Jing Li 21902204

Table of Contents

[Task 1: One security risk : A1 –Injection that is most likely to occur in development or the most important issue. 1](#_Toc19359210)

[1. Explain reasons of chosen 1](#_Toc19359211)

[2. Describe the security risk 2](#_Toc19359212)

[3. Analysis of application vulnerable 2](#_Toc19359213)

[4. Solution of prevention 2](#_Toc19359214)

[5. Two Example Attack Scenarios 2](#_Toc19359215)

[Task 2: TWO of the dangerous errors that is most likely to occure in development or is the most important issue from Commom Weakness Enueration: CWE-89 SQL injection ; CWE-78, OS command injection 4](#_Toc19359216)

[1.Introduction 4](#_Toc19359217)

[2. Technical details 4](#_Toc19359218)

[3. Code examples; 4](#_Toc19359219)

[4. Detection methods 5](#_Toc19359220)

[5. Conclusion 5](#_Toc19359221)

[Task 3: Summary to above tasks 6](#_Toc19359222)

[1. Identify vulnerabilities exposed in the above two tasks 6](#_Toc19359223)

[2. Correct explaination of exploited vulnerabilities 6](#_Toc19359224)

[3. Describle how to detect the exploit 6](#_Toc19359225)

[4. Suggest defences for these types of vulnerability on secure coding issues 6](#_Toc19359226)

[5. Policy to cover the security risk exposed in the above two tasks: 6](#_Toc19359227)

# Task 1: One security risk : A1 –Injection that is most likely to occur in development or the most important issue.

## 1. Explain reasons of chosen

As we all know, Data is extremely important to many companies todays.

At the same time, almost any kind of data could be an injection vectors,

external and internal web services, and every types of users.

So I choose A1 –Injection, the NO. 1 security risk based on Open Web Application Security Project (OWASP)-rating 2017.

## 2. Describe the security risk

Injection may result in data loss, corruption, or disclosure. Injection sometimes

leads to take over the host completely.

An attacker's malicious data may trick the interpreter into executing an

unexpected command or accessing the data without proper authorization.

## 3. Analysis of application vulnerable

Today more and more vulnerable web-based applications are deployed.

An Injection vulnerability occur when an attacker can send malicious data to

the interpreter. Injection defects are very common, frequently found injection

holes in SQL, OS command and so on.

Malicious data is used or connected directly so that SQL or commands contain

structural and malicious data ,dynamic queries, commands or stored

procedures.

## 4. Solution of prevention

Preventing injection requires separating data from commands and queries.

The preferred option is to use a secure API to avoid using full interpreter or

provide a parameterized interface, or migrate to use the Object Relational

Mapping Tools (ORMs).

Use LIMIT and other SQL controls in the query to prevent a large number of

records were disclosed in the case of SQL injection.

## 5. Two Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the

following vulnerable SQL call:

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

This query can be exploited when the attacker modifies the ‘id’ parameter

value in their browser to send: ' or '1'='1.

For example, http://example.com/app/accountView?id=' or '1'='1

This can cause the database table to return all the rows.

Scenario #2: Hackers with it can make nobody can enter the website.

Common Gateway Interface (CGI) programs can be written in any programming language and executed by virtually all web servers.

Imagine a form in a Webpage with two input textboxes: “username”, and “password”.

**CGI script code for SQL: string query = "SELECT \* FROM users WHERE username = '" + userName + "' AND password = '" + password.Text + "'";**

**CGI intended generated SQL string:**

**SELECT \* FROM users WHERE username = <userName> AND password = <password>;**

**User enters: “Administrator” as username and “secret'; DELETE FROM users; --” as password**

**SQL query result is:**

**SELECT \* FROM users WHERE username = ‘Administrator' AND password = ‘secret';**

**DELETE FROM users;**

**--';**

**Result is 3 separate SQL queries separated by semicolon.**

**1st might fail.**

**2nd will delete all entries in table “users”.**

**3rd is just a comment.** In this case, the comment character is used to delete trailing single quotes left in the modified query.

