

Lab Notebook 2 for Andrew Stevenson

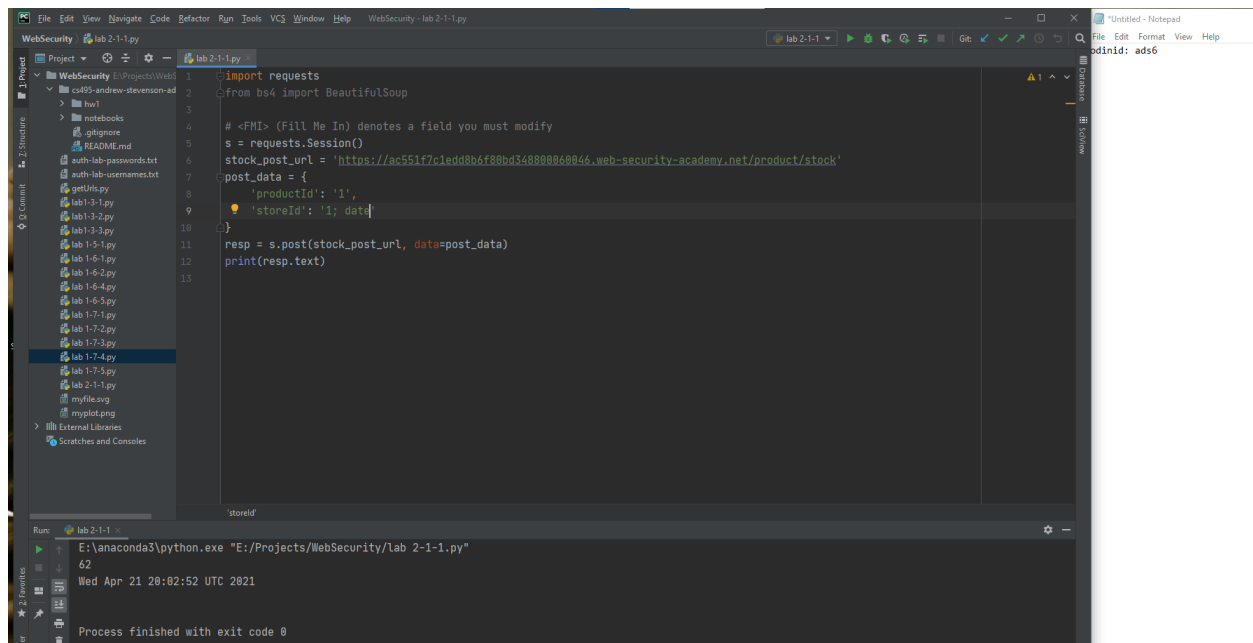
Lab 2.1

- [1. os-command-injection \(1\)](#)
- [2. os-command-injection \(2\)](#)
- [3. os-command-injection \(3\)](#)
- [4. os-command-injection \(4\)](#)
- [5. sql-injection \(1\)](#)
- [6. sql-injection \(2\)](#)
- [7. sql-injection/union-attacks \(1\)](#)
- [8. sql-injection/union-attacks \(2\)](#)
- [9. sql-injection/union-attacks \(3\)](#)
- [10. sql-injection/examining-the-database \(1\)](#)
- [11. sql-injection/examining-the-database \(2\)](#)

Lab 2.1

1. os-command-injection (1)

- Take a screenshot for your lab notebook of the output



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help WebSecurity - lab 2-1-1.py
WebSecurity lab 2-1-1.py
1 import requests
2 from bs4 import BeautifulSoup
3
4 # <FMI> (Fill Me In) denotes a field you must modify
5 s = requests.Session()
6 stock_post_url = 'https://ac551f7c1edd8b6f88bd348888648846.web-security-academy.net/product/stock'
7 post_data = {
8     'productId': '1',
9     'storeId': '1; data'
