Cross Site Scripting XSS
Athmanathan Panneerselvam, Venkatachalam
Computer Science Department
San Jose State University
San Jose, CA 95192
appraveencse@gmail.com, sujandharanv@gmail.com

1. Abstract
Internet is used by vast number of people. Security is the most important factor in almost all applications which run over the internet. Though high security is enforced in the operating systems, servers, firewalls, and in client browsers there are some vulnerabilities that do exist in web based applications. Cross site scripting is a kind of security vulnerability in these applications. This is often caused by injecting venomous code into the web pages. This paper shows the types of cross site scripting, possible ways of injection, and how to extenuate these attacks. The attacks are illustrated with appropriate scenarios for better understanding. It is true that these attacks are hard to eliminate completely but can be mitigated to higher extent.

2. Introduction
Cross site scripting often called as XSS is a web attack that happens at application layer. Web pages are classified as static and dynamic. Static web pages do not suffer by this attack as there is no user session involved in it. Whereas, in dynamic web pages the server customizes the screen for each user and this customization is tracked through sessions. This session can be stolen by the attacker and the attacker can impersonate as the authenticated user. The code which is capable of running in the client side is injected to the web page and it is usually in the form of HTML or JavaScript. The attacker may be a malicious user or robot. HTML forms act as an entry point for this attack, the malicious code entered through the form is stored in the server and cause problems when this information is taken back from the server and displayed to the users. When the page is loaded along with the malicious script, it may send important information like cookies and form values of the user to the attacker or malicious website.

Cross Site Scripting had its acronym as CSS, this usage has not continued as it is confusing with Cascading Style Sheet. Apart from Javascript and HTML, technologies like ActiveX, VBScript, ActionScript, and Jscript are used to create XSS attacks.

In many cases this attack gets in to effect through hyperlink, hyperlink will have all the data collected from user’s page and encodes the malicious content in the link. When the user clicks this link, all the information gathered will be sent to the attacker. Since the attacker has your session, he or she can claim as you. XSS exploits the trust a user has for a website.

3. Threats Involved
- Ignorant user may execute the malicious scripts on viewing the pages which are dynamically generated by the attacker
- Attacker can use the user session before it gets expired.
- Attacker can redirect the user to malicious web site or server
- If the user clicks the link provided by the attacker, he/she gains the rights of the user. For the rest of the connection period, he/she can execute commands with the user privileges that accessed the URL.

4. Characters
Trudy is the attacker
Alice is the normal user
Assume my.sjsu.edu is vulnerable to XSS
5. **Attack Initialization**
Trudy first checks a particular website whether it is vulnerable to cross site scripting or not. If the website is vulnerable then Trudy would employ a way to attack. There are many ways that she can perform an attack, few of the methods are described below. In all the techniques she would use any of the ECMAScript to be executed in Alice’s system with the Alice’s privilege. Hereafter, anything from stealing cookies, corrupting or changing user settings, hijacking user account is possible.

6. **Scenarios**
The following scenarios illustrates the various attacks of Cross site scripting

6.1 **Malicious Link Attack**
Trudy forms a spiteful link by embedding malicious script in the link. She sends this link to Alice either through email or instant messages or she can put inside vulnerable XSS website which Alice often visits. We will look how to insert new fields and links in XSS vulnerable websites in the following sections. The link may look like

```html
<A HREF=http://my.sjsu.edu/registration.cgi?clientprofile=<SCRIPT>malicious code</SCRIPT>>
```

Click here to see your status</A>

Alice who is studying at SJSU in some department (not computer science) might click the link, thinking she can look at her status.

When the SJSU server sends back the response which include the value of the user profile along with some important information, the malicious code will start to execute once the response reaches the client browser.

6.2 **Cookie Theft**
Trudy knows that my.sjsu.edu is vulnerable to XSS. If she finds the usage of cookies in the website, she formulates an attack which can steal those cookies. In this attack Trudy first register a page with malicious script. When the page is displayed on the browser, the malicious script would come in effect and process the required information as told by Trudy. Finally, gathers all the cookie information and redirect the request to Trudy’s server. Trudy takes this request and sends that to my.sjsu.edu and masquerades as Alice. Since my.sjsu.edu only relies on the cookie information, it authenticates the request without knowing that the request was made by Trudy and not Alice. Trudy now can do anything what Alice can do in my.sjsu.edu

Collecting these kinds of sensitive information from other website is called Phishing. Let’s look in detail how Trudy can embed malicious script in a page.
For example consider the help page of my.sjsu.edu and there is text box to search items and an action button called “Search” to perform search.

![Image]

**Figure 3.1**

Trudy can frame javascript to include username and password like

```html
Login to get privileged access<br/>
Username: <input type="text" name="username"/>
Password: <input type="password" name="password" />
```

Trudy puts the above code in the search box and submits the search. The response page will look like

![Image]

**Figure 3.2**

Unknowing Alice will give her credentials and click login to get privileged access. Alice thinks that the username and password are indeed provided by my.sjsu.edu. After the click, Alice sensitive information is sent to Trudy. Trudy gets the credentials and stores in her place and redirects the page back to my.sjsu.edu/help with fake successful login message. Alice would not even know that her username and password is hijacked. Now Trudy can send unauthorized request to my.sjsu.edu.

