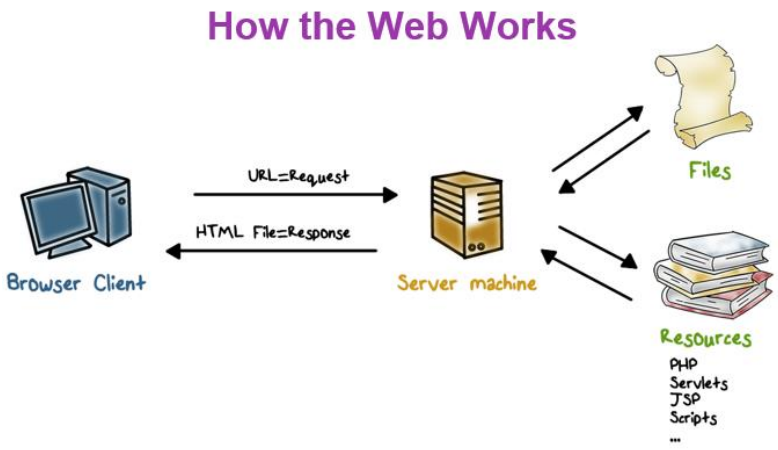
Reference: **Computer Security by Stallings and Brown, Chapter (not specified)**

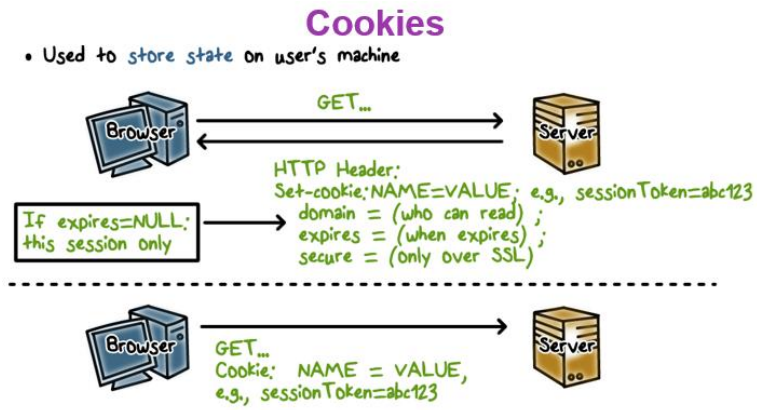
Web Security
Lesson Summary

- Overview of Web and security vulnerabilities
- Cross Site Scripting
- Cross Site Request Forgery
- SQL Injection

The web is an extension of our computing environment, because most of our daily tasks involve interaction with the web. In this lesson, we'll first reveal how the web works and discuss its major threat vectors. We will then discuss several attacks, including Cross Site Scripting, Cross Site Request Forgery and SQL Injection.



Let's first briefly review how the web works. The web browser and the web server communicate using the Hypertext Transfer Protocol or HTTP. The browser requests documents through a URL. The server responds with document in hypertext markup language, or HTML, which can include, not just the text, but also graphics, video, audio, Postscript, JavaScript, etc. The browser displays HTML documents and embedded graphics. It can run JavaScript and other helper applications.



HTTP is a stateless protocol, each request is its own TCP connection. For example, if you log into your bank's website, each click on a URL generates a separate TCP connection. In order to carry information across multiple HTTP requests such as user authentication, cookies are used.

A cookie is created by the web server when the user first logs into the site. It contains not only user identity information, but also security information such as access, expiration time, and if SSL is required. The user's browser stores the cookie and includes it in subsequent requests so that the server knows that these new requests are related. For example, they belong to the same user login session.

Cookie Quiz

Which of the following are true statements?

- ☐ Cookies are created by ads that run on websites
- ☐ Cookies are created by websites a user is visiting
- ☐ Cookies are compiled pieces of code
- ☐ Cookies can be used as a form of virus
- ☐ Cookies can be used as a form of spyware
- ☐ All of the above

Now let's do a quiz on cookies. Which of the following statements are true? First, cookies are created by ads that run on websites. Second, cookies are created by websites a user is visiting. Third, cookies are compiled pieces of code. Four, cookies can be used as a form of virus. Fourth, cookies can be used as a form of spyware. Sixth, all of the above.

