Identify vulnerabilities and remediation techniques

Vulnerability management overview

Vulnerability management is the process of identifying, assessing, and remediating vulnerabilities within systems and applications. It is a vital component of information security, as it helps to protect sensitive data from unauthorized access and potential breaches.

Key Points:

Identifying vulnerabilities early in the development process is generally more cost-effective than dealing with security breaches later.

Regularly scanning for vulnerabilities can help identify and address weaknesses before malicious attacks, thus mitigating potential threats proactively.

Vulnerability scanning provides insight into an application's attack surface, helping to understand potential avenues of exploitation and priorities critical areas for improvement.

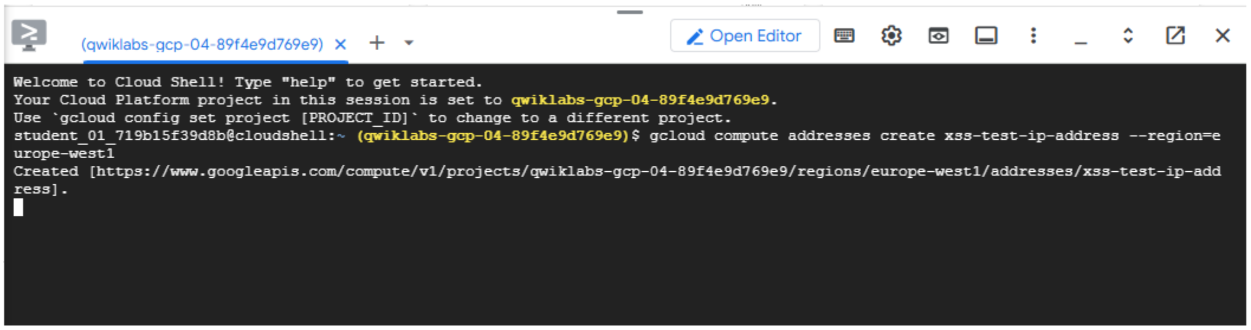
Setting Up Static IP and VM for XSS Testing

This section outlines the process of creating a static IP address and launching a virtual machine to run a vulnerable application for cross-site scripting (XSS) testing.

