Mitigating Cross-Site Scripting Attacks with a Content Security Policy

Vutukuru Parvathi[1], Dr. G V Ramesh Babu[2]
M.C.A, M.Tech, Ph.D

Abstract- **A content security policy (CSP) can help Web application developers and server administrators better control website content and avoid vulnerabilities to cross site scripting (XSS). In experiments with a prototype website, the authors' CSP implementation successfully mitigated all XSS attack types in four popular browsers.**

Among the many attacks on Web applications, cross site scripting (XSS) is one of the most common. An XSS attack involves injecting malicious script into a trusted website that executes on a visitor's browser without the visitor's knowledge and thereby enables the attacker to access sensitive user data, such as session tokens and cookies stored on the browser.1 With this data, attackers can execute several malicious acts, including identity theft, key logging, phishing, user impersonation, and webcam activation.

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP is designed to be fully backward compatible; browsers that don't support it still work with servers that implement it, and vice-versa. Browsers that don't support CSP simply ignore it, functioning as usual, defaulting to the standard same-origin policy for web content. If the site doesn't offer the CSP header, browsers likewise use the standard same-origin policy.

Enabling CSP is as easy as configuring your web server to return the Content-Security-Policy HTTP header. (Prior to Firefox 23, the X-Content-Security-Policy header was used). See Using Content Security Policy for details on how to configure and enable CSP.

INTRODUCTION

A primary goal of CSP is to mitigate and report XSS attacks. XSS attacks exploit the browser's trust of the content received from the server. Malicious scripts are executed by the victim's browser because the browser trusts the source of the content, even when it's not coming from where it seems to be coming from.

CSP makes it possible for server administrators to reduce or eliminate the vectors by which XSS can occur by specifying the domains that the browser should consider to be valid sources of executable scripts. A CSP compatible browser will then only execute scripts loaded in source files received from those whitelisted domains, ignoring all other script (including inline scripts and event-handling HTML attributes).

As an ultimate form of protection, sites that want to never allow scripts to be executed can opt to globally disallow script execution.

Even major application services such as Facebook, Google, PayPal, and Twitter suffer from XSS attacks, which have grown alarmingly since they were first reported in a 2003 Computer Emergency Response Team advisory. The Open Web Application Security Project ranked XSS third on its 2013 list of top 10 Web vulnerabilities (the latest list as of February 2016), calling it the "most prevalent Web application security flaw. Underscoring the widespread risk of XSS intrusions, WhiteHat Security's May 2013 Web Security Statistics Report noted that 43 percent of Web applications were vulnerable to this kind of attack (www .whitehatsec.com/assets/WPstatsReport_052013.pdf).

Researchers have proposed a range of mechanisms to prevent XSS attacks, with content sanitizers dominating those approaches. Although sanitizing eliminates potentially harmful content from untrusted input, each Web application must manually implement it—a process prone to error. To avoid this problem, we use a different technique. Instead of sanitizing harmful scripts before they are injected into a website, we block them from loading and executing with a variation of the content security policy (CSP), which provides server administrators with a white list of accepted and approved resources. The Web application or website will block any input not on that list and thus there is no need for sanitizing. The white list also guards against data exfiltration and extrusion—the unauthorized downloading of data from a website visitor's computer.

EXISTING SYSTEM

Thus system will send request with identity. After that all the collected information will be send to collection database server. It not only protects clients from XSS attacks but also inform the vulnerable web servers.
This mechanism requires minimal effort and low performance overhead. Also, it will prevent all the types of XSS attacks.

Disadvantages
How to use the collected information in database is not addressed.
How to make system deployed universally has also not been addressed.
It requires modifications in the frameworks or installation of additional frameworks.
Approved scripts have to be identified by the website.
There is no single policy for all the documents.
Creating policies manually is a very tough task.
This approach incurs runtime overhead due to interception of HTTP traffic.
It requires user-defined security policies which can be labor-intensive.

