Group Members:
Omar Salih - 314.637.1110 or 314.380.0123 (Google Hangouts Call / Text), omarmsalih@gmail.com
Will Smith, 314-566-0334 whs2p8@umsl.edu; yokoref@gmail.com (Google Hangouts Call / Text)
Katrina Clark, 314-324-1954, kckatmel1@gmail.com (Google Hangouts, Call/Text)
Rebecca Craine, 636-232-6021 text preferred, sabrieldjc@gmail.com for hangouts, school email rchw5@mail.umsl.edu preferrd email
Nicole Gaehle 314-630-1591 tex preferred; nmgb79@mail.umsl.edu; ngaehle@gmail.com for hangout only

**Tasks & Percentage for the Group Project And Guidelines**

TASKS POINTS/PERCENTAGE ASSIGNED

Find 10 CWEs that would be useful for exploiting. List the CWEs out. [PowerPoint] **25%**

Provide three sentences on at least 5 CWEs on how to exploit or reverse engineer. How would one take advantage of this weakness? [PowerPoint] **25%**

Use at least 2 tools at a minimum to find 7 weakness, bugs, or vulnerabilities in software. Try to map these vulnerabilities found to CWEs. Perform a simple mapping. In the notes section be to sure to write at least 200 word analysis on this weakness, bug, or vulnerability[PowerPoint - Screenshot] **25%**

Provide all raw reports of scans separately. **25%**

Project

1. Find a vulnerability,
2. Take it or the source code and run a vulnerability test on it

CWEs Can be found on:
http://cwe.mitre.org/data/index.html

Program to Run CWE testing on : Java

Each member will choose 2 CWEs
Out of the 2 CWEs each member will choose the weakest and write the 3 sentences.
Leading to 5 weaknesses

CWE VIEW: Weaknesses in Software Written in Java
https://cwe.mitre.org/data/definitions/660.html

**Tools (on Linux use)**

Bug Finder http://findbugs.sourceforge.net/

Should be able to do sudo apt-get install findbugs

Download the derbyclient.jar that is in the google docs

To have the gui run for findbugs it is findbugs -gui

Java Decompiler http://jd.benow.ca

**JAVA written Software to run vulnerabilities:**

**CWEs**

Nikki

Use of NullPointerException Catch to Detect NULL Pointer Dereference - *(395)*

*https://www.owasp.org/index.php/Catch_NullPointerException*

Description: Catching NullPointerException should not be used as an alternative to programmatic checks to prevent dereferencing a null pointer.

Extended Description:

Programmers typically catch NullPointerException under three circumstances:

- The program contains a null pointer dereference. Catching the resulting exception was easier than fixing the underlying problem.
- The program explicitly throws a NullPointerException to signal an error condition.
- The code is part of a test harness that supplies unexpected input to the classes under test.

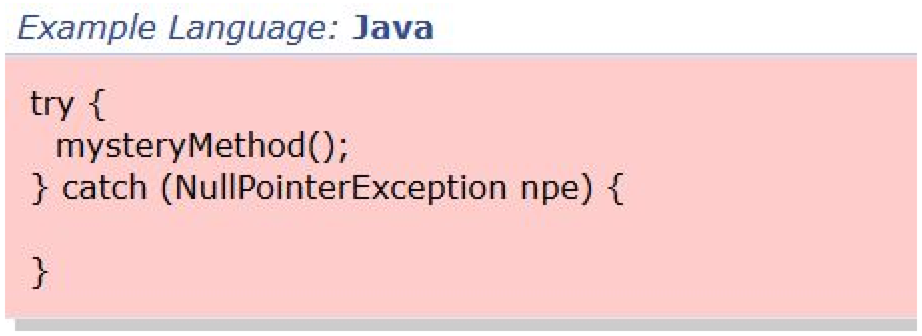
Of these three circumstances, only the last is acceptable.

Languages: Java

Scope: availability

Effect: Technical Impact: *DoS: resource consumption (CPU)*

Example of Code: The following code mistakenly catches a NullPointerException.



```
try {
  mysteryMethod();
} catch (NullPointerException npe) {

}
```

Potential Mitigation: Phases: Architecture and Design; Implementation

Do not extensively rely on catching exceptions (especially for validating user input) to handle errors. Handling exceptions can decrease the performance of an application.

