



# SANS/CWE Top 25 Programming Errors

**OWASP**  
April 2, 2009

**Michael S. Menefee, CISSP**  
**North Carolina Chapter Leader**  
**WireHead Security, LLC**  
mmenefee@gmail.com  
(919) 271-8883

**Fred W. Williams Jr.**  
**SAS Institute**  
[fredwi@yahoo.com](mailto:fredwi@yahoo.com)  
(919) 793-4675

Copyright © The OWASP Foundation  
Permission is granted to copy, distribute and/or modify this document  
under the terms of the OWASP License.

**The OWASP Foundation**  
<http://www.owasp.org>

# Agenda

- Introductions
- Quick Re-cap from last week
- Discussions of errors #10 - #18
- What is the OWASP Top 10 List and how does it relate to the SANS / CWE Top 25 List?
- Open Discussion/Questions

# Welcome, Introductions

## ■ Presenters

- ▶ Michael S. Menefee, CISSP
  - Principal Consultant
  - WireHead Security, LLC
  - [mmenefee@gmail.com](mailto:mmenefee@gmail.com)
  - (919) 271-8883
- ▶ Fred W. Williams Jr.
  - Software Developer – Team Lead Publication Services
  - SAS Institute
  - [fredwi@yahoo.com](mailto:fredwi@yahoo.com)
  - (919) 793-4675

# Common themes to remember

- SANS/CWE categorized each error into 3 areas: prevalence, remedy cost, and frequency of occurrence.
- Several errors are factors within other errors.  
More bang for your buck.
- Give users just enough to do their job.
- Use encryption whenever possible. SSL.
- Testing is important. FindBugs, CodePro.
- Keep up! Podcasts, News services.



# CWE Top 25 Brief Overview

## Insecure Interaction Between Components

These weaknesses are related to insecure ways in which data is sent and received between separate components, modules, programs, processes, threads, or systems.

- [CWE-20](#): Improper Input Validation \*
- [CWE-116](#): Improper Encoding or Escaping of Output \*
- [CWE-89](#): Failure to Preserve SQL Query Structure (aka 'SQL Injection')
- [CWE-79](#): Failure to Preserve Web Page Structure (aka 'Cross-site Scripting')
- [CWE-78](#): Failure to Preserve OS Command Structure (aka 'OS Command Injection')
- [CWE-319](#): Cleartext Transmission of Sensitive Information
- [CWE-352](#): Cross-Site Request Forgery (CSRF)
- [CWE-362](#): Race Condition
- [CWE-209](#): Error Message Information Leak

# CWE Top 25 Brief Overview, Continued

## Risky Resource Management

The weaknesses in this category are related to ways in which software does not properly manage the creation, usage, transfer, or destruction of important system resources.

- [CWE-119](#): Failure to Constrain Operations within the Bounds of a Memory Buffer
- [CWE-642](#): External Control of Critical State Data
- [CWE-73](#): External Control of File Name or Path
- [CWE-426](#): Untrusted Search Path
- [CWE-94](#): Failure to Control Generation of Code (aka 'Code Injection')
- [CWE-494](#): Download of Code Without Integrity Check
- [CWE-404](#): Improper Resource Shutdown or Release
- [CWE-665](#): Improper Initialization
- [CWE-682](#): Incorrect Calculation

# CWE Top 25 Risky Resource Mgmt #10 - 18

## ■ Error #10: CWE 119 Buffer Overflow

- ▶ **Prevalence:** High, **Remedy:** Low, **Frequency:** Often
- ▶ Buffer overflows are when software permits read or write operations on memory outside of allocated range.
- ▶ Attackers can insert malicious code or make the program operate in an unintended way
- ▶ Buffer overflows were a big problem ~ 10 years ago and been remarkable resistant to elimination. Today we have BO variants.
- ▶ Java susceptible to BO? Possible but not likely.
  - Java by default checks the buffer or input sizes
  - However, Java is used for connecting web systems written in any type of languages so it potentially could pass overflows.