### 6.2.1 Solution

Cookies read and write access can be prevented by setting the cookie attribute ‘HTTPOnly’. When HTTPOnly is turned on, browsers will not let the client side script to read or write the cookie. Not all browsers follow the rule of HTTPOnly. Some would allow read access and some would allow write access and some would/wouldn’t allow both. Another approach is to tie cookie to the ipaddress so it cannot be used by a malicious third party.

### 6.3 Stored XSS

This is similar to the previous scenario. The attack is handled in the same way but on the form fields which are going to be persisted. The previous scenario would not work out for Trudy if Alice closes her session and comes after the cookie is expired. In order to persist the changes made by Trudy, Trudy has to choose the fields which are going to be persisted. Say, Trudy has access to my.sjsu.edu, she logs in and edit her fields. She puts the malicious script in one of her profile's information field (ex: Address). After she updates her profile her malicious script is going to reside in the server database. Later Bob who is professor at my.sjsu.edu (he is not teaching security) tries to view information about Trudy. When he opens her profile, the malicious script gets in action and does what Trudy told.

### 6.3.1 Solution

This attack can be prevented by validating the input. If some scripts are present inside the form fields, server can disable the action of the script by simply stripping out the slashes. In this case even though Trudy's malicious code is stored in the server, the script cannot get executed when it reaches the client browser. The scripts which are already stored in the server before our input validation come in effect, can be prevented executing on the client.
server by *encoding* the output sent to the client browser. It is generally a good practice to encode both input and output form fields which goes in and out from the server. Here we are not stripping any slashes and this helps the webpage not to lose any information especially when Alice pastes some code snippets in a blog or forum.

### 6.4 Reflected XSS

The first phase involved in “Cookie Theft” is similar to this attack. Trudy embeds some malicious code in the webpage and posts the data to the server. The server in turn sends the response; the malicious code will get *reflected* in the response and get into action once the page reaches the client browser. This scenario happens in the stored XSS, in addition to that the form fields are stored in “Stored XSS” attack. The solution for this attack is same as the stored XSS attack.

### 6.5 Cross Site Request Forgery

CSRF or XSRF attacks by misusing the trust which the site has for the logged in user. Once the user logs in the system the website trusts that the subsequent tasks request are from the logged in user. Trudy can exploit this trust by accessing the task URL directly by using image object without bringing any notice to Alice. For example envious Trudy can order 100 items in shopping website when Alice is logged in. Alice would not know until she receives the summary of the shopping through email/mail.

This attack is achieved through HTML image tag or Javascript Image object and these objects will be embedded to the web page. When Alice loads the page, these objects perform a web request as mentioned by Trudy.

```html
<img src="www.shopitems.com/buy/laptop/id=1234&quantity=5" width="1" height="1" />
```

Assuming Alice has already registered her bank account details with the website shopitems.com loading the URL given above would simply order those items.

This can be achieved through javascript too by

```javascript
var objImage = new Image();
objImage.src = "www.shopitems.com/buy/laptop/id=1234&quantity=5";
</script>
```

Browsers are not the only applications which support CSRF. Files like .doc (word), .flv (flash), .dat(Movie), Atom, RSS (Feeds) can be used to embed the script. When these files are opened in their corresponding software it will make web request and will perform the action.

### 6.5.1 Solution

Token can be appended in each request. These tokens have to be associated with user session and the validity of the token must be limited to certain time period. If the token is not present form data won’t be used to take actions.

As a user make sure to log-off sites before browsing elsewhere.

### 6.6 Cross Site Tracing

Alice visits a webpage which is vulnerable to XSS. The server sends back the page with the malicious script in it which was stored earlier. The script executes and sends an HTTP Trace request to the web page which was recently visited by Alice in order to get information. The recently visited web page then sends cookies or other important information to Trudy.

XST is only a historic attack. This problem has been fixed in all the new browsers.

### 7. WebGoat

Webgoat is an amazing project which explained the discovered web attacks precisely. Attention to minute details, hints, and videos made us to like the tool very much. It does not provide explanation for all scenarios; however, it
provides a good foundation to start learning about the attacks.

8. Acknowledgements
We thank Dr. Mark Stamp, professor in the department of computer science at San Jose State University for giving us an opportunity to explore the web site attacks. The errors and inconsistencies in this article remain to us.

9. Conclusion
The survey results from 2007 indicate XSS carried out on the websites constitutes 80% of the known attacks and at least 68% of websites are open to XSS attacks on their users. This article covers almost all of the XSS attacks and the preventive measures. Escaping the slashes, input validation, cookie security, and disabling scripts are discussed to prevent the various attacks. If preventive measures are not taken appropriately, it will lead to other attack “SQL injection”. Every developer who works with web applications must study about these attacks and must consider all preventive measures while developing web applications. By doing this we can at least prevent novice and intermediate attacker from attacking the website.

10. References