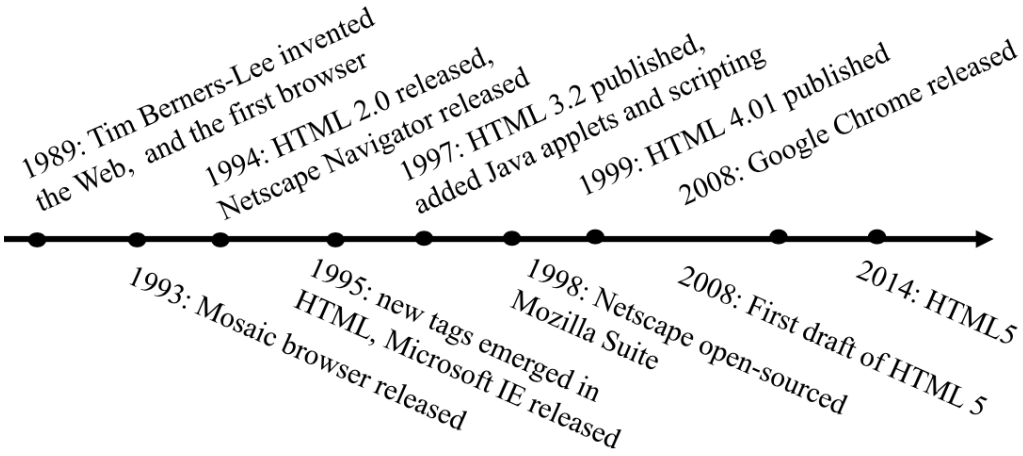
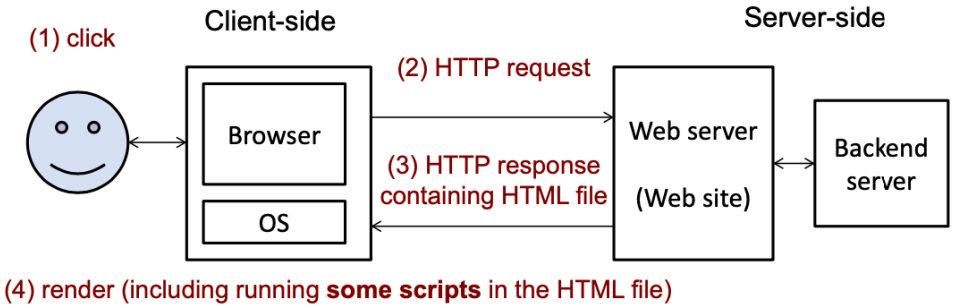
Web Security

Background



Overview of HTTP



1. A user clicks on a URL link e.g. luminus.nus.edu.sg/
2. A HTTP request is sent to the server, with cookies if any
3. Server constructs and include a HTML file inside its HTTP response to the browser, likely with cookies
4. The browser renders the HTML file, which describes the layout to be rendered and presented to the user, and the cookies are stored in the browser.

Sub-Resources of a Web Page

A HTML page may contain sub-resources such as images, multimedia files, CSS and scripts, including ones from external or third-party websites. When parsing a page with sub-resources, the browser also contacts the respective server for each sub-resource. A separate HTTP request for every single file on a page is thus made, since each file requires its own.

Interactive Query-Page

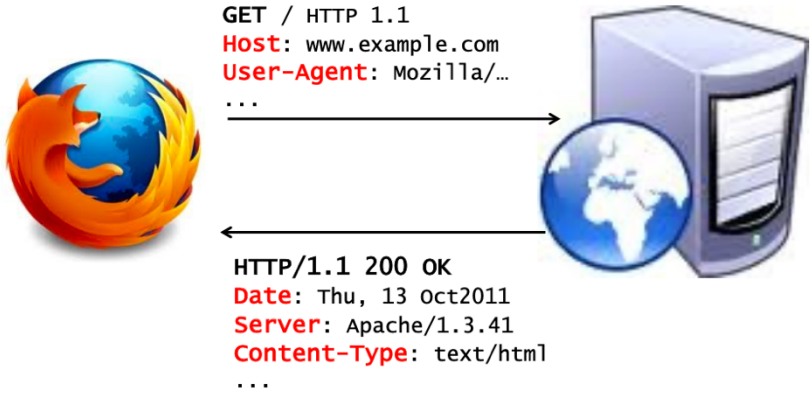
Let us take a look at Google.