# CWE 10 – Buffer Overflow

Try to avoid using C / C++ for new development.

```
void hostlookup(char *user_data)
char hostname[64];
strcpy(user_data , hostname)
```

What if user data had a length > 64 bytes?

Using CodePro Analytix <http://www.instantiations.com/codepro/analytix/about.html>

Audits code and produces reports on violations....

SQLInjectionTestExampleCode.java

```
22 public class SQLInjectionTestExampleCode {  
23  
24     public void nameLookup(ServletRequest req,  
25             int userId) throws SQLException {  
26  
27         ServletRequest servletRequest;  
28         Connection connection;  
29         Statement statement;  
30  
31         servletRequest = req;  
32         connection = DriverManager.getConnection("www.example.com", "myUserName", "myPassword");  
33         statement = connection.createStatement();  
34  
35         String firstName = servletRequest.getParameter("firstName");  
36         String query = "SELECT * FROM user_data WHERE firstName = '" + firstName + "'";  
37         statement.executeQuery(query);  
38     }  
39 }
```

Audit (SQLInjectionTestExampleCode.java at 10/14/08 9:52 AM) -- High: 1 Medium: 0 Low: 0

SQL Injection (Line 37)

Description

SQL Injection

Explanation

The data path below was found as a potential SQL Injection scenario. The top of the path shows where the potentially harmful user data can enter SQL queries. The bottom location in the path shows where the use

at com.instantiations.testing.audit.security.SQLInjectionTestExampleCode.exampleCode(SQLInjectionTestExampleCode.java:37)

Recommendation

1. Linear data should never directly be sent into SQL queries. Either the path should be eliminated or extra

# CWE Top 25 Risky Resource Mgmt #10 - 18

## ■ Error #11: CWE 642 External Control of Critical State Data

- ▶ **Prevalence:** High, **Remedy:** Medium, **Frequency:** Often
- ▶ Attackers can snoop in unsecure data stores: cookies, hidden form fields, configuration files, registry keys.
- ▶ Solution: Encrypt data before storing
- ▶ Solution: Store state information on the server side.
- ▶ You can use frameworks such as Spring Security (formerly Acegi).
- ▶ Examine your settings for cookie management.

# CWE Top 25 Risky Resource Mgmt #10 - 18

## ■ Error #12: CWE 73 External Control of File name or path

- ▶ **Prevalence:** High, **Remedy:** Medium, **Frequency:** Often
- ▶ When using outside inputs to construct file names, an attacker can use combinations of "../" to navigate outside of intended directories.
- ▶ Problem when attackers modify or delete files outside of intended folder structure. Or provides a location of a trojan.
- ▶ Most effective solution: Goes back to proper input validation.
- ▶ Combined with: Practicing least privileges for users
- ▶ Solution: Run in a jailed environment



# CWE 73 – External control of file name

```
String fName = request.getParameter("fileName");
File newFile = new File ("/usr/local/apr/reports" + fName);
...
newFile.delete();
```

What if user supplies:  
“`../../tomcat/conf/server.xml?`

# CWE Top 25 Risky Resource Mgmt #10 - 18

## ■ Error #13: CWE 426 Untrusted Search path

- ▶ **Prevalence:** Low, **Remedy:** Medium, **Frequency:** Rarely
- ▶ Applications rely on a collection of configuration files, code libraries, internationalization property files, etc. If an attacker can gain access and modify the file path used by the application to access these files, then the attacker can point the file path to his own resources.
- ▶ Similar in theory to #12 CWE 73.
- ▶ Solution: Again, goes back to proper input validation.
- ▶ Solution: Use fully qualified path names and reduce reliance on system variables such as \$PATH that attackers can easily modify.

