Web application security vulnerabilities

**Abstract**

The world is exceedingly dependent on the Internet, Web applications are one of the most prevalent platforms for information and services Exchange over Internet today. Nowadays, web security is greatest challenge in the corporate world. As almost all organizations has using the web application service to share or store sensitive information.

So Web applications are inclined to security attacks and Number of security vulnerabilities in web application has grown with the tremendous growth of web application in last two decades. so web applications become a well known and important target for security attacks by attackers. So it is very vital to secure a web application from attacks by unauthorized users. A lot of the issues that occur over a web application is basically due to the improper input provided by the client.

This paper reviews the area of web application security Vulnerabilities, with the point of systematizing the current strategies into a major picture that advances future research. in paper firstly we discuss the working of web application and the basic Architecture of web application. This paper also discusses the different web application vulnerabilities. and The main web application vulnerabilities like sql injection, cross site scripting(XSS) ,malicious file upload ,broken authentication, improper session management, cross site request forgery(CSRF),server side request forgery(SSRF)and their remediation for minimizing these type of web vulnerabilities.

**Introduction**

World Wide Web has advanced from a framework that delivers static pages to a stage that supports distributed applications, known as web applications and become one of the foremost predominant technologies for information and service delivery over Web.

Web application advancements give a promising system of coordinating numerous useful segments over the web and therefore empower people and associations to cooperate each other utilizing application program interface along enormous topographical separations. Billions of people everywhere throughout the world use web application advancements to exchange data, perform money related exchanges, and have fun and communicate and to socialize themselves [3, 5, 6].Web application grew tremendously in the last few decades and it has brought great benefits to the people, however, these benefits are associated with some challenges and one of the most important challenges is that of security. Security in web application refers to the threat which occurs due to flaws in software design, coding, testing and implementation. Web application services are more prone to cyber attacks due to their public access. And web applications are increasingly used to deliver security critical services; they become a valuable target for security attacks. Many web applications interact with back-end database systems, which may store sensitive information (e.g., financial, health), the compromise of web applications would result in breaching an enormous amount of information, leading to severe economical losses, ethical and legal consequences [7,8]. The Web platform is a complex ecosystem composed of large number of components and technologies, including HTTP protocol, web server and server-side application development technologies (e.g., CGI, PHP, ASP), web browser and client-side technologies (e.g., JavaScript, Flash). Web application built and hosted upon such a complex framework faces characteristic challenges postured by the highlights of those components and innovations and the irregularities among them. Current widely-used web application advancement and testing systems, on the other hand, offer constrained security back. In this way secure web application advancement is an error-prone prepare and requires considerable endeavors, which may be unreasonable beneath time-to-market weight and for individuals with insufficient security skills or awareness. As a result, a high rate of web applications sent on the Web is uncovered to security vulnerabilities. According to a report by the Internet Application Security Consortium, around 49% of the internet applications being looked into contain vulnerabilities of tall hazard level and more than 13% of the websites can be compromised totally naturally [1]. A later report [2] uncovers that over 80% of the websites on the Web have had at least one serious Vulnerability.

Vulnerability refers to a weakness in system’s security requirement, design, coding or operation that could accidently occur or intentionally violated and result in security failure. In last few years, number of reported web application security vulnerabilities has increased. Some commonly reported web application vulnerabilities include SQL injection, cross site scripting, command line injection, cross site request forgery and malicious file execution [3, 4].

**Understanding How A Web Application Works**

Web application could be a distributed application that's executed over the web platform. It is an fundamentally portion of today’s Web biological system that enables dynamic data and benefit conveyance. As shown in Fig. 1, a web application may comprise of code on both the server side and the client side. The server-side code will generate dynamic HTML pages either through execution (e.g., Java servlet, CGI) or elucidation (e.g., PHP,JSP).

