IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 2 Issue 4, April 2015.
www.ijiset.com
ISSN 2348 – 7968

Security Testing for Web Applications

Purnima Bindal[1], Purnima Khurana[2] and Kamlesh Kumar Raghuvanshi[3]

[1,2,3] Assistant Professor ,
Department of Computer Science PGDAV College(M),
University of Delhi, Delhi, India

Abstract

With the wide use of computer, software is also being used on a large scale and is becoming more and more complicated, which also results in increasingly growing software security problems. Software security is the ability of software to provide required function when it is attacked. Security testing is a process that is performed with the intention of revealing flaws in security mechanisms and finding the vulnerabilities or weaknesses of software applications and taking necessary actions to overcome these problems.

Keywords: *Security testing, Vulnerability, Web Application*

1. Introduction

There is growing concern about security testing, because it is regarded as an important means to improve security of software. Software security testing is the process to identify whether the security features of software implementation are consistent with the design. Software security testing can be divided into security functional testing and security vulnerability testing. Security functional testing ensures whether software security functions are implemented correctly and consistent with security requirements based on security requirement specification. Software security requirements mainly include data confidentiality, integrity, availability, authentication, authorization, access control, audit , privacy protection, security management etc. Security vulnerability testing is to discover security vulnerabilities as an attacker. Vulnerability refers to the flaws in system design, implementation, operation, management. Vulnerability may be used to attack, resulting in a state of insecurity. Security vulnerability testing is to identify software security vulnerabilities. In this paper, the current methods, techniques and tools of security vulnerability testing are analyzed and summarized. We will be focusing on security of Web Applications in our paper [1].

2. What is Security Testing?

The prime objective of security testing is to find out how vulnerable a system may be and to determine whether its data and resources are protected from potential intruders. Online transactions have increased rapidly of late making security testing as one of the most critical areas of testing for such web applications. Security testing is more effective in identifying potential vulnerabilities when performed regularly [2].

Security testing is a type of software testing carried out by specialized team of software testers. Objective of security testing is to make the software secure from external or internal threats caused either by humans or malicious programs. Security testing basically checks, how good is software's authorization mechanism, how strong is authentication, how software maintains confidentiality of the data, how does the software maintain integrity of the data, what is the availability of the software in an event of an attack on the software by hackers and malicious programs. Security testing requires good knowledge of application, technology, networking, security testing tools. With increasing number of web applications necessarily of security testing has increased to a greater extent [3].

Security testing is a testing technique to determine if an information system protects data and maintains functionality as intended. It also aims at verifying 6 basic principles as listed below:

- *Confidentiality:* It is a set of rules or a promise that limits access or places restrictions on certain types of information. Information has value, especially in today's world. Bank account statements, personal information, credit card numbers, trade secrets, government documents. Everyone has information they wish to keep a secret. Protecting such information is a very major part of information security.

- *Integrity:* Data integrity means maintaining and assuring the accuracy and consistency of data over its entire life-cycle. This means that data

ISSN 2348 – 7968

cannot be modified in an unauthorized or undetected manner.

- *Availability:* The information must be available when it is needed. This means that the computing systems used to store and process the information, the security controls used to protect it, and the communication channels used to access it must be functioning correctly. High availability systems aim to remain available at all times, preventing service disruptions due to power outages, hardware failures, and system upgrades. Ensuring availability also involves preventing denial-of-service attacks, such as a flood of incoming messages to the target system essentially forcing it to shut down.

- *Authentication:* In information security, it is necessary to ensure that the data, transactions, communications or documents (electronic or physical) are genuine. It is also important for authenticity to validate that both parties involved are who they claim to be. Some information security systems incorporate authentication features such as "digital signatures", which give evidence that the message data is genuine and was sent by someone possessing the proper signing key.

- *Authorization:* The function of specifying access rights to resources related to information security and computer security in general and to access control in particular.

- *Non-repudiation*: One's intention to fulfill their obligations to a contract. It also implies that one party of a transaction cannot deny having received a transaction nor can the other party deny having sent a transaction[4], [5],[6].

3. Why Security Testing ?

Security testing, in the current scenario, is a must to identify and address web application security vulnerabilities and to avoid any of the following:

- Security breaches resulting in loss of data, service downtime and brand damage highly cost businesses.
- Website downtime, time loss and expenditures in recovering from damage (reinstalling services, restoring backups, etc.)
- Cost associated with securing web applications against future attacks.
- Related legal implications and fees for having lax security measures in place.
- Disturbance to your online means of revenue generation/collection.
- Loss of customer trust.