# CWE Top 25 Risky Resource Mgmt #10 - 18

## ■ Error #14: CWE 94 Code Injection

- ▶ **Prevalence:** Medium, **Remedy:** High, **Frequency:** Sometimes
- ▶ Occurs when software allows inputs that contain programming code that alters intended control of software.
- ▶ Leads to arbitrary code execution.
- ▶ Solution: Again, proper input validation and proper encoding.
- ▶ Suffers from same flaws as SQL injection and XSS.
- ▶ Run in a jailed environment that protects between processes and operating systems.
- ▶ CodePro has an specific audit rule to look for Code injection



# CWE 94 Code Injection

Using the Java “ScriptEngine” class that allows for script executions...

```
String script = request.getParameter("input");
ScriptEngine scriptEngine = ...;
...
scriptEngine.eval(script);
```

What if attacker provides scripting code?

# CWE Top 25 Risky Resource Mgmt #10 - 18

## ■ Error #15: CWE 494 Download of code without integrity check

- ▶ **Prevalence:** Medium, **Remedy:** Med - High, **Frequency:** Rarely
- ▶ Revolves around users downloading and executing code from untrusted and unverified sources.
- ▶ Solution: Digitally sign your downloads to provide authentication
- ▶ Examples follow:

# CWE 494 Download of code without integrity check

Using the Java “URL” class...

```
URL[] new = new URL[]{  
    new URL("file: subdir/");  
}
```

What if attacker substitutes a malicious file?

# CWE Top 25 Risky Resource Mgmt #10 - 18

## ■ Error #16: CWE 404 – Improper resource release or shutdown

- ▶ **Prevalence:** Medium, **Remedy:** Medium, **Frequency:** Rarely
- ▶ When application resources have reached end of life – memory, files, sessions, database connections, cookies– dispose of them properly or attackers can misuse.
- ▶ Problem: database connection pools can become exhausted, attackers can snoop sensitive data from old files, DDoS attacks can be launched.
- ▶ Problem: employees leave the company but still have their old accounts active.
- ▶ Solution: use languages that auto garbage collect.
- ▶ Solution: clean up unneeded cookie data.



# CWE 404 – Improper resource release or shutdown

## Close Database connections:

```
Connection conn = null;  
try {  
    conn = getConnection(datasource);  
} catch (Exception e) {  
    ....  
} finally {  
    conn.close();  
}
```

# CWE Top 25 Risky Resource Mgmt #10 - 18

## ■ Error #17: CWE 665 – Improper initialization

- ▶ **Prevalence:** Medium, **Remedy:** Low, **Frequency:** Sometimes
- ▶ Software requires a proper starting point in order to function properly. Variables and other resources such as session data, needs to proper start up values.
- ▶ Problem: especially important, reason it made this list, when authentication comes into play.
- ▶ Solution: use languages that flag no initialization variables like Java.
  - **Compiler will flag this:** String authenticationCode; **Should be:** String authenticationCode = "";
- ▶ Solution: Proper input validation can mitigate – especially if applied towards external input to uninitialized variables.



# CWE Top 25 Risky Resource Mgmt #10 - 18

## ■ Error #18: CWE 682 – Incorrect calculations

- ▶ **Prevalence:** High, **Remedy:** Low, **Frequency:** Often
- ▶ Problem: Software performs calculations that generate incorrect or unintended results that compromise security.
- ▶ Problem: Can produce DDoS attacks if an incorrect calculation causes a system crash.
- ▶ Solution: Our favorite input validation especially on numeric values to ensure within range.
- ▶ Java suffers from floating point errors especially in monetary calculations.



# CWE 682 – Incorrect calculations

Divide by 0 errors:

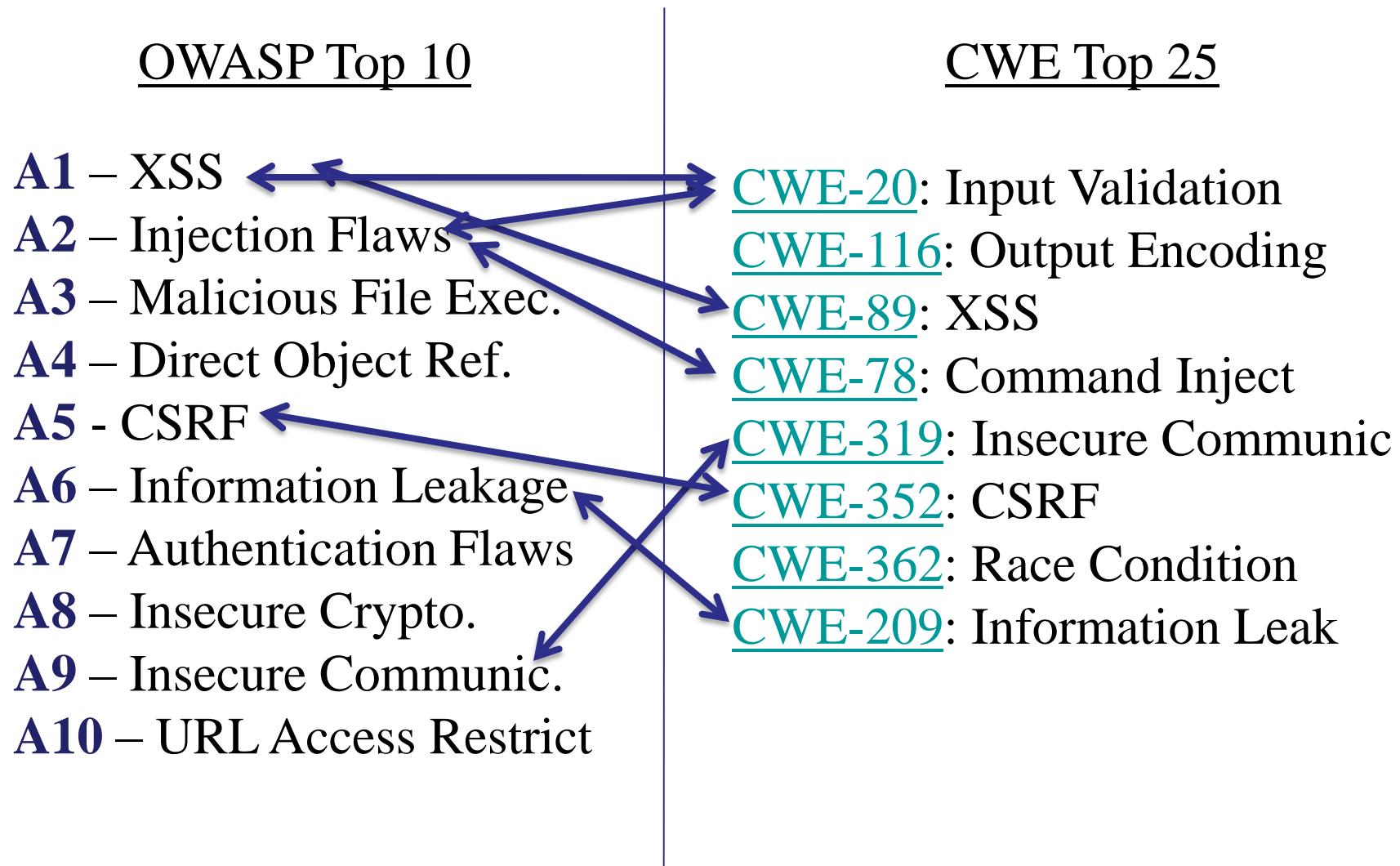
```
int touchdowns = request.getParameter("tds");
int yardage= request.getParameter("yards");
System.out.println("Average is: " + yardage / touchdowns);
```

What if a team has not scored a touchdown or attacker injects a 0?  
Could lead to a loss of availability if software does not properly handle such a condition.