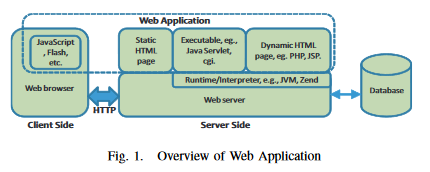


Fig. 1. Overview Of Web Application

During the execution of the server-side code, the internet application may interact with nearby record framework or back-end database for storing and retrieving information. The client-side code(e.g., in JavaScript) are implanted within the HTML pages, which is executed inside the browser. It can communicate with the server-side code (i.e., AJAX) and powerfully overhauls the HTML pages. In what takes after, we portray three one of a kind perspectives of the internet application advancement, which separate web applications from conventional applications.

2.**1 Web Application Architecture**

Web application architecture is often structured as a three-tiered application The architecture of web applications consists of web browser, web server, web application and database server. Tier 1 architecture consists of web browser and web server, Tier 2 for web application and Tier 3 for database.

•Tier 1 – web browser A web browser is also known as web client, functions as the user interface to web server to get input from web application or database server. Web server receives input and interacts with client through web browser by using Hyper Text Transfer Protocol (HTTP) or via secure protocol HTTPS. There are many type of web servers where Apache and Internet Information Server (IIS) are the most popular web server in the world.

•Tier 2 – web application architecture Web application consists of a collection of scripts such as Javascript, VBscript, which reside on a web server and interact with databases or other sources of dynamic content. The common example, the data input using a web browser is processed and stored into database. Java Server Pages (JSP), PHP, Active Server Pages (ASP), Perl and Common Gateway Interface (CGI) are among the technology used to build web based application. Using the infrastructure of the Internet, web applications allow service providers and clients to share and manipulate information in a platform-independent manner. Normally web application server is attached on top of web server and works as interface from web client and database server. Web server will manage the page requested from web client by sending to application server and application server constructs code dynamically and passed back to web server. The flow of data among tiers gives rise to the input validation problem for the web application server; it must check and/or modify incoming input before processing them further or incorporating them into output that it passes to other tiers to execute. Failure to check or sanitize input appropriately can compromise the web application’s security [9].

•Tier 3 – database architecture Stores and manages all the processed users input data. The database tier is responsible for the access of authenticated users and the rejection of malicious users from the database.

**Web application vulnerabilities**

Because web applications are open to the world, they are more vulnerable to attacks and prone to a great variety of vulnerabilities. In this section, we describe some of the most common and well-known web application vulnerabilities based on OWASP Top Ten lists 2010 [10]. OWASP stands for Open Web Applications Security Project, and is an open-source collaboration of web based security tools, technologies and methodologies from industry leaders, educational organizations and individuals from around the world. The OWASP Top Ten is a valuable document for developers and testers because its focus on web applications. The OWASP Top 10 2017 has listed the ten most critical web application security vulnerabilities as shown in Table 1. The OWASP Top 10 2017 refers to the top 10 web attacks as seen over the year by security experts, and community contributors to the project.

|  |
| --- |
| OWASP TOP 10 |
| A1:2017-injection |
| A2:2017-Broken Authentication |
| A3:2017-Sensitive Data Exposure |
| A4:2017-Xml External Entities(XXE) |
| A5:2017-Broken Access control |
| A6:2017-Security misconfiguration |
| A7:2017-Cross-site Scripting |
| A8:2017-Insecure Deserialization |
| A9:2017-Using components with known Vulnerabilities |
| A10:2017-Insufficient logging and Monitoring |

**Table1.** OWASP Top Ten Vulnerability 2017 [10]

**Injection:**

Injection flaws, such as SQL, NoSQL, OS, and LDAP 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. Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter[12].

Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws. Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access[12]. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.

An application is vulnerable to attack when:

1. User-supplied data is not validated, filtered, or sanitized by the application.
2. Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
3. Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records…
4. Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.

The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections, closely followed by thorough automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs. Organizations can include static source (SAST) and dynamic application test (DAST) tools into the CI/CD pipeline to identify newly introduced injection flaws prior to production deployment.

The most common injection types are:

* Expression Language (EL)
* Object Graph Navigation Library (OGNL)
* LDAP
* Object Relational Mapping (ORM)
* OS command
* NoSQL
* SQL

SQL is possibly the most common and widespread injection type. Instead of a Python, Pearl, or other script type, malicious SQL commands are embedded into the content of the user input [11].