First, cookies are created by ads that run on websites. This is true, cookies are created by ads, widgets, and elements on the web page the user is visiting.

Second, cookies are created by websites a user is visiting. This is true, as we have just explained.

- ☑ Cookies are created by ads that run on websites
- ☑ Cookies are created by websites a user is visiting
- ☐ Cookies are compiled pieces of code
- ☐ Cookies can be used as a form of virus
- ☑ Cookies can be used as a form of spyware
- ☐ All of the above

Third, cookies are compiled pieces of code. This is false, cookies are plain text. They're not compiled code.

Fourth, cookies can be used as a form of virus. This is false, since cookies are not compiled code, they cannot be used as a virus. They cannot replicate themselves. And they cannot be executed and are not self-executing.

Fifth, cookies can be used as a form of spyware. This is true, cookies store user preferences and browsing history, and therefore they can be used as spyware.

The Web and Security

- **Web page contains both static and dynamic contents, e.g., JavaScript**
- Sent from a **web site(s)**
- Run on the **user's browser/machine**

Now let's look at the main web security issues. The browser accepts contents and often runs dynamic contents such as scripts from websites. The question is, can a browser trust these contents.

In some cases, the browser can authenticate the website, but in many cases, authentication is not required. But even if a website is authenticated, the contents that it sends may not be trustworthy because the website may have

security vulnerabilities that allow attackers to inject malicious contents that get passed to the browser. Or the website includes contents or links to other websites which may also have security vulnerabilities.

The Web and Security

- Web sites run applications (e.g., PHP) to generate response/page
- According to requests from a user/browser
- Often communicate with back-end servers

On the server side a website runs applications that process requests from browsers and often interacts with back-end servers to produce content for users. These web applications, like any software, may have security vulnerabilities. Furthermore, many websites do not authenticate users. That is, attackers are not prevented from sending requests designed to exploit the security vulnerabilities in these web applications.

Web Browser Quiz

Mark each statement as true or false.

- ☐ Web browser can be attacked by any web site that it visits
- ☐ Even if a browser is compromised, the rest of the computer is still secure
- ☐ Web servers can be compromised because of exploits on web applications

Mark each statement as true or false. First, web browser can be attacked by any web site that it visits. Second, even if a browser is compromised, the rest of the computer is still secure. Third, web servers can be compromised because of exploits on web application.

First, web server can be attacked by any web site that it visits. This is true. As we have discussed, we can not authenticate all websites, and even if a website is authenticated, it may still have vulnerabilities.

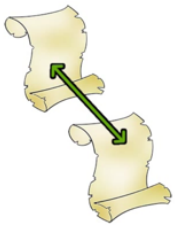
- [T] Web browser can be attacked by any web site that it visits
- [F] Even if a browser is compromised, the rest of the computer is still secure
- [T] Web servers can be compromised because of exploits on web applications

Second, even if a browser is compromised, the rest of the computer is still secure. This is false, because if a browser is compromised, it can lead to malware installation on the computer.

Third, web servers can be compromised because of exploits on web applications. This is true. The security vulnerabilities of web applications can lead to attacks that deface websites or the backend servers can be compromised as well. For example, credit card information can be stolen from the backend servers.

Cross-Site Scripting (XSS)

If a website allows users to input content without controls, **then attackers can insert malicious code as well**.