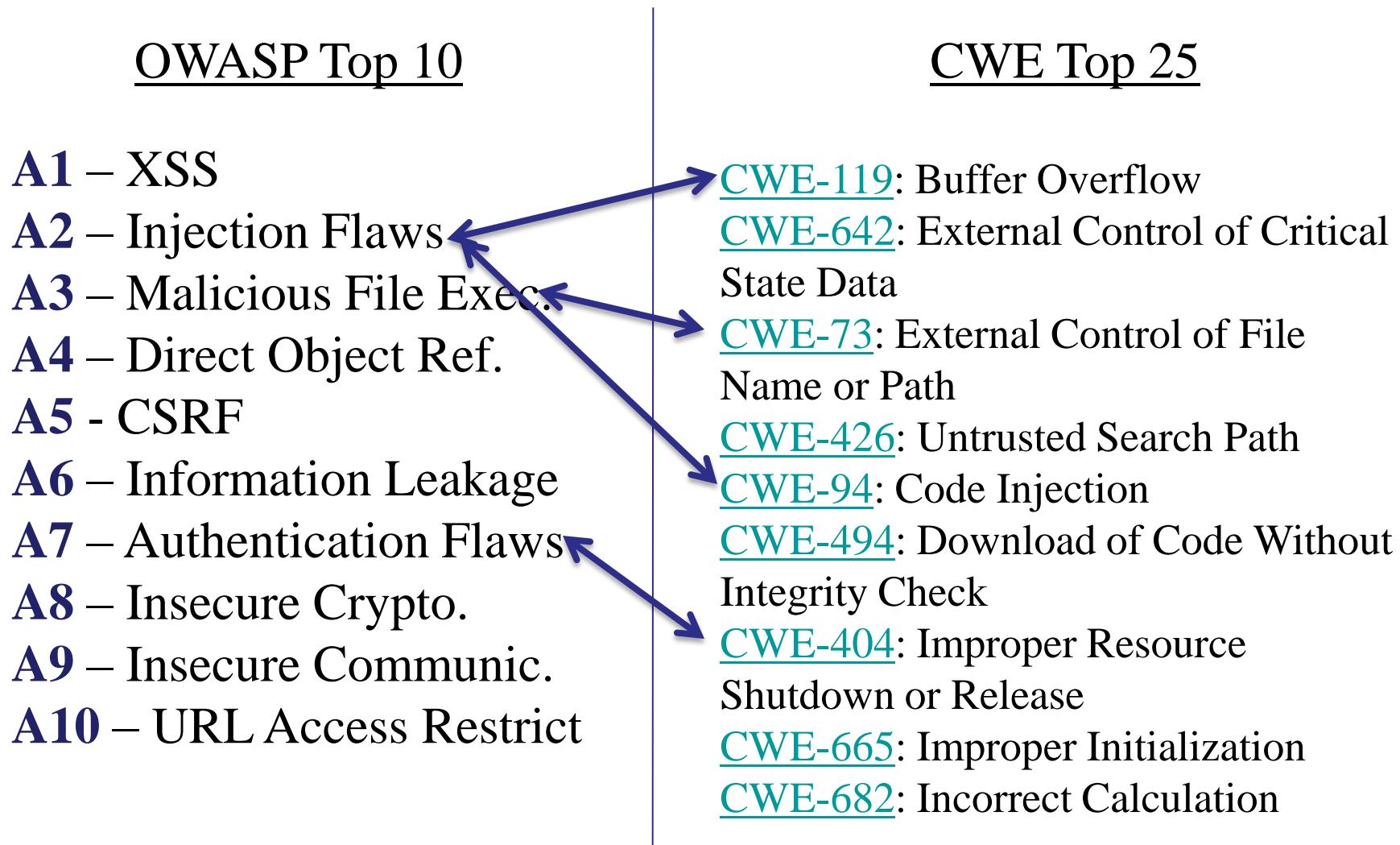
# Current OWASP Top 10 List (2007)

- A1- Cross-Site Scripting (XSS)\*
- A2- Injection Flaws\*
- A3- Malicious File Execution
- A4- Insecure Direct Object Reference
- A5- Cross-Site Request Forgery
- A6- Information Leakage & Improper Error Handling\*
- A7- Broken Authentication & Session Management\*
- A8- Insecure Cryptographic Storage\*
- A9- Insecure Communications\*
- A10- Failure to Restrict URL Access

