A Survey on Website Attacks Detection And Prevention

Satish L. Kuchiwale
Smt. Indira Gandhi College of Engineering
Koparkhairane, Navi Mumbai, India.
slbsatish@gmail.com

Shobha Lolge
Lokmanya Tilak College of Engineering,
Koparkhairne, Navi Mumbai, India.
lolgeshobha@gmail.com

ABSTRACT: Securing web applications has become extremely important as the information processed by web applications has become critical to corporations, customers, organizations, and countries. Web applications manage a wide array of information including financial data, medical records, social security numbers, intellectual property and national security data. Web based attacks are considered by security experts to be the greatest and least understood of all risks related to confidentiality, availability, and integrity. This paper will introduce and address web based attacks from attack to detection and solutions for the same. Attacks covered are SQL Injection (SQLI), Cross Site Scripting(CSS), Cross Site Request Forgery(CSRF), Directory Traversal, Unvalidated Input, Broken Access Control, Broken Authentication and Session Management and Parameter Modification.

KEYWORDS- Website attacks, Security, Vulnerability detection and Web applications.

Introduction

A Web application is an application program that is stored on a remote server and delivered over the Internet through a browser interface.

A web application or web service is a software application that is accessible using a web browser or HTTP(s) user agent. A web application security deals with the security of websites, web applications and web services. A poor application security measures can lead to breaches in data [1]:

- **Authentication**
  The process of uniquely identifying the clients of applications and services. These might be end users, other services, processes, or computers.

- **Authorization**
  The authenticated client is permitted to access the resources(include files, databases, tables, rows, and so on) and operations(performing transactions such as purchasing a product, transferring money from one account to another, or increasing a customer's credit rating ).

- **Integrity**
  Protection of information from tampering, forgery, or accidental changes.

- **Confidentiality**
  Ensures that applications and data is accessible to only the users intended and authorized to have access. It is the process of making sure that data remains private and confidential, and that it cannot be viewed by unauthorized users or eavesdroppers who monitor the flow of traffic across a network.

- **Availability**
  Ensures that authorized users have access to the application and the data when required. From a security perspective, availability means that systems remain available for legitimate users.

- **Accountability**
  Ensure accuracy of data and guide against unauthorized modifications.

Commonly web servers, application servers, and web application environments are susceptible to following types of vulnerabilities.

1. **SQL Injection**

In SQL-Injection we exploited the vulnerability by injecting SQL Queries as user inputs. A web application is vulnerable to SQL injection attacks when malicious content can flow into SQL queries without being fully sanitized, which allows the attacker to trigger malicious SQL operations by injecting SQL keywords or operators [2].
**SQL Injection.**

**Detection:** Content filtering, penetration testing, and defensive coding, IDS Approach, Generic Signatures, Accurate and Efficient Taint Propagation, Syntax-Aware Evaluation, Minimal Deployment Requirements [4].