- **Social networking sites**, blogs, forums, wikis
- Suppose **a website echoes user-supplied data**, e.g., his name, back to user on the html page

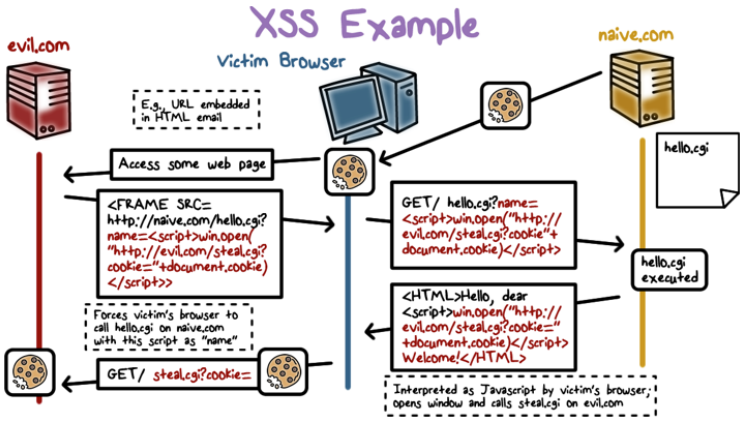
Now let's look at several attacks on the web.

The first is the cross-site scripting attack. Many websites allow user to input data and then simply display or echo data back. The website include the user input data in the HTML page to the user's browser, such websites include social networking sites, blogs, etc.

But what if the users are attackers, and the user input is malicious? Let's use an example to illustrate. Suppose a website echoes the user-supplied data, such as name, back to the user.

Suppose user Joe visits a website, and he supplies his name, Joe, to the website, then the website will send back a page saying, hello Joe. Now, suppose, instead of sending the user's name, Joe, to the website, the browser sends a script as his name. What's going to happen? The website will take this script as the user's name and echo it back to the web browser. That is the script will be included in the HTML page sent to the browser. Therefore, when the browser displays this webpage, the script will run, the webpage will display Hello World. Now, this is a benign case.

Now what if the script is malicious? When the browser gets the HTML page from the web server, it would just execute the script. So if the script is malicious, then the malicious script will be executed by the browser. But why would the browser send a malicious script to the website without a user knowing about it? How can this happen?



This can happen in a cross site scripting attack. Here's an example.

The user has logged into a vulnerable site, naive.com. And his browser now stores the cookie from naive.com. The user is then phished and clicks a URL to visit evil.com. Evil.com returns a page that has a hidden frame that forces the browser to visit naive.com and invoke hello.cgi web application on naive.com with a malicious script as the name of the user.

When the user's browser displays the HTML page from evil.com, he will be forced to visit naive.com and call hello.cgi with this malicious script as the user's name. Hello.cgi at naive.com then echoes the malicious script in the HTML page that is sent back to the user's browser. The user's browser displays the HTML page and executes the malicious script. The result is that the cookie to naive.com is stolen and sent to the attacker. You may ask, so what? If evil.com gets the cookie to naive.com? Because that

cookie can include session authentication information for naive.com. Therefore, by stealing the cookie the attacker can now impersonate the user.

XSS Query Quiz

Mark each statement as true or false.

☐ When a user's browser visits a compromised or malicious site, a malicious script is returned

☐ To prevent XSS, any user input must be checked and preprocessed before it is used inside html

Mark each statement as true or false. First, when a user's browser visits a compromised or malicious site, a malicious script is returned. Second, to prevent cross site scripting, any user input must be checked and preprocessed before it is used inside html.

First, when a user's browser visits a compromised or malicious site, a malicious script is returned. This is true because this a required step in the cross-site scripting attack.

☐ T When a user's browser visits a compromised or malicious site, a malicious script is returned

☐ T To prevent XSS, any user input must be checked and preprocessed before it is used inside html