**Sql injection:**

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure, or perform a denial-of-service attack.

These common mistakes must be avoided with good programming habits. The programmer must apply the following methods for the protection from SQLIA's:-

* Avoid building dynamic SQL statement from user input.
* Length of input string must be limited.
* Escape query delimiter, SQL keyword, character data string delimiter and single line comment in user input.
* Use different Database account for different level of privileges.
* Error messages must be customized to hide the details of injectable parameters.
* Use parameterized queries for Database access.
* Use stored procedures to avoid direct access of Database.
* Void building SQL statements from cookie and HTTP variables.

Impact of SQL Injection As we already mentioned SQL injection attack is accomplished by providing data (inclusion of SQL queries) from an external source which is further used to dynamically construct a SQL query. The impact and consequences of SQL injection attacks can be classified as follows:

1. **Confidentiality:** Confidentiality loss is a significant issue with SQL Injection attacks since SQL databases generally hold sensitive and critical information which could be viewed by unauthorized users as a consequence of successful SQL injection attack.
2. **Integrity:** Successful SQL injection attack allows external source to make unauthorized modifications such as changing or even deleting destination database data.
3. **Authentication:** Poorly formulated SQL queries do not correctly validate User identity that allow unauthenticated users act as an authenticated user without out knowledge of the password or even user name.
4. **Authorization:** The successful exploitation of SQL injection vulnerability enables an attacker to alter permission and obtain higher privileges ,data in the database. It is very difficult to identify SQL injecting before effect in reality.In most number of scenarios, The unlawful transaction is carried out by the intruder using legitimate customer identification or the intrinsic characteristics of the database application such as malicious modification of existing SQL Queries of web application that are accessing critical sections of the affected databases.

**Broken Authentication and Session Management:**

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities temporarily or permanently [10].

Broken Authentication is a kind of web vulnerability which occurs due to the misconfiguration of session management. After an authentication process completed, a session will be created which will be activated for data communication between the server and a particular user. Fig. 1 represents the problem of Broken Authentication by exploiting session mismanagement problem. If any intruder can get access in the active session of any specific user bypassing the authentication process, the scenario is treated as broken Exploiting Authentication problem of the given application. Fig. 1 represents the overall process of user authentication and session management.

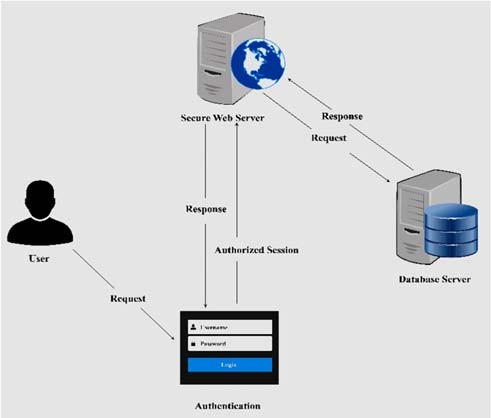


Figure 1. Authentication and Session Management Process

A session request is raised by a user of a web application through the login page where the user credential has been provided. Once the given request has been sent from the client side to server side, the server initiates a query to the database for checking whether the user provided credential is matched [13].

A session request is raised by a user of a web application through the login page where the user credential has been provided. Once the given request has been sent from the client side to server side, the server initiates a query to the database for checking whether the user provided credential is matched with the record of the database or not. As soon as the validation process is successful, a session with a specific ID will be allocated for the user to communicate the application. A user then can access the system with a given privileges provided by the administrator of the system for getting different services.

There is some exploitation techniques used to exploit Broken Authentication & Session Management. The Types are given below:

1. *Session Misconfiguration Attack:*

Session duration isone of the major facts in maintaining a secure authentication process of the web applications. As soon as the user credential is validated from a system, it assigns a session for the particular user with a session ID for a limited period of time. In case the developer of the web application sets the session duration parameter with a large value, the session will remain active for that specific period if the user not logged off their account as directed by the designer of the application. Therefore, that session can be reestablished to re-using by an intruder which leads to Broken Authentication. Session misconfiguration is one of the most critical areas for Broken Authentication and Session Management vulnerability.

