Input Validation Vulnerabilities in Web Applications

Navdeep Kaur and Parminder Kaur
Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar-143005, India

Corresponding Author: Parminder Kaur, Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar, 143005, India

ABSTRACT
With the growth of internet, web applications have become very popular and used in every environment like medical, financial and military. But in the race to develop these online services, web applications have been developed and deployed with minimal attention given to security risks which leads to vulnerabilities in web application. Developers are mandated to deliver functionality on time and on budget but not to develop secure web applications, resulting in development of vulnerable web applications. Removing vulnerabilities after development wastes cost as well as time. So, why not Security is implemented throughout software development lifecycle it will save time and cost. Thousands of vulnerabilities are there in existing web application but this study focused on input validation vulnerabilities i.e., SQL injection and Cross Site Scripting (XSS), as they are more prevalent and have high risk. A brief introduction of web application vulnerabilities is discussed in this study. How cross site scripting and SQL injection vulnerabilities are addressed throughout the software development lifecycle is discussed. Different activities to be performed to mitigate them are suggested.

Key words: Software security, SQL injection, cross site scripting (XSS), vulnerability, input validation vulnerabilities

INTRODUCTION
Now a days, web applications have become very popular. They are used in every environment such as medical, financial, military systems and so on. In the race to develop these online services, many well-known software vendors don’t yet understand that security is not an add-on feature. They continue to design and create products at alarming rates, with little attention paid to security. They start to worry about security only after their product has been publicly broken by someone. Then they rush out a patch instead of coming to the realization that designing security in from the start may be a better idea. Web applications have been developed and deployed with minimal attention given to security risks, resulting in surprising number of corporate sites that are vulnerable to hacker’s attacks. Web based attacks are rising consistently and there are number of reasons that make applications so vulnerable like poor design, configuration mistakes, insecure coding techniques, complexity, unchecked user input, password management flaws and so on. According to CERT/CC (Mead and McGraw, 2005), more than 90% of vulnerabilities leak out during development. They are result of ignoring known vulnerabilities found in software.

There are three major approaches to develop secure web application. They are discussed below:

• Penetrate and patch means applying patches to fix vulnerable applications after the development. It is a common approach to securing applications but, the cost of finding and
fixing a bug after a software product has been released can be 100 times more expensive than solving the problem during the development phases (Boehm, 1987). Furthermore, patches can contain yet vulnerabilities.

- Securing the operational environment relies on devices external to a software system such as intrusion detection systems and firewalls. Although these protection mechanisms can provide some security, do little to aid application resist attack against software implementation or design. Also, they can be used only when the development process is complete and software is operational.

- Secure software engineering means to implement well-structured processes, from requirements analysis through design and implementation, with security in mind.

Nevertheless, the dominating idea, addressing security during development, offers reduction in future probable expenditures as well as more in-depth defensive layers (Howard and LeBlanc, 2003). Today, thousands of vulnerabilities are present in web applications. Vulnerability classification is done by different organizations like OWASP (Williams and Wichers, 2013; Microsoft, 2003b) based on the risk rating of vulnerability according to exploitability, prevalence, detectability and impact. Among the classes of vulnerabilities exhibited by web applications, input validation vulnerabilities (XSS and SQL injection) remain among the most serious and prevalent threats to web application security. As such, much attention in the security research community has focused on removing or mitigating the effect of these vulnerabilities after deployment. But it is far better to mitigate during development as to bolt them at the end. So, this study will attempt to focus on input validation vulnerabilities i.e., SQL injection and Cross Site Scripting (XSS), how these can be addressed during different phases of a software development lifecycle starting from requirement analysis till testing.

RELATED WORK

There have been plenty of works made around software security. A number of major organizations and government departments have devoted resources themselves to develop strategies, policies and guidelines aimed at managing the risks from the open nature of web applications.

OWASP is Open Web Application Security Project (Williams and Wichers, 2013). An open and freely-accessible community has been established to coordinate worldwide efforts aimed at reducing the risks associated with web application software and to improve the security of web applications. This community works to create freely-available articles, methodologies, documentation, tools and technologies related to security.

