# CIS3165 – WEB SECURITY

BY PELIN

# 1. THREATS

# CROSS SITE SCRIPTING

XSS for short

Hacker can inject JavaScript into a web page

Used to trick users into running JavaScript code

Used to steal cookies

# CROSS SITE SCRIPTING - STORED CROSS-SITE SCRIPTING ATTACKS

Attackers will attempt to inject JavaScript code into the database

web server will write out the JavaScript when it renders HTML.

Malicious JavaScript can be planted in a database by using:

the legitimate methods to store information in the databases (forms)

SQL injection

# XSS - MITIGATIONS

**Escaping HTML Characters:** prevent all dynamic content coming from a datastore so that the browser knows to treat it as the content of HTML tag

| Character | Entity encoding |
|---|---|
| " | &quot; |
| & | &amp; |
| ' | &apos; |
| < | &lt; |
| > | &gt; |

# XSS - MITIGATIONS

**Implement a Content Security Policy**

Modern browsers allow websites to set a content security policy, which you can use to lock down JavaScript execution on your site.

content security policy in your HTTP response headers,

tell the browser to never execute inline JavaScript.

The browser will execute JavaScript on your page only if it is imported via a **src** attribute in the <script> tag

# REFLECTED CROSS-SITE SCRIPTING ATTACKS

If your site takes part of an HTTP request and displays it back in a rendered web page

your rendering code needs to protect against attacks that inject malicious JavaScript via the HTTP request.

***https://www.google.com/search?q=rwanda***.

# REFLECTED CROSS-SITE SCRIPTING - MITIGATION

**Escape Dynamic Content from HTTP Requests**

escaping control characters in dynamic content that the website interpolates into HTML page

# 2. CROSS SITE REQUEST FORGERY

- CSRF for short
- Hackers trick users into making requests to your server
- Can be used for fraudulent clicks
- Can take advantage of user's logged in state

```
Online vote: "Who is the smartest hacker?"


https://fun-poll.com/vote?hacker=5674


<img src="https://fun-poll.com/vote?hacker=5674" />
```

```
// You log into your online banking account

// You close browser window, but don't log out


<img src="https://bank.com/transfer?amount=1000&to=987654321" />
```

# CSRF SOLUTION

GET Requests should be idempotent

Idempotent: a get request (URL) should always return the same value when called repeatedly

Only Use POST requests for making changes

Store a **form token** in user's session

Add a hidden form field with form token as value

Compare session form token and submitted form token

Store the token generation time in user's session

Check if too much time has passed

# 4. SQL INJECTION

attacks target websites that use an underlying SQL database and construct data queries to the database in an insecure fashion

Hacker is able to execute arbitrary SQL requests

Can be used to:

- probe database schema
- steal database data (username, password, credit cards,..)
- Add or change database data (assign elevated privileges, place orders)
- Destroy databases

# ANATOMY OF SQL INJECTION ATTACK

*Poorly constructed query*

```
Connection connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);

Statement statement = connection.createStatement();

String sql = "SELECT * FROM users WHERE email="' + email + '" AND encrypted_password="' + password + '"";

 statement.executeQuery(sql);
```

# SQL INJECTION EXAMPLE

```
SELECT * FROM users
WHERE username = '${username}' AND password = '${password}';

username = "jsmith"
password = "secret"

SELECT * FROM users
WHERE username = 'jsmith' AND password = 'secret';
```

```
SELECT * FROM users
WHERE username = '${username}' AND password = '${password}';

username = "jsmith' OR 1 = 1; --"
password = "blank"

SELECT * FROM users
WHERE username = 'jsmith' OR 1 = 1; --' AND password = 'blank';
```

```
SELECT * FROM articles WHERE title = '${query}';

query = "q' OR 1=(SELECT COUNT(*) FROM tblAdminLogins); --"

query = "q' UNION SELECT username, password FROM users; --"

query = "q'; DROP TABLE customers; --"

query = "q'; UPDATE users SET admin=1 WHERE username='hacker'; --"

query = "q'; INSERT INTO orders (customer_id, product_id,
amount_paid) VALUES (2906, 567, 995.0); --"
```

# SQL INJECTION MITIGATION

**Mitigation 1. Use Prepared Statements**:

```
SET @sql = "SELECT * FROM articles WHERE title = ?";

PREPARE stmt FROM @sql;

EXECUTE stmt USING @query;
```

With parameterized statements, the **control characters** used in SQL injection will be prefixed with **escaped characters** when binding the parameters

# SQL INJECTION MITIGATION

**Mitigation 2: Use Object-Relational Mapper (ORM)**

ORMs use bind parameters under the hood, they protect against injection attacks in most cases

**User.objects.filter(email="billy@gmail.com")** (Django ORM)

Most ORMs have features of using raw sql statements that can be easily attacked

When using raw sql in ORMs Mitigation 1 (parameterized statements) can be a solution

# SQL INJECTION MITIGATION

**Other Mitigations:**

Give limited privileges to application's database user.