1. When we visit their site, a HTML file is sent by the server to the browser. The browser renders the site.
2. The user enters the search keywords “CS2107 NUS”.
3. The browser, by running the HTML file, constructs a query, for instance: https://www.google.com/search?client=safari&source=hp&ei=MJHSXdiAEoHLvgTY64qAAw&q=**CS2107+NUS**&oq=**CS2107+NUS**&gs_l=... as URL parameters. These information are useful for the server, and could even be **in the form of a script.**

4. The server constructs a reply. In some cases, the reply contains substrings sent in Step 3.

HTTP Request and Response Messages
For every message, both request and response, headers are used. There are many headers used.

The key ones we would need to be aware of are:



HTTP Request

- Request line
  - GET /test.html HTTP/1.1
- Request headers
  - Accept
    - image/gif
    - image/jpeg
    - */*
  - Accept-Language
    - us-en
    - fr
    - cn
  - Cookie
    - theme=light;
    - sessionToken=abc123;
- Empty/blank line
- Optional message body

HTTP Response

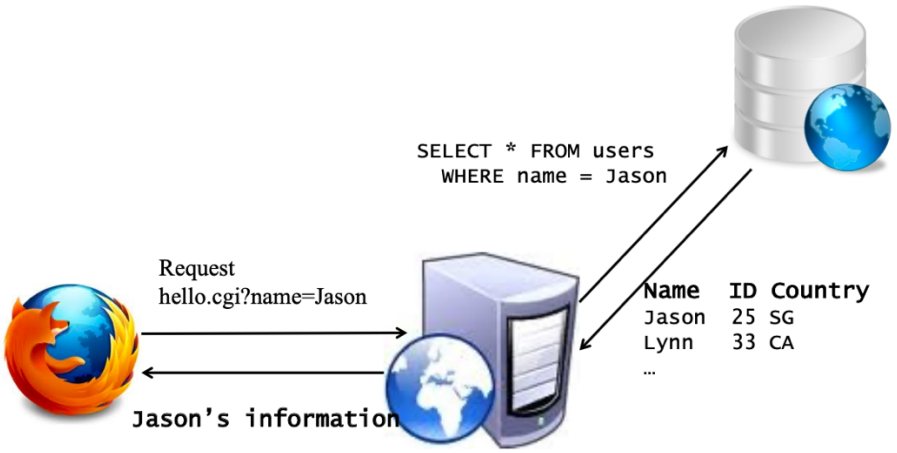
- Status line containing status code & reason phrase
  - HTTP/1.1 200 OK
  - HTTP/1.0 404 Not Found
- Response headers
  - Content-Type
    - text/html
  - Content-Length
    - 35
  - Set-Cookie
    - theme=light;
    - sessionToken=abc123;
    - Expires=Wed, 09 Jun 2021 10:18:14 GMT

Web Client and Server Components
Client-side Components

- Hypertext Markup Language (HTML): Webpage Content
- Cascading Style Sheets (CSS): Webpage Presentation
- Javascript: Webpage Behaviour – making pages interactive and responsive

Server-side Components

- Web server: nowadays a scripting language is typically used as well, e.g. PHP
- Database server



Understanding Javascript
Example of Javascript in HTML:
<script type="text/javascript"> document.write('Hello World!"); </script>

What can Javascript do in a browser?

- Write a variable or text into a HTML page
  - document.write("<h1>" + studentname + "</h1>")
- Read and change HTML elements
  - var doc = document.childNodes[0];
- React to events, such as when a page has finished loading or when a user clicks on a HTML element
  - <a href="someURL" onclick="alert('User just clicked me!')">
- Validate user data, such as form inputs
- Access cookies
  - var doccookie = document.cookie;
- Interact with the server, such as using AJAX (Asynchronous Javascript And XML)

Understanding Hypertext Preprocessor (PHP)
PHP is a widely used, free server scripting language for making dynamic web pages.

