proxy has to

- Ensure that replies to an authenticated user are modified in such a way that future requests originating from this document (i.e., through hyperlinks and forms) will contain a valid token, and
- Take countermeasures against the requests of authenticated users that do not contain a valid token.
- An essential prerequisite for this mechanism is the proxy's ability to associate a user's session with a valid token. To this end, the proxy maintains a token table with entries that map session IDs to tokens.

Drawback:

- **Does not discriminate between hyperlinks**

It does not discriminate between hyperlinks back to the web application and hyperlinks to other web sites. If the web application links to another site, the remote site will receive a copy of the user's CSRF token [13].

- **Does not defend against login CSRF**

It does not defend against login CSRF because it only validates the CSRF token if the user already has a session identifier. Although this oversight is repairable, it demonstrates the complexity of implementing secret token validation correctly [13].

Unfortunately, the Referer contains sensitive information that impinges on the privacy of web users. Therefore using Referer header is widely suppressed. [14]

The Origin header improves on the Referer header by respecting the user's privacy:

- The Origin header includes only the information required to identify the principal that initiated the request (typically the scheme, host, and port of the active document's URL). In particular, the Origin header does not contain the path or query portions of the URL included in the Referer header that invade privacy without providing additional security.
- The Origin header is sent only for POST requests, whereas the Referer header is sent for all requests. Simply following a hyperlink (e.g., from a list of search results or from a corporate intranet) does not send the Origin header, preventing the majority of accidental leakage of sensitive information.

By responding to privacy concerns, the Origin header will likely not be widely suppressed. [13]

This [14] approach relies on the matching of parameters and values present in a suspected request with a form's input fields and values that are being displayed on a webpage (visibility). To overcome an attacker's attempt to circumvent form visibility checking, the response content type of a suspected request with the expected content type are compared.

VI. COMPARATIVE ANALYSIS AND SUGGESTIONS

Following are the comparative study of the above discussed techniques to detect and prevent SQLIA, XSS and CSRF.

Technique	Mechanism	Approach	Type of attack	Testing /Deployment platform	Static / dynamic	Performance	Future scope
Access control method	Parameterized cursor	Detection + Prevention	All / None	Oracle 10g + Oracle Internet Developer Suite 10g	Both – audit log of users	Degrade performance by 75%	Improve performance to optimal level
Network based vulnerability scanner	Penetration testing	Detection	All	JAVA + MYSQL	Static – attack library	Good Performance	Test for unknown attacks
Random4	Encryption	Prevention	Tautology + Piggy backed + Blind injection	C# 5 PHP application + MYSQL	Dynamic	Good – cipher text is difficult + time consuming	Advancement to other type of attacks too

Table 1 Comparison of SQLIA prevention/detection techniques

Technique	Mechanism	Approach	Implementation Approach	Technology used	Type of XSS attack	Advantages	Future scope
Dynamic Cookies Rewriting	Rewriting cookies with random values	Detection + Prevention	Web Proxy approach	Java	Reflected	Web content caching, NAT, Authentication, Authorization and content filtering etc	Intercept HTTPs
Comment Injection	Injecting comments with random token (Method Call Injection)	Injection + Detection	Server side approach	JSP	Reflected	No information need to be sent to the browser	Automated way for preprocessing
Multi agent Scanner	Using multi agent architecture	Detection	Standalone scanner	Java + Apache Tomcat + MYSQL	Stored	No modification of code	Improve scanning, injection point and attack vectors

Table 2 Comparison of XSS prevention/detection techniques

Technique	Mechanism	Approach	Deployment location	Tested on	Type of attack	Advantage	Limitation	Future scope
Secret Token	Instrumenting request and reply with shared token	Detection Prevention	Server side	Open source PHP applications from SourceForge	Reflected	No modification of code but need to implement proxy	Wrong CSRF alarms Login CSRF cannot be defended	Performance can be optimized using C/C++
Referer Header	Validating HTTP Referer header	Prevention	Client side/ Browser based	Mozilla Firefox + Internet Explorer	Reflected	Distinguishes a same site request from cross site request	Violates the user privacy	Strict Referer validation to protect against login CSRF
Origin Header	Including Origin header with POST request	Prevention	Client side/ Browser based	Firefox	Reflected – Login CSRF	Preserves the user privacy, Detects Login CSRF	Modifying browsers to send header with POST	Enforcing discipline to prevent side effect in response to GET
Client Side Detection	Visibility and content checking of suspected requests	Detection Prevention	Client side	Firefox browser on 3 real PHP program	Reflected Stored	Independent of cross origin policy, No URL or token is required to be matched	More complex	Detection of complex multi-step attacks and attacks with no parameter and values

Table 3 Comparison of CSRF prevention/detection techniques

VII. CONCLUSIONS

To help organizations and developers reduce their application security risks in a cost effective manner, OWASP has produced numerous free and open resources that you can use to address application security in your organization. The following are some of the many resources OWASP has produced to help organizations produce secure web applications. [2]

- **Guidelines for Developers:**

➢ **Establish & Use Repeatable Security Processes and Standard Security Controls**

➢ **Application Security Requirements:**

To produce a secure web application, developers must define what secure means for that application. OWASP recommends to use the OWASP Application Security Verification Standard (ASVS), as a guide for setting the security requirements for application(s).

