Potential System Vulnerabilities

# Vulnerabilities

## SQL Injection

When searching user defined criteria in databases using SQL, there is a risk of SQL injection. This is where the user enters SQL code into a text field that is then executed. This can be used to access or delete sensitive information.

For example:

|  |  |  |
| --- | --- | --- |
| *Example Database* | | |
| **ID** | **Username** | **Password** |
| 00 | John | 0000 |
| 01 | Luke | 1234 |
| 02 | James | 1999 |

The following code is used in a program; it asks the user for an ID and returns the corresponding Username entry.

cursor.execute(“SELECT Username FROM Example\_Database WHERE ID=” + input())

Then, when the user is prompted to enter an ID, they enter this:

0 OR 1=1;

The program will interpret the “OR 1=1” part as an extension of the SQL instruction, and as such will return all usernames to the user.

Within our program, we use this code when getting inputs from the user:

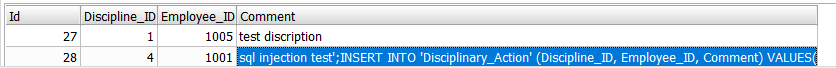
cursor.execute("INSERT INTO 'Disciplinary\_Action' (Discipline\_ID, Employee\_ID, Comment) VALUES(?, ?, ?)", (str(comboDisciplinary.current()+1), str(TxtEmployeeId.get("1.0", "end-1c")), str(txtDescription.get("1.0", "end-1c"))))

This avoids the problem by using the ‘?’ keyword – the inputs are not treated like normal SQL commands and as such users cannot inject their own code into the program.

Attempted SQL injection:



Resulting Entry Created:



## String Formatting

python allows for formatted string to be enter using the % symbol. This make python check the number of parameters is equal to the number specified. If it is not it will produce an error message If there is no handler. This can cause denial of service attacks on the program as it will terminate on an error.

This can cause an attacker to view the contents of a variables.

userdata = "password" : "secret" }

passwd = raw\_input("Password: ")

if (passwd != userdata["password"]):

print ("Password \"" + passwd

+ "\" is wrong for user")

else:

print "Welcome!"

if the user enter ‘wrong’ for the password it would display the wrong user message: *Password "green" is wrong for user jdoe.*

However if the user enters ‘%(password)s’ the program will out the variable password contents instead: *Password "secret" is wrong for user jdoe.*

## Buffer Overflow Attack

Buffer overflows occur when user given information cannot fit into the memory assigned by the program. This is more of an issue in C and C++ but can occur in Python (Peterson et al, 2015). This can be exploited to manipulate parts of memory that would otherwise be inaccessible to the user.

## Backup Management

This is not very complex but still important; if hardware fails or files get corrupted somehow, a lot of progress can be lost very quickly. This is why making regular backups of the project is important. When developing our program, we backed up our project to GitHub after each significant change. This means that if a section of our program broke somehow, we have easy access to earlier iterations of the code.

# Sources

* P. Peterson, P. Reiher (2015) *Exploits: Buffer Overflows, Pathname Attacks, and SQL Injections.* Available at: <https://cs.slu.edu/~chambers/spring11/security/assignments/lab05.html> (Accessed: 3 May 2019)
* W3schools <https://www.w3schools.com/sql/sql_injection.asp> (no date) (Accessed: 3 May)
* H. Burch, R. Seacord (2007) *Programming Language Format String Vulnerabilities* Available at: <http://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/DDJ/2007/0703/070201hb01/070201hb01.html> (Accessed: 3 May)