A sample PHP page:

```
<!DOCTYPE html>
<html>
<body>
<?php
echo "My first PHP script!";
?>
</body>
</html>
```

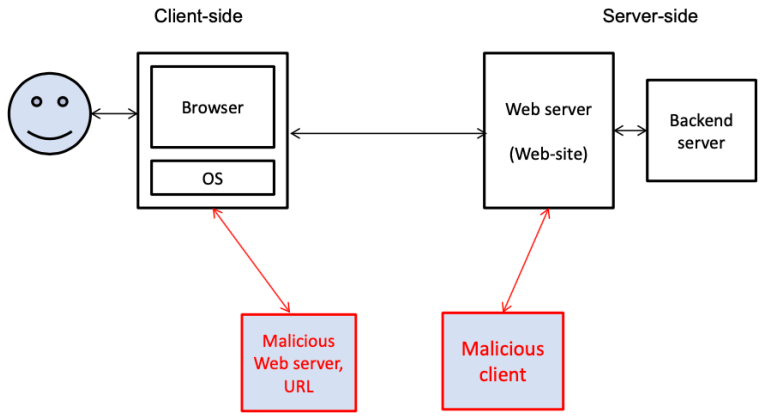
Security Issues and Threat Models

There are many reasons why Web Security is tough.

- Browsers run with the same privileges as the user – they can thus access the user's files
- At any instance, multiple servers with different domain names could provide the content. Access isolation among sites is thus required.
- Browsers support a rich command set and controls for content providers to render the content
- Browsers keep user's information and secrets. For example, some are stored in cookies.
- For enhanced functionality, many browsers support plugins, add-ons and extensions by third parties. The exact definitions and differences between these three may not be clear.
- Users can update content in the server, and more and more sensitive data is stored in the Web or on the cloud
- For the PC, the browser is becoming the main/super application. To some extent, the browser **IS** the OS.

Threat Model 1: Attackers as Another End Systems

In networking jargon, the computers that are connected to a computer network are sometimes referred to as **end systems**, because they sit as the edge of the network. End systems are the devices that provide information or services.



In Threat Model 1, the attackers are just another end systems. For example, it could be a malicious web server that lures the victim to visit it, or a malicious web client who has access to the targeted server.

Attacker Types in Threat Model 1

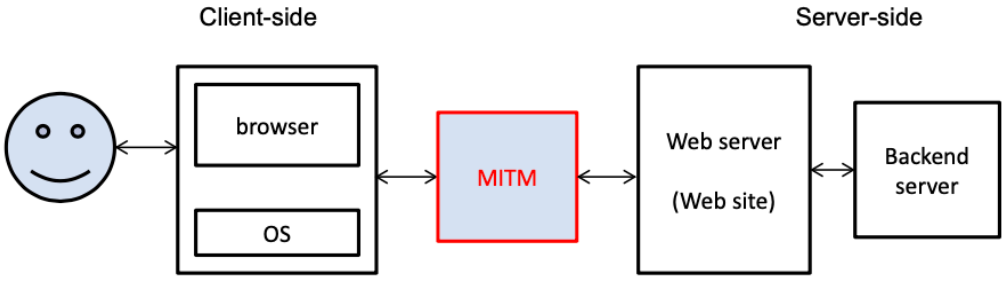
These are the different types of attackers:

1. Forum Poster
    a. The weakest attacker type who simply baits you to click something
        i. Link may direct you to a spoofed site or perform stored XSS (covered later)
    b. A user of an existing web app e.g. spreading on existing forums
    c. Does not register domains or host his own application content
2. Web Attacker
    a. Owns a valid domain and web server with an SSL certificate
    b. Entices a victim to visit his site
        i. "Click here to get a free iPad"
        ii. Or via an advertisement, thus no clicks needed
    c. Cannot intercept or read traffic for other sites
    d. It is also the most commonly-assumed attacker type

        i. It is quite effortless these days to get one's own domain and become a Web Attacker
3. Related-domain Attacker
    a. A web attacker who is able to host content in a **related domain** of the target web application
    b. It may be a sibling or a child domain of the target application.
        i. attacker.target.com
        ii. attacker.app.target.com
        iii. In an attempt to target app.target.com
4. Related-path Attacker
    a. A web attacker who is able to host an application on a **different path** than the target application, but **within the same origin**
        i. www.comp.nus.edu.sg/~attacker
        ii. In an attempt to target www.comp.nus.edu.sg/~target
    b. Very high level, difficult

Threat Model 2: Attackers as a Man-in-the-Middle

A man-in-the-middle attack is an attack where the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other.



In Threat Model 2, the attacker is a Man-in-the-Middle (at the IP layer). For example, an attacker can be a malicious café-owner who provides the free WIFI services in our previous examples.

Attacker Types in Threat Model 2

These are the different types of attackers:

1. Passive Network Attacker
    a. An attacker who can passively eavesdrop on network traffic, but cannot manipulate or spoof traffic (Eve)
    b. Can additionally act as a web attacker
2. Active Network Attacker
    a. An attacker who can launch active attacks on a network (Mallory)
    b. Can additionally act as a web attacker
    c. This is the most powerful attacker type
    d. Yet, the attacker is generally considered to be incapable of presenting valid certificates for HTTPS sites that are not under his control.
        i. Unless the user himself clicks through i.e. accepts the invalid certificate
        ii. Or the attacker is a rouge CA, which we have touched upon before
        iii. Both conditions are generally hard to fulfil

Ultimately, it's still difficult to clearly classify web attacks, since many use a combination of various other attacks. We will now look at some of these web attacks, and the relevant and common protection mechanisms.

Attacks on the Secure Communication Channel (SSL/TLS)

As we know, HTTPS protocol stands for HTTP over TLS/SSL, where the latter includes Netscape SSL 2.0 in 1993 to TLS 1.3 in 2018.

It provides a data channel that has confidentiality, integrity and authenticity between two programs, providing security against a computationally-bounded "network attacker".

HTTPS works via:

- Ciphers negotiation
- Authenticated key exchange (AKE)
- Symmetric key encryption and MAC

Attack on a Secure Channel by a MITM

There are two conditions of a Man-in-the-Middle attack:

- The attacker is a MITM between the browser and the web server
- The attacker is able to sniff and spoof packets at the TCP/IP layers

If the connection is through HTTPS, the MITM is unable to compromise both confidentiality and authenticity **unless** the web user accepts a forged certificate or a rouge CA (which is the MITM in this context).

Yet, this might not always be the case as there exist vulnerabilities in the protocol or its implementation. Some examples are:

- FREAK Attack
    - Factoring RSA Export Keys is a security exploit of a cryptographic weakness in the SSL/TLS protocols long ago. There were limitations on exportable software to use only public key pairs with RSA moduli of 512 bits or less so that the NSA can crack it but not other organisations with lesser computing resources.
    - However, in the early 2010s, increases in computing power meant that anyone could break it. Combined with the ability of a MITM attack to manipulate the initial cipher suite negotiation between the endpoints in the connection and the fact that the finished hash only depended on the master secret, this meant that a MITM attack with only a modest amount of computation could break the security of any website that allowed the use of 512-bit export-grade keys.
    - Discovered in 2015, but existed since 1990s.
- Superfish Attack
    - It's a preinstalled MITM on Lenovo machines since 2014 which sits between the machine and the web server. The installation included a universal self-signed certificate authority, allowing the Superfish Visual Search software to intercept your HTTPS communications and inject advertisements via proxy re-encryption.
    - However, this CA had the same private key across laptops, allowing third-party eavesdroppers to intercept or modify HTTPS secure communications without triggering browser warnings.
    - Its HTTPS-decrypting and interception software is basically an SSL hijacker.
- Heartbleed Attack
    - Heartbleed is a security bug in the OpenSSL cryptography library, which is a widely used implementation of the Transport Layer Security (TLS) protocol. It was introduced into the software in 2012 and publicly disclosed in April 2014.
    - It may be exploited regardless of whether the vulnerable OpenSSL instance is running as a TLS server or client. It results from improper input validation due to a missing bounds check in the implementation of the TLS Heartbeat Extension.