10 }
11 resp = s.post(stock_post_url, data=post_data)
12 print(resp.text)
13
```

Run: lab 2-1-1

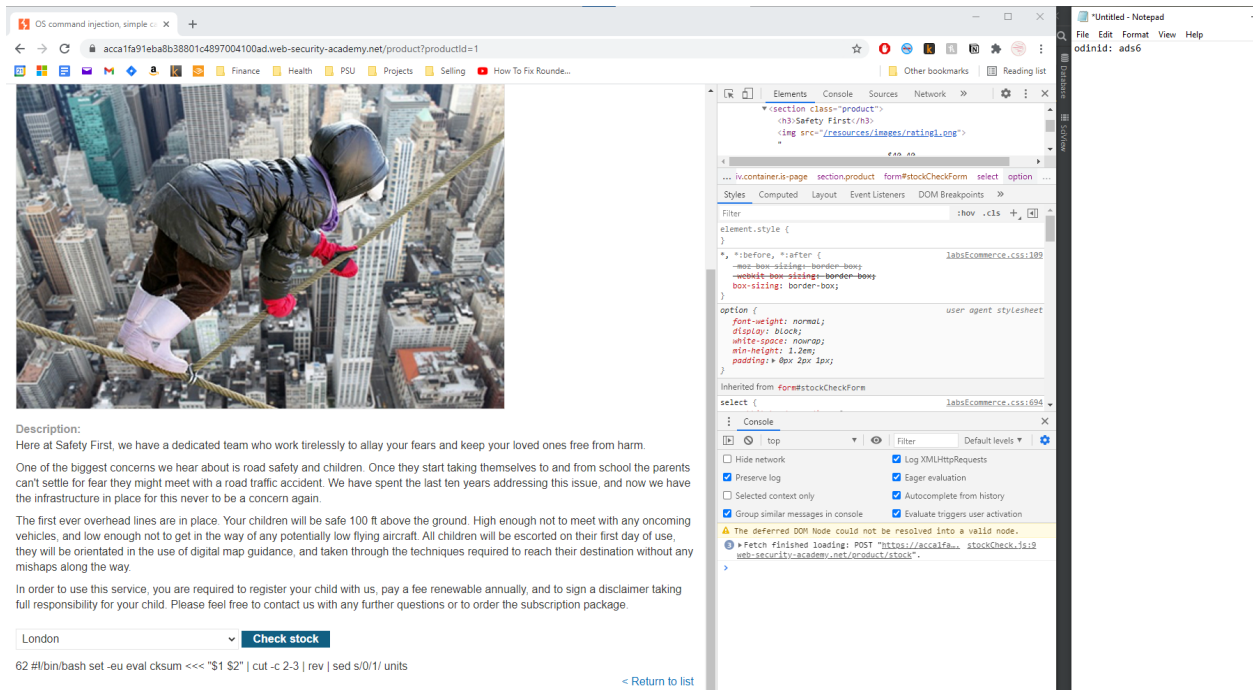
E:\anaconda3\python.exe "E:/Projects/WebSecurity/lab 2-1-1.py"

62

Wed Apr 21 20:02:52 UTC 2021

Process finished with exit code 0

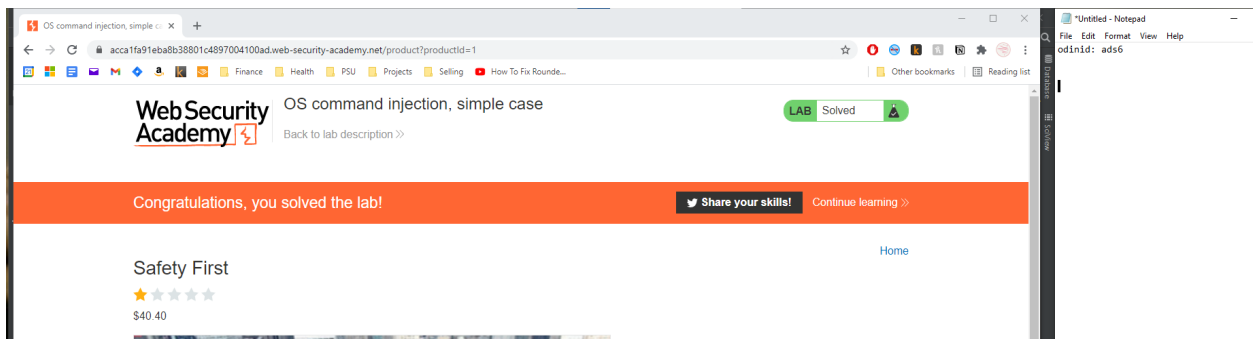
- Take a screenshot for your lab notebook of the output



- Take a screenshot showing completion of the level that includes your OdinId

Vulnerability: The unsanitized client-provide POST data structure is used in a shell command, allowing the adversary to execute any command they please.

Remediation: Sanitize the values by mapping user input onto storeID values before running the script.

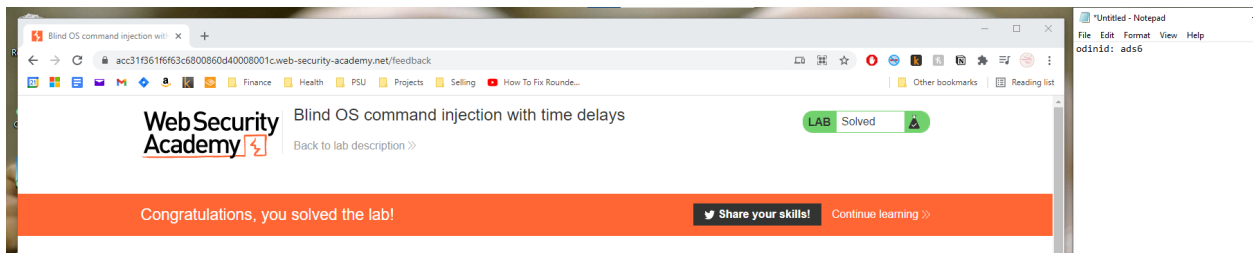


2. os-command-injection (2)

- Take a screenshot showing completion of the level that includes your OdinId

Vulnerability: The email field of the feedback form uses unsanitized user input, allowing the adversary to inject any command-line instruction they like surrounded by the || operator.