4. Security Testing Types

4.1 Dynamic application security testing (DAST)

It is often referred to as testing from the outside in. Dynamic analysis involves performing tests on a running instance of an application and is also known as black box testing. The security test will involve sending requests to the application and observing the responses to see if there was any indication that security vulnerability may be present. Dynamic analysis can be an effective way to test applications, but it has some limitations as well. First of all, because the testing is based on analyzing request and response patterns, the results obtained are really only a guess about the internal state of the application -- the tester typically has no knowledge of the actual application source code and what the actual internal state of the application is. In addition, because the tester is only looking at the observable behavior of the application and cannot know the entire attack surface, there is a chance that areas of the application and components of its functionality will be excluded from the test. Also some responses might not obviously indicate that a security vulnerability is present. These factors lead to the potential for false negatives -– situations where there is a security vulnerability that goes unnoticed and unreported. DAST tools and processes are applied to applications that are nearly ready for production, or are already in production. DAST tools and services apply different attack scenarios to see how well the applications react.

4.2 Static application security testing (SAST)

It is often referred to as testing from the inside out. SAST tools and processes evaluate the application code for possible vulnerabilities. Static analysis involves reviewing application assets like source code, configuration files and so on when they are static -- or at rest. This is also known as source code analysis or white box testing. Static analysis opens up opportunities for a more thorough

ISSN 2348 – 7968

analysis because the analysis being performed has access to the "ground truth" of the source code. Analysts do not have to observe the behaviour of an application and make guesses about the internal state of the system; instead the analyst has access to the actual instructions the software will follow when put into production. This can help to reduce false positives as well as reduce false negatives. One drawback to static analysis is that it can fail to identify security issues that are bound up in the specific configuration of the deployed system -- for example, static analysis will not be able to identify issues that would arise due to administrators failing to install Web server or operating system patches. SAST is typically performed before the application is released to production [7] , [2],[7].

5. Some Key Terms used In Security Testing

Before we go further, it will be useful to be aware of a few terms that are frequently used in web application security testing:

What is "Vulnerability"?
This is a weakness in the web application. The cause of such a "weakness" can be bugs in the application, an injection (SQL/ script code) or the presence of viruses.

What is "URL manipulation"?
Some web applications communicate additional information between the client (browser) and the server in the URL. Changing some information in the URL may sometimes lead to unintended behavior by the server.

What is "SQL injection"?
This is the process of inserting SQL statements through the web application user interface into some query that is then executed by the server.

What is "XSS (Cross Site Scripting)"?
When a user inserts HTML/ client-side script in the user interface of a web application and this insertion is visible to other users, it is called XSS.

What is "Spoofing"?
The creation of hoax look-alike websites or emails is called spoofing [8].

6. Steps while Preparing and Planning for Security Testing

- *Security Architecture Study:* The first step is to understand the business requirement, security goals and objective in terms of security compliance of the organization. The test planning should consider all security factors.
- *Security Architecture Analysis:* Understand and analyze the requirements of the application under test.
- *Classify Security Testing:* Collect all system setup information used for development of Software and Network like Operating Systems, technology, hardware.
- Make out the list of Vulnerabilities and Security Risks.
- *Threat Modeling:* Based on above step prepare Threat profile.
- *Test Planning:* Based on identified Threat, Vulnerabilities and Security Risks prepare test plan to address these issues.
- *Traceability Matrix:* For each identified Threat, Vulnerabilities and Security Risks prepare Traceability Matrix.
- *Security Testing Tool Identification:* All security testing cannot possible to execute manually, so identify the tool to execute the all security test cases faster & more reliable.
- *Test Case Preparation:* Prepare the Security tests case document.
- *Test Case Execution:* Perform the Security Test cases execution and retest the defect fixes. Execute the Regression Test cases.
- *Reports:* Prepare detailed report of Security Testing which contains Vulnerabilities and Threats contained, detailing risks, and still open issues etc.