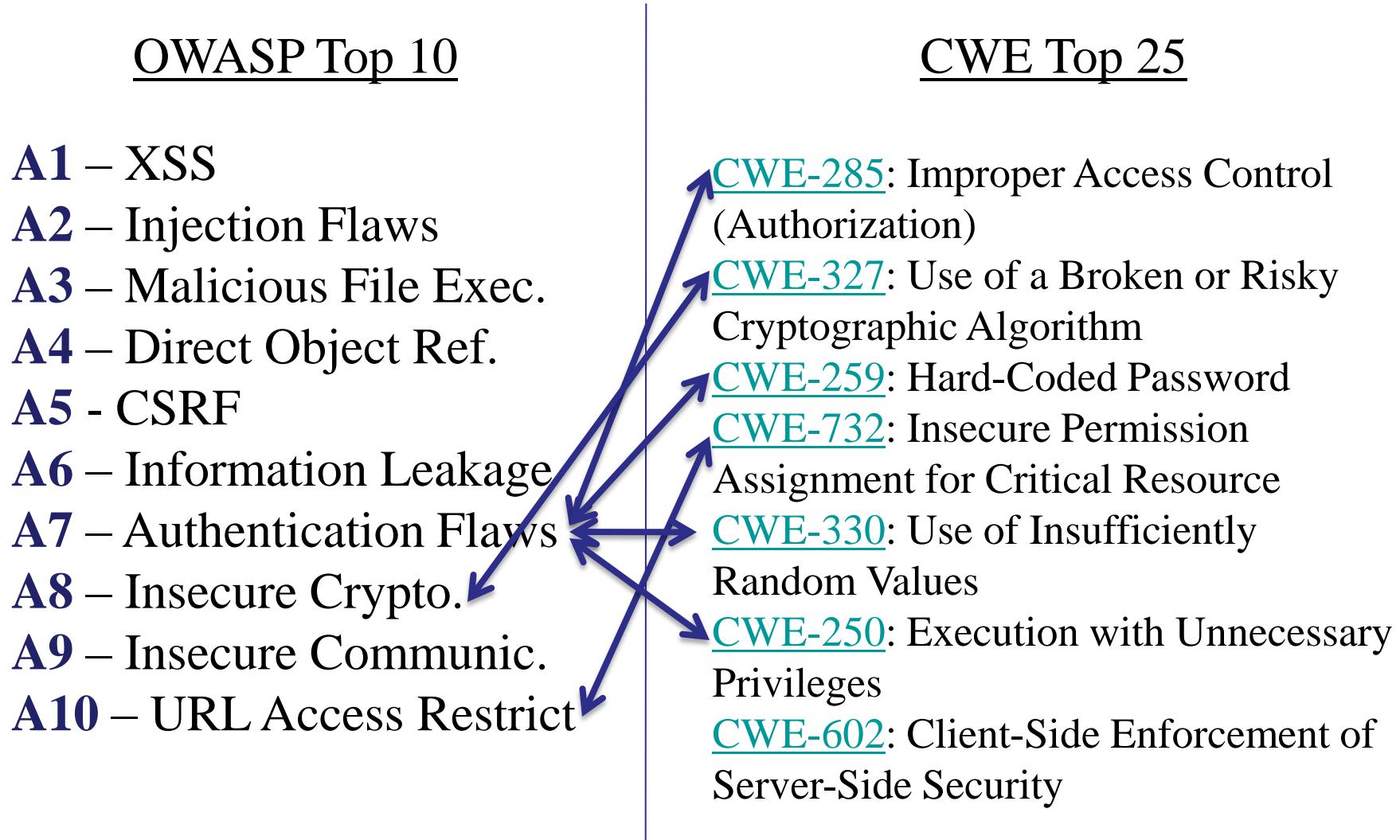
# OWASP Top 10 to CWE Top 25 Mapping



# OWASP Top 10 to CWE Top 25 Mapping



# OWASP Top 10 to CWE Top 25 Mapping



# Open Discussion/Questions?