1. *Using Cracking/ Guessing Weak Password Exploitation*

Due to lack of awareness about password management, some non-technical users keep their password in a generalize form like admin, password, my password, password123, admin1997 etc. and also in some cases, user remains the default password for their access into the system which will be easy to guess for an attacker to get access in the system. It is an automated process of cracking/ guessing user’s weak passwords. Attacker gives user login link in Hydra in which it checks predefined dataset for trying to find username and password

1. *Exploiting Authentication problem*

Web applications authentication systems are handled by using conditional quires to check username and password against one user for authentication. If these conditional queries get infected or not properly handled, it could easily compromised by an intruder to get access into the system without proper authentication.

1. *Decoding Inadequate Encryption*

In some web applications privacy measures are not properly handled by the developers. Therefore, an attacker can steal the session ID against one user by exploiting the security flaws of disclosing the session ID in the URL of the system,

*e.g. http://www.demosite.com/transactions/saleitems?sessioni d=7892384838&dest=demo user*

The example shows the general transaction’s session id of demo user has been disclosed publicly in the URL. As such, it is not very critical for an attacker to steal some other user’s session id just only changing the session ID value into the URL. The attack process is feasible for the inadequate encryption in the value of session ID. After changing the value in session ID, it will look like as below: *http://www.demosite.com/transactions/saleitems?sessionid= 7892384839&dest=attacker name.*

1. *Other Vulnerabilities:*

Web application vulnerabilities allow users to disclose users/ systems sensitive information. It also causes major harm to other circumstance e.g. it allows users to execute malicious quires in the system if the system is vulnerable to XSS vulnerability, it also allows attackers to post malicious links for phishing to steal session of the victim, etc. Forgotten password functionally, relying on IP address for session, emailing user credentials, not authenticating a user before changing password, and not having adequate timeouts for inactive session are also reason for Broken Authentication.

**How to Prevent**

* Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute force, and stolen credential re-use attacks.
* Do not ship or deploy with any default credentials, particularly for admin users.
* Implement weak-password checks, such as testing new or changed passwords against a list of the top 10000 worst passwords.
* Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets or other modern, and evidence based password policies [14].
* Ensure registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
* Limit or increasingly delay failed login attempts. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
* Use a server-side, secure, built-in session manager that generates a new random session ID with high entropy after login. Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute timeouts.

**Sensitive Data Exposure:**

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser [10].

The potential impact is always considered high. What the data consists of varies and so does the impact. The danger lies in the data being exposed, and the potential impact reflects the data’s sensitivity. For example, if credit card data is stolen, the attacker can empty the victim’s bank account. If passwords are exposed, the attacker can abuse these credentials. If certificates are stolen, the attacker can pretend to be the target. It all depends on what kind of data is at risk of being exposed.

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information and business secrets require extra protection, particularly if that data falls under privacy laws, e.g. EU's General Data Protection Regulation (GDPR), or regulations, e.g. financial data protection such as PCI Data Security Standard (PCI DSS) [12]. For all such data:

* Is any data transmitted in clear text? This concerns protocols such as HTTP, SMTP, and FTP. External internet traffic is especially dangerous. Verify all internal traffic e.g. between load balancers, web servers, or back-end systems.
* Is sensitive data stored in clear text, including backups?
* Are any old or weak cryptographic algorithms used either by default or in older code?
* Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?
* Is encryption not enforced, e.g. are any user agent (browser) security directives or headers missing?
* Does the user agent (e.g. app, mail client) not verify if the received server certificate is valid? [12].