MITRE Corporation developed a Common Weakness Enumeration (CWE) list (http://cwe.mitre.org/top25). CWE is a community developed dictionary of software weakness type. CWE is targeted to developers and security practitioners.

MICROSOFT has developed a security development lifecycle (Microsoft, 2010). SDL is a security assurance process that is focused on secure software development. SDL aims to reduce the number and severity of vulnerabilities in software. This process adds a series of security-focused activities and deliverables to each phase of microsoft’s software development process. Typical tasks included in the plan are identifying security risks, eliciting and defining security requirements, secure design and code reviews, use of static analysis tools, unit tests and fuzz testing and final security review.

Different researchers put their efforts in integrating security in software development lifecycle (Mead and McGraw, 2005) have developed several software security best practices,
knowledge and tools that developers and security engineers can apply throughout the software development life cycle. A high-level approach to iterative risk analysis should be deeply integrated throughout the software development life cycle.

Davis (2005) has discussed the concept of defect removal filters in software development lifecycle. Each defect removal activity can be thought of as a filter that removes some percentage of defects that can lead to vulnerabilities from the software product.

Cheng (2007) has discussed different security activities (Threat modeling, risk Analysis) which should be integrated in each phase of SDLC to develop a Vulnerability free software.

Daud (2010) has focused on security requirements at each phase of software life cycle. He said that iterative development of software is a key solution that provides us with a guide with the ease to recheck our security requirements, if they are unnoticed at any step of software life cycle. He designed a secure iterative lifecycle model for software and explains its different stages.

WHAT IS WEB APPLICATION VULNERABILITY?

Vulnerability is a hole or weakness in application which can be design flaw or implementation bug that allows an attacker to cause harm to stakeholders of an application.

Formally, vulnerability is defined as “The existence of a weakness, design, or implementation error that can lead to an unexpected, undesirable event compromising the security of the computer system, network, application, or protocol involved” (Enisa, 2014).

Vulnerabilities are caused because of poor design, configuration mistakes, inappropriate and insecure coding techniques, complexity of software, unchecked user input, weak password management. The impact of vulnerabilities is very harmful, if a hacker obtains the bank account details of an individual, he can misuse this information (like account number, account balance, loan amount, etc.) and can also alter the data to cause harm to the concerned individual. Vulnerability management in web applications is the cyclical practice of identifying, classifying, remediating and mitigating vulnerabilities. Today, thousands of vulnerabilities are present in web applications. Vulnerability classification is done by different organizations like OWASP and Microsoft (Williams and Wichers, 2013; Microsoft, 2003b) based on their risk rating according to exploitability, prevalence, detectability and impact. Classification according to OWASP, top 10 application security risks are given below:

- **SQL injection**: Injection flaws 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. Different injection flaws are SQL, OS and LDAP injection. They occur due to poor input validations
- **Cross site scripting**: XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. XSS flaw occur due to poor input validations or unsanitized input
- **Broken authentication and session management**: 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 user's identities. They occur by putting session ID in URL, application's timeouts aren't set properly
- **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, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

- **Security misconfiguration**: Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server and platform. Secure settings should be defined, implemented and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

- **Sensitive data exposure**: Many web applications do not properly protect sensitive data, such as credit cards, tax IDs and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

- **Missing function level access control**: Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

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

- **Using components with known vulnerabilities**: Components, such as libraries, frameworks and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.

- **Unvalidated redirects and forwards**: Web applications frequently redirect and forward users to other pages and websites and use untrusted 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.

Among the classes of vulnerabilities exhibited by web applications, input validation vulnerabilities (XSS and SQL injection) remain among the most serious and prevalent threats to web application security. This study is focused on SQL injection and XSS vulnerabilities. These are input validation vulnerabilities whose risk is very high. These vulnerabilities are discussed below.