    - The Heartbeat Extension provides a way to test and keep alive secure communication links without the need to renegotiate the connection each time.
    - The vulnerability is classified as a buffer over-read, where more data can be read than should be allowed.
- Re-negotiation Attack
    - Covered in the section on HTTPS, under Tutorial 5.
    - Basically, there was a lack of continuity between communications before and after re-negotiation, allowing attackers to "combine" their communications with a victim's.
- BEAST Attack
    - The Browser Exploit Against SSL/TLS Attack attacked the fact that the Cipher Block Chaining mode of the AES used the last ciphertext block (which is visible) of one packet as the initial IV of the next packet.

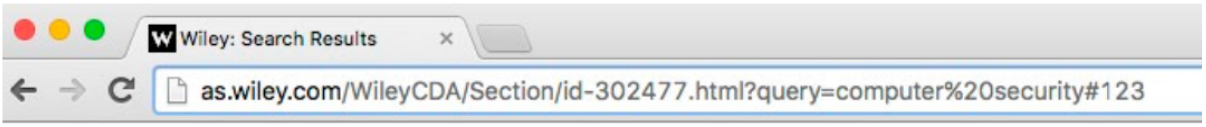
URL and Address Bar Insecurities (i.e. Mislead the User)

The Uniform Resource Locator (URL) consist of a few components:

- Scheme
- Authority (aka the hostname)
- Path
- Query
- Fragment

For example:
http://www.wiley.com/WileyCDA/Section/id-302477.html?query=computer%20security#12
Scheme: http
Authority: www.wiley.com
Path: WileyCDA/Section/id-302477.html
Query (?): query=computer%20security
Fragment (#): 12

A browser usually helps with this by displaying the authority and everything else with different levels of intensity i.e. different font colour.



URL Confusion

Suppose that there is no clear visual distinction between the hostname and the path of a URL. The delimiter that separates the hostname and the path can be a character **in the hostname or path** itself. For example, in the previous example, the character is "/", which is also found in the path.

Thus, a malicious website whose hostname contains the targeted hostname followed by some character that looks like the delimiter "/" may confuse users:
www.wiley.com.lwiley.in/Section/id-302477.html

The actual domain is lwiley.in. Another example is nuslogin.789greeting.co.uk, where 7 supposedly looks like a "/".

For all these, however, the displayed different intensities can help the user spot the attack. But what if the address bar itself is spoofed?

Address Bar Spoofing
The address bar is an important component to protect, as it is the only indicator of what URL the page is actually rendering. What if it can be spoofed? An attacker could trick someone to visit malicious URL X, all the while making the user believe that the URL is Y. This may be possible with a poorly-designed browser.

An example would be how in the early design of browsers, a web page could render objects or pop-ups in an arbitrary location. This allowed a malicious page to overlay a spoofed address bar on top of the actual address bar.

Current versions of popular browsers have mechanisms to prevent this issue. However, a recent attack, CVE-2015-3830: Android Browser All Versions – Address Bar Spoofing Vulnerability, occurred.

Cookies and Same-Origin Policy
Note that the same-origin policy is not an attack, but a protection mechanism to protect cookies.

A HTTP cookie is a piece of textual data sent by a Web server and stored on the user's web browser while the user is browsing. It is sent in a HTTP response's "Set-Cookie" header field.

A cookie consists of a name-value pair, and can be used to indicate a user preference, shopping cart content, server-based session identifier, etc. Whenever a client revisits the site i.e. submit another HTTP request, the browser automatically sends in all "in-scope" cookies back to the server in its HTTP request's "Cookie" header.

