I2

Vulnerability Report for OS Comand and SQL Injection

Hiren Shah

3/18/21

University of Maryland Global Campus

SDEV 325 – Detecting Software Vulnerabilities

# Executive Summary

In this homework I used cloud9 to create two unique examples of an insecure interaction between components. The first example contained an improper neutralization of special elements used in an OS Command “OS Command Injection.” I was successful in fixing the vulnerable example by using input validation in the code.

The second example I chose is the sql injection vulnerability. This example contained an improper neutralization of of special elements used in an SQL command. I was successful in fixing the vulnerable example by using a parameterized statement.

# Example 1 – [CWE-78: OS Command Injection]

## Overview

The first vulnerability that I chose was the OS Command Injection. This vulnerability (also known as shell injection) is a web security vulnerability that allows an attacker to execute arbitrary operating system (OS) commands on the server that is running an application, and typically fully compromise the application and all its data. Very often, an attacker can leverage an OS command injection vulnerability to compromise other parts of the hosting infrastructure, exploiting trust relationships to pivot the attack to other systems within the organization. The example applications performs a ping to a given ip address or URL, it is written in JAVA.

NOTE: I tried using the same code in AWS environment.. the ping worked but having multiple commands on one line in the terminal did not work. The example below is working in Eclipse, so I used this screenshot instead