Remediation: Make sure the email field is really an email address before using it on the command line.

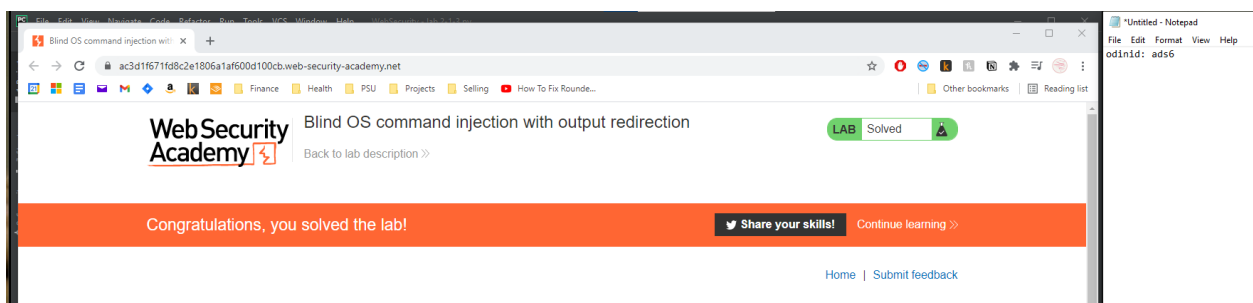


3. os-command-injection (3)

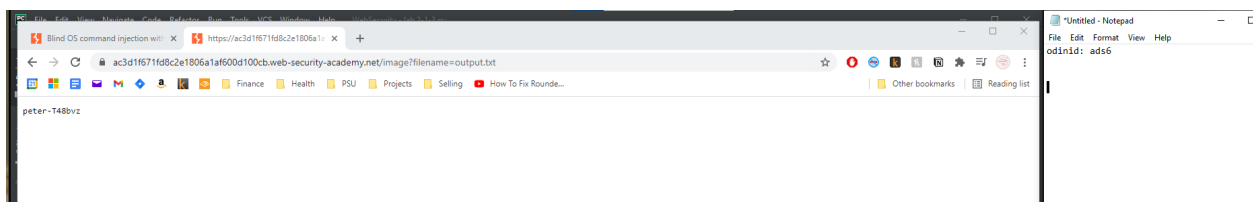
- Take a screenshot showing completion of the level that includes your OdinId

Vulnerability: As above, except the linux web user account also has file write privileges, allowing the adversary to exfiltrate all sorts of data to a file on a public web server.

Remediation: Don't give the web user write privileges.



- Take an additional screenshot that shows the username that runs the site

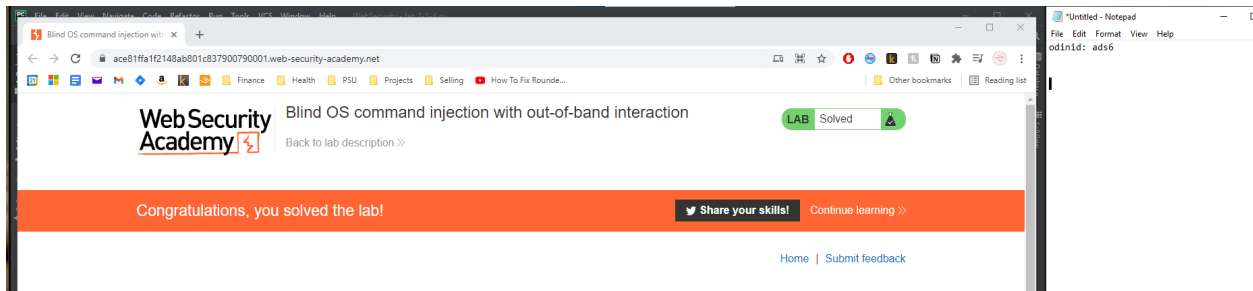


4. os-command-injection (4)

- Take a screenshot showing completion of the level that includes your OdinId

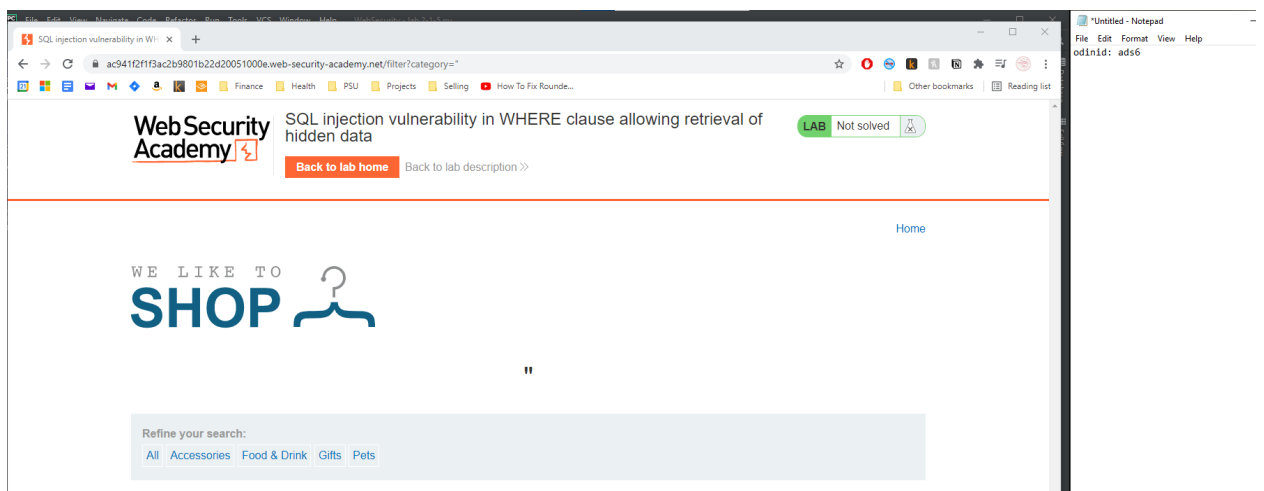
Vulnerability: As above, except this time the adversary injected a dig command in order to initiate a request and thus exfiltrate the stolen data.

