

---

THE GEORGE WASHINGTON UNIVERSITY

---

WASHINGTON, DC

# Web App Security

CSCI 2541 Database Systems & Team Projects

Wood

# Reminders

## Database Schema:

- Saying it is 3NF doesn't necessarily mean it is 3NF - look closely! Repeating columns were common (for pre-reqs, prior degrees, etc)
- You need to actually define your foreign and primary keys! Your schema will be worth more points in phase 2!

## Engagement Points:

- CS Diversity & Inclusion event - Thursday SEH B1220
- SEAS R&D Showcase - Friday SEH Atrium
- UTA/LA feedback form
- Sophomore focus group (last week)
- Check #engage

# SQL Injection

- The insertion or injection of a SQL query via the input data from the client to the application.
- An injection allows for an attacker to:
  - Read sensitive data
  - Modify database data
  - Execute Admin ops
  - Issue commands to the operating system

# How does it work?

## SQL Injection

*id = request.getParameter("id")*

```
query = "SELECT * FROM accounts WHERE custID='" +  
request.getParameter("id") + "'";
```

**Navigate to:**

```
http://example.com/app/accountView?id='1; Drop table customers;'
```

- The database will execute the select statement first and then drop the table customers from the database.
- This works because SQL will execute commands on the same line as long as there is a semi-colon present.

# How do we fix it

Use mysql's parameterization.

```
query = "SELECT * FROM accounts WHERE custID='%s'";  
mysql.execute(query, [request.getParameter("id")])
```

---

- The mysql library will automatically escape semi-colons and other dangerous tokens to make sure your query is safe.

# Broken Authentication

- Broken Authentication permits attackers to break or work around the authentication methods of an application.
- Broken Authentication allows:
  - Credential stuffing with lists of passwords
  - Brute Force attacks
  - ~~Weak/ineffective Password Recovery~~
  - Plain Text Passwords
  - Lack of session invalidation

# How do we fix it?

## Broken Authentication

- Limit password attempts to 3 times
- Require strong passwords with symbols, numbers and mixed case *length*
- Use strong password recovery options such as multi-factor authentication
- Hash passwords when storing in database
- Invalidate login sessions after 15 minutes

# Sensitive Data Exposure

- Sensitive Data Exposure occurs when an application inadvertently exposes personal information of its users. Essentially data is left unguarded and exposed to attackers.
- Sensitive Data Exposure occurs when:
  - Data and databases are not properly secured to store data.
  - Data in transit is plaintext and not properly protected. *http vs https*
  - Weak cryptographic algorithms



# How do we fix it?

## Sensitive Data Exposure

- Use HTTPS and SSL with your API's. All data in transit should be protected.
- Secure your databases against sequel injects to prevent data from being exposed.
- Properly hash your passwords with a salt when storing in a database.
- Use strong encryption algorithms as recommended by NIST.
- Encrypt your data when storing in a database and in the cloud.
- Rotate your encryption keys often and don't reuse them.

# Broken Access Control

- Access control is how an application grants access to data and functions in an application. Also known as authorization.
- Broken Access control is when a user is able to elevate their permissions or get around an access control mechanism to perform unauthorized functions.
- Its one of the harder vulnerabilities to get right and prevent.
  - It occurs as a result of:
    - Not properly guarding certain endpoints
    - Using access policies that grant wide permissions
    - Path Traversal: Being able to hit a resource directly
    - Client Side Caching
    - Privilege Escalation

# How does it work?

## Broken Access Control

- An attacker modifies the 'acct' parameter in the browser to send any account number. If not properly verified, the attacker can access any user's account.

`http://example.com/app/accountInfo?acct=notmyacct`

- An attacker forces a browser to target URLs directly. If you can access them without logging in then theres a problem.
  - `http://example.com/app/getappInfo`
  - `http://example.com/app/admin_getappInfo`

# How do we fix it?

## Broken Access Control

- Ensure all endpoints are properly guarded by a permission barrier
- Use the concept of least privilege. Users should have the bare minimum permissions to accomplish what they need.
- Make sure resources are guarded by a permission barrier even when directly accessible. A user shouldn't be able to navigate a URL path to reach a file by getting around the permission barrier.
- Make sure to invalidate client side caches when a user logs out, so data is not left exposed.

