Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 9, Issue, 03, pp.47370-47373, March, 2017

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

ISSN: 0975-833X

RESEARCH ARTICLE

DEBUNKING OF COMMON ATTACKS IN WEB APPLICATIONS

***Vamsi Mohan, V. and Dr. Sandeep Malik**

Department of Computer Science, School of Engineering and Technology, Raffles University, Neemrana

ARTICLE INFO

Article History:
Received 19th December, 2016
Received in revised form
10th January, 2017
Accepted 15th February, 2017
Published online 31st March, 2017

Key words:

Web Attacks, SQL injections,
Cross Site Scripting (XSS).

ABSTRACT

Cyber-attacks are becoming commonplace in our society these days as the world becomes more connected. Security attacks on information security infrastructure have continued to evolve steadily and legacy network based attacks have been replaced by more sophisticated web application based attacks. It is vitally important to discuss web application attacks considering the number of attacks happened in the recent years. In this paper, we define each of them in detail and emphasize their differences. We also conclude our studies in this area with providing a diagram which gives a comprehensive overview about Web attacks.

Citation: Vamsi Mohan, V. and Dr. Sandeep Malik, 2017. "Debunking of common attacks in web applications", *International Journal of Current Research*, 9, (03), 47370-47373.

INTRODUCTION

The rapid and tremendous growth of Information and Communication Technology has increased access to web applications. The increased usage of the Internet and network technology has changed the focus in assessing computers. Increased access has paved the way for security and vulnerable threats in the form of attacks in web applications. Web based attacks are considered by security experts to be the greatest and frequent times the least understood of all risks related to confidentiality, stability, availability, and integrity. The purpose of a web based attack is significantly different than any other attacks. Web based attacks focus on an application itself and functions on layer 7 of the OSI. John Pescatore of the Gartner group claims that nearly 70% of all attacks occur at the application layer (Desmond, Paul, 2004). J. Fonseca, M. Vieira, and H. Madeirastated in their publication, vulnerabilities are viewed as sensible on the grounds that they are gotten from the broad field study on genuine web application vulnerabilities introduced in (Fonseca *et al*., 2010), and are infused by set of delegate limitations and tenets characterized in (Buehrer *et al*., 2005). According to the OWASP, there are top 10 web application attacks namely SQL Injection (SQLi), Broken Authentication and Session Management, Cross-Site Scripting (XSS), Insecure Direct Object References, Security Miscon figuration, Sensitive Data Exposure, Missing Function Level Access Control, Cross-Site Request Forgery, Using Components With Known Vulnerabilities and Unvalidated Redirects and Forwards (OWASP top 10 security flaws, 2013).

Corresponding author:* **Vamsi Mohan, V.
Department of Computer Science, School of Engineering and Technology, Raffles University, Neemrana

1.OWASP Top 10 Web Application Attacks

According to the TechTarget, the OWASP Top Ten is a list of the 10 most dangerous current Web application security flaws, along with effective methods of dealing with those flaws. OWASP (Open Web Application Security Project) is an organization that provides unbiased and practical, cost-effective information about computer and Internet applications (TechTarget OWASP top ten, 2013).

SQL Injection (SQLi)

SQL Injection (SQLi) refers to an attack where an attacker can inject malicious SQL statements (referred as a malicious payload) that control a web application's database server. SQL Injection vulnerabilities could affect any web application that usesthe SQL-based database. SQLi is the oldest and critical vulnerability. It is most prevalent and dangerous for web applications. By leveraging SQL injection vulnerability, an attacker can bypass a web application's authentication and authorization mechanism and retrieve the contents of an entire database. SQL Injection can be used to add, modify and delete records in a database and it affects the data integrity. SQL Injection can provide attackers with unauthorized access to sensitive data including, bank account details, trade secrets, intellectual property rights and other sensitive information.

How SQL Injection (SQLI) works?