**Prevention:**
- An Authentication Scheme using Hybrid Encryption.
- Effective SQL Injection Attack Reconstruction Using Network Recording.
- Insecure Query Processing in the Delay/Fault Tolerant Mobile Sensor Network (DFT-MSN) and Mobile Peer to Peer Network.
- Secure Query Processing In Delay Tolerant Network Using Java Cryptography Architecture.
- SQL Injection Attack Detection using the Removal of SQL Query Attribute Values [4].
- Dynamic Candidate Evaluations Approach to prevent SQL injection.
- Obfuscation-based Analysis of SQL Injection Attacks.
- SQL injection Detection via Automatic Test Case Generation of Programs.
- Combinatorial Method for Preventing SQL Injection Attacks.
- An Approach for SQL Injection Vulnerability Detection- AMNeSIA [4].
- Use language specific libraries to perform the same functions as shell commands and system calls.
- Check for existing reusable libraries to validate input, and safely perform system functions, or develop your own.
- Perform design and code reviews on the reusable libraries to ensure security.
- Use stored Procedures.
- Data validation (to ensure input isn’t malicious code).
- Run commands with very minimal privileges.
- A clustering approach for web vulnerabilities detection.
- Error pattern approach.
- SAFELI framework aims at identifying the SQL Injection attacks during the compile-time.
- Thomas et al., in suggest an automated prepared statement generation algorithm.
- Ruse et al. propose a technique that uses automatic test case generation.
- Haixia and Zhihong propose a secure database testing design for Web applications.
- Roichman and Gudes, in order to secure Web application databases, suggest using a fine-grained access control to Web databases.
- Shin et al. suggest SQLUnitGen, a Static-analysis-based tool that automate testing for identifying input manipulation vulnerabilities.
- Kemalis and Tzouramanis in suggest using a novel specification-based methodology.
- SQLrand approach using randomized SQL query language, targeting a particular CGI application is proposed by Boyd and Keromytis.
- Parse Tree Validation Approach compared the parse tree of a particular statement at runtime and its original statement.
- Algorithm with SQLCHECK on a real time environment by Su and Wassermann.
- In Manual approaches, defensive programming and code review are applied.
- Automated approach: Static analysis FindBugs and Web vulnerability scanning.

**2. Cross Site Scripting (CSS or XSS)**

In XSS, we inject code (basically client side scripting) to the remote server. A web application is vulnerable to XSS attacks when malicious contents can flow into web responses without being fully sanitized, which allows the attacker to execute malicious scripts in victims’ browsers, since the web browser trusts the contents returned by the web application under the same-origin policy.
Common consequences of XSS attacks include disclosure of users’ sensitive information, such as cookie details and credit card information.

There exist different forms of XSS attacks, depending on how malicious code is submitted to the vulnerable application and later echoed from the application to its users.

2.1 Non-persistent or Reflected XSS attack

This attack occurs when the victim interacting with a specially-crafted link which points to the vulnerable application and embeds the malicious code (e.g., as the value of a parameter or the name of a resource). When the link is activated the vulnerable web application immediately reflects the code to the user (e.g., as part of an error message). The code is then executed in the context of the vulnerable site and has access to all the information associated with the attacked application, such as authentication cookie or session information.

How Does Reflected XSS Work?

1. Attacker sends evil email
2. Victim clicks on link, sends request to vulnerable bank.com
3. Vulnerable bank website takes data from request and includes in valid web page
4. Victim’s browser now trusts the attacker’s script is from bank.com
5. Attacker has full access to victim’s account

http://bank.com/?id=’&gt;&lt;img src=x onerror=http://evil.com/attack.js&gt;

Figure 2.1 Reflected Cross Site Scripting

2.2 Persistent or Stored XSS attack

This attack occurs when a victim visits a page that has been exploited by a malicious user. The malicious code is first stored by the vulnerable application, and then, at a later time, it is presented to its users. In this case, the security of a user is compromised each time he/she visits a page whose content is determined using the stored malicious code [3].

In other words, Persistent XSS happens when the malicious scripts are sent to the application back-end database as, for example, forum posts, comments, etc, and stored for a period of time. The malicious scripts are triggered by the victim later when he visits a web page which contains the scripts.

Typical examples of vulnerable applications include guestbook applications or blog systems. If they allow users to submit entries containing scripting code, then they are vulnerable to persistent XSS attacks.

2.3 Document Object Model (DOM) Based XSS

This attack launched when injected code manipulates sites JavaScript code or variables, rather than HTML objects. JavaScript often use user inputs to modify the DOM. Input can be URL parameters XHR Responses, HTTP headers etc. Server side input validation logic fails at data sanitization.

A data structure which permits to access and modifies the content, structure, and style of HTML documents dynamically in client-side scripting code i.e. DOM. Browser populates some of its properties relies on request parameters, rather than document characteristics.