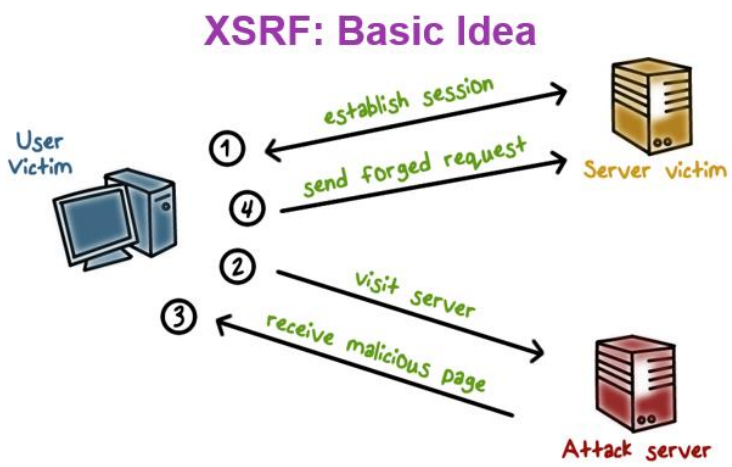
Second, to prevent cross-site scripting, any user input must be checked and preprocesseed before it used inside HTML. This is true. For example, the website can check that the name of a user should not be a script.

XSRF: Cross-Site Request Forgery

- A browser **runs a script from a "good"** site and **a malicious script from a "bad" site**
- Malicious script **can make forged requests** to "good" site with user's cookie

Cross-Site Request Forgery is another web-based attack.

A user's browser may be running a script from a good site and also malicious script from a bad site. This can happen when the user has logged into the good site and kept the session alive. For example, the user has logged into Gmail and has not logged off. Meanwhile, the user maybe browsing other sites include the bad site that sends malicious script to the browser. The malicious script can then forge a request to the good side using the users cookie. The good side does not know that the request was not sent by the user.



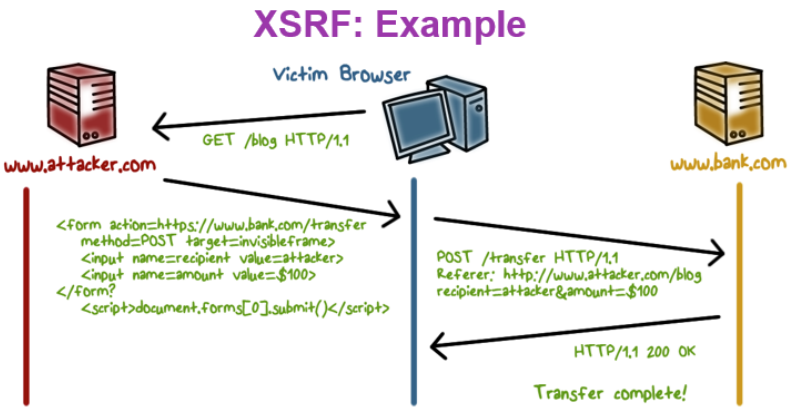
Here's an illustration.

The user logs in, and establishes a session with a good site, and keeps the session alive.

Meanwhile, the user browses a bad site. For example, because he's phished. And the browser runs a malicious script from the bad site. The malicious script then sends forged requests to the good site.

Here's a realistic example.

A user logs into bank.com and forgets to sign off. The session cookie remains in browser. The user then visits a malicious website which sends an HTML page that contains a hidden I frame that includes this malicious content, that is when the user's browser displays the HTML page, actions will be performed on the bill payment form of bank.com as if users are entering these values. Because the I frame is invisible the user knows nothing about it. The browser will send a request on behalf of the user and without his consent or knowledge.



And here is the illustration of this example.

The user logs into bank.com and keeps a session alive. So the browser has the cookie to bank.com. Meanwhile, the user is phished to visit attacker.com. And in return, the web browser gets this malicious content.

When this malicious page is displayed on the user's browser, the browser will make the request on the user's behalf to bank.com. And since the user is still logged into bank.com, the user's cookie is also sent to bank.com along with the request. And so the bank website believes that the request is from the user, and therefore, the payment request is fulfilled.

XSRF vs XSS

- **Cross-site scripting**
  - User trusts a **badly implemented** website
  - Attacker **injects a script** into the trusted website
  - User's browser **executes attacker's script**
- **Cross-site request forgery**
  - A badly implemented website trusts the user
  - **Attacker tricks user's browser** into issuing requests
  - Website executes attacker's requests

Now let's compare cross-site request forgery and cross-site scripting.