**INPUT VALIDATION VULNERABILITIES**

Input validation vulnerability is characterized by the ability to decide at each step of the execution whether or not the program is in a safe state. Input validation refers to how your application filters, scrubs, or rejects input before additional processing. They can be removed by constrain input, reject known bad input, sanitize input, validate data for type, length and range. Input validation vulnerabilities are further classified as.
**SQL injection:** SQL Injection are attacks by which an attacker alters the structure of the original SQL query by injecting SQL code in the input fields of the web form in order to gain unauthorized access to the database. Figure 1 shows the scenario how SQL injection happens.

There are several methods which an attacker can use to gain access to the database of a web application like tautology based, statement injection, union query, inferences and so on. Examples related to these attacks are given below:

- **Tautology based:** Attack is very simple SQL injection attack done via text box. Tautology is a conditional statement which always evaluates to true. This type of method is used to attack the SQL queries which use the where clause. Here the intention of the attacker is to make the condition in the query as true. To do so, the attacker adds the most common tautology condition (e.g., 1 = 1) in the input field of the web form. The SQL query being executed at the back end is:
  - **Original query:** Select accountno, balance FROM accounts WHERE loginid = 'abc' and pwd = 'xyz'
  - **Injected query:** SELECT accountno, balance FROM accounts WHERE loginid = ' ' or 1=1-- ' and pwd = ' not required

  The single quote entered by the attacker closes the User name field and the double dashes comments out everything after the dashes. Therefore, the query retrieves all the records in the LOGIN table and returns them back to the attacker.

- **Statement injection:** It is used to inject a new SQL query to the original SQL query, through the front-end of an application using query delimiter " ; ";
  - **Original query:** SELECT * FROM orders WHERE orderID = 12345
  - **Injected query:** SELECT * FROM orders WHERE orderID = 12345; DROP TABLE orders
1. Attacker places bad code on a vulnerable web site
2. User navigates to the vulnerable web site and submits a cookie
3. The Web site allows the user to log on
4. The malicious code sends the user’s cookie to the attacker

Fig. 2: XSS attack (Microsoft, 2010)

- **Union query**: The original SQL query is concatenated to the injected query by using the SQL keyword UNION, to gather information from other tables related to the application
- **Original query**: Select salary from employee where, empid = '1234'
- **Injected query**: Select salary from employee where, empid = 'Union select * from employee'

It will return all the records from the employee table.

**Cross Site Scripting (XSS)**: XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. Figure 2 shows the scenario how XSS attack happens.

Cross site scripting can be performed by passing scripts in form of: Text box, query string, cookie, session variables. Purpose of attacker to perform cross site scripting is session hijacking, cookie poisoning, defacing website, redirect user to malicious site. Examples related to these attacks are given below:

- **Defacing website**: It is a very simple XSS attack done via Text Box. Defacing means changing the content of the website to hacker content. Hackers do this just to harm the website, when users view the website they always see the Hacker content (like a image on a whole web page) instead of actual content of website. Let’s take an example in which name is entered in the textbox and then display the user’s name on the page with some welcome message as shown in Fig. 3.

The code behind for the button click looks like:
protected void Button1_Click(object sender, EventArgs e)
{
    lblMessage.Text = "Hello " + TextBox1.Text;
}

Now everything will work fine under normal input scenarios but as soon as we try pausing some HTML and JavaScript in the textbox, the problem will come in front of us. Let's put an image (Fig. 4) on the page on top of everything else on page by using the following input in the textbox:

<img src="dilbert-03.jpg" style="position: absolute; top:0;left:0;display:block"/>
This is one of the web page defacement methods, done using XSS Attack. It is very to inject client side script in the pages using cross site scripting attacks. This Attack can be prevented by proper input validation, constraint or sanitize input:

- **Cookie stealing**: It is a very common and serious attack done by using XSS. In this, attacker can steal cookie of victim user and then use it for his benefit. Say that a user enters the following in a text box

  My name is <a href="javascript:document.location=\"http://attacker.example.com/\"+document.cookie\">Amy</a>

This text allows a cross-site scripting attack that would pass the user's cookie back to the attacker's site. A user who views this page and click on hyperlink "Amy" is inadvertently sending the document cookie (which may contain her authentication token) to attacker's site, presumably where hackers are waiting to pounce on it and steal her identity.