Remediation: Make sure the email field is really an email address before using it on the command line.



5. sql-injection (1)

- Take a screenshot of the output.



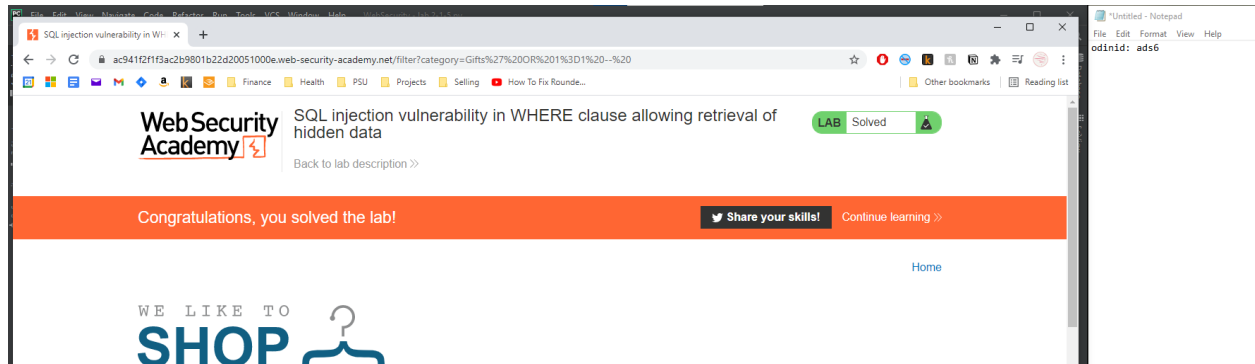
- What output is returned?

Internal Server Error

- Take a screenshot showing completion of the level that includes your OdinId

Vulnerability: The unsanitized url query argument is used on the server to form a sql string; thus, the adversary can easily inject their own sql strings in to the url to view unauthorized data.

Remediation: Use a sql encode function on the client data before using it in a sql string.



6. sql-injection (2)

- Is the username field vulnerable to SQL injection? If so, what character breaks syntax?

Yes; a single quote breaks the syntax.

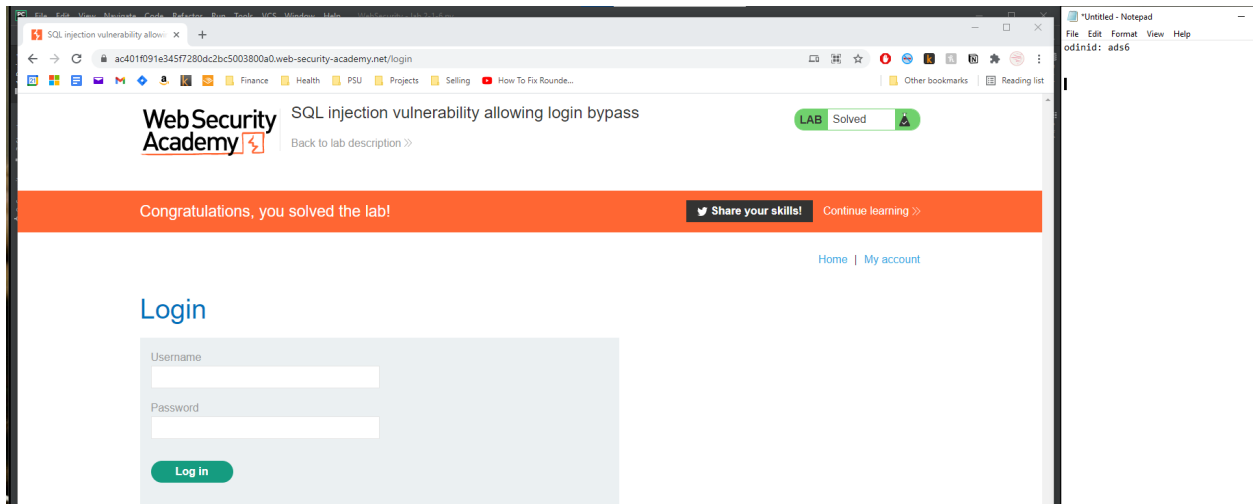
- Is the password field vulnerable to SQL injection? If so, what character breaks syntax?

