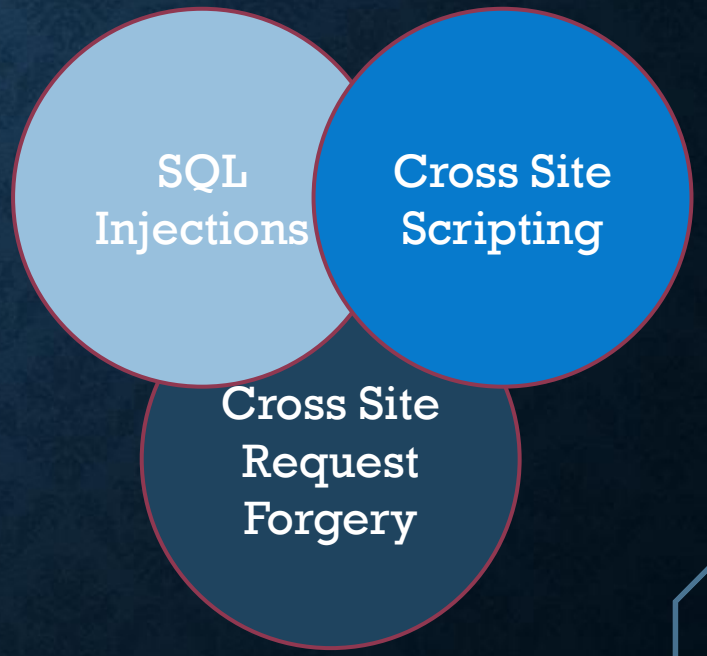
SECURITY

SQL Injections, Cross Site Scripting, Request Forgery, and Cookie Stealing

COS216
AVINASH SINGH
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF PRETORIA

WEBSITE ATTACKS

- Websites can be "hacked" in many different ways
- There are 3 groups of attacks that are commonly used, since most novice web developers forget about them



SQL INJECTIONS

- SQL injections is caused when SQL queries are sent to your sever from the client side by a malicious user
  - If you have some input fields on your website that sends data to your server, a user might add unexpected code to that input field.

- SQL injections can be triggered by
  - Unescaped special characters (quotes, equal sign, semicolon, etc)
  - Unhandled encoding schemes (ASCII vs various UTF encoding schemes)
  - Various other subtle means

SQL INJECTIONS

- The basic idea to cause the problem
  - Somehow cause a valid SQL query to fail (eg: add unexpected quotes)
  - Add a semicolon after the failed statement (which indicates that the previous statement ends)
  - Add some malicious SQL query after the added semicolon

- Add the malicious code to a HTML input on the website:

Comment: “; DROP DATABASE users; Submit

SQL INJECTIONS

- To solve SQL injections
  1. Remove malicious code via JavaScript (unsafe, JS code can be manipulated in a browser)
  2. Remove malicious code on the server (safer, but requires a lot of manual code, and the programmer might not have removal for all malicious code)
  3. Use prepared statements (safest, existing built-in functionality to remove all known malicious code for you – all malicious code is seen as strings instead of code)
     - Both mysqli and PDO have prepared statements

SQL INJECTIONS

```php
<?php
        // Connect to database like normal

        $statement = $connection->prepare(
                "INSERT INTO student (firstname, lastname) VALUES (?, ?)");
        $statement->bind_param("ss", $firstname, $lastname);

        $firstname = "Jacob";
        $lastname = "Zuma";
        $statement->execute();

        $firstname = "Mmusi";
        $lastname = "Maimane";
        $statement->execute();

        // Disconnect from database like normal
?>
```

SQL INJECTIONS

- The bind_param function takes 2 parameters
  - 1st Parameter: A sequence of data type of the parameters as a string
    - i = integer
    - d = double/float
    - s = string
    - b = blob
  - 2nd Parameter: The parameters to bind
    - Can add any number of parameters here
    - Must be in the same order as the 1st parameter's sequence
    - Are passed as reference to the function, so you can change the values after calling the function
    - One can also bind specific parameters before each execution of the query