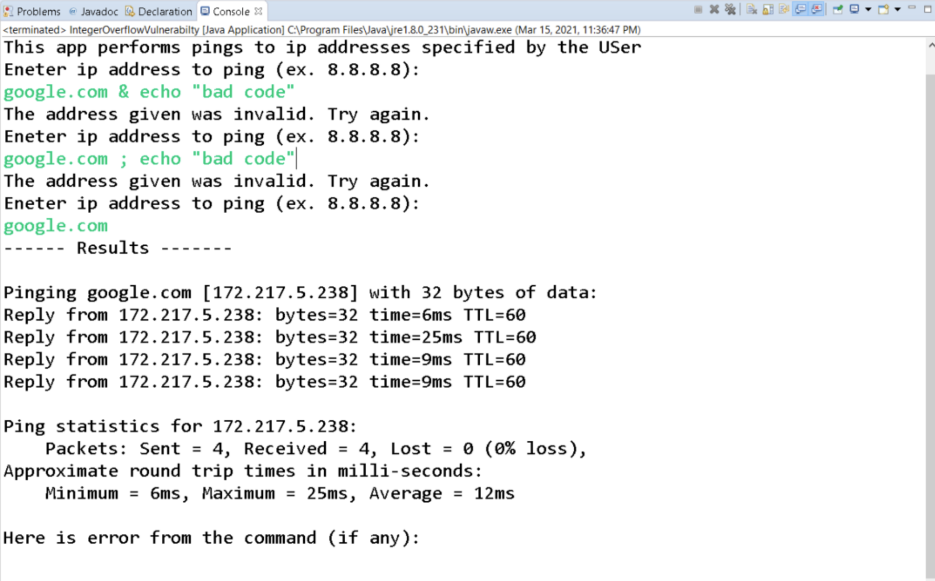
## Analysis of the Vulnerability

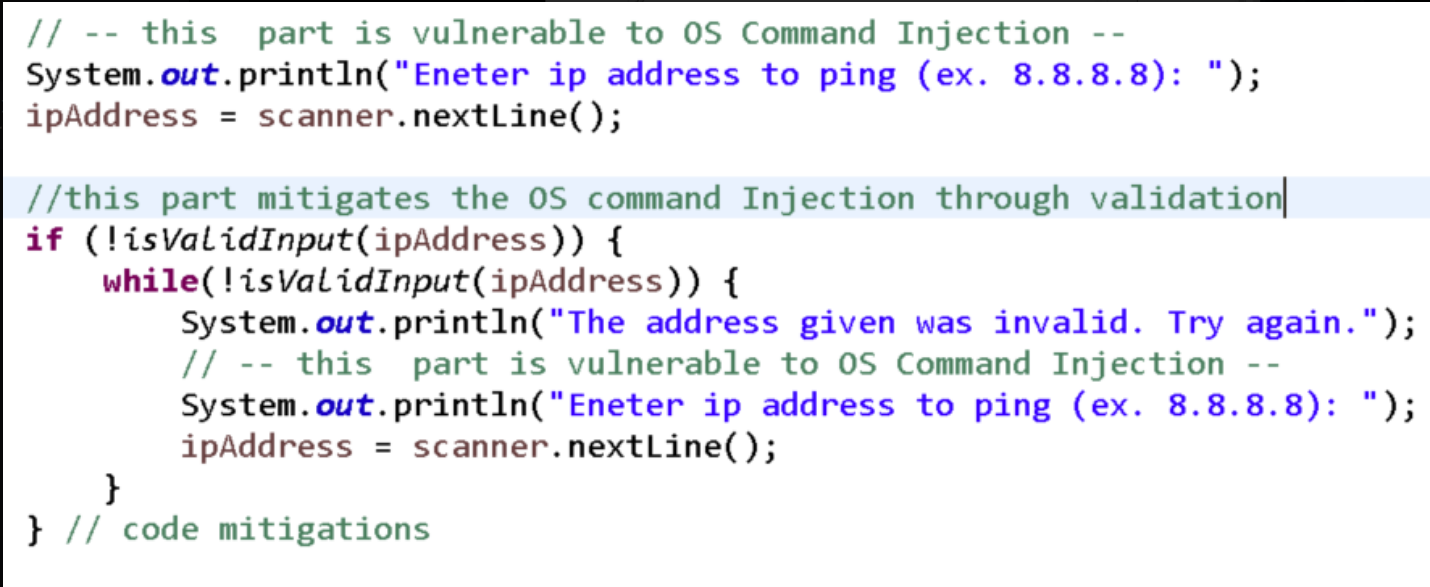
In this example a code is asking the user to input a i.p. address to ping, in addition to the i.p address, a user can also submit a malicious code to break the program or get access to certain system architecture.