To run malicious SQL queries against a database server, first an attacker must find an input within the web application that is included inside of an SQL query. For attacking the web application, the vulnerable website needs to directly include user input within an SQL statement. Then, an attacker can insert a payload that will be included as part of the SQL query and run against the database server. The following server-side pseudo-code is used to authenticate users to the web application.

```
# Define variables
txtUserName = getRequestString("username");
txtPasswd = getRequestString("password");

# SQL query vulnerable to SQLi
sql = "SELECT id FROM users WHERE username='" + txtUserName+ "' AND password='" +
txtPasswd + "'";
```

The above code is a simple example of authenticating a user with a username and a password against a database table. The written code is vulnerable to SQL Injection because an attacker could submit malicious input in a way that would alter the SQL statement being executed by the database server. A simple example of an SQL Injection payload could be as simple as setting the password field to password' OR 1=1. The result would be in the following SQL query being run against the database server.

```
SELECT id FROM users WHERE username='username' AND password='password' OR 1=1'
```

An attacker can also comment out the rest of the SQL statement to control the execution of the SQL query further. Once the query executes, the result is returned to the application to be processed, resulting in an authentication bypass.

```
-- MySQL, MSSQL, Oracle, PostgreSQL, SQLite
' OR '1'='1' --
' OR '1'='1' /*
-- MySQL
' OR '1'='1' #
-- Access (using null characters)
' OR '1'='1' %00
' OR '1'='1' %16
```

Broken Authentication and Session Management

An attacker (an anonymous attacker, a user with own account who may attempt to steal data from other accounts, or an insider malicious actions) uses leaks or flaws in the authentication or session management functions to impersonate other users. The applications which is poorly coded related to authentication and session management are often allow attackers to compromise passwords, keys, or session tokens, or to exploit implementation flaws. Developers build custom authentication and session management techniques while developing web site.

According to CodeDx enterprise, there are seven reasons an application may be vulnerable (Code, 2004).

1) User authentication credentials aren't protected, when stored using hashing or encryption techniques.
2) Credentials can be guessed in case of weak account management functions.
3) Session IDs are exposed in the URL.
4) Session IDs are vulnerable to session fixation attacks.
5) Sessions don't timeout or SSO is not invalidated during logout.
6) Session IDs aren't rotated after a successful login.
7) Passwords, session IDs and other credentials are sent over unencrypted channels.

Cross-Site Scripting (XSS)

Cross-site Scripting (XSS) refers to the client-side code injection attack where an attacker can execute malicious scripts (referred as malicious payload) into a legitimate website or web application. XSS is the most rampant of web application vulnerabilities and occurs when a web application makes use of unvalidated user input within the output it generates. According to Incapsula security blog, Cross site scripting (XSS) is a common attack vector that injects malicious code into a vulnerable web application. XSS differs from other web attacks (e.g., SQL injections), in that it does not directly target the application itself. Instead, the users of the web application are the ones at risk (Incapsula security blog, 2016). There are primarily three types of XSS attacks (Vikas K. Malviya *et al.*, 2013; https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS); http://excess-xss.com/#xss-attacks) i.e., Persistence XSS, Non- Persistence XSS and DOM based XSS. XSS will take advantage of VBScript, ActiveX code, and Flash files. However, it will impact majorly on Javascript, which is widely used in the web applications and portals.

How Cross site scripting (XSS) works?

To run malicious javascript code in the victim's web browser, the attacker must first find a way to inject a payload into a web page. Generally, attackers use social engineering techniques to convince user to visit a vulnerable page with an injected JavaScript payload. An XSS attack to take place the vulnerable website needs to directly include user input or malicious code in its pages. The following server-side pseudo-code is used to display the comments on a web page. The above web page is vulnerable to Cross Site scripting (XSS) because an attacker could submit a comment that contains a malicious payload. Eg: <script>executeMaliciousCode();</script>. Users visiting the web page will get served the following HTML page.

```
print "<html>"
print "<h1>User Comment</h1>"
print database.userComment
print "</html>"
```

When the page loads in the victim's browser, the attacker's malicious script will execute, without the user permission or realization.

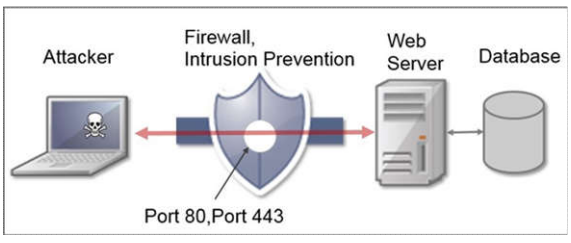
```
<html>
<h1>User Comment</h1>
<script>executeMaliciousCode();</script>
</html>
```