SCRIPT INJECTIONS

- Cross Site Scripting (XSS)
- Similar to SQL injections, just with JavaScript (persistant)

- Malicious user enters JavaScript code into an HTML input (eg: comment box)
- The JS code is saved as a string on the server
- Other users who visit the site also see the posted comment which now has embedded hidden malicious JavaScript code
  - The malicious code can be used to do pretty much anything (eg: stealing cookies, making external GET/POST requests, creating annoying popups, etc)

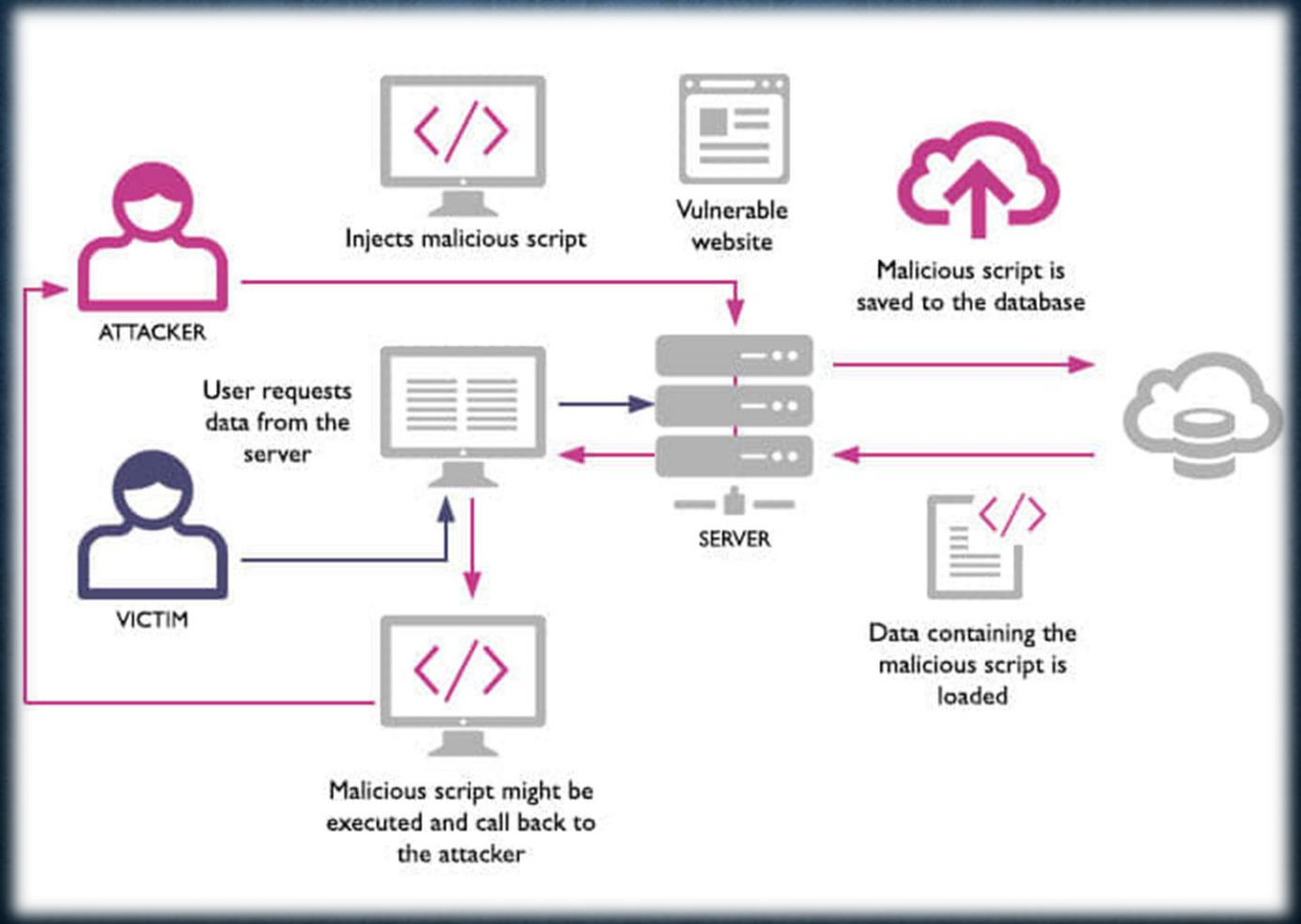
SCRIPT INJECTIONS

- Add the malicious code to a HTML input on the website:

Comment: <script>alert("Hacked!")</script> Submit

- To solve script injections
  1. Remove malicious code via JavaScript (unsafe, JS code can be manipulated in a browser)
  2. Remove malicious code on the server (safe, since only the site admin has access to the server code)
  3. Avoid malicious DOM manipulation in your JavaScript (safe, beyond the scope of this course)

SCRIPT INJECTIONS



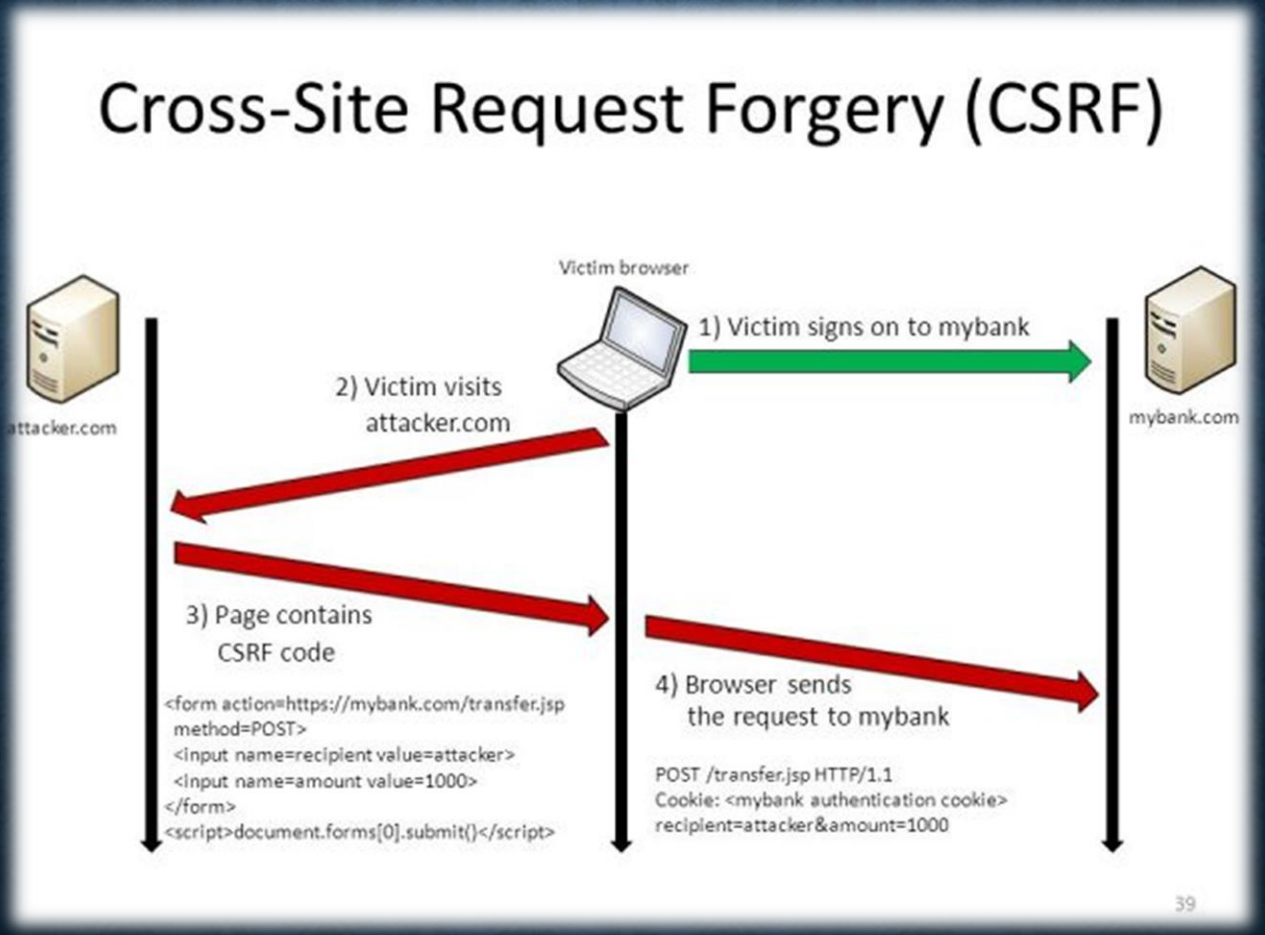
CROSS SITE REQUEST FORGERY

- Cross Site Request Forgery (CSRF)
- User is logged into some website in one of the browser tab
- If the user opens up your malicious site (or alternatively any website that was infected with your XSS code)
  - The malicious code makes a request to the authenticated website loaded in the other tab
  - Since the other tab is already authenticated, the server will simply accept the requests you sent to it

CROSS SITE REQUEST FORGERY

- To solve cross site request forgery
  1. Inspect the HTTP header on the server to see from which website the request originated. If the requesting website is not yours, reject the request (safe, but might annoy end users, epically if your code is callable from other sites, eg: public API)
  2. Use a hidden token for all the forms on your website and check the token on the server (safe, but requires effort to add this to every form)

CROSS SITE REQUEST FORGERY



COOKIE STEALING

COOKIE STEALING

- Some websites use cookies to store data on clients' machines
- Sessions also use cookies to save the session ID on the client
- Cookies can be accessed by any website that is opened in the client browser
  - As long as the website knows the name of the cookie

- You can write a malicious website that accessed the information stored in cookies by other websites
- You can use the session ID in cookies in order to make authenticated requests on that site (this is actually cross site request forgery)

COOKIE STEALING

- To solve cookie stealing
  1. Encrypt the cookie data (safe, but this requires you to contact the server in order to get the key to decrypt the cookie – kind of counter productive)