What is "in-scope"? Cookies are only sent back to the same cookie origin, i.e. to the server that is the "origin" of the cookies. Scheme and protocol checking **may be** optional.

Viewing Cookies
On Firefox
- Right-click → View Page Info → Security → View Cookies; or
- Tools → Web developer → Developer toolbar → Storage

On Chrome
- chrome://settings/content/cookies

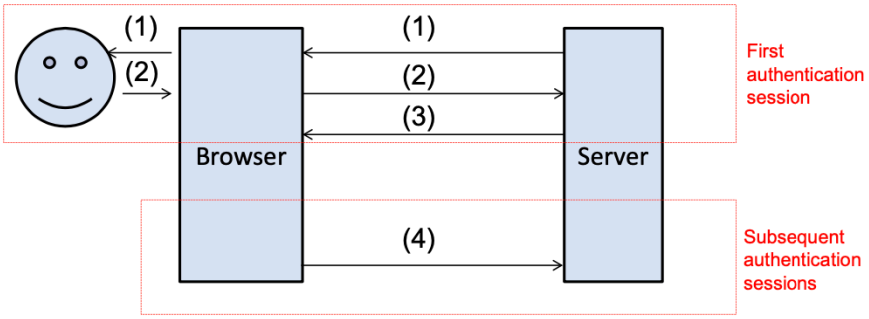
Usage
There are a few types of cookies, such as:
- Session cookie
  - Deleted after the browsing session ends
- Persistent cookie
  - Expires at a specific date or after a specific length of time
- Secure cookie
  - Can only be transmitted over HTTPS

The checking of scheme when doing the "same origin" check for cookies is optional, except for secure cookies that strictly require HTTPS.

Since HTTP is stateless (and thus HTTPS as well), there is a need to keep track of a web session. Cookies are thus commonly used to set and indicate a session ID. It is a better approach than attaching the session ID as a URL-encoded parameter in the HTTP request or as a form field, but it does have its own issues.

Token-based Authentication
To ease a web user's tedious task of repeated logins, many websites use "token-based" authentication. After a user A is authenticated e.g. entering the right username and password, the server sends a value t, known as the token, to A. In subsequent connections, whoever presents the token t is thus accepted as the authentic user A.

This token t typically has an expiry date, and it can identify a session, thus it's also called a session ID (SID). In web applications, a token is often stored as a cookie.



1. Authentication challenge e.g. asking for password
2. Authentication response that involves the user e.g. providing the password
3. Server sends a token t and browser keeps the token t.
4. Browser presents the token t with every HTTP request, and the server verifies it

We assume that the communication channel is secure – done over HTTPS (with server being authenticated) and the HTTPS being free from vulnerabilities.

Choice of Token
A token t needs to be random and sufficiently long. However, if the token t is a randomly chosen number, then the server would need a table to store all issued tokens, which can take up a lot of space.

To avoid storing the table, one could use:
- Insecure method
    - The cookie is some meaningful information concatenated with a predictable sequence number
    - Example: t = "alicetan:16/04/2015:128829"
    - Insecure as an attacker, who knows how the token is generated by observing their own token, can forge it
    - This is the weakness of security by obscurity
        - We assume that the attackers do not know the format, which is false
- Secure method
    - The cookie consists of two parts
        - Randomly chosen value or meaningful information like the expiry date
        - MAC computed with the server's secret key
    - Example: t = "alicetan:16/04/2015:adc8213kjd891067ad9993a"
    - Secure as it relies on the security of MAC

Both for methods, when the server finds out that the provided token t is not in the correct format or does not contain the correct content, the server rejects the token.

Scripts
A script that runs in a browser can access cookies. The scripts must then be limited in the cookies they are able to access. Due to security concerns, browsers employ the access control mechanism called Same-Origin Policy

Same-Origin Policy (SOP)
SOP states that a script in web page A (identified by its URL) can access cookies stored by another web page B (identified by its URL) only if A and B have the same origin. The origin is defined as the combination of protocol, hostname and port number.