Detection Methods:

**Automated Static Analysis - Binary / Bytecode**

**Dynamic Analysis with manual results interpretation**

**Manual Static Analysis - Source Code**

**Automated Static Analysis - Source Code**

**Architecture / Design Review**

*Public Data Assigned to Private Array-Typed Field* - *(496)*

*https://owasp.org/index.php/Public_Data_Assigned_to_Private_Array-Typed_Field*

Description: Assigning public data to a private array is equivalent to giving public access to the array.

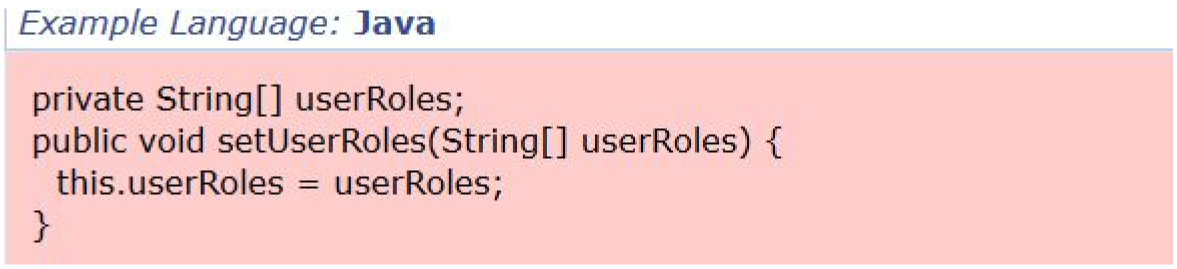
Time of Introduction: Implementation

Languages: Java, C, C++, .NET

Scope: Integrity

Effect: Technical Impact: *Modify application data*

The contents of the array can be modified from outside the intended scope.
Example of Code: In the example below, the setRoles() method assigns a publically-controllable array to a private field, thus allowing the caller to modify the private array directly by virtue of the fact that arrays in Java are mutable.


Example Language: **Java**

```
private String[] userRoles;
public void setUserRoles(String[] userRoles) {
  this.userRoles = userRoles;
}
```

Potential Mitigation: Phase: Implementation Do not allow objects to modify private members of a class.

White Box Definitions: A weakness where code path has a statement that assigns a data item to a private array field and the data item is public

Katrina
*Access to Critical Private Variable via Public Method - (767)*
*http://cwe.mitre.org/data/definitions/767.html*

*Description*
*Description Summary*
*The software defines a public method that reads or modifies a private variable.*
*Extended Description*
*If an attacker modifies the variable to contain unexpected values, this could violate assumptions from other parts of the code. Additionally, if an attacker can read the private variable, it may expose sensitive information or make it easier to launch further attacks.*

*Applicable Platforms*
*Languages*
*C++*
*C#*
*Java*

*Common Consequences*

| ***Scope*** | ***Effect*** |
|---|---|
| *Integrity*<br>*Other* | *Technical Impact: Modify application data; Other* |

Likelihood of Exploit

Low to Medium

Demonstrative Examples

Example 1

The following example declares a critical variable to be private, and then allows the variable to be modified by public methods.

(Bad Code)

Example

Language: C++

```
private: float price;
public: void changePrice(float newPrice) {
price = newPrice;
}
```

Example 2

The following example could be used to implement a user forum where a single user (UID) can switch between multiple profiles (PID).

(Bad Code)

Example

Language: Java

```
public class Client {
private int UID;
public int PID;
private String userName;
public Client(String userName){
PID = getDefaultProfileID();
UID = mapUserNametoUID( userName );
this.userName = userName;
}
public void setPID(int ID) {
UID = ID;
}
}
```

Omitted Break Statement in Switch - (484)

http://cwe.mitre.org/data/definitions/484.html

Description Summary

The program omits a break statement within a switch or similar construct, causing code associated with multiple conditions to execute. This can cause problems when the programmer only intended to execute code associated with one condition.

Extended Description

This can lead to critical code executing in situations where it should not.