Creating a Static IP Address

1. Access the Google Cloud console and activate Cloud Shell.

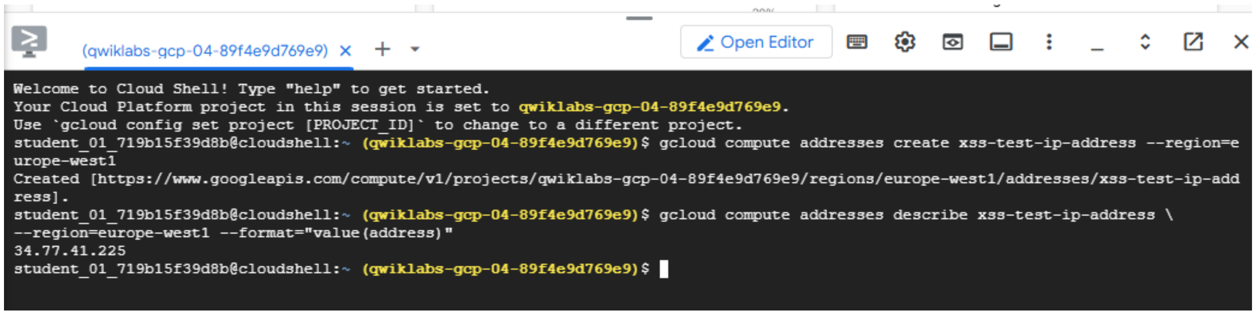
2. Execute the following command to create a static IP address:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-04-89f4e9d769e9.
Use `gcloud config set project [PROJECT_ID]` to change to a different project.
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$ gcloud compute addresses create xss-test-ip-address --region=e
urope-west1
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-04-89f4e9d769e9/regions/europe-west1/addresses/xss-test-ip-add
ress].
```

This command creates a static IP named `xss-test-ip-address` in the `us-west1` region.

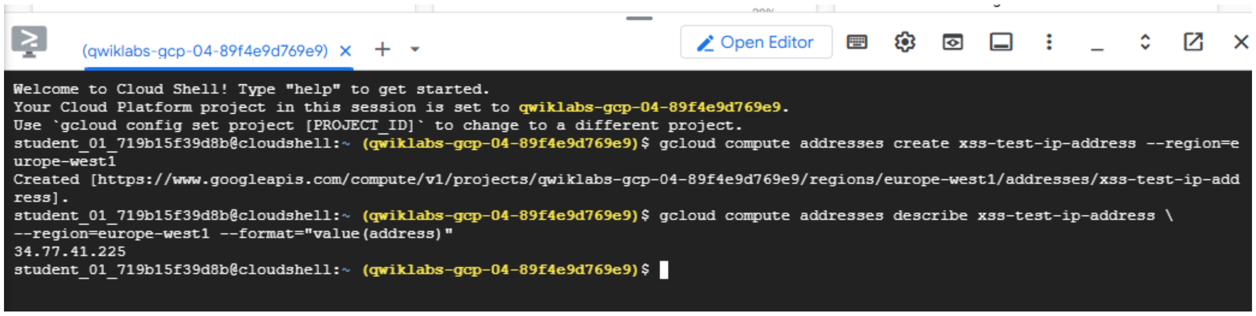
3. Retrieve the generated static IP address:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-04-89f4e9d769e9.
Use `gcloud config set project [PROJECT_ID]` to change to a different project.
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$ gcloud compute addresses create xss-test-ip-address --region=e
urope-west1
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-04-89f4e9d769e9/regions/europe-west1/addresses/xss-test-ip-add
ress].
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$ gcloud compute addresses describe xss-test-ip-address \
--region=europe-west1 --format="value(address)"
34.77.41.225
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$
```

3. Retrieve the generated static IP address

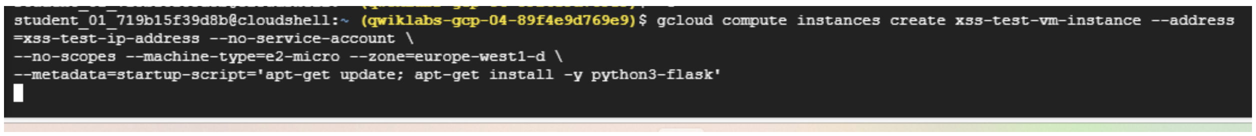
4. Note down the IP address displayed in the output for future use.



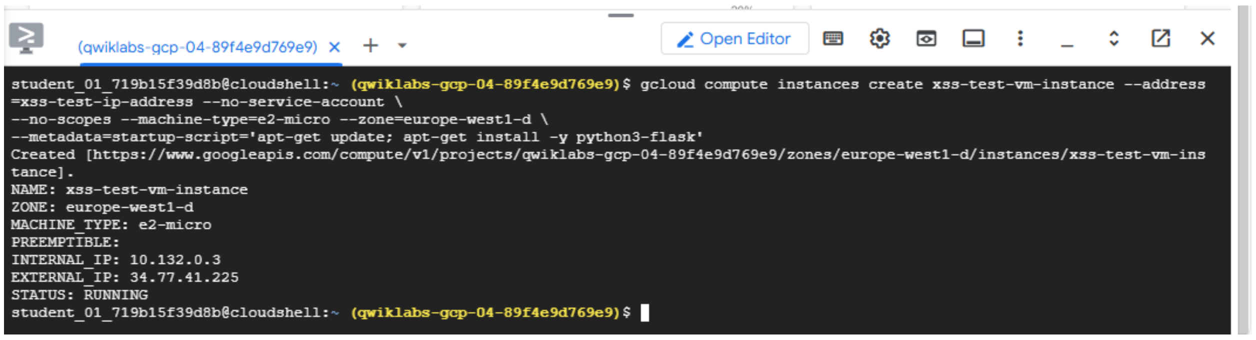
```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-04-89f4e9d769e9.
Use `gcloud config set project [PROJECT_ID]` to change to a different project.
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$ gcloud compute addresses create xss-test-ip-address --region=e
urope-west1
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-04-89f4e9d769e9/regions/europe-west1/addresses/xss-test-ip-add
ress].
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$ gcloud compute addresses describe xss-test-ip-address \
--region=europe-west1 --format="value(address)"
34.77.41.225
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$
```