Sounds safe and simple, but unfortunately there are complications.

Let us assume we have this URL: http://www.example.com. Then what we have is as such:

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password@**www.example.com** /dir2/other.html	Success	Same protocol, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

There are many exceptions, resulting in confusion and this being error-prone. For example, unlike other browsers, the Microsoft IE does not include the port in the calculation of the origin, using the Security Zone in its place.

Cross Site Scripting (XSS) Attacks
There are two types of XSS Attacks:

1. **Reflected (Non-Persistent) XSS Attack**

In many sites, the client can enter a string into the browser, which is to be sent to the server. The server then responds with a HTML containing s, which is then rendered and displayed by the client's browser. If the string contains a script, then the page will actually execute the script!

Note that this won't work if the server performs HTML (entity) encoding, which replaces the special characters with safe versions of the character e.g. replaces "<" with < or %3c.

The steps are as follows:
1. The attacker tricks the user to click on a URL
    a. The URL contains the target website and a malicious script s
    b. The link could be sent via email, href as "click me", or is a link in a malicious window
2. The request is sent to the server
3. The server constructs a response HTML
    a. The server does not check the request carefully
    b. The response contains s
4. The browser renders the HTML page and runs the script s

A script may be benign, but a malicious one could deface the original webpage or steal the user's cookies. Due to the same-origin policy, as the script now comes from the target web server, it can access cookies previously sent by the web server.

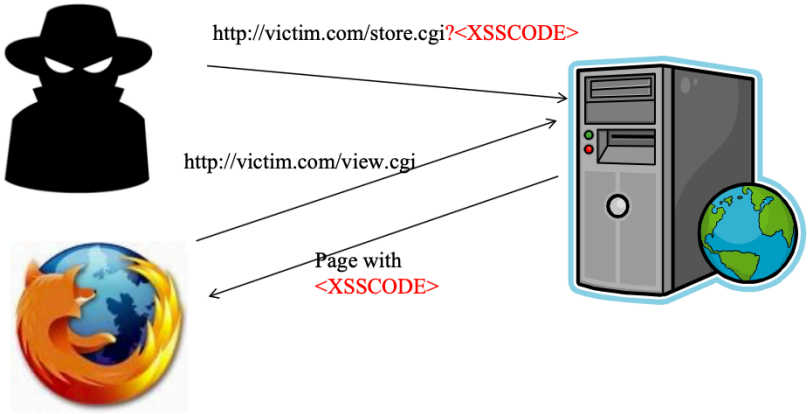
This is an example of privilege escalation – the malicious script from the attacker now has the privileges of the web server and can read the cookies. The attack exploits the client's trust of the server, as the browser believes that the injected script is from the server.

2. **Stored (Persistent) XSS Attack**

The script s is **stored** in the target web server. For example, it may be posted onto the forums, thus stored in a forum page. The attacker is a malicious forum poster.

An example would be the Samy XSS works on MySpace.com, where Samy became a friend of 1 million users in less than 20 hours.

This is more dangerous than reflected XSS attacks as the malicious script is rendered automatically without the need to lure target victims to a third-party website. The victim to script ratio is also many to one.



What is XSS in short?

It is a type of injection attack on web apps, whereby a forum poster or web attacker attacks another web user by causing the latter run a (malicious) script from the former in the execution context of a page from an involved web server, thus subverting the Same Origin Policy.

The attack works by exploiting the victim's trust of the involved server:

- In reflected XSS, the web server that **returns** a page reflecting the injected script
- In persistent XSS, the web server that **stores** a page containing the injected script

Defence Mechanisms

Most defence rely on mechanisms carried out on the server-side:

- Filter and remove any malicious script in a HTTP request while constructing its response page
- Filter and remove any malicious script in a user's post before it is saved into the forum database

Some example techniques are Script Filtering and Noscript Region. A Noscript Region is basically a region of a web page where JavaScript is not allowed to appear.