Yes; a single quote breaks the syntax.

- Take a screenshot showing completion of the level that includes your OdinId

Vulnerability: The unsanitized url query argument is used on the server to form a sql string; to validate the login credentials; thus, the adversary may use a sql injection in the password field (for example, a password of "a' OR 1 = 1 --") to successfully login.

Remediation: Use a sql encode function on the username and password fields before using it in a sql string.

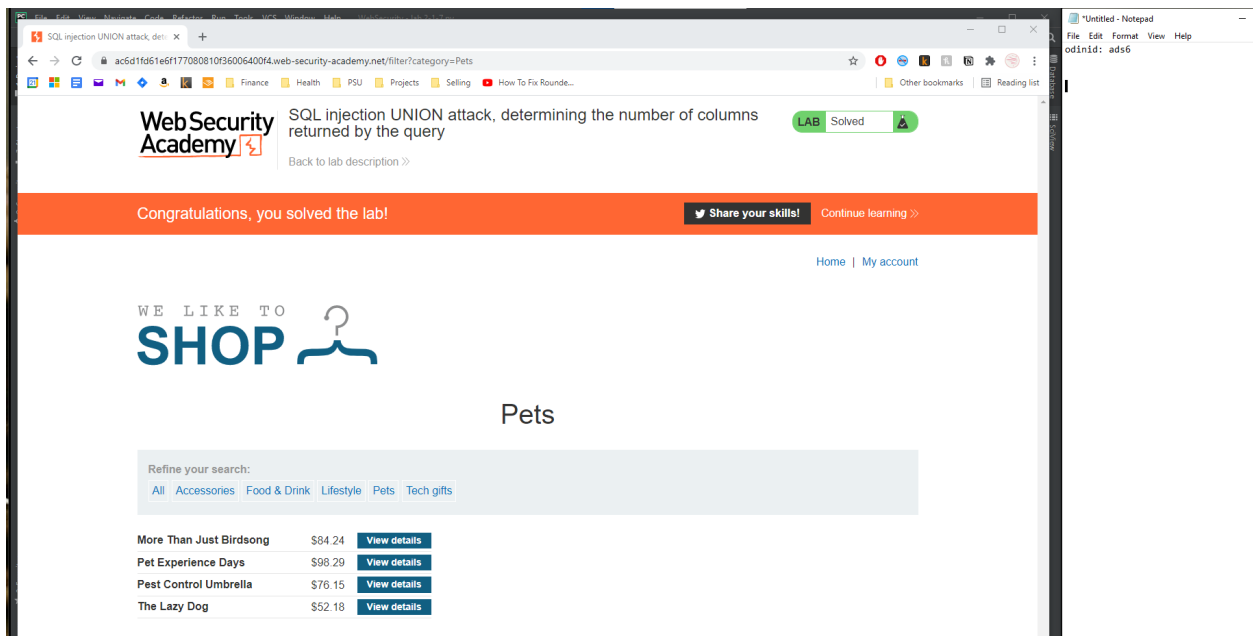


7. sql-injection/union-attacks (1)

- How many columns does the products table contain?
- Take a screenshot showing completion of the level that includes your OdinId

Vulnerability: Because the server uses unsanitized client data from the query string, an adversary may, by brute force, determine the correct number of columns in order to append a UNION statement disclosing the contents of any table within the database.

Remediation: Pass the client input through a sqlencode function before forming the sql string.