Launching the Virtual Machine

1. Create a VM instance to run the vulnerable application:



```
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$ gcloud compute instances create xss-test-vm-instance --address
=xss-test-ip-address --no-service-account \
--no-scopes --machine-type=e2-micro --zone=europe-west1-d \
--metadata=startup-script='apt-get update; apt-get install -y python3-flask'
```



```
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$ gcloud compute instances create xss-test-vm-instance --address
=xss-test-ip-address --no-service-account \
--no-scopes --machine-type=e2-micro --zone=europe-west1-d \
--metadata=startup-script='apt-get update; apt-get install -y python3-flask'
Created [https://www.googleapis.com/compute/v1/projects/qwiklabs-gcp-04-89f4e9d769e9/zones/europe-west1-d/instances/xss-test-vm-ins
tance].
NAME: xss-test-vm-instance
ZONE: europe-west1-d
MACHINE_TYPE: e2-micro
PREEMPTIBLE:
INTERNAL_IP: 10.132.0.3
EXTERNAL_IP: 34.77.41.225
STATUS: RUNNING
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$
```

The command configures a virtual machine (VM) with specific attributes. The VM is named “xss-test-vm-instance” and utilises a pre-existing static IP address. There is no service account or scopes associated with the VM. It employs the “e2-micro” machine type and resides in the “us-west1-a” zone. The startup script for this VM involves the installation of Python3-Flask, a popular web framework for building web applications in Python. This setup enables the VM to host Flask-based applications. The startup script installs python-flask, a

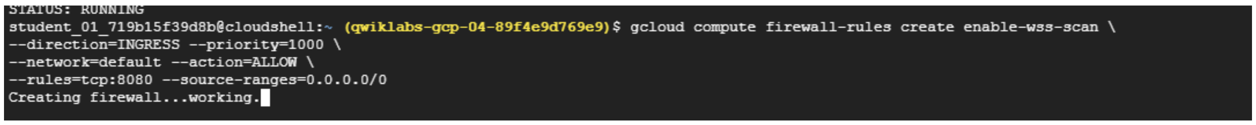
Web Application Framework used to run a simple Python application. This application is designed to demonstrate cross-site scripting (XSS) vulnerability, a common web application security issue.

Setting Up and Running the Vulnerable Application

This section outlines the process of downloading, extracting, and deploying a vulnerable web application for security testing purposes.