**To prevent the sensitive data exposure:**

* Classify data processed, stored, or transmitted by an application. Identify which data is sensitive according to privacy laws, regulatory requirements, or business needs.
* Apply controls as per the classification.
* Don’t store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
* Make sure to encrypt all sensitive data at rest.
* Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
* Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, cipher prioritization by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security (HSTS).
* Disable caching for responses that contain sensitive data.
* Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt, or PBKDF2.
* Verify independently the effectiveness of configuration and settings[12]..

**Xml External Entities (XXE)**

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks[10].

**Broken Access control**

Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users’ data, change access rights, etc[10].

**Security misconfiguration**

Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely fashion[10].

**Cross-site Scripting**

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. 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[10].

**Insecure Deserialization**

Insecure Deserialization often leads to remote code execution. Even if Deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks[10].

**Using components with known Vulnerabilities**

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts [10].

**Insufficient logging and Monitoring**

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring [10].

References

[1]WebApplicationSecurityStatistics,“http://projects.webappsec.org/w/page/13246989/WebApplicationSecurityStatistics.”

[2]White Hat Security, “White Hat website security statistic report 2010.”

[3] Zhang, D., Liu, D., Csallner, C., Kung, D., & Lei, Y. (2014). A distributed framework for demand-driven software vulnerability detection. Journal of Systems and Software, 87, 60-73.

[4] Huang, C. C., Lin, F. Y., Lin, F. Y. S., & Sun, Y. S. (2013). A novel approach to evaluate software vulnerability prioritization. Journal of Systems and Software,86(11), 2822-2840.

[5] Kaur, N., & Kaur, P. (2014). Input Validation Vulnerabilities in Web Applications. Journal of Software Engineering, 8(3), 116-126.

[6] Austin, A., Holmgreen, C., & Williams, L. (2013). A comparison of the efficiency and effectiveness of vulnerability discovery techniques. Information and Software Technology, 55(7), 1279- 1288.

[7] Shar, L. K., & Tan, H. B. K. (2012). Automated removal of cross site scripting vulnerabilities in web applications. Information and Software Technology, 54(5), 467-478.

[8] Avancini, A., & Ceccato, M. (2013). Comparison and integration of genetic algorithms and dynamic symbolic execution for security testing of cross-site scripting vulnerabilities. Information and Software Technology, 55(12), 2209-2222.

[9] Shklar, L., Rosen, R.: Web Application Architecture: Principles, Protocols and Practices, 2nd edn. John Wiley & Sons (2009).

[10] The Open Web Application Security Project: The Ten Most Critical Web Application Security Vulnerabilities, “https://www.owasp.org/index.php/Main\_Page:OWASP\_Top\_Ten\_Project”

[11] What is injection vunlerability“https://resources.infosecinstitute.com/2017-owasp-a1-update injection/#gref”.

[12] The Open Web Application Security Project, “https://www.owasp.org/images/7/72/OWASP\_Top\_10-2017\_%28en%29.pdf.pdf”.

[13] M HASSAN et al: Broken authentication and session management vulnerability: a case study (2006), DOI 10.5013/IJSSST

# [14] NIST Special Publication 800-63BDigital Identity Guidelines Authentication and Lifecycle Management

“https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret”.

In general, there are three kinds of security vulnerabilities among web applications at completely different levels: (1) input validation vulnerability at the single request level, (2) session management vulnerability at the session level, and (3) application logic vulnerability at the extent of the whole application. In what follows, description of the above three kinds of vulnerabilities are presentedand the common attacks that exploit these vulnerabilities.