Sanitize input

Escape for SQL

**Blind and Nonblind SQL Injection**

- Blind: non descriptive error messages when running a SQL related command
- Nonblind: descriptive feedback messages when running a SQL related command

# 4. URL MANIPULATION

- Editing the URL string to probe the site
- It can be used to reveal private information
- It can be used to perform restricted actions

```
http://yoursite.com?invoice=A-17391

http://yoursite.com/authorize?UserID=9876543210

http://yoursite.com?SESSIONID=AG8B3190CFAF48231A55E

http://yoursite.com/products?preview=false

http://yoursite.com/images/small/rockymtns.jpg

http://yoursite.com/reports/export/sales
```

# URL MANIPULATION SOLUTIONS

Realize that URL are exposed and editable
- Pages can be accessible via "deep links"

Don't use obscurity for access control

Keep error messages vague

Clarify POST and GET requests
- GET requests should be Idempotent (no changes)
- POST requests should be used to make changes

# 5. FAKED REQUESTS AND FORMS

Request header information can be manufactured

"faked" or "spooked"

plugins and tools for all major browsers

Advanced tools for command line

can be part of the script

```
GET /somepath HTTP/1.1
Host: somesite.com
Referer: www.spoofy.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:
25.0) Gecko/20100101 Firefox/25.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Cookie: PHPSESSID=3e3531fa4ba0fc482b398e703f044cb8;user_id=4983
```

# FAKED FORM

HTML Forms can be duplicated

can be used to add or remove form values

Can be used to edit hidden field values

Can be used to remove Client-side Validations

```
<form action="/add_comment" method="post">
  <input name="username" type="hidden" value="pjones" />
  <input type="text" name="subject" maxlength="100" />
  <textarea name="comment" onkeyup="limit(this,200);">
  </textarea>
</form>

<script type="text/javascript">
  function limit(field, maxLength) {
    if (field.value.length > maxLength) {
      field.value = field.value.substring(0, maxLength);
    }
  }
</script>
```

# FAKED FORM SOLUTIONS

Do not rely for form structure for validation

Do not rely on client-side validations

Use HTTP Referrer to enforce same-domain forms

Use CSRF protections (token, stamp)

# 6. COOKIE VISIBILITY AND THEFT

Cookie data is visible to users

Cookie data can be stolen using XSS attacks

Cookies can be sniffed by observing network traffic (using tools like Wireshark)

Write software tests for common security concerns

# COOKIE SOLUTIONS

Only put non sensitive information in cookies (example: preferences)

Use HttpOnly cookies

Use secure cookies (applicable to HTTPs only)

Set expiration date

set cookie domain and path

Encrypt cookie data

Use server-side sessions instead of client-side cookies

# 7. SESSION HIJACKING

Similar to cookie theft but much more valuable

Can be used to assume your identity and logged in status

Can be used to steal personal info, change password

Often done by network eavesdropping

**Solutions:**

# SESSION HIJACKING SOLUTIONS

Save user agent in session and confirm it.

Check IP address

Use HttpOnly cookies

Regenerate session identifier periodically, at key points.
- Important to regenerate after login

Expire/remove old session files regularly
- Keep track of last activity in session

Use SSL

Use Secure cookies

# 8. SESSION FIXATION

Trick users into using a hacker provided session identifier

Can be used to assume your identity and logged in status

Can be used to steal personal info, change password

Successful if user authenticates a known session identifier

```
GET /login HTTP/1.1
Host: yourbank.com
Cookie: SESSION_ID=3e3531fa4ba0fc482b398e703f044cb8


http://yourbank.com/login?SESSION_ID=a1b2c3d4e5a1b2c3d4e5

GET /login HTTP/1.1
Host: yourbank.com
Cookie: SESSION_ID=a1b2c3d4e5a1b2c3d4e5
```

# SESSION FIXATION SOLUTIONS

Do not accept session identifiers from GET or POST variables

Regenerate session identifier periodically, at key points.

- Important to regenerate after login

Expire/remove old session files regularly

- Keep track of last activity in session

# 9. REMOTE SYSTEM EXECUTION

Remotely run operating system commands on a webserver

Can be used to do **anything** on your operating system can do

Most powerful hacks

Typically hardest to achieve

Most programming languages do not allow casual access to the underlying operating system

Most programming language offer special commands which can access the underlying operating system

```
system
exec
shell
sh
shell_exec
open
popen
proc_open
```

```
call
subprocess
spawn
passthru
eval
%x
`
```

# REMOTE SYSTEM EXECUTION SOLUTIONS

Avoid system execution keywords

Perform system execution which extreme caution

Sanitize any dynamic data carefully

Understand the commands and their syntax completely

Add additional data validations

# FILE UPLOAD ABUSE

Abuse of allowed file upload features

Can be used to upload too much (Quantity, file size)

Can be used to upload malicious files

# FILE UPLOAD ABUSE SOLUTIONS

Require user authentication, no anonymous uploads

Limit maximum upload size

Limit allowed file formats and file extensions

Use caution when opening uploaded files

Do not host uploaded files which have not been verified

# 11. DENIAL OF SERVICE

"DoS" for short

Attempt to make a server unavailable to users

Usually performed by overloading a server with requests

Includes DNS and routing disruption

Includes using up disk space, processor power, bandwidth

Attacks often performed by distributed network ("DDoS")

Cheap to launch, difficult to prevent

Usually performed by person or group with a grudge

Can be used as a distraction from other hacking attemps

# DOS SOLUTIONS

Firewalls

Switches and routes

Load management hardware / software

Collection of reverse proxies

Map your infrastructure

Keep infrastructure up to date

Purchase high-quality hosting and equipment

Make network traffic visible

Develop a response plan

Change IP address

Black hole or null route traffic

# SUMMARY

Do (Thorough) Code Reviews

Test Your Code

Anticipate Malicious Input

Neutralize File Uploads

Escape Content While Writing HTML

Be Suspicious of HTTP Requests from Other Sites

Don’t Admit Who Your Users Are

# SUMMARY

Protect Your Cookies

Protect Sensitive Resources

Detect and Be Ready for Surges in Traffic

# THANK YOU