# Security Misconfiguration

- Improper server or web application configuration that leads to abuse by attackers.
- Potential vulnerabilities include:
  - Leaving debugging logs enabled
  - Using default passwords/accounts
  - Directory listing available allowing attackers to browse and download code
  - Error logs returning excess information that can lead to abuse by attackers
  - Unnecessary features enabled

# How do we fix it?

## Security Misconfiguration

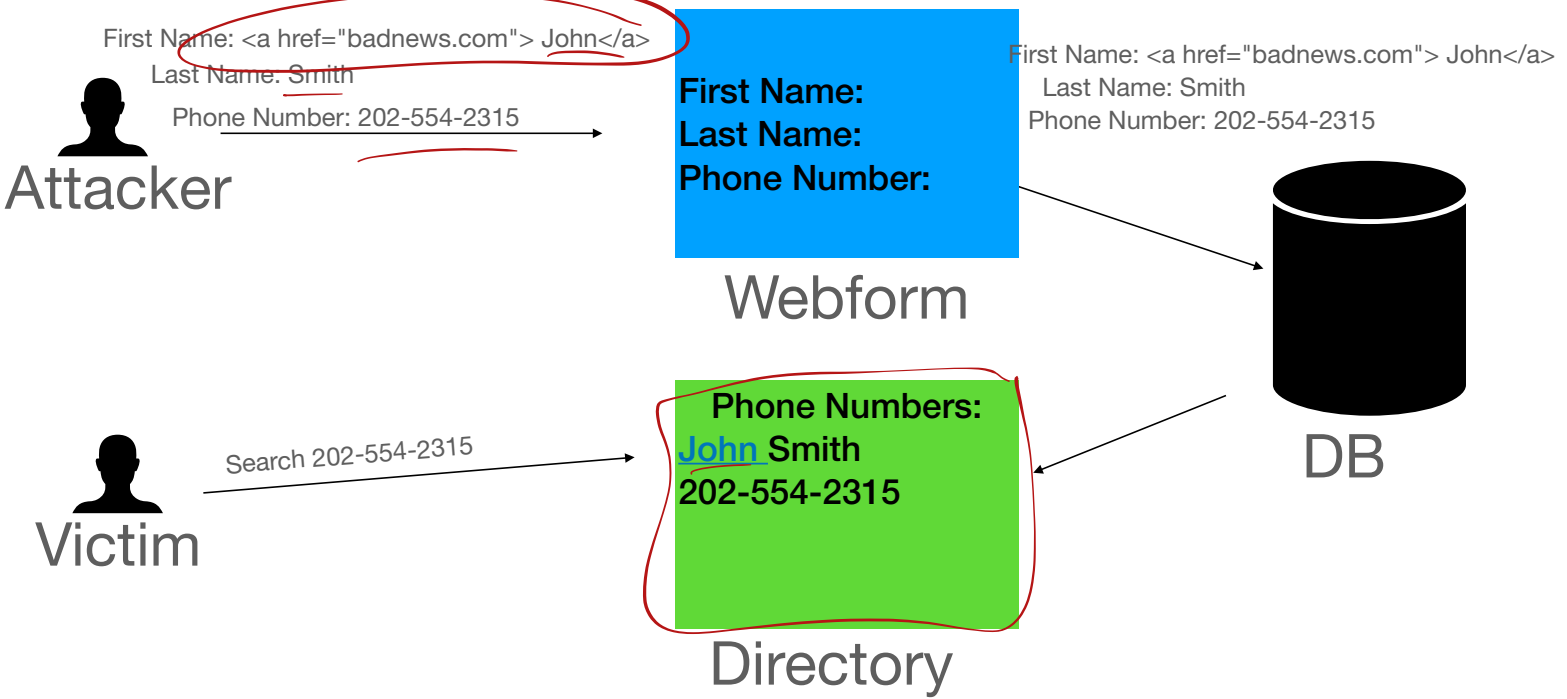
- Enable production logging that logs only the bare necessary information.
- Change default passwords to secure usernames and passwords
- Turn off directory listing so that users can't download code or files to look for attack surfaces.
- Control your logs so that no stacks or information is returned that can be used to find vulnerabilities.
- Disable features on your site that aren't needed to limit attack vectors.
- Use a repeatable hardening process that can be run every time you promote your software to production.

# Cross Site Scripting (XSS)

- When an application includes untrusted data in a new web page without validation or escaping. Affects web pages where users submit data and html/js is generated from the result.
- Allows for users to hijack a clients browser session, steal credentials, redirect users to harmful websites, download untrusted code or run malicious scripts.
- Reflected XSS: The untrusted data presents a link that a user clicks on and redirects them to a harmful website.
- Stored XSS: The untrusted data is stored in the database and presented to users at a later time.
- DOM XSS: JS Frameworks that dynamically include untrusted data and leads to stealing of credentials, session hijacking, and malicious downloads

# How does it work?

## Cross Site Scripting (XSS)





# How do we fix it?

## Cross Site Scripting (XSS)

WTFORM

- Use frameworks that automatically escape XSS of data input. Make sure you know the limitations of each so you can cover the rest.
- Escape untrusted html request data to prevent reflected attacks.
- Bottom line: Escape and sanitize any input you receive from ~~users~~ before presenting it in the browser or storing it in the database.