7. Security Testing Techniques

To prevent all of the security testing threats/flaws and perform security testing on a web application, it is required to have good knowledge of the HTTP protocol and an understanding of client (browser) – server communication through HTTP. Also, basic knowledge of SQL injection and XSS is required [2]. The following techniques will help in performing quality security testing:

ISSN 2348 – 7968

7.1 Ethical Hacking

Ethical hacking means hacking performed by a company or individual to help identify potential threats on a computer or network. An ethical hacker attempts to bypass the system security and search for any vulnerability that could be exploited by malicious hackers aka Black hats. White hats may suggest changes to systems that make them less likely to be penetrated by black hats [2].

7.2 Password Cracking

Password cracking is the most critical part while doing system testing. In order to access the private areas of an application, hackers can use a password cracking tool or can guess a common username/password. Common usernames and passwords are easily available online along with open source password cracking applications. Until a web application enforces a complex password (e.g. a long password with a combination of numbers, letters, and special characters), it is easy to crack the username and password. Another way of cracking the password is if username/password is to target cookies if cookies are stored without encryption [2]

7.3 SQL Injection

Entering a single quote (‘) in any textbox should be rejected by the application. Instead, if the tester encounters a database error, it means that the user input is inserted in some query which is then executed by the application. In such a case, the application is vulnerable to SQL injection.
SQL injection attacks are very critical as attackers can get vital information from the server database. To check SQL injection entry points into your web application, find out code from your code base where direct MySQL queries are executed on the database by accepting some user inputs .
SQL Injection Testing can be done for:

- Apostrophes
- Brackets
- Commas
- Quotation marks [2]

7.4 Cross-Site Scripting (XSS)

The tester should additionally check the web application for XSS (Cross site scripting). Any HTML e.g. <HTML> or any script e.g. <SCRIPT> should not be accepted by the application. If it is, the application can be prone to an attack by Cross Site Scripting.
Attackers can use this method to execute malicious scripts or URLs on a victim’s browser. Using cross-site scripting attackers can use scripts like JavaScript to steal user cookies and information stored in the cookies .
Cross Site Scripting Testing can be done for:

1. Apostrophe
2. Greater-Than Sign
3. Less-Than Sign[2]

7.5 Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. An attacker can manipulate direct object references to access other objects without authorization, unless an access control check is in place.
For example, in Internet Banking applications, it is common to use the account number as the primary key. Therefore, it is tempting to use the account number directly in the web interface. Even if the developers have used parameterized SQL queries to prevent SQL injection, if there is no extra check that the user is the account holder and authorized to see the account, an attacker tampering with the account number parameter can see or change all accounts .

HOW TO TEST

To test this vulnerability the tester first needs to map out all locations in the application where user input is used to reference objects directly. For example, locations where user input is used to access a database row, a file, application pages and more. Next the tester should modify the value of the parameter used to reference objects and assess whether it is possible to retrieve objects belonging to other users or otherwise bypass authorization.

The best way to test for direct object references would be by having at least two (often more) users to cover different owned objects and functions. For example, two users each having access to different objects (such as purchase information, private messages, etc.), and (if relevant) users with different privileges (for example administrator users) to see whether there are direct references to application functionality. By having multiple users the tester saves valuable testing time in guessing different object names as he can attempt to access objects that belong to the other user [10],[11].

7.6 *Cross-Site Request Forgery (CSRF)*

Cross-Site Request Forgery is an attack that forces an end user to execute unwanted actions on a web application in

ISSN 2348 – 7968

which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), 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 their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application .

HOW TO TEST

The tester must know URLs in the restricted (authenticated) area. The tester may induce a legitimate, logged in user into following an appropriate link. This may involve a substantial level of social engineering.

Either way, a test case can be constructed as follows:

- Let *u* the URL being tested; for example, u = http://www.example.com/action
- Build an html page containing the http request referencing URL u make sure that the valid user is logged on the application;
- Induce him into following the link pointing to the URL to be tested (social engineering involved if you cannot impersonate the user yourself);
- Observe the result, i.e. check if the web server executed the request[12],[13].

7.7 Unvalidated Redirects and Forwards

Unvalidated redirects and forwards are possible when a web application accepts untrusted input that could cause the web application to redirect the request to a URL contained within untrusted input. By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts may have a more trustworthy appearance. Unvalidated redirect and forward attacks can also be used to maliciously craft a URL that would pass the application's access control check and then forward the attacker to privileged functions that they would normally not be able to access .