Creating a Firewall Rule

1. Execute the following command in the Cloud Shell terminal to create a firewall rule:

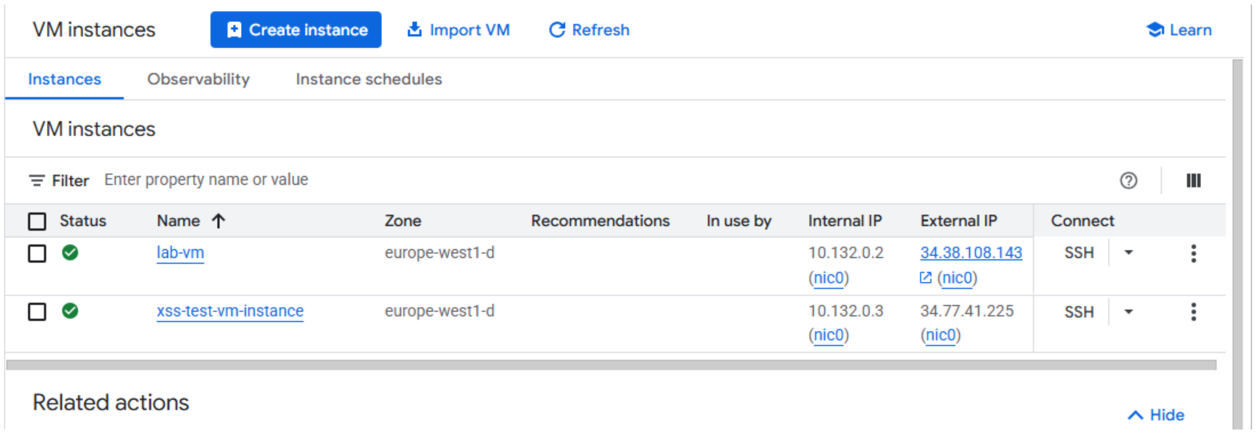


```
STATUS: RUNNING
student_01_719b15f39d8b@cloudshell:~ (qwiklabs-gcp-04-89f4e9d769e9)$ gcloud compute firewall-rules create enable-wss-scan \
--direction=INGRESS --priority=1000 \
--network=default --action=ALLOW \
--rules=tcp:8080 --source-ranges=0.0.0.0/0
Creating firewall...working.
```

This rule allows access to the web application from any source IP address, enabling Web Security Scanner to access and scan the vulnerable application.

Connecting to the VM Instance

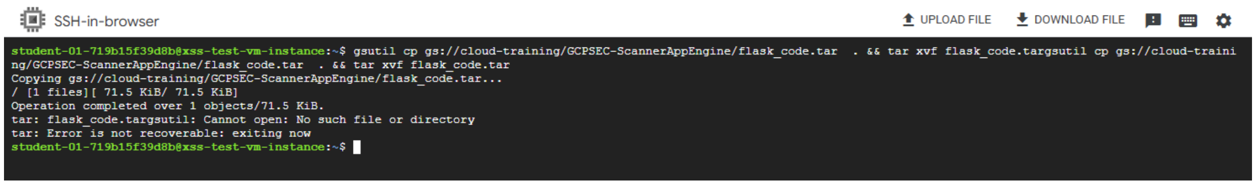
To initiate an SSH connection, begin by navigating to the Google Cloud console menu and selecting ‘Compute Engine’ followed by ‘VM instances’. On the subsequent page, identify your test instance within the list and click the ‘SSH’ button adjacent to it, located in the ‘Connect’ column. Should a prompt appear, grant the necessary authorization for the SSH in-browser connection to proceed.



Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	lab-vm	europe-west1-d			10.132.0.2 (nic0)	34.38.108.143 (nic0)	SSH
✓	xss-test-vm-instance	europe-west1-d			10.132.0.3 (nic0)	34.77.41.225 (nic0)	SSH