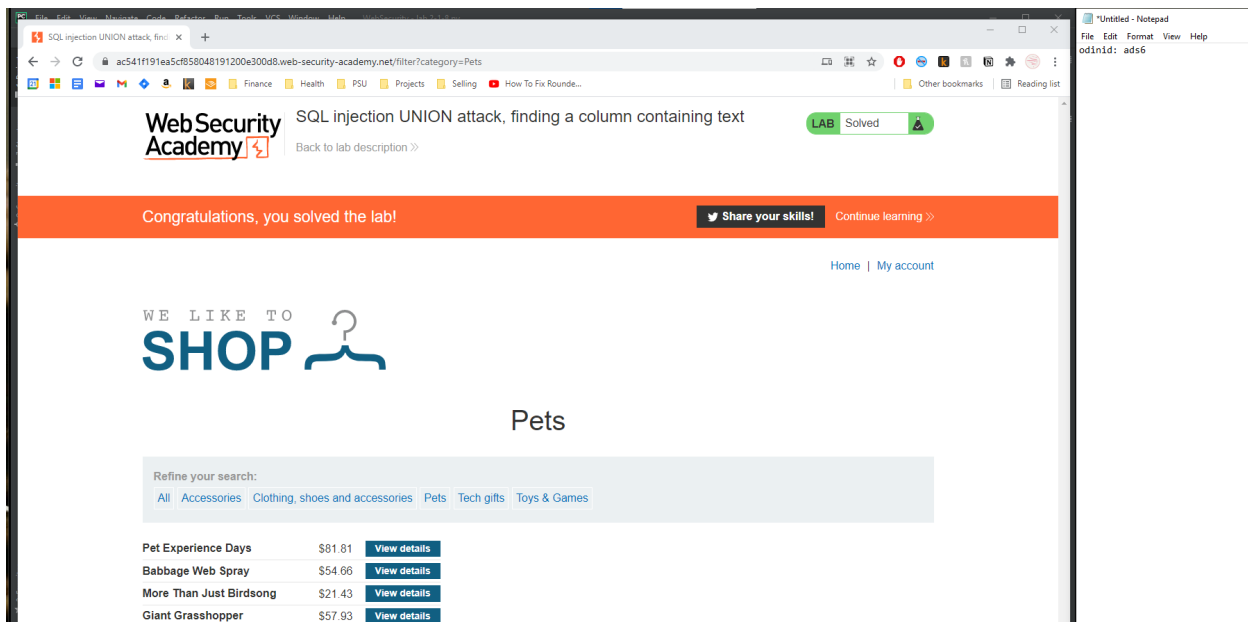


8. sql-injection/union-attacks (2)

- Take a screenshot showing completion of the level that includes your OdinId

Vulnerability: Because the server uses unsanitized client data from the query string, an adversary may, by brute force, determine the correct number of columns in order to append a UNION statement disclosing the contents of any table within the database, as above. In this level, the adversary may also determine whether the columns are text or not using a second brute-force attack (though they might have done so above as well...)

Remediation: Pass the client input query string through a sqlencode function before forming the sql string.

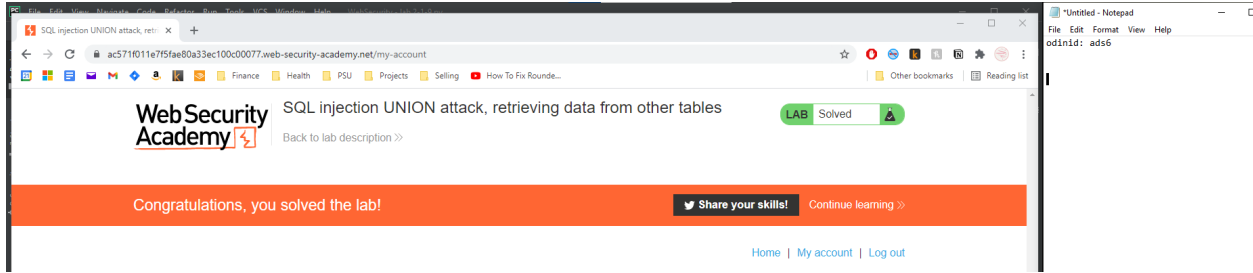


9. sql-injection/union-attacks (3)

- Take a screenshot showing completion of the level that includes your OdinId

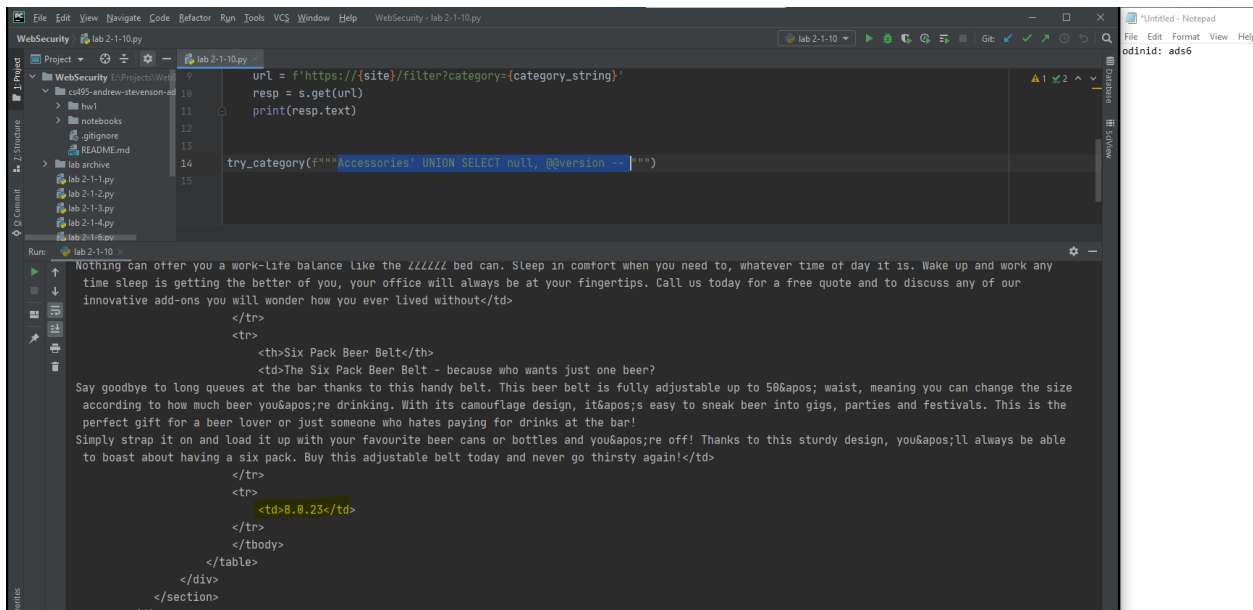
Vulnerability: As above, the server uses unsanitized client data from the query string, so an adversary may, by brute force, determine the correct number of columns in order to append a UNION statement disclosing the contents of the user/password table.