In this case, the vulnerable application presents to the users an HTML page that uses data from parts of its DOM in insecure ways. In other words, DOM-based XSS occurs when the malicious scripts are injected into the client-side JavaScript code for execution, even without sending to the server side. It is worth noting that DOM-based XSS is extremely difficult to handle using only server-side defenses.
For example, the document.URL and document.location properties are set to the URL of the document by the browser. If an HTML page contains code that dynamically changes the appearance of the page using the content of document.URL (e.g., to show to the user the URL associated with the page), it is possible to use a maliciously crafted URL to execute malicious scripting code.

### 3. Cross-Site Request Forgery (CSRF):

CSRF attacks are also known as XSRF, "Sea Surf", Session Riding, Cross-Site Reference Forgery, Hostile Linking and One-Click attack. A CSRF attack forces a logged-on victim’s browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim’s browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks.

### Detection:

- Manual Analysis
  - Binary / Bytecode
  - Source Code
- Automated Static Analysis
  - Binary / Bytecode
  - Source Code
- Dynamic Analysis with automated results interpretation
- Dynamic Analysis with manual results interpretation
- Architecture / Design Review
- WhiteHat Security’s Approach
  - Dynamic Analysis Software Testing (DAST)

### Prevention:

- Protecting Cookies from Cross Site Script attacks Using Dynamic Cookies Rewriting Technique.
- An Execution-flow Based Method for Detecting Cross-Site Scripting Attacks.
- Automatic Creation of SQL Injection and Cross-site Scripting (XSS) Attack.
- Perform data integrity checks on data prior to their submission to ensure the data are reasonable.
- When possible, restrict all end-user input to alphanumeric content.
- Filter output by converting text/data which might have dangerous HTML characters to its encoded format:
  - '<' and '>' to '&lt;' and '&gt;'
  - '(' and ')' to '&amp;' and '&lt;40;' and '&amp;#41;'
  - '#' and '&' to '&#35;' and '&amp;#38;'
- Recommend filtering on input as much as possible [5].
• Static Analysis Software Testing (SAST)
• human source-code reviews

Prevention:

• Eliminate any Cross Site Scripting vulnerabilities
  ▪ Not all CSRF attacks require XSS
  ▪ However XSS is a major channel for delivery of CSRF attacks
  ▪ Generate unique random tokens for each form or URL, which are not automatically transmitted by the browser.
• Do not allow GET requests for sensitive actions. For sensitive actions, re-authenticate or digitally sign the transaction.

4. Directory or Path Traversal

Web server consist of directories where files, information, application functions are stored to provide the services which do not have access to users. But attackers obtain these unauthorized directories, by traversing the directory in the address area of web browser and may misuse it[1].

Directory traversal is a type of HTTP exploit that is used by attackers to gain unauthorized access to restricted directories and files.

The goal of a Directory Traversal attack is to execute commands that will access files that are intended to be restricted. This type of attack uses HTTP to bypass Web server and Web application security. It is enabled by insufficient and missing security measures in servers and websites.

These attacks can be viewed in two basic groups: attacks that target directory traversal vulnerabilities in the web server and attacks that target vulnerabilities in application code.

Detection:

• Enterprises commonly rely on vulnerability scanning and manual penetration testing techniques to detect directory traversal vulnerabilities.
• Directory traversal vulnerabilities can be located in web server software/files or in application code that is executed on the server.

Prevention:

• Input validation assures that attackers cannot use commands that leave the root directory or violate other access privileges.
• Filters can be used to block certain user input. Enterprises typically employ filters to block URLs containing commands and escape codes that are commonly used by attackers.
• Escape codes and directory paths should be filtered out to ensure that only safe inputs are passed to the Web server.
• Recent web servers, including IIS and Apache, provide protection against this type of directory traversal attack.
• Web servers use two types of mechanisms to prevent access unauthorized access to restricted directories and files:
  i. The Root Directory – Limits user access to a specific directory (known as the root directory) and its subdirectories. The root directory is the top-most directory on a server file system. User access is confined to the root directory, meaning users are
unable to access directories or files outside of the root.

ii. Access Controls Lists (ACLs) – Limits access to specific files within the root directory. Only users who are listed in the ACL for a file can access that file. Administrators use Access Control Lists to define user access rights and privileges for viewing, modifying, and executing files.