In cross-site scripting, the user's browser trusts a badly implemented website that does not check input content. That is, the attacker is able to inject a script into the trusted website and as a result, the user's browser executes the attacker's script.

In cross-site request forgery a website trusts that requests are from the user. But in fact, the attacker forges user requests to the website. As a result, the website executes the attackers malicious actions.

Both cross-site scripting and cross-site request forgery are the results of security weakness of websites. In particular, the lack of authenticating and validating user input.

XSRF Quiz

Which of the following methods can be used to prevent XSRF?

- ☐ Checking the http Referer header to see if the request comes from an authorized page.
- ☐ Use synchronizer token pattern where a token for each request is embedded by the web application in all html forms and verified on the server side.
- ☐ Logoff immediately after using a web application.
- ☐ Do not allow browser to save username/password and do not allow web sites to "remember" user login
- ☐ Do not use the same browser to access sensitive web sites and to surf the web freely
- ☐ All the above

Which of the following methods can be used to prevent XSRF forgery? First, checking the http referer header to see if the requests comes from an authorized page. Second, use synchronized token pattern, where a token for each request is embedded by the web application in all HTML forms and verified on a server site. Third, log off immediately after using a web application. Fourth, do not allow browser to save username/password, and do not allow web sites to remember user login. Fifth, do not use the same browser to access sensitive web sites and to surf the web freely. Sixth, all of the above.

The correct answer is all the above.

For example, by checking the HTTP referer header, bank.com would notice that attacker.com is the referer, and then bank.com can stop the payment requests.

- ☐ Checking the http Referer header to see if the request comes from an authorized page.
- ☐ Use synchronizer token pattern where a token for each request is embedded by the web application in all html forms and verified on the server side.
- ☐ Logoff immediately after using a web application.
- ☐ Do not allow browser to save username/password and do not allow web sites to "remember" user login
- ☐ Do not use the same browser to access sensitive web sites and to surf the web freely
- ☑ All the above

Similarly, bank.com can generate a token and embed it with bill pay form that is sent to the user's browser, so that when it receives request from the browser it can verify that the request are from that bill pay form. In addition, these practices will prevent a malicious script from using the live session to good site.

Structured Query Language (SQL)

- Widely used **database query language**
- Retrieve a set of records, e.g.,

  SELECT * FROM Person WHERE Username='Lee'
- Add data to the table, e.g.,

  INSERT INTO Key (Username, Key) VALUES ('Lee', Ifoutw2)
- Modify data, e.g.,

  UPDATE Keys SET Key=ifoutw2 WHERE PersonID=8

Before we discuss SQL injection attacks, let's quickly review SQL.

SQL is the most widely used database query language. We can use it to retrieve database records, modify the database, for example by adding records to a table or modifying the specific values of a record.

Nowadays, many databases have a web front end to allow users to create a database using the web.

The website typically offers a web form for entering the query and runs a program to form the input into SQL query and send it to the backend database server.

Sample PHP Code

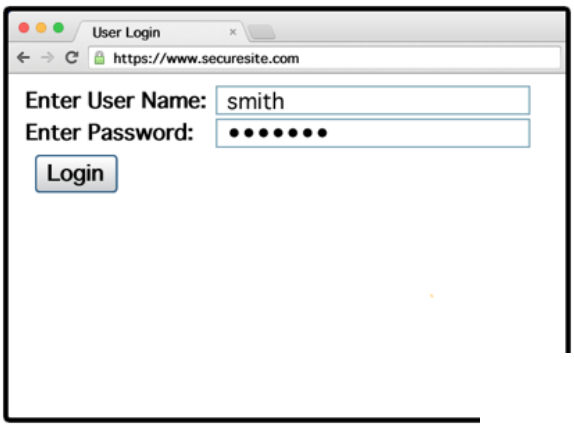
- **Sample PHP**

```
$selecteduser = $_GET['user'];
$sql = "SELECT Username, Key FROM Key".
       "WHERE Username='$selecteduser'";
$rs = $db->executeQuery($sql);
```

- What if **'user' is a malicious string** that changes the meaning of the query?