Applicable Platforms

Languages

C

C++

Java

.NET

PHP

Common Consequences

| cope | Effect |
|---|---|
| ther | Technical Impact: Alter execution logic<br>This weakness can cause unintended logic to be executed and other unexpected application behavior. |

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1

In both of these examples, a message is printed based on the month passed into the function:

(Bad Code)

Example

Language: Java

```
public void printMessage(int month){
switch (month) {

case 1: print("January");
case 2: print("February");
case 3: print("March");
case 4: print("April");
case 5: print("May");
case 6: print("June");
case 7: print("July");
case 8: print("August");
case 9: print("September");
case 10: print("October");
case 11: print("November");
case 12: print("December");
}
println(" is a great month");
}
```

(Bad Code)

Example Languages: C and C++

```
void printMessage(int month){
```

```
switch (month) {

case 1: printf("January");
case 2: printf("February");
case 3: printf("March");
case 4: printf("April");
case 5: printff("May");
case 6: printf("June");
case 7: printf("July");
case 8: printf("August");
case 9: printf("September");
case 10: printf("October");
case 11: printf("November");
case 12: printf("December");
}
printf(" is a great month");
}
```

*Both examples do not use a break statement after each case, which leads to unintended fall-through behavior. For example, calling "printMessage(10)" will result in the text "OctoberNovemberDecember is a great month" being printed.*

▼ *Potential Mitigations*

*Phase: Implementation*

*Omitting a break statement so that one may fall through is often indistinguishable from an error, and therefore should be avoided. If you need to use fall-through capabilities, make sure that you have clearly documented this within the switch statement, and ensure that you have examined all the logical possibilities.*

*Phase: Implementation*

*The functionality of omitting a break statement could be clarified with an if statement. This method is much safer.*

Omar

[Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')](#) - (362)
[https://cwe.mitre.org/data/definitions/821.html](https://cwe.mitre.org/data/definitions/821.html) (821)

Rebecca

[Struts: Incomplete validate() Method Definition](#) - *(103)*

CWE- 103: Struts: Incomplete validate () Method Define
Description summary: The application has a validator that either does not define validate() method, or defines a validate() method but does not call super.validate().
Extended Description: If you do not call super.validate(), the Validation Framework cannot check the contents of the form against a validation form. In other words, the validation framework will be disabled for the given form.
Languages: Java
Scope: other Confidentiality, integrity, availability
Effect
Technical Impact *other*: disabling the validation framework for a form exposes the application to numerous types of attacks. Unchecked input is the root cause of vulnerabilities like cross-site scripting, process control, and sql injection.
Technical Impact *other*: Although J2EE applications are not generally susceptible to memory corruption attacks, if a J2EE application interfaces with native code that does not perform array bounds checking, an attacker may be able to use an input validation mistake in the J2EE application to launch a buffer overflow attack.
Example 1
In the following Java example the class Registration Form is a Struts framework ActionForm Bean that will maintain user input data from a registration webpage for an online business site. The user will enter registration dta and the RegistrationForm bean in Struts framework will maintain the user data. The RegistrationForm class implements the validate method to validate the user input entered into the form

*Example Language:* **Java bad code**

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {
// private variables for registration form
private String name;
private String email;
...
public RegistrationForm() {
super();
}
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
ActionErrors errors = new ActionErrors();
if (getName() == null || getName().length() < 1) {
errors.add("name", new ActionMessage("error.name.required"));
}
return errors;
}
// getter and setter methods for private variables
...
}
```

Altho the validate method is implemented in this example the method does not call the validate method of the ValidatorForm parent class with a call to super.validate(). Without the call to the parent validator class only the custom validation will be performed and the default validation will not be performed. The following example shows that the validate method of the ValidatorForm class is called within the implementation of the validate method

*Example Language:* **Java good code**

```
public class RegistrationForm extends org.apache.struts.validator.ValidatorForm {