The best way to find out if an application has any unvalidated redirects or forwards is to:

1. Review the code for all uses of redirect or forward. For each use, identify if the target URL is included in any parameter values. If so, verify the parameter(s) are validated to contain only an allowed destination, or element of a destination.
2. Also, spider the site to see if it generates any redirects (HTTP response codes 300-307, typically 302). Look at the parameters supplied prior to the redirect to see if they appear to be a target URL or a piece of such a URL. If so, change the URL target and observe whether the site redirects to the new target.
3. If code is unavailable, check all parameters to see if they look like part of a redirect or forward URL destination and test those that do [14],[15].

7.8 URL manipulation through HTTP GET methods

HTTP GET method is used between application client and server to pass on the information. The tester needs to verify if the application is passing vital information in the query string. The information via HTTP is passed in parameters in the query string. To test this, a parameter value can be modified in the query string to check if the server accepts it.

Generally user information is passed through HTTP GET request to the server for either authentication or fetching data. Hackers can manipulate the input of this GET request to the server so that the required information can be gathered or to corrupt the data. Any abrupt behavior of application or web server, in such condition, is the key for a hacker to slip into the application.

Ad hoc Data Testing can also be done as a part of security testing:

- Testing random data which is included in requests.
- Testing random data which is included as parameters.
- Testing encoded random data included as parameters [2].

7.9 Service Access Points (Sealed and Secure Open)

Today, businesses depend and collaborate with each other, same holds good for applications especially websites. In such case, both the collaborators should define and publish some access points for each other. So far the scenario seems quite simple and straightforward but, for some web based product like stock trading, things are not so simple and easy. When there is large number of target audience, the access points should be open enough to facilitate all users, accommodating enough to fulfill all users' requests and secure enough to cope with any security-trial.

How to Test Service Access Points:

In some cases these access points can be sealed for unwanted applications or people**.** This depends upon the

ISSN 2348 – 7968

business domain of application and its users, e.g. a custom web based Office Management System may recognize its users on the basis of IP Addresses and denies to establish a connection with all other systems (applications) that do not lie in the range of valid IPs for that application.
Tester must ensure that all the inter-network and intra-network access to the application is from trusted applications, machines (IPs) and users. In order to verify that an open access point is secure enough, tester must try to access it from different machines having both trusted and untrusted IP addresses. Different sort of real-time transactions should be tried in a bulk to have a good confidence of application's performance. By doing so, the capacity of access points of the application will also be observed clearly [16].

7.10 Broken Authentication and Session Management Attack

Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users identities.
Account credentials and sessions tokens are often not properly protected

- A third party can access to anyone's account
- Attacker compromise password, keys or authentication token

Risks

- Undetermined authorization and accountability controls
- Cause privacy violation
- Identity Theft

Method of attack: Use weaknesses in authentication mechanism

- Logout
- Password Management
- Timeout
- Remember me
- Secret question and account update
- Session Hijacking

All known web servers, application servers, and web application environments are susceptible to broken authentication and session management issues. Session management assets like user credentials and session IDs are not properly protected. You may be vulnerable if:

1. User authentication credentials aren't protected when stored using hashing or encryption.
2. Credentials can be guessed or overwritten through weak account management functions (e.g., account creation, change password, recover password, weak session IDs).
3. Session IDs are exposed in the URL (e.g., URL rewriting).
4. Session IDs are vulnerable to session fixation attacks.
5. Session IDs don't timeout, or user sessions or authentication tokens, particularly single sign-on (SSO) tokens, aren't properly invalidated during logout.
6. Session IDs aren't rotated after successful login.
7. Passwords, session IDs, and other credentials are sent over unencrypted connections.

For example, Application's timeouts aren't set properly. User uses a public computer to access site. Instead of selecting "logout" the user simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated .

Prevention/Recommendation for 'Broken Authentication and Session Management

The primary recommendation for an organization is to make available to developers:
1. A single set of strong authentication and session management controls. Such controls should strive to: Meet all the authentication and session management requirements defined in OWASP's Application Security Verification Standard (ASVS) areas V2 (Authentication) and V3 (Session Management).Have a simple interface for developers.

2. Strong efforts should also be made to avoid XSS flaws which can be used to steal session IDs.
Careful and proper use of the authentication and session management mechanisms should significantly reduce the problem in this area. Defining and documenting your site's policy for securely managing users credentials is a good first step. Ensuring that your implementation consistently enforces this policy is to have a secure and robust authentication and session management mechanism. Some critical areas include:

- Password Strength – passwords should have restrictions that require a minimum size and complexity for the password. Complexity typically requires the use of minimum combinations of alphabetic, numeric, and/or non-alphanumeric characters in a user's password (e.g., at least one of each). Users should be required to change their password periodically. Users should be prevented from reusing previous passwords.