➢ **Application Security Architecture:**

Rather than retrofitting security into applications, it is far more cost effective to design the security in from the start. OWASP recommends the OWASP Developer's Guide, and the OWASP Prevention Cheat Sheets as good starting points for guidance on how to design security in from the beginning.

➢ **Standard Security Controls:**

Building strong and usable security controls is exceptionally difficult. A set of standard security controls radically simplifies the development of secure applications. OWASP recommends the OWASP Enterprise Security API (ESAPI) project as a model for the security APIs needed to produce secure web applications. ESAPI provides reference implementations in Java, .NET, PHP, Classic ASP, Python, and Cold Fusion.

➢ **Secure Development Lifecycle:**

To improve the process an organization follows when building such applications, OWASP recommends the OWASP Software Assurance Maturity Model (SAMM). This model helps organizations formulate and implement a strategy for software security that is tailored to the specific risks facing their organization.

- **Application Security Education:**

The OWASP Education Project provides training materials to help educate developers on web application security and has compiled a large list of OWASP Educational Presentations.

- **Guidelines for Verifier's:**

- **Code Review:**

OWASP has produced the OWASP Code Review Guide to help developers and application security specialists understand how to efficiently and effectively review a web application for security by reviewing the code.

- **Security and Penetration Testing:**

OWASP produced the Testing Guide to help developers, testers, and application security specialists understand how to efficiently and effectively test the security of web applications. This enormous guide, which had dozens of contributors, provides wide coverage on many web application security testing topics. [2].

REFERENCES

[1] "Web Server Security Guidelines", Online: Available, http://delhi.gov.in/wps/wcm/ connect/CISG-2004-04.pdf

[2] "The Ten Most Critical Web Application Security Risks", "sql-injection-prevention-cheat-sheet", "css-prevention-cheat-sheet","csrf-prevention-cheat-sheet", Online : available, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[3] "Improving Web Application Security: Threats and Countermeasures", online: available, http://msdn.microsoft.com/en-us/library/ff648653.aspx

[4] Online: available, http://www.veracode.com/security/sql-injection.

[5] Online: available, http://www.veracode.com/security/sql-prevention-cheat-sheet.

[6] Jan, Z.; Shah, M.; Rauf, A.; Khan, M.A.; Mahfooz, S., "Access Control Mechanism For Web Databases By Using Parameterized Cursor," In Future Information Technology (FutureTech), 2010 5th International Conference on , vol., no., pp.1,6, 21-23 May 2010.

[7] Singh, A.K.; Roy, S., "A network based vulnerability scanner for detecting SQLI attacks in web applications," Recent Advances in Information Technology (RAIT), 2012 1st International Conference on , vol., no., pp.585,590, 15-17 March 2012.

[8] Avireddy, S.; Perumal, V.; Gowraj, N.; Kannan, R.S.; Thinakaran, P.; Ganapthi, S.; Gunasekaran, J.R.; Prabhu, S., "Random4: An Application Specific Randomized Encryption Algorithm to Prevent SQL Injection," Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on , vol., no., pp.1327,1333, 25-27 June 2012.

[9] Rattipong Putthacharoen, Pratheep Bunyatnoparat ,"Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewriting Technique", Advanced Communication Technology (ICACT), 2011 13th International Conference on , vol., no., pp.1090,1094, 13-16 Feb. 2011.

[10] Hossain Shahriar and Mohammad Zulkernine, "Injecting Comments to Detect JavaScript Code Injection Attacks", Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual , vol., no., pp.104,109, 18-22 July 2011.

[11] E. Galan, A. Alcaide, A. Orfila, J. Blasco ,"A Multi-agent Scanner to Detect Stored-XSS Vulnerabilities", Internet Technology and Secured Transactions (ICITST), 2010 International Conference on , vol., no., pp.1,6, 8-11 Nov. 2010.

[12] Nenad Jovanovic, Engin Kirda, and Christopher Kruegel "Preventing Cross Site Request Forgery Attacks" In IEEE International Conference on Security and Privacy in Communication Networks (SecureComm), Securecomm and Workshops, 2006 , vol., no., pp.1,10, Aug. 28 2006-Sept. 1 2006

[13] Adam Barth, Collin Jackson, John C. Mitchell, "Robust Defenses for Cross-Site Request Forgery", CCS'08, October 27–31, 2008, Alexandria, Virginia, USA. Copyright ACM 978-1-59593-810-7/08/10.$5.00, 2008.

[14] Hossain Shahriar and Mohammad Zulkernine, "Client-Side Detection of Cross-Site Request Forgery Attacks", Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on , vol., no., pp.358,367, 1-4 Nov. 2010.

[15] Online:available,http://www.veracode.com/security/cross-site-scripting

[16] Online:available,http://www.linuxforu.com/2010/11/securing-apache-part-3-xsrf-csrf/

[17] Siddiqui, M.S.; Verma, D., "Cross site request forgery: A common web application weakness," Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on, vol., no., pp.538,543, 27-29 May 2011.