Insecure Direct Object References

The threat of insecure direct object reference flaws has become common with the increased complexity of web applications that provide varying levels of access to enable users to gain entry to some components. When a developer exposes a reference to an internal implementation object such as a file, directory or key a direct object reference will be created. Without an access control check, attackers can manipulate these references to access the unauthorized data. These vulnerabilities result from a name or key of an object being used during web site development. Potential threats will be introduced from an authorized user of the system who alters a parameter value that directly points to an object that the user doesn't authorized to access. The user may be an authorized user to the system. However, he or she may not have access to specific objects or sections depends on the user role such as database records and reports. If the application won't verify the user for that specific object, it can result in an insecure direct object reference flaw. It is easy to detect these flaws. Each location that a user can supply input and points directly to reference objects needs to be tested. There are many tools available to track the flows. By manipulating parameter values, testers can identify the flaw and analyze the code to determine whether a user is able to bypass authorization and retrieve objects that are not intended for them. To prevent these vulnerabilities, it is important to have a strong access control policies are in place in the organizations. Developers need to ensure that users have proper authorization to gain access to the direct references and restricted access where they are not entitled to access the confidential modules or information.

Security Misconfiguration

Application Misconfiguration attacks exploit many configuration weaknesses found in web applications. Strong application security configurations are vital for any organization. Security misconfiguration is simply that incorrectly assembling the safeguards such as parameters, firewalls for a web application. These mis-configurations typically occur when faults in the system configuration by systems administrators, DBAs or developers. These can occur at any level in the application stack including the platform, web server, application server, database, framework and custom code modules. These security mis-configurations can lead an attacker right into the system and result in a partially or even totally compromised the system. Attackers find these mis-configurations in the system by trying through unauthorized access to accounts, unused web pages, flaws, unprotected files and directories. If the system is not robust respect to security configurations, data can be stolen or modified over time and can be a time consuming and costly affair to recover back.



Sensitive Data Exposure

Portswigger web security online portal states, Sensitive data exposure vulnerabilities can occur, when an application does not adequately protect the sensitive information from being disclosed to attackers (Portswigger web security portal, 2016). For many applications this may be limited to information such as passwords, health records, credit card data, session tokens, or other authentication credentials. This is not a vulnerability that you can investigate same as other vulnerabilities. Most vulnerabilities within this category cannot be scanned.

The usage of cryptography, SSL usage and data protection are recommended for security sensitive data.

To Prevent 'Sensitive Data Exposure':

1. Consider using encryption techniques both at rest and in transit (https) requests.
2. Don't store sensitive data unnecessarily until it is vitally required. Discard it as soon as possible.
3. Ensure standard algorithms and strong keys are used, and proper key management is in place to store the data.
4. Ensure passwords are stored with an algorithm specifically designed for password protection.
5. Disable autocomplete form features collecting sensitive data and disable caching for pages that contain sensitive data.

Missing Function Level Access Control

According to GitHub Security (2013), Function level access control vulnerabilities could result from insufficient protection of sensitive request handlers within an application. An application may hide access to sensitive actions, unable to enforce sufficient authorization for certain actions, or accidentally expose an action through a user controlled request parameter. These vulnerabilities could be much more complex and be the result of delicate edge-cases in the underlying application logic. Developers ignores to remove the input controls from the code, when it is not necessary. They hide or disable the input controls from UI (frontend). Attackers take this as an advantage and use various tools to access it and enable the form controls to perform for bad operations.

To Prevent Missing Function Level Access Controls':

1. Proper authentication and authorization should be performed on both front end and backend
2. Critical validations need to be conducted at both client side and server side instead of client side validations.
3. Do not hard code the values in the application code.

Cross-Site Request Forgery

OWASP states that Cross-Site Request Forgery (CSRF or XSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. Since the attacker has no way to see the response to the forged request, XSRF attacks specifically target state-changing requests. With a little help of social engineering (through sending phishing mails), an attacker may trick the users of a web application into executing actions of the attacker's choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing credentials, doing transactions forcefully, changing email address, and so forth. If the victim is an administrative account, even XSRF can compromise the entire web application (OWASP Cross-Site

Request Forgery (CSRF), 2016). An attacker will use XSRF to trap the victim to access a website or to click a URL that contains malicious, unauthorized requests. XSRF attack will use the identity and privileges of the victim and impersonate them in order to perform any actions desired by the attackers. For performing a XSRF attack, the user should be authenticated with the website initially. Assuming the victim is authenticated, the attacker can include a link or script in a third-party website that victim visits. Then, when the victim visits that website or link, the rogue script will be executed without the victim being noticed of it.