ISSN 2348 – 7968

- Password Use – Users should be restricted to a defined number of login attempts per unit of time and repeated failed login attempts should be logged. Passwords provided during failed login attempts should not be recorded, as this may expose a user's password to whoever can gain access to this log. The system should not indicate whether it was the username or password that was wrong if a login attempt fails. Users should be informed of the date/time of their last successful login and the number of failed access attempts to their account since that time.
- Password Change Controls: A single password change mechanism should be used wherever users are allowed to change a password, regardless of the situation. Users should always be required to provide both their old and new password when changing their password (like all account information). If forgotten passwords are emailed to users, the system should require the user to re-authenticate whenever the user is changing their e-mail address, otherwise an attacker who temporarily has access to their session (e.g., by walking up to their computer while they are logged in) can simply change their e-mail address and request a 'forgotten' password be mailed to them.
- Password Storage – All passwords must be stored in either hashed or encrypted form to protect them from exposure, regardless of where they are stored. Hashed form is preferred since it is not reversible. Encryption should be used when the plaintext password is needed, such as when using the password to login to another system. Passwords should never be hardcoded in any source code. Decryption keys must be strongly protected to ensure that they cannot be grabbed and used to decrypt the password file.
- Protecting Credentials in Transit – The only effective technique is to encrypt the entire login transaction using something like SSL. Simple transformations of the password such as hashing it on the client prior to transmission provide little protection as the hashed version can simply be intercepted and retransmitted even though the actual plaintext password might not be known.
- Session ID Protection – Ideally, a user's entire session should be protected via SSL. If this is done, then the session ID (e.g., session cookie) cannot be grabbed off the network, which is the biggest risk of exposure for a session ID. If SSL is not viable for performance or other reasons then session IDs themselves must be protected in other ways. First, they should never be included in the URL as they can be cached by the browser, sent in the referrer header, or accidentally forwarded to a 'friend'. Session IDs should be long, complicated, random numbers that cannot be easily guessed. Session IDs can also be changed frequently during a session to reduce how long a session ID is valid. Session IDs must be changed when switching to SSL, authenticating, or other major transitions. Session IDs chosen by a user should never be accepted.
- Account Lists – Systems should be designed to avoid allowing users to gain access to a list of the account names on the site. If lists of users must be presented, it is recommended that some form of pseudonym (screen name) that maps to the actual account be listed instead. That way, the pseudonym cannot be used during a login attempt or some other hack that goes after a user's account.
- Browser Caching – Authentication and session data should never be submitted as part of a GET. Instead POST should be used. Authentication pages should be marked with all varieties of the no cache tag to prevent someone from using the back button in a user's browser to backup to the login page and resubmit the previously typed in credentials. Many browsers now support the auto complete=false flag to prevent storing of credentials in auto complete caches.
- Trust Relationships – Your site architecture should avoid implicit trust between components whenever possible. Each component should authenticate itself to any other component it is interacting with unless there is a strong reason not to (such as performance or lack of a usable mechanism). If trust relationships are required, strong procedural and architecture mechanisms should be in place to ensure that such trust cannot be abused as the site architecture evolves over time.
- Token should be independent of the browser.
- Session tokens should be expired on the server, and destroyed when a browser is closed.
- Avoid writing your own routines to authenticate, end sessions, tokens, etc. Use a well-tested tool.
- Test for when a "back browser" action is taken or use of an expiring timestamp.
- Do not enumerate account lists.
- If the web server is within a shared environment (multiple services on the same server), do not allow sharing of directories. Verify that permissions are set up correctly.

ISSN 2348 – 7968

- Remove all demo code, and guard against path traversal attacks.
- Verify that the server configuration is proper for your environment. Do not accept the server defaults without analysis. Defaults are usually bad since they are often too open.

3. User account management:
   - Use best practices for user account management, i.e. annual account review using active personnel list or files. If a user has not logged in for a specified period of time, disable/deactivate.
   - Do not use generic user accounts.
   - You MUST not use generic administrator accounts.
   - Use different administrator accounts and passwords for each server.