PROPOSED SYSTEM

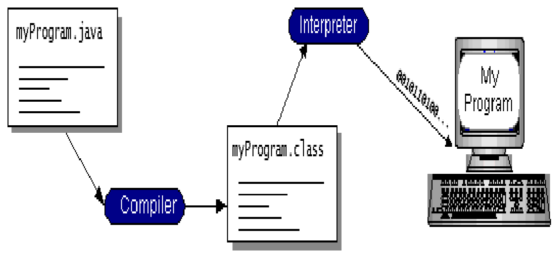
A client-side tool that acts as a Web proxy, disallows requests that do not belong to the website and thus thwarts stored XSS attacks. Browser-enforced embedded policies (BEEPs) let the Web application developer embed a policy in the website by specifying which scripts are allowed to run.With a BEEP, the developer can put genuine source scripts in a white list and disable source scripts in certain website regions. Document Structure Integrity (DSI) is a client-server architecture that restricts the interpretation of untrusted content. DSI uses parser-level isolation to isolate inline untrusted data and separates dynamic content from static content. However, this approach requires both servers and clients to cooperatively upgrade to enable protection.

SOFTWARE ENVIRONMENT

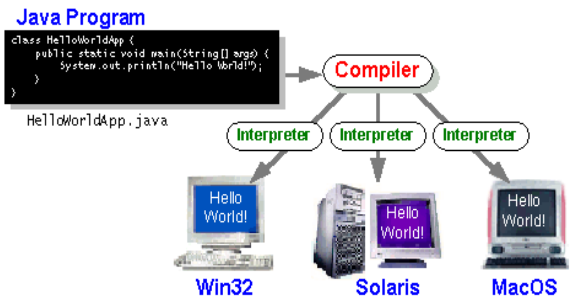
Java Technology
Java technology is both a programming language and a platform.

The Java Programming Language
The Java programming language is a high-level language that can be characterized by all of the following buzzwords:
- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



You can think of Java byte codes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make "write once, run anywhere" possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

The Java Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

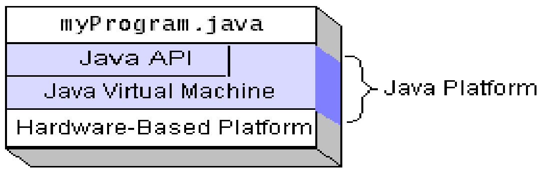
The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

What Can Java Technology Do?

The most common types of programs written in the Java programming language are *applets* and *applications*. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

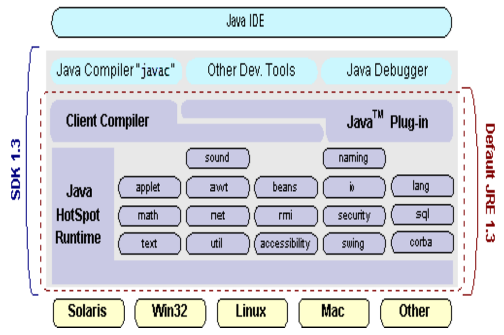
However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a *servlet*. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- The essentials: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- Applets: The set of conventions used by applets.
- Networking: URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.
- Internationalization: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- Security: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- Software components: Known as JavaBeans^{TM}, can plug into existing component architectures.
- Object serialization: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- Java Database Connectivity (JDBC^{TM}): Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.



