Injection Attacks

Objectives

- Explore how command inject attacks introduce arbitrary server-side execution.
- Discuss how databases store and organize information; examine how SQL statements are used to retrieve data.
- Explore how SQL-injection introduces opportunities to reveal confidential information.

References

- https://owasp.org/www-community/attacks/Command_Injection
- https://owasp.org/www-community/attacks/SQL_Injection

Command Injection

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation

Text copied from: https://owasp.org/www-community/attacks/Command_Injection

Command Injection Example

On the server there is code that takes the user supplied server and executes the command ping –c 1 *server;* however this is not properly validated; how might we abuse it?

```
@app.route('/command', methods=['GET', 'POST'])
def command():
    if request.method == 'GET':
        message = 'Enter the IP of a Server to Ping'
        return render_template('command.html',message=message)
    elif request.method == 'POST':
        server = request.form['server']
        results = subprocess.getoutput('ping -c 1 %s' % server)
        return render_template('command.html',message=results)
```

Command Injection Example

Benign Request – ping server as intended

	server	
ping –c 1	8.8.8.8	ping –c 1 8.8.8.8

Malicious Request – arbitrarily execute additional command

	server	
ping –c 1	8.8.8.8; cat flag.txt	ping –c 1 8.8.8.8; cat flag.txt

SQL Injection

A SQL injection attack consists of insertion or “injection” of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

Text copied from: https://owasp.org/www-community/attacks/SQL_Injection

SQL Injection Example

On the server there is code that takes the user supplied username and processes a sql query to recover the associated email address. However, it is not properly validated. *How might we abuse it?*

```
@app.route('/sqli')
def sqli():
    db = sqlite3.connect('users.db')
    cursor = db.cursor()
    query = request.args.get('q')
    try:
        if query:
            stmt = f"SELECT email from USERS where user == '{query}'"
            cursor.execute(stmt)
            results = cursor.fetchall()
```

SQL Injection Example

Benign Request – return the email address for adam

	name		result
SELECT email from USERS where user == '	adam	'	SELECT email from USERS where user == 'adam'

Malicious Request – return all the email addresses

	name		result
SELECT email from USERS where user == '	adam' or 1=1 --	'	SELECT email from USERS where user == 'adam' or 1=1 -- '

SQL Injection Example

Malicious Request – return all the email addresses

	name		result
SELECT email from USERS where user == '	adam' or 1=1 --	'	SELECT email from USERS where user == 'adam' or 1=1 -- '

- *'adam' or 1=1: is always True (for all records)*
- *-- is a comment. It comments out the extra single quote*

Additional helpful queries

We can connect two SQL statements with a union

select user from users union select password from passwords

For sqlite databases, we can iterate out the database structure

SELECT name FROM sqlite_master WHERE type='table'

SELECT name FROM pragma_table_info('<tablename>')