# Task 2: TWO of the dangerous errors that is most likely to occure in development or is the most important issue from Commom Weakness Enueration: CWE-89 SQL injection ; CWE-78, OS command injection

## 1.Introduction

CWE-89 - SQL injection refers to the attacker inserting the SQL command into the string submitted by the web form to achieve the purpose of causing the server to execute a malicious SQL command.

If SQL queries are used in security controls such as authentication, an attacker could change the logic of these queries to bypass security.

CWE-78-OS command injection is where the application interacts with the operating system.

Your software is usually a bridge between outsiders on the network and the inside of the operating system. When you call another program on the operating system, but allow untrusted input to be provided to the command string that you generated to execute the program, you will invite the attacker to cross the bridge into wealth by executing your own commands. Your land.

## 2. Technical details

SQL injection：Because SQL is an interpreted language. As an interpreted language, it is run by a runtime component. The statement that interprets the code and executes the instructions contained in it.

This can be used to change the query logic to bypass security checks, or to

insert other statements that modify the backend database, possibly including

executing system commands.

OS command injection: This may allow an attacker to execute an unexpected

dangerous command directly on the operating system.

This weakness can lead to an attacker's inability to directly access

vulnerabilities in the operating system's environment, such as in a web

application.

## 3. Code examples:

Example 1: Hackers with it can enter the website without knowing password.

Imagine a form in a Webpage with two input textboxes: “username”, and “password”.

**CGI script code for SQL:**

**string query = "SELECT \* FROM items WHERE username = '" + userName + "' AND password = '" + password.Text + "'";**

**CGI intended generated SQL string:**

**SELECT \* FROM items WHERE username = <userName> AND password = <password>;**

**User enters:**

**“Administrator” as username and “secret' OR 'a'='a” as password**

**SQL query result is:**

**SELECT \* FROM items WHERE username = ‘Administrator' AND password = ‘secret' OR 'a'='a';**

Result is that the right part of the OR statement is always true and the user always gets authenticated as Administrator bypassing the check.

Example 2: The following example reads the name of a shell script to be executed from a system property. It is affected by the second variant injected by the OS command.

*Example Language:***Java**

**String script = System.getProperty("SCRIPTNAME");**

**if (script != null)**

**System.exec(script);**

If an attacker controls this property, they can modify the property to point to a dangerous program.

## 4. Detection methods

First I can use automation detection method: I use Puma Scan and Security IntelliSense online tools to be installed in MS Visual Studio to detect.

Secondly, I can use code more smart to detect. Take the above for example , I will use “if” code in background to limit the password input with “or” or” ； “ and so on.

## 5. Conclusion

# Task 3: Summary to above tasks

It is expected the following questions to be answered as a summary to the above two tasks.

## Identify vulnerabilities exposed in the above two tasks

SQL injection is to insert the SQL command into the Web form to submit or enter the query string of the domain name or page request, and finally reach the spoofing server to execute malicious SQL commands.

## Correct explaination of exploited vulnerabilities

## Describle how to detect the exploit

#### Solution

Regex rgx = new Regex(@"^[a-zA-Z0-9]+$");

if(rgx.IsMatch(input))

{

var p = new Process();

p.StartInfo.FileName = "exportLegacy.exe";

p.StartInfo.Arguments = " -user " + input + " -role user";

p.Start();

}

## Suggest defences for these types of vulnerability on secure coding issues

Include one to three paragraphs on secure coding issues in your answer.

## Policy to cover the security risk exposed in the above two tasks:



SCS0001 - Command Injection

1. The dynamic value passed to the command execution should be validated.

2. Risk: If a malicious user controls either the FileName or Arguments, he might be able to execute unwanted commands or add unwanted argument.

This behavior would not be possible if input parameter are validate against a white-list of characters.