For example, the web application can be a PHP and it takes user input and forms a SQL query. The security threat here is that the input may be malicious that is the SQL to the database server is malicious and can lead to compromise of data confidentiality and integrity.

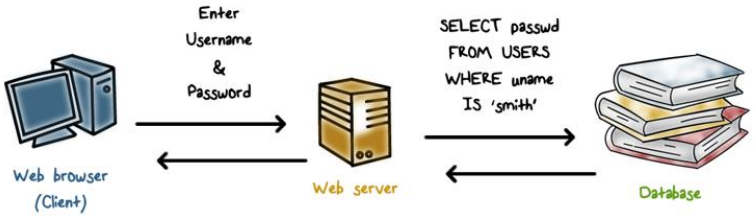
Example Login Prompt



Here's an example of a web form.

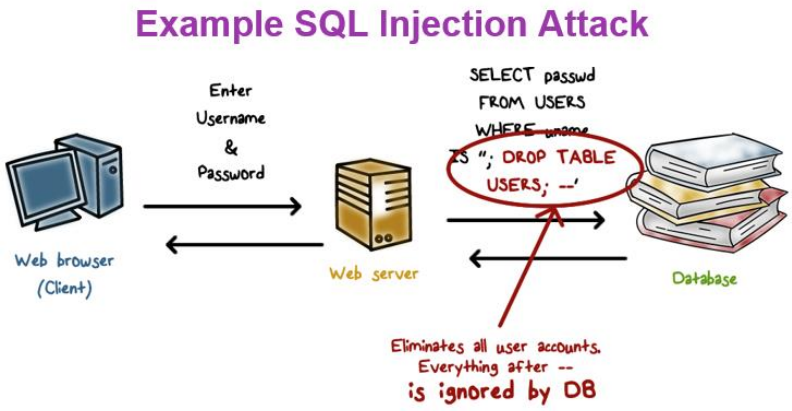
A user enters his name and password. The password is sent in encrypted hash form to the web server, and the web server needs to retrieve the user's password hash from the database and compare and authenticate the user.

Normal Login



The web server issues a SQL query to the database, so this is the example of normal usage.

Suppose an attacker enters this malicious string as the user name. What's going to happen?



The actual SQL query sent from the web server to the backup database server would look like this, and the result of running this query to database is to delete all user records.

SQL Injection Quiz

Which is the better way to prevent SQL injection?

Now let's do a quiz on SQL injection attack. Which of the following is the better way to prevent SQL injection? Is it using blacklisting to filter out bad input or using whitelisting to allow only well-defined set of safe values?

- ☐ Use blacklisting to filter out “bad” input
- ☐ Use whitelisting to allow only well-defined set of safe values

The correct answer is use whitelisting to allow only well-defined set of safe values.

Blacklisting is very hard to implement, because there can be many, many possible ways to inject malicious strings. That is, it's very hard to have a complete blacklist.

- ○ Use blacklisting to filter out "bad" input
- ◉ Use whitelisting to allow only well-defined set of safe values

Web Security
Lesson Summary

- Both browser and servers are vulnerable: dynamic contents based on user input
- XSS: attacker injects a script into a website and the user's browser executes it
- XSRF: attacker tricks user's browser into issuing request, and the website executes it
- SQL injection: attacker inject malicious query actions, and a website's back-end db server executes the query

Both the web browsers and web service are vulnerable, because they need to execute or generate dynamic content based on user input, which can be malicious.

In cross-site scripting, the attacker injects a script into a website, and the user's browser executes this script.

In cross-site request forgery, the attacker tricks the user's browser into issuing a request. And the website executes this request.

In SQL injection, the attacker injects malicious query actions. And a website's backend database server executes this malicious query.