Difference between Cross Site Scripting (CSS) and Cross-Site Request Forgery (XSRF)

According to Acunetix a website security audit organization, CSRF attack includes a malicious exploit of a website in which a user will transmit malicious requests that the target website trusts without the user's permission. In Cross-Site Scripting (XSS), the attacker exploits the trust a user has for a website, on the other hand with XSRF, the attacker exploits the trust a website has against a user's browser.

Using Components With Known Vulnerabilities

It is extremely common existing vulnerabilities in the third party libraries, APIs and software, which could be used to compromise the security of the systems using the software. Over the period of several years approximately 4500 Common Vulnerabilities and Exposures have been published per year. Speaking at the recent RSA Conference Europe 2012 in London, Don Smith, technology director at Dell Secure Works, says that "Organizations fails to deal with known vulnerabilities stems from the vast increase in IT complexity over the past couple of decades" (RSA Conference, 2012). According to Outpost24, "Components with known vulnerabilities are often, but not always, the first vulnerabilities that are detected during a penetration testing assignment" (https://www.outpost24.com/using-components-known-vulnerabilities-owasp10).

```
<%    String redirectURL = "http://www.RedirectedPage.com/";
      response.sendRedirect(redirectURL);
      return;
%>
```

Unvalidated Redirects and Forwards

Web applications generally redirect and forward users to other web pages and websites, and use un-trusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Preventing Unvalidated Redirects and Forwards

1. Avoid using redirects and forwards.
2. In case of redirects or forwards, avoid allowing the URL as user input for the destination. If redirection is necessary, validate the URL before redirection.
3. It is recommended any destination input should be mapped to a value, rather than the actual URL or partial URL.
4. Sanitize input by creating a list of trusted URL's before developing the applications.
5. Introduce intermediate pages to notify the users that the control redirects to external website.

Conclusion

Web application attacks are increasing proportional to the increasing of digital transactions. As users have moved more towards web for their daily transactions, fraudsters and attackers have followed, targeting to creating security threats based on Web applications. In this paper, we explained OWASP top 10 vulnerabilities and preventions. Because the most serious threat which occurs commonly in today's web attacks are SQL injections (SQLI) and Cross Site Scripting (CSS), we tried to concentrate more on them and tried explaining with detailed examples.

REFERENCES

Acunetix Cross Site Request Forgeryhttp://www.acunetix.com/websitesecurity/csrf-attacks/

Buehrer, G., B. Weide, and P. Sivilotti, "Using Parse Tree Validation to Prevent SQLi Attacks," Proc. Int"l Workshop Software Eng. and Middleware, 2005

Code Dx enterprisehttps://codedx.com/broken-authentication-session-management/

Desmond, Paul 2004, May 17. All-out blitz against Web app Attacks Retrieved December 30, 2006, from networkworld.com Web site: http://www.networkworld.com/techinsider/2004/0517techinsidermain.html

Fonseca, J., M. Vieira, and H. Madeira,"The Web Attacker Perspective- A Field Study," Proc. IEEE Intl. Symp. Software Reliability Eng., Nov. 2010

GitHub Security portal https://bounty.github.com/classifications/missing-function-level-access-control.html

http://excess-xss.com/#xss-attacks

https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS)

Incapsula security bloghttps://www.incapsula.com/web-application-security/cross-site-scripting-xss-attacks.html

Outpost 24 journal https://www.outpost24.com/using-components-known-vulnerabilities-owasp10

OWASP Cross-Site Request Forgery (CSRF)https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

OWASP top 10 security flaws https://www.owasp.org/index.php/Top_10_2013-Top_10

Portswigger web security portalhttps://support.portswigger.net/customer/portal/articles/1965730-using-burp-to-test-for-sensitive-data-exposure-issues

RSA Conference https://www.rsaconference.com/events/eu12/agenda/sessions/610/breaking-the-failure-cycle-why-breaches-from-known

TechTarget OWASP top ten http://searchsoftwarequality.techtarget.com/definition/OWASP-Top-Ten

Vikas K. Malviya, SaketSaurav, Atul Gupta, "On Security Issues in Web Applications through Cross Site Scripting (XSS)", APSEC-2013, pp 583-588.

****