5. Unvalidated Input

Web application request inputs from user to determine how to respond in accordance to provide web service. The user enters input values to web application request. Attackers may pass harmful information to the web application which tries to bypass the website’s security mechanisms.

**Detection:** All user input that provided to the web applications requested, need to check by against strict format that specifies exactly what input must be allowed. Ensure that all parameters are validated before they are used. A tool or library is most effective, as the performing the checking should be placed.

**Prevention:** All the input parameters to the web application should be validated against specification such as use of data type, length of all fields, whether null, duplicates are allowed, parameter is required or not, numeric range, specific legal values (enumeration), patterns (regular expressions).

6. Broken Access Control

In web application the users are categorized in the different level of privileges. Access control determines how the web application allows access to functions to some users and not others, also called authorization. But attacker may be access higher level of authority.

**Detection:** The code implementation of the access control policy should be verified. Penetration testing can be useful in verifying if there are problems in the access control. In the web application, if there is categories of users that can be accessed through the interface, verify each interface to make sure that only authorized users can allowed access.

**Prevention:** Attackers will use path traversal method, which provide path information as part of a request for information. Such attacks may try to access files that are normally not directly accessible. Such attacks can be submitted in URLs as well as any other input that ultimately accesses a file. In such case redirect the page to custom error page message [1].

7. Broken Authentication and Sessions Management

Web application creates session when the user logged in, which specify the period of time that a unique user interacts with a web application. Using session maintains state by providing the client with a unique id. This id is stored in a cookie which is used between the user browser and web server. If this session’s details are not protected correctly, attacker can steal it and misuse it [1].

**Detection:** Detailed review of authentication mechanisms to ensure that user’s credentials are protected and only an authorized user can change them. Review session management mechanism, that session identifiers are always protected.

**Prevention:** Ensuring that implementation consistently enforces to have a secure authentication and session management mechanism which include passwords should have restrictions that require a minimum size; securing session id, a user's entire session should be protected; browser cache protection, authentication and session information never submitted as part of a GET parameter. Authentication pages must be specified with no cache tag to protect from using the back button in a user's browser; session tokens should be expired on the server, and destroyed when a browser is closed; Everywhere authenticate must be provided.

8. Parameter Modification

Parameter modification is the problem where the attacker’s do not fill the form but rather passes the parameters from URL itself, bypassing the form validations. Therefore it may lead to ambiguous effect on the form data and the overall site data.

**Detection:** Ensure that application must not allow parameter values, query string or form GET parameters to the URL. These are handled through dynamic parameter detection.
Prevention: In this type of vulnerability attacker involves providing the input parameter in the URL of the web application in such case don’t allow the URL as for example “………../User Login Page? Username = xyz & password =xyz123”. In such case redirect the page to custom error page message.

CONCLUSION

Web applications have been growing extremely fast with innovative programming languages and technologies. This results in challenges for web application security, which requires extensive and continuous efforts from security researchers. This paper provides survey on few web site attacks such as SQL Injection (SQLI), Cross Site Scripting (CSS), Cross Site Request Forgery (CSRF), Directory Traversal, Unvalidated Input, Broken Access Control, Broken Authentication and Session Management and Parameter Modification with detection and prevention schemes. But with various complexities such as an increasing amount of application code and logic, multiple web applications are integrated and embedding third-party programs with security it is not possible to provide clean solution to all these attacks. As new types of attacks are always up-and-coming which requires security professionals to quickly act in response and put a massive number of web applications at risk.

REFERENCES