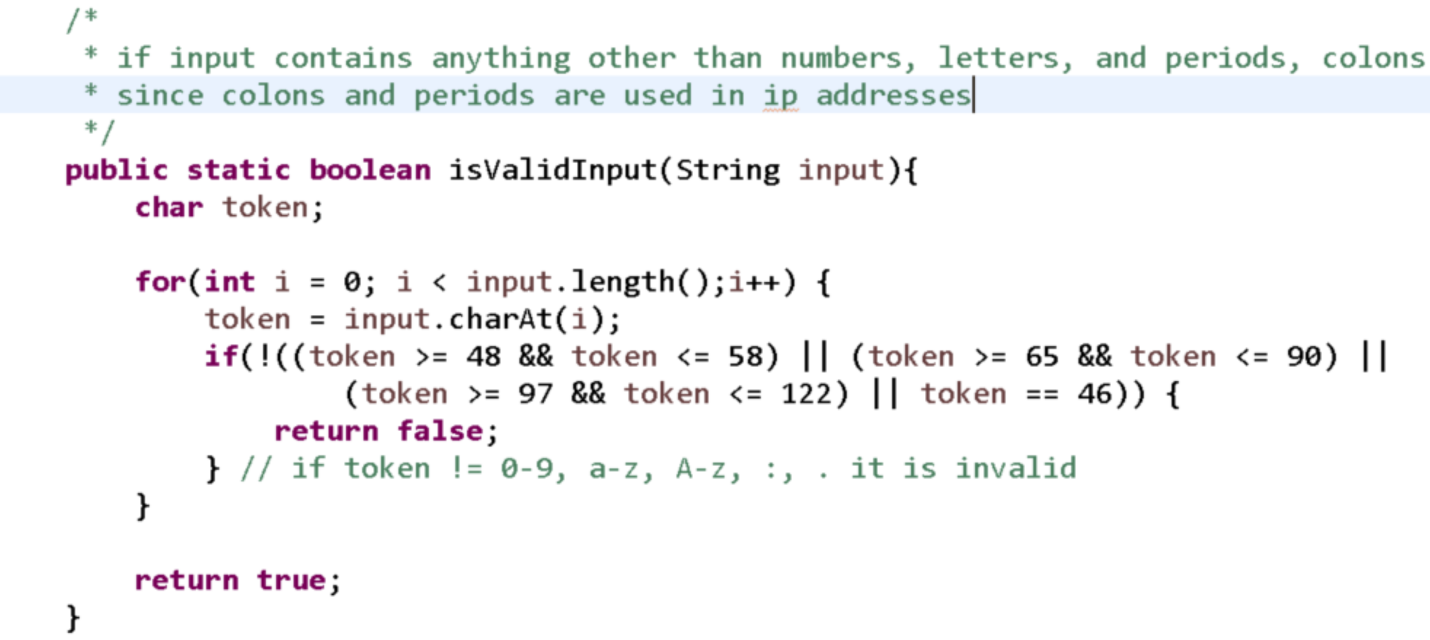
## 

## Mitigation

Mitigation would be to perform input validation. In the example, we looped through the string inputted by the user and if it contained any of the characters other than numbers, letters, periods, and colons, we marked it as invalid input. This is because ip addresses only need letters, numbers, colons, and periods.

Figure 2. Showing output of code when using “malicious code”

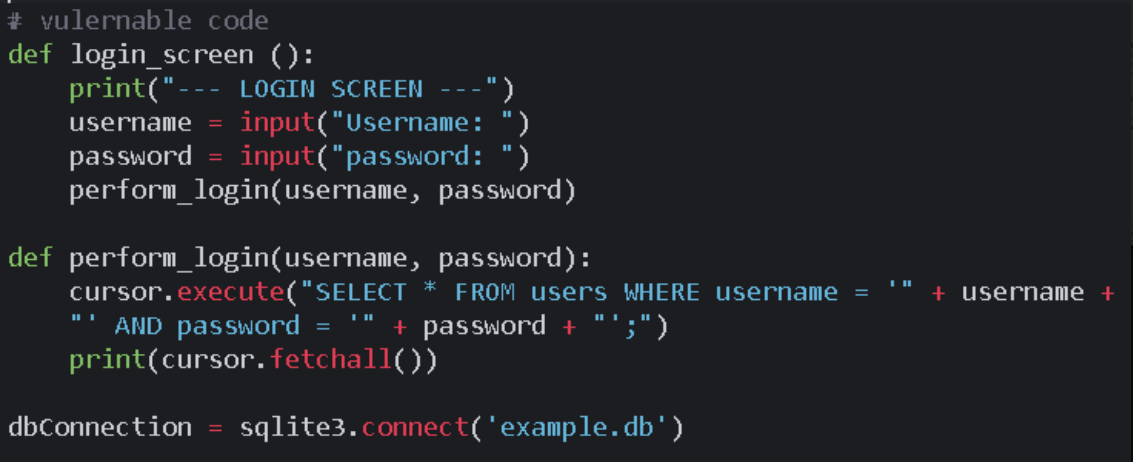
Figure 3. Showing the code that mitigates the OS command Injection through input validation.

Figure 4. Showing the helper method for input validation.