However, this is not fool-proof. To additionally detected reflected XSS attacks, some browsers employ a client-side detection mechanism e.g. XSS auditor.

Cross Site Request Forgery (CSRF) Attacks
There are also two types of CSRF attacks:

1. **Victim clicks on a URL**

This is also known as “sea-surf”, cross-site **reference** forgery or session riding. The attack goes as such:

- Suppose a client Alice is already authenticated by a target website such as www.bank.com and this site accepts an authentication token cookie
- The attacker Bob tricks Alice into clicking a URL of this site, which maliciously requests for a service. For example, transfer $1000 to Bob:
    - www.bank.com/transfer?account=Alice&amount=1000&to=Bob
- Alice’s cookie will also be automatically sent to the site, indicating that the request comes from the already-authenticated Alice
- Hence, the transaction will be carried out

2. **Victim doesn’t click on a URL**

A web attacker can also perform a CSRF attack without UI actions from the victim. An example would be:

- Again, suppose Alice is already authenticated by a target website www.bank.com and the site accepts an authentication token cookie
- Alice visits the attacker’s site, whose page contains the following:
    -
- Alice’s browser issues another HTTP request to obtain the image
- Alice’s cookie will also be automatically sent to the website
- Hence, the transaction will be carried out

What is CSRF in short?
It is a type of authorization attack on web apps, whereby a web attacker attacks a web user by issuing a forged request to a vulnerable web server ‘on behalf’ of the victim user.

The attacker disrupts the integrity of the target user’s session. This is, in a way, the reverse of XSS – it exploits the server’s trust of the client. The server believes that the request is from the client.

Defence Mechanisms
This is relatively easier to prevent compared to XSS. The SID/authentication-token cookie that is automatically sent by the browser is deemed as insufficient. The server must issue and require extra information i.e. anti-CSRF token. For example, the server may include a dynamic anti-CSRF token in its money transfer request page.

- The anti-CSRF token can then be included in a URL
    - www.bank.com/transfer?account=Alice&amount=1000&to=Bob&Token=xxk34n890ad7casdf897e324
- It is also possible to include the anti-CSRF token inside a HTTP request header or a hidden form field.

Other Web Attacks and Terminologies
Here is a list:

- Drive-by download
    - Unintended download of computer software from the Internet
        - May be authorised by user but without understanding the consequences e.g. downloads which install some unknown executable automatically
        - Any download that happens without a person’s knowledge, often a virus, spyware, malware or crimeware

    - May happen when visiting a website, opening an email attachment, clicking a link, or clicking on a deceptive pop-up window
- Web beacon / Web bug / Tracking bug / Tag / Page tag
    - A technique used on web pages and emails to unobtrusively, usually invisibly, allow checking that a user has accessed some content.
    - They are typically used by third parties to monitor the activity of users at a website for purposes of web analytics or page tagging.
    - May also be used for email tracking.
- Clickjacking (User Interface Redress Attack)
    - A malicious technique of tricking a user into clicking something different from what the user perceives, thus potentially revealing confidential information or allowing others to take control of their computer while clicking on seemingly innocuous objects, including web pages.
    - It is usually embedded code or script that can execute without the user's knowledge, such as clicking on a button that appears to perform another function
- Click fraud
    - A type of fraud that occurs on the Internet in pay-per-click online advertising. As owners of sites that post these ads receive money based on how many visitors click on the ads, they may use a person, automated script or computer program that imitates a legitimate user of a web browser to click on such an ad without having an actual interest in the target of the ad's link.
    - This may even be in the form of employing low-wage workers to repeatedly click on each AdSense ad on their website, thereby generating money to be paid by the advertiser to the publisher and to Google.
- Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA)
    - A challenge-response test used in computing to determine whether or not the user is human.
    - The most common type requires someone to correctly evaluate and enter a sequence of letters or numbers perceptible in a distorted image displayed on their screen.
    - Because the test is administered by a computer, in contrast to a standard Turing test that is administered by a human, a CAPTCHA is sometimes described as a reverse Turing test.