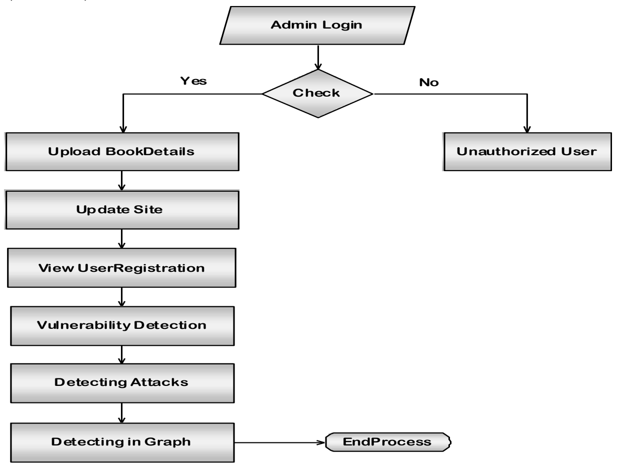
SYSTEM DESIGN

Data Flow Diagram / Use Case Diagram / Flow Diagram

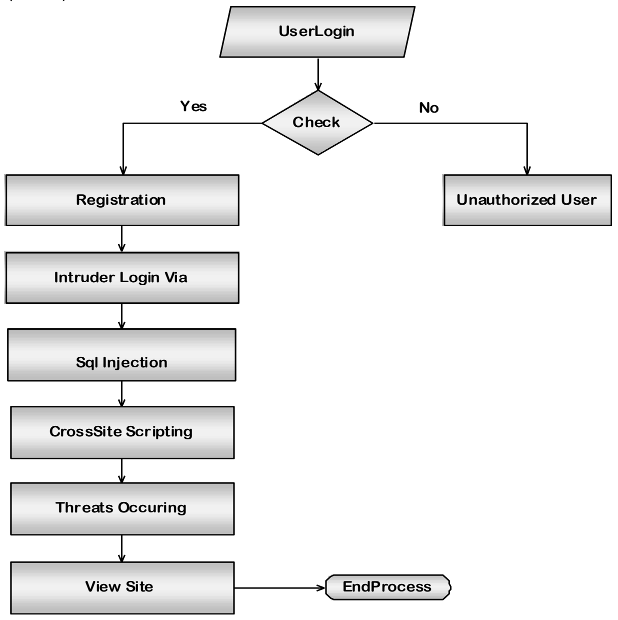
The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system.

Data Flow Diagram:

(Admin)

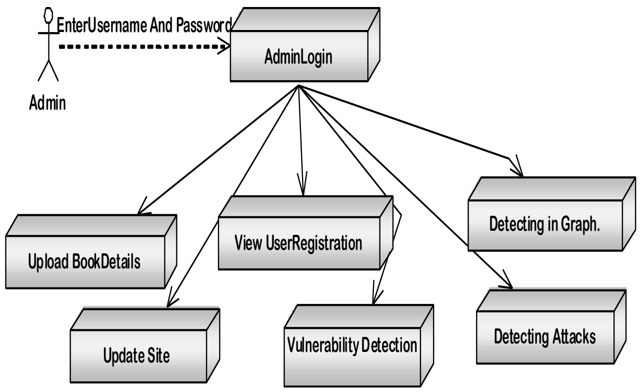


(User):

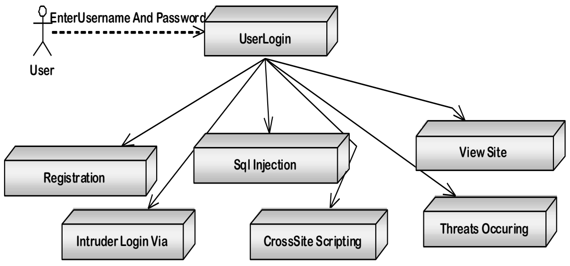


Component Diagram:

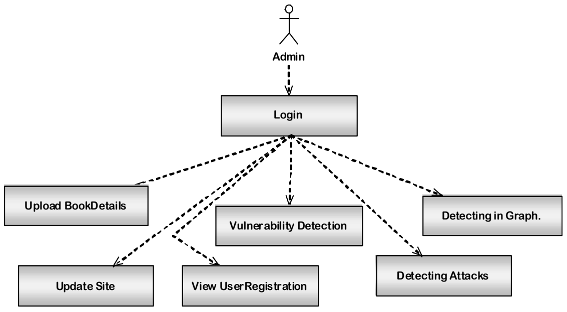
Admin:



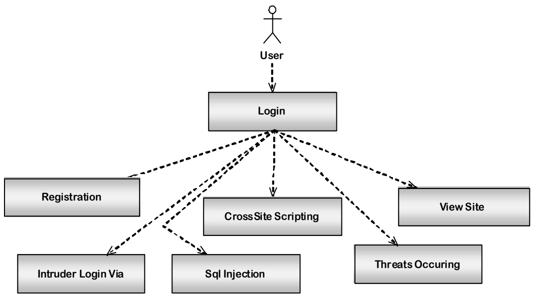
User:



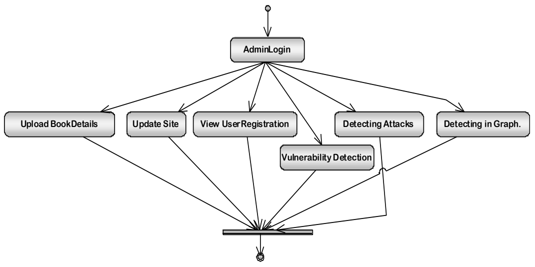
Use case Diagram:(Admin)



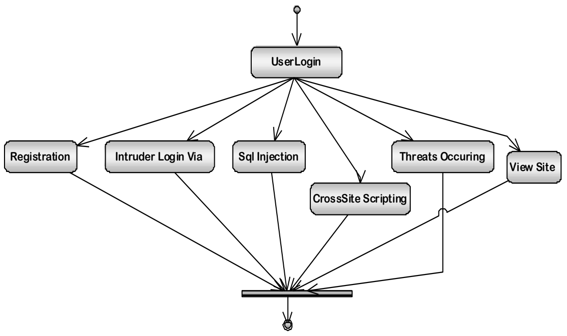
User:



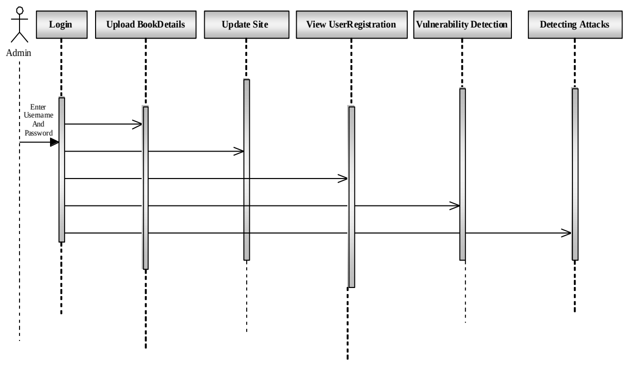
ACTIVITY DIAGRAM: (Admin)



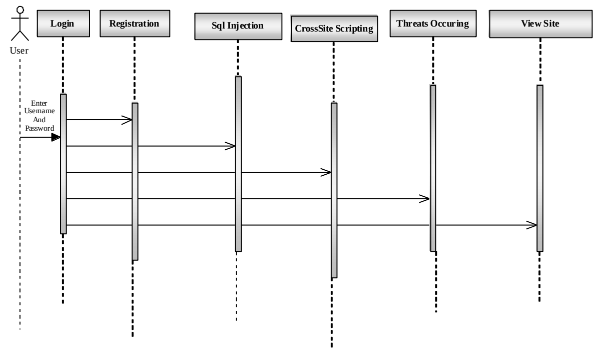
User:



Sequence Diagram:(Admin)



User:



CONCLUSION

Although our CSP has many benefits, it is not intended as a primary defense mechanism against XSS attacks. Rather, it would best serve as a defensein- depth mitigation mechanism. A primary defense involves tailored security schemes that validate user inputs and encode user outputs.

Cross site scripting has been a major threat for web applications and its users from past few years. Lot of work has been done to handle XSS attacks which include:

- Client side approaches
- Server side approaches
- Testing based approaches
- Static and dynamic analysis based approaches

Each kind of solution has been discussed in this paper. Different approaches have their own advantages and disadvantages. Major problems faced are:

- Requirement of complex frameworks
- Additional runtime overhead
- Intensive labor requirements
- Not being able to cover all types of XSS attacks
- Prone to human error
- Requires client action

- Not able to detect web content manipulation
- False positives and false negatives
- Effectives depend on completeness of specification

Based on our requirements we can choose among the possible solutions. However, there is no ideal solution for the detection and prevention of XSS attacks.