4. Server security management:
   - Be careful about privileges and administrative interfaces. Do not use elevated privileges.
   - Limit access to administrators and only use "secure shell" or console privileges.
   - Authenticate for all levels.
   - Use audit trails and logging. Preferably log to a log server[2], [9].

8. Open Source/Free Security Testing Tools[17]:

Table 1 : Open Source Tools

Product	*Vendor*	*URL*
FxCop	Microsoft	https://www.owasp.org/index.php/FxCop
FindBugs	The University of Maryland	http://findbugs.sourceforge.net/
FlawFinder	GPL	http://www.dwheeler.com/flawfinder/
Ramp Ascend	GPL	http://www.deque.com

9. Commercial Security Testing Tools[17]:

TABLE 2: Commercial Security Testing Tools

Product	**Vendor**	**URL**
Armorize CodeSecure	Armorize Technologies	http://www.armorize.com/index.php?link_id=codesecure
GrammaTech	GrammaTech	http://www.grammatech.com/
Appscan	IBM	http://www03.ibm.com/software/products/en/appscan-source
Veracode	VERACODE	http://www.veracode.com

CONCLUSIONS

This paper elucidates about Security testing for Web Applications, its vulnerabilities and types and techniques to deal with these vulnerabilities. Security testing is an important and integral part of the software developmental process. We need to conduct this test to find security loopholes and later close them with appropriate security measures and techniques. It is one of the most important tests that we should conduct before introducing it to the commercial domain. Businesses should try to incorporate safety measures right inside the applications they use and not around it as most security scanners have limited capabilities. Secure applications ensure system safety and security as it can impede attacks by hackers. So, we should make the system safe and secure with sturdy applications.

References

[1] Gu Tian-yang, Shi Yin-sheng, and Fang You-yuan ,"Research On Software Security Testing", World Academy of Science, Engineering and Technology 69 2010

[2] The approaches-tools-techniques-for-security-testing homepage on 3pillarglobal. [Online]. Available: http://www.3pillarglobal.com/insights/approaches-tools-techniques-for-security-testing

[3] The types-of-software-testing-complete-list homepage on testingexcellence. [Online]. Available: http://www.testingexcellence.com/types-of-software-testing-complete-list/

[4] The security_testing homepage on tutorialspoint. [Online]. Available: http://www.tutorialspoint.com/software_testing_dictionary/security_testing.htm

[5] The ecurity-services homepage on world4engineers. [Online]. Available: http://be.world4engineers.com/security-services/

ISSN 2348 – 7968

[6] Kevin Roebuck ,"SIEM - Security Information and Event Managers: High-impact Strategies" Emereo Publishing, 24-Oct-2012, page no. 186
[7] Jon Oltsik, Jane Wright, Jennifer Gahm, "Web Application Security Testing Tools and Services," The Enterprise Strategy Group, Inc April 2013.
[8] The security-testing-of-web-applications homepage on softwaretestinghelp. [Online]. Available: http://www.softwaretestinghelp.com/security-testing-of-web-applications/
[9] The broken-authentication-and-session-management homepage on triadsquare. [Online]. Available: http://www.triadsquare.com/broken-authentication-and-session-management
[10] The Top_10_2013-A4-Insecure_Direct_Object_References homepage on owasp. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References
[11]The Testing_for_Insecure_Direct_Object_References_(OTG-AUTHZ-004) homepage on owasp. [Online]. Available: https://www.owasp.org/index.php/Testing_for_Insecure_Direct_Object_References_(OTG-AUTHZ-004)
[12] The Cross-Site_Request_Forgery_(CSRF) homepage on owasp. [Online]. Available: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
[13] The Testing_for_CSRF_(OTG-SESS-005) homepage on owasp. [Online]. Available: https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005)
[14] The Unvalidated_Redirects_and_Forwards_Cheat_Sheet homepage on owasp. [Online]. Available: https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet
[15] The Top_10_2013-A10-Unvalidated_Redirects_and_Forwards homepage on owasp. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-A10-Unvalidated_Redirects_and_Forwards
[16] The how-to-test-application-security-web-and-desktop-application-security-testing-techniques homepage on softwaretestinghelp. [Online]. Available: http://www.softwaretestinghelp.com/how-to-test-application-security-web-and-desktop-application-security-testing-techniques/
[17] The security_testing homepage on tutorialspoint. [Online]. Available: http://www.tutorialspoint.com/software_testing_dictionary/security_testing.htm