  2. Don't use cookies or don't store sensitive data in cookies (safe, but sometimes you have to use cookies, eg sessions)
  3. Rather use API keys for authentication (safe, since temporary key is stored in memory, not in cookie/file, however difficult/annoying to implement when using refresh tokens)
  4. Only use cookies for sessions and nothing else, and validate on the server that any incoming data is checked for their host the request originated from (safe, but requires some additional coding)

COOKIE STEALING

- Due to modern browser's Same Origin Policy you cannot access a cookie from another domain
  - Only the domain that created the cookie can later read it
  - Also applies to subdomains: one subdomain cannot access the cookies from another subdomain, even if they are both part of the same main domain

- Some cookie stealing still works
  - Using CSRF
  - Some limited access through iframes and AJAX calls

BROKEN AUTHENTICATION

- This could be due to multiple problems or issues with the way authentication is done.
- Here are the most common ways this happens:
  - URL contains the session ID or adds the session when getting external sources (AJAX)
  - No password encryption or hashing
  - Session IDs might be predictable (e.g. if session IDs are not complex and are sequential)
  - No SSL, not implementing timeouts for sessions

SECURITY MISCONFIGURATION

- Typically this is the most important step when securing your website/server, some of the common misconfigurations include:
  - Running debug mode
  - Enabling Directory listing
  - Running outdated software with vulnerabilities (WordPress, etc)
  - **Not Changing DEFALT keys and passwords**
  - **Revealing Error/Exception information (tracebacks)**

A list of known vulnerabilities: https://www.cvedetails.com/vulnerability-list/

There are other resources too.

SOME VIDEOS

- SQL Injections: https://www.youtube.com/watch?v=_jKylhJtPmI
- Cross Site Scripting: https://www.youtube.com/watch?v=L5l9lSnNMxg
- Request Forgery: https://www.youtube.com/watch?v=vRBihr41JTo
- Cookie Stealing: https://www.youtube.com/watch?v=T1QEs3mdJoc