Remediation: Pass the client input query string through a sqlencode function before forming the sql string.



10. sql-injection/examining-the-database (1)

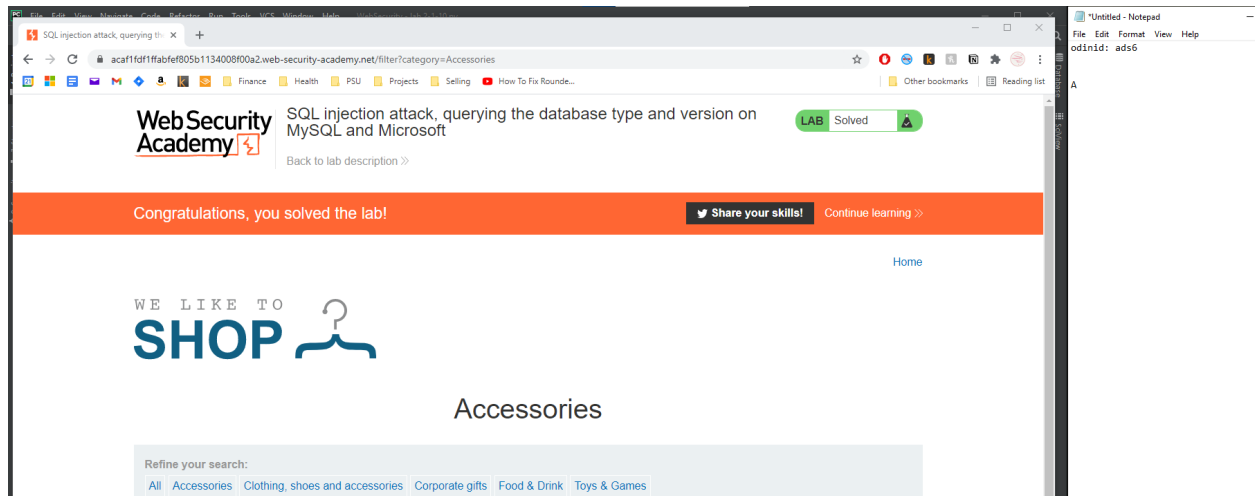
- Take a screenshot showing the version number for your lab notebook



- Take a screenshot showing completion of the level that includes your OdinId

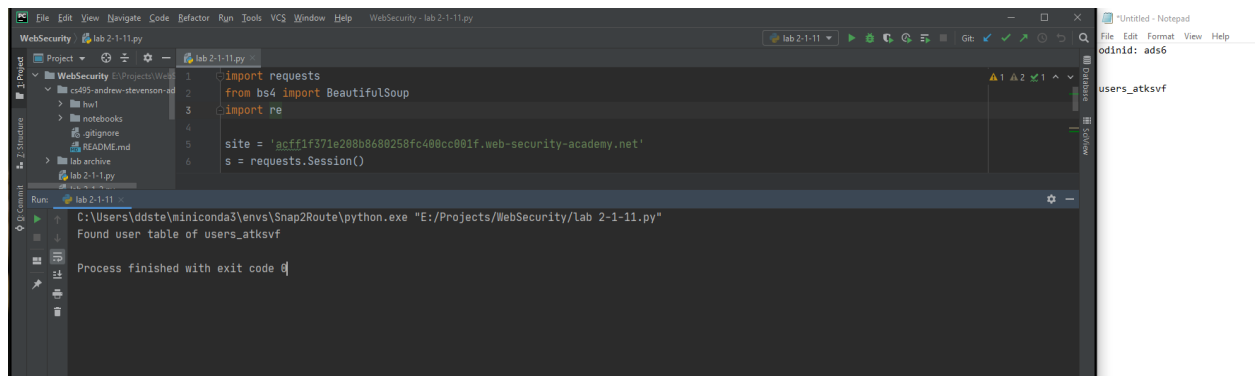
Vulnerability: As above, the server uses unsanitized client data from the query string, so an adversary may, by brute force, determine the correct number of columns in order to append a UNION statement disclosing the version of the database.

Remediation: Pass the client input query string through a `sqlencode` function before forming the sql string.