Log4j - library

## Using Components with Known Vulnerabilities

- You can build the most secure castle in the world but if the bricks are vulnerable so are you.
- Libraries are commonly used to make code reproducible and accomplish common functionality instead of reinventing the wheel.
- Occasionally new vulnerabilities are found in these libraries that require remediation. If you don't stay on top of these your code becomes vulnerable to attack even if the code you wrote yourself is secure.
- This can lead to application compromise and make all your hard work invalidated.

# How do we fix it?

## Using Components with Known Vulnerabilities

- Stay up to date with the latest Common Vulnerabilities and Exposures (CVE) reports.
- Remove unused dependencies so as to limit attack vector surfaces.
- Use a vulnerability scanner to continuously monitor your dependencies and check for vulnerabilities.
- Only use libraries from official sources so as not to rely on potentially untrusted code and to receive the latest versions.
- Stay up to date with libraries and move to those that are currently maintained.

# Insufficient Logging & Monitoring

- Studies show the time to detect a breach is over 200 days and often detected by a third party.
- Insufficient logging and monitoring allows attackers to scour the kingdom undetected and cause steady damage and data leakage in an application.
- Sources of insufficient logging & monitoring
  - Auditable events such as logins, admin actions, and failed actions are not being logged
  - Logs only stored locally
  - Applications unable to detect, escalate, and alert for suspicious activity in real time.
  - Application logging is visible to users

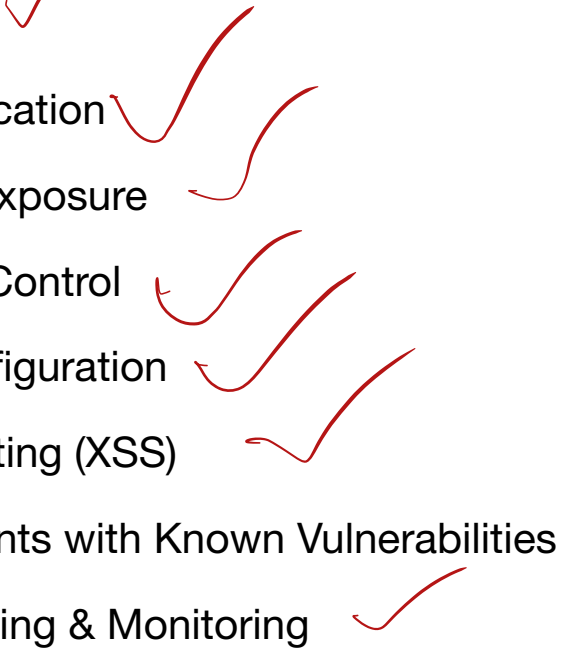
# How do we fix it?

## Insufficient Logging & Monitoring

- Ensure events that are auditable are logged. Log all events that are admin related or suspicious (failures) with appropriate context to auto detect failures.
- Ensure logs are in a format that can be consumed from a log aggregation tool.
- Ensure there are audit trails for high value operations so that attacks can be discovered in real time and escalated.
- Configure alerting and monitoring tools to alert on suspicious events.
- Ensure logs are stored for a long enough duration that can be studied post breach.

# Summary

## OWASP Top 10

- SQL Injection
  - Broken Authentication
  - Sensitive Data Exposure
  - Broken Access Control
  - Security Misconfiguration
  - Cross Site Scripting (XSS)
  - Using Components with Known Vulnerabilities
  - Insufficient Logging & Monitoring
- 

# Securing your app

Don't store passwords as plain text

- Solution: store **hash (password + salt)**
- Ideally **hash()** should be slow to compute - why?

hash  
xbae  
com

Don't store your DB credentials in your git repo

- Do you trust GitHub with keys to your bank account or company secrets?
- Solution: use a separate **config file not added to the repo**
- See: <https://github.com/cs2541-22s/flask-sample-mysql>

Check permissions in any endpoints where the user controls input parameters

- Check or avoid URLs like `/transcripts/12345`

Limit access to your DB server

- **AWS Security groups** provide a firewall to control who can access
- (hard to do in practice if you are running flask on your own laptop)