* 1. input validation vulnerability

A common security observes is input data validation, since user input data can't be trusty. Data validation is that the method of guaranteeing that a program operates on clean, correct and helpful input data. Once inputs don't seem to be sufficiently or properly valid, attackers are ready to craft distorted inputs, which might alter program executions and gain unauthorized access to resources. Input validation vulnerability may be a durable drawback in software system security. Incorrect or depleted input validation may invite a range of attacks, like buffer overflow attacks and code injection attacks. Web applications might contain a large vary of input validation vulnerabilities. Since the whole web request, as well as request headers and payload data, is beneath the entire management of users, a web application must make sure that user inputs are processed and utilized in a very secure manner throughout the execution.**SQLI (SQL Injection)**:-SQL Injection is acode injection technique where attacker injectsmalicious code in to strings that are later passed to SQL server for execution.A web application is at risk of SQL injection attacks once malicious content will flow into SQL queries while not being absolutely sanitized, that permits the offender to trigger malicious SQL operations by injecting SQL keywords or operators. For example, the offender will append a separate SQL query to the present query, causing the application to drop the complete table or manipulate the comebackresult. Malicious SQL statements may be introduced into a vulnerable application victimization many various input mechanisms [1] as well as user inputs, cookies and server variables **Cross-site Scripting (XSS):**vulnerabilities arise from associate application‟s failure to properly validate user input before it's came to a user. Mistreatment this vulnerability, associate offender will force a consumer, like a user application, to execute attacker-supplied code, like JavaScript, within the context of a trusty computing machine [5]. As a result, the attacker‟s code is granted access to security-critical data that was issued by (or is associated with) the trusty website.

* 1. session management vulnerability

Session management is essential for a web application to keep track of user inputs and maintain application states. Within the OWASP top-ten security risks [3], three are related to session management vulnerabilities: (1) Broken Authentication and Session Management, (2) Cross-Site Request Forgery and (3) Insufficient Transport Layer Protection.In web application development, session management is accomplished through the collaboration between the client and the server. A common approach is that the server sends the client a unique identifier (i.e., a session ID) upon successful user authentication, through which the server recognizes the client on subsequent requests and indexes his session variables stored at the server side. Since session ID is the only proof of the client‟s identity, its confidentiality, integrity and authenticity need to be ensured to avoid session hijacking.First, the session ID should be random for each client‟s visit and expire after a short period of inactivity. Weak session identifier generation allows attackers to hijack the victim‟s web sessions by predicting his session ID. Second, transmission of the session ID should always be protected by a secure transport layer protocol (i.e., over SSL). Otherwise, attackers are able to sniff the session ID and hijack the session. Third, the client needs to make sure that his session ID is provided by the server and is unique. Adopting a session ID from an external source opens up a vulnerability to session fixation, where attackers can set the session ID to a value that is known to them.Securing the session ID alone is not sufficient for secure session management. Session hijacking can also be achieved through malicious web requests that are associated with a valid session ID. Cross-site request forgery (CSRF) is a popular attack of this type, where attackers trickthe victim into sending crafted web requests on their behalf. The vulnerable web application cannot differentiate if the incoming web requests are malicious, since they are associated with valid session information. For example, attackers may forge a web request that instructs a vulnerable banking website to transfer the victim‟s money to his account. Login CSRF [6], on the other hand, tricks the victim into logging in to a target website using the attacker‟s credential through a forged request. This attack allows the attacker to harvest the information about the victim‟s activities under the attacker‟s account

* 1. application logic vulnerability

The decentralized structure of web applications poses significant challenges to the implementation of business logic. First, since web application modules can be accessed directly through their URLs, interface hiding mechanism has been commonly used as a measure for access control in web applications. However, this mechanism alone, which follows then principle of “security by obscurity”, is not sufficient to enforce the control flow of a web application. Application logic vulnerabilities are highly dependent on the intended functionality of a web application. For example, a vulnerable e-commerce website may have a specific logic vulnerability that allows attackers to apply the same coupon multiple times to reduce prices. Despite the heterogeneous application functionalities, there are several types of logic flaws that correspond to common business logic patterns in many applications. One common type is access control vulnerability, which allows attackers to access unauthorized sensitive information or operations. Another type is workflow violation, which allows attackers to violate the intended steps within business workflows. For example, a vulnerable e-commerce website may allow attackers to bypass the tax calculation step during the checkout procedure.Theclass of attacks that target application logic vulnerabilities are generally referred to as logic attacks or state violation attacks Depending on how attacks are launched, they can be given several other terms. Forceful browsing [7] is one attack vector, where attackers directly point to hidden but predictable web links to access sensitive information. Parameter tampering [8] is launched by manipulating certain values in web requests to exploit application logic