**MITIGATION OF SQL INJECTION AND XSS VULNERABILITIES IN SDLC**

As we know addressing security during development of software, offers reduction in future probable expenditures as well as more in-depth defensive layers. Different activities adopted by Microsoft's SDL (Microsoft, 2010) are taken care in proposed work to mitigate vulnerabilities (SQLI and XSS). The thought of iterative risk analysis is incorporated from the model developed by Mead and McGraw (2005). The idea of mitigating vulnerability in each respective phase is taken from the concept of defect removal filters as introduced by Davis (2005), so that flaw of one phase cannot pass to next phase.

Relying on the above said research work of secure web development, this study describes how SQL injection and XSS are addressed in each phase of software development life cycle starting from requirement analysis till testing.

The activities mentioned in Fig. 5 do not grantee absolute secure web applications but can help web applications to fortify against attacks. Figure 5 clearly shows all activities to be performed to make vulnerability (SQLI and XSS) free web application. The activities (like misuse cases, threat modeling, code reviews, penetration testing) are either performed manually or by using tools.

Risk analysis is an analytical assessment of selected programs, to consider how to identify and eliminate risk based on the knowledge about vulnerabilities, threats, impacts and probability. Risk analysis helps to define preventive measures and identify counter measures to successfully deal with vulnerabilities. Risk analysis is an iterative process performed in each phase of secure software development lifecycle (starting from Requirements phase till testing).

How risk analysis to be performed in order to mitigate SQLI and XSS vulnerabilities is briefly described below:

- In requirement phase, risk analysis is performed by using UMLsec, secure UML, misuse cases and abuse cases (Mead and McGraw, 2005)
- In design phase, risk analysis is performed by threat modeling (Microsoft, 2003a) as shown in Fig. 6. Threat modeling allows systematically identifying and rating the threats that are most likely to affect the system. By identifying and rating threats based on a solid understanding of the architecture and implementation of the application, we can address threats with
Fig. 5: Mitigation of SQL injection and XSS vulnerability in SDLC

Fig. 6: Threat modeling process (Microsoft, 2003a)

appropriate counter measures in a logical order, starting with the threats that present the greatest risk. The output of the threat modeling process includes documentation of the security aspects of the architecture of application and a list of rated threats. Threat modeling should not be a one-time only process. It should be an iterative process that starts during the early phases.
of the design of application and continues throughout the application life cycle. There are two reasons for this. First, it is impossible to identify all of the possible threats in a single pass. Second, because applications are rarely static and need to be enhanced and adapted to suit changing business requirements, the threat modeling process should be repeated as application evolves

- In implementation phase, same threat model is used by developers who write code can use it to mitigate risks
- In testing phase, threat model is used by testers who can write test cases to test if the application is vulnerable to the threats identified by the analysis

In this way, risk analysis is used to mitigate vulnerabilities (SQLi and XSS) in each phase of software development lifecycle.

CONCLUSIONS
Integrating security throughout all phases of software development life cycle can save time and cost as compare to removing flaws after the software has been developed. Relying on this idea of secure web development, how different researchers integrate security in to software development lifecycle is discussed. As vulnerabilities are the major cause which break security, so it is necessary to mitigate them during software development. As input validation vulnerabilities (SQL injection, XSS) are common and severe so, this study focused on the mitigation of SQL injection and XSS during each phase of software development lifecycle. It is suggested to detect mitigate and prevent SQL injection and XSS in each phase of lifecycle. So that a vulnerability (SQLi, XSS) free software can be developed. The activities suggested to mitigate SQLi and XSS, do not guarantee absolute secure web applications but can help web applications to fortify against attacks. It is very important that web developers keep themselves up to date with new risks which their applications could face and also understand their nature to ensure that the mitigations implemented are effective.

REFERENCES