// private variables for registration form
private String name;
private String email;
...
public RegistrationForm() {
super();
}
public ActionErrors validate(ActionMapping mapping, HttpServletRequest request) {
ActionErrors errors = super.validate(mapping, request);
if (errors == null) {
errors = new ActionErrors();
}
if (getName() == null || getName().length() < 1) {
errors.add("name", new ActionMessage("error.name.required"));
}
return errors;
}
// getter and setter methods for private variables
...
}
```

Potential Mitigations: Phase Implementation

Implement the validate() method and call super.validate() within that method.

Use of Dynamic Class Loading - (545)

CWE-545 Use of Dynamic Class Loading

Description Summary- Dynamically loaded code has the potential to be malicious

Languages- Java

Scope: other Confidentiality, integrity, availability

Effect: Technical Impact: other; execute unauthorized code or commands

An attacker could execute malicious code that they have included in the loaded class. The malicious code can be executed without calling a specific method if the malicious code is hidden within the static class initializer.

Example of code: The Code below dynamically loads a class using the java refelction api

```
String className = System.getProperty("customClassName");
Class clazz = Class.forName(className);
```

Potential Mitigations: **Phase: Architecture and Desgin**

Avoid the use of class loading as it greatly complicates code analysis. If the applicatin requires dynamic class loading, it should be well understood and documented. All classes that may be loaded should be predefined and void the use of dynamically created classes from byte arrays.

Other notes

The class loader executes the static initializers when the class is loaded. A malicious attack may be hidden in the static initializer and therefore does not require the execution of a specific method. An attack may also be hidden in any other method in the dynamically loaded code. The use of Dynamic code could also  enable an attacker to insert an attack into an application after it has been deployed. The attack code would not be the baseline, but loaded dynamically while the application is running.

**Used in Tools (Need 7)**

## CWE-395: Use of NullPointerException Catch to Detect NULL Pointer Dereference

## CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Description summary: Software constructs all or part of an SQL command using externally-influenced input without correctly neutralizing special elements that could modify the intended SQL commands.

Extended Description: Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data. This can be used to alter query logic to bypass security checks, or to insert additional statements that modify the back-end database, possibly including execution of system commands.
SQL injection has become a common issue with database-driven web sites. Flaws in the software can be easily detected and exploited. This is because SQL makes no real distinction between control and data planes, no matter the size of user base involved.

Enabling Factors for Exploitation: Application dynamically generates queries that contain user input.

Languages: All

Technology Classes: Database-Server

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Modes of Introduction: Data-rich applications that save user inputs in a database.

Common Consequences:

| Scope | Effect |
|---|---|
| Confidentiality | Technical Impact: Read application data<br>Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem. |
| Access Control | Technical Impact: Bypass protection mechanism<br>If poor SQL commands are used, it may be possible to connect to a system as another user without a password.<br>Or<br>If authorization information is held in a SQL database, it may be possible to change this information through the SQL injection vulnerability. |
| Integrity | Technical Impact: Modify application data<br>Possible to change or delete db information with a such an attack. |

Likelihood of Exploitation: Very High

Detection Methods:

- Automated static analysis tools
- Dynamic tools and techniques that interact with the software
- Manual analysis
- Bytecode Weakness Analysis
- Binary Weakness Analysis
- Database scanners
- Web Application Scanner
- Web Services Scanner
- Fuzz Tester
- Framework-based Fuzzer

·      Manual Source Code Reviews
·      Focused Manual Spotcheck - Focused manual analysis of source

If poor SQL commands are used to check usernames and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.
Access Control

Technical Impact: Bypass protection mechanism
If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL injection vulnerability.
Integrity

Technical Impact: Modify application data
Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL injection attack.

**“Vulnerable” code = Bad**

```
String accountBalanceQuery =

"SELECT accountNumber, balance FROM accounts WHERE account_owner_id =
"

  + request.getParameter("user_id");

try {
     Statement statement = connection.createStatement();
     ResultSet rs = statement.executeQuery(accountBalanceQuery);
     while (rs.next()) {
        page.addTableRow(rs.getInt("accountNumber"),
rs.getFloat("balance"));
     }
} catch (SQLException e) { ... }
```

**The developer could mitigate this vulnerability by using a prepared statement to create a parameterized query as below:**

**Good or at least better code**

```
String userLoginQuery =
  "SELECT user_id, username, password_hash FROM users WHERE username
= ?";
…
try {
    PreparedSatement statement =
connection.prepareStatement(userLoginQuery);
    statement.setString(1, request.getParameter("user"));
    ResultSet rs = statement.executeQuery();
…
```