# Example 2 – [SQL Injection: CWE 89]

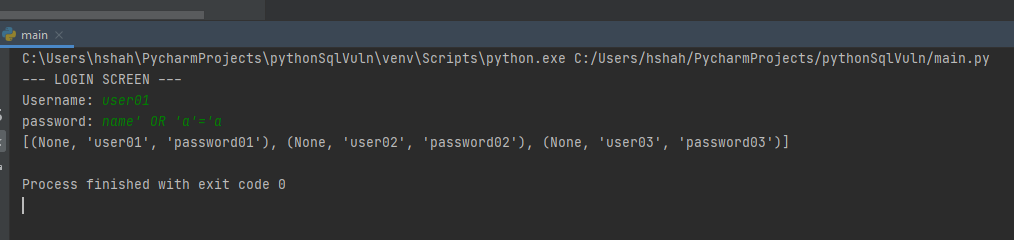
## Overview

The example I wrote creates a table users and creates columns. Then it takes in user input for username and password and displays if correct. The code was written in Python.

Figure 5. Vulnerable pythong/sql code.

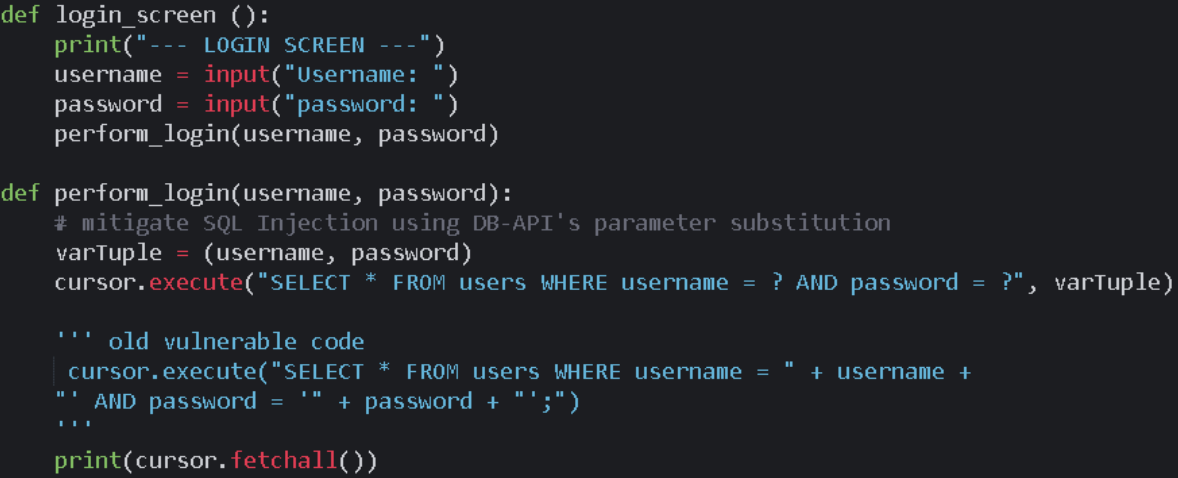
## Analysis of the Vulnerability

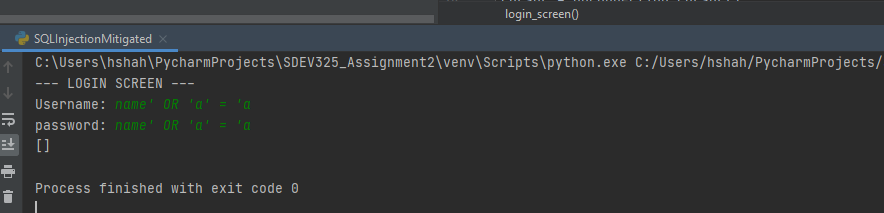
This sequal injection vulnerability in my example exists because the execute query uses python string operations to assemble the query. This allows a user to escape the intended query and add additional queries.

Figure 6. Output of code showing vulnerability.

## Mitigation

In order to mitigate SQL Injection attacks, we need to use DB-API’s parameter substitution. This is where we pre-compile an SQL statement so that all that needs to be supplied are the parameters for that statement. In python, this is as simple as putting the username nad password into a tuple and including that as an argument in the execute function.

Figure 7. Showing the new code that uses parameter substitution.

Figure 8. Showing the mitigated output when using an attack techique of name’ OR ‘a’=’a

# References