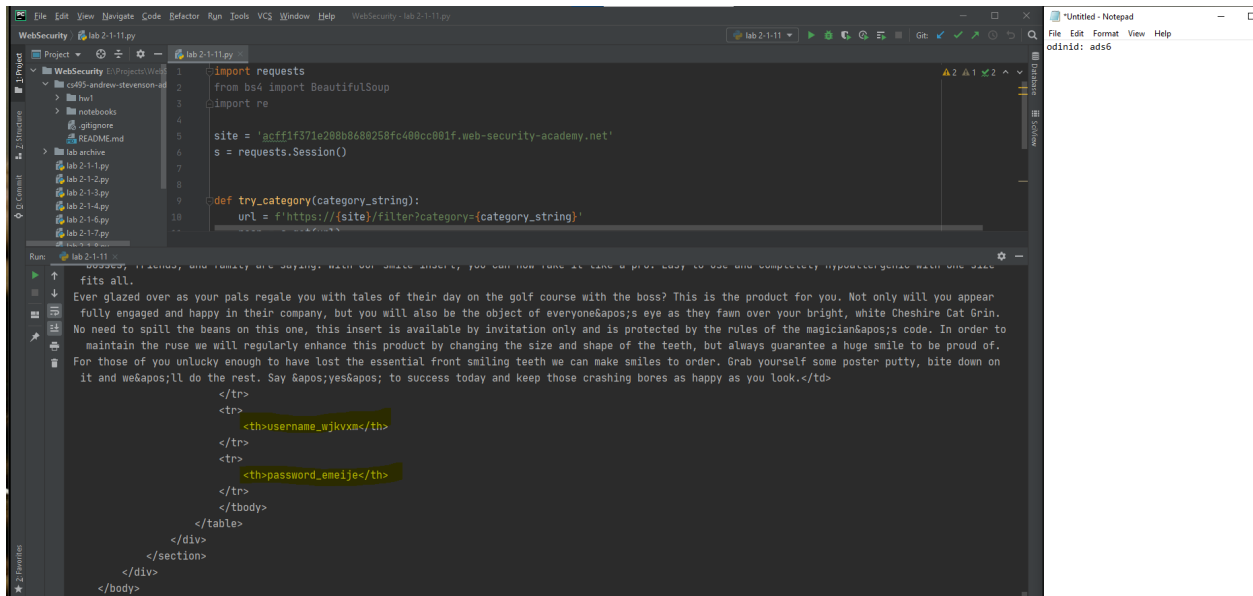


11. sql-injection/examining-the-database (2)

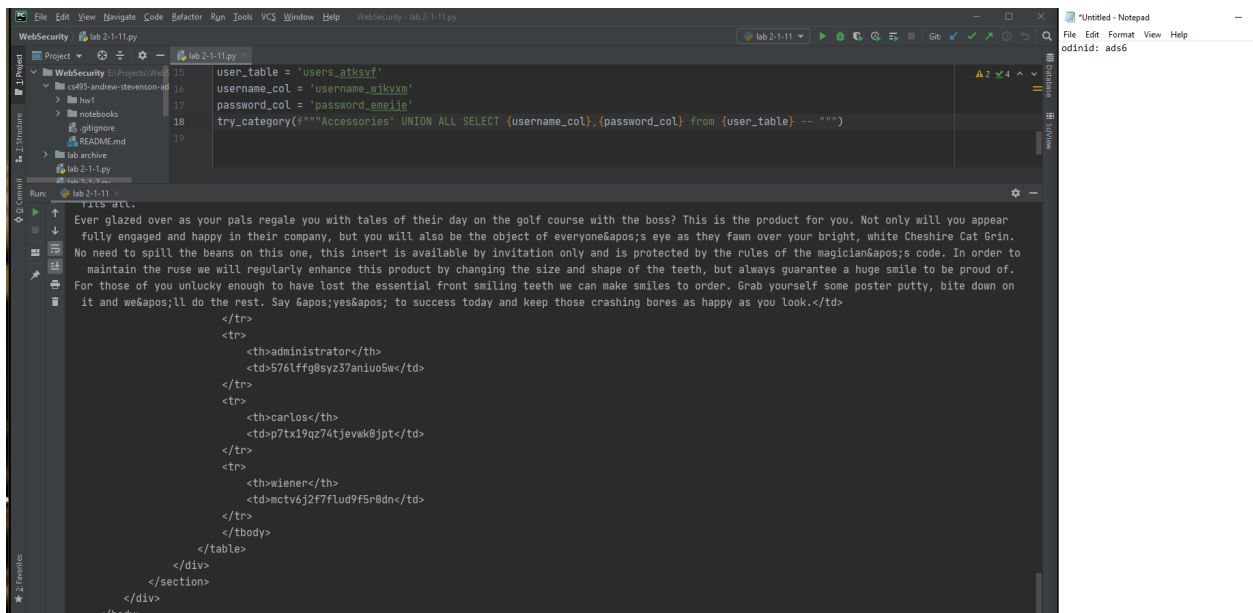
- Take a screenshot of the results showing the name of the user table



- Take a screenshot of the results showing the column names of the user table.



- Take a screenshot of the results showing all of the users and their passwords for the site.



- Take a screenshot showing completion of the level that includes your OdinId

Vulnerability: The same sql injection vulnerability as above permits the adversary to successively discover the names of all database tables, then the columns of all database tables, and finally the contents of a sensitive table.

Remediation: Pass the client input query string through a sqlencode function before forming the sql string.