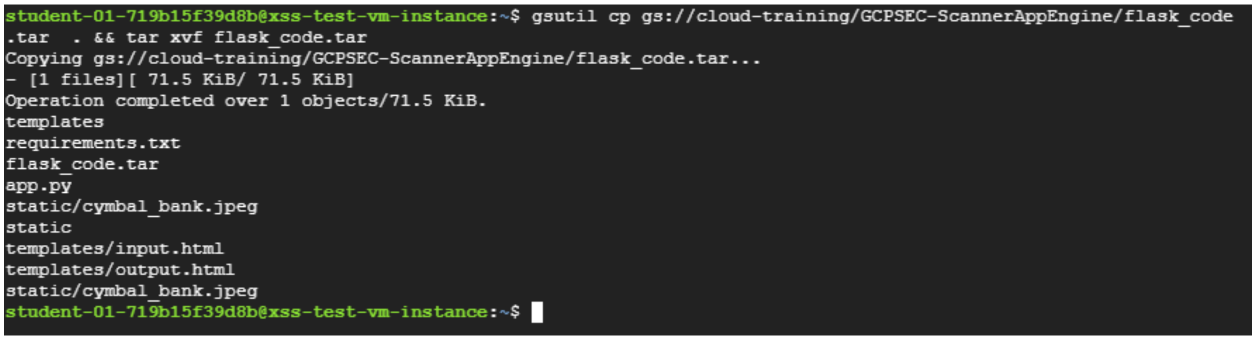
Extracting Web Application Files

1. In the SSH-in-browser window, run the following command:



```
student-01-719b15f39d8b@xss-test-vm-instance:~$ gsutil cp gs://cloud-training/GCPSEC-ScannerAppEngine/flask_code.tar  . && tar xvf flask_code.targsutil cp gs://cloud-traini
ng/GCPSEC-ScannerAppEngine/flask_code.tar  . && tar xvf flask_code.tar
Copying gs://cloud-training/GCPSEC-ScannerAppEngine/flask_code.tar...
/ [1 files][ 71.5 KiB/ 71.5 KiB]
Operation completed over 1 objects/71.5 KiB.
tar: flask_code.targsutil: Cannot open: No such file or directory
tar: Error is not recoverable: exiting now
student-01-719b15f39d8b@xss-test-vm-instance:~$
```

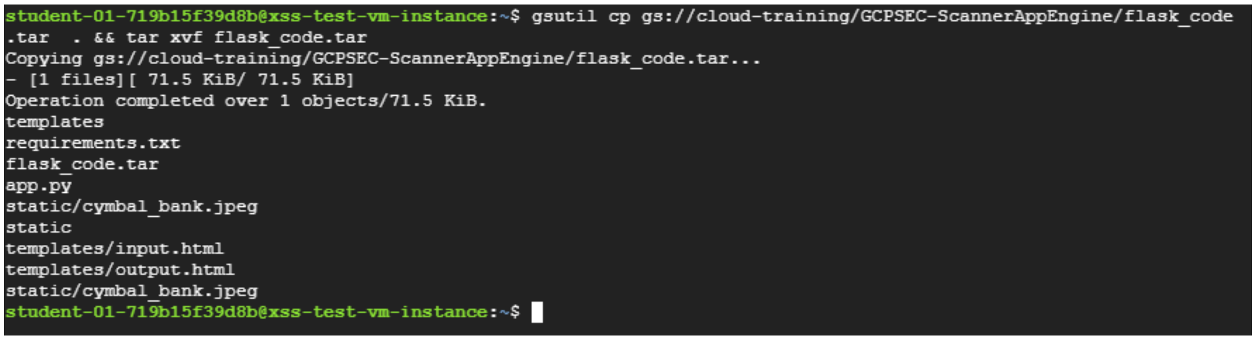
This command downloads and extracts the vulnerable web application files.



```
student-01-719b15f39d8b@xss-test-vm-instance:~$ gsutil cp gs://cloud-training/GCPSEC-ScannerAppEngine/flask_code
.tar  . && tar xvf flask_code.tar
Copying gs://cloud-training/GCPSEC-ScannerAppEngine/flask_code.tar...
- [1 files][ 71.5 KiB/ 71.5 KiB]
Operation completed over 1 objects/71.5 KiB.
templates
requirements.txt
flask_code.tar
app.py
static/cymbal_bank.jpeg
static
templates/input.html
templates/output.html
static/cymbal_bank.jpeg
student-01-719b15f39d8b@xss-test-vm-instance:~$
```

Running the Application

1. Start the application by executing:



```
student-01-719b15f39d8b@xss-test-vm-instance:~$ gsutil cp gs://cloud-training/GCPSEC-ScannerAppEngine/flask_code
.tar  . && tar xvf flask_code.tar
Copying gs://cloud-training/GCPSEC-ScannerAppEngine/flask_code.tar...
- [1 files][ 71.5 KiB/ 71.5 KiB]
Operation completed over 1 objects/71.5 KiB.
templates
requirements.txt
flask_code.tar
app.py
static/cymbal_bank.jpeg
static
templates/input.html
templates/output.html
static/cymbal_bank.jpeg
student-01-719b15f39d8b@xss-test-vm-instance:~$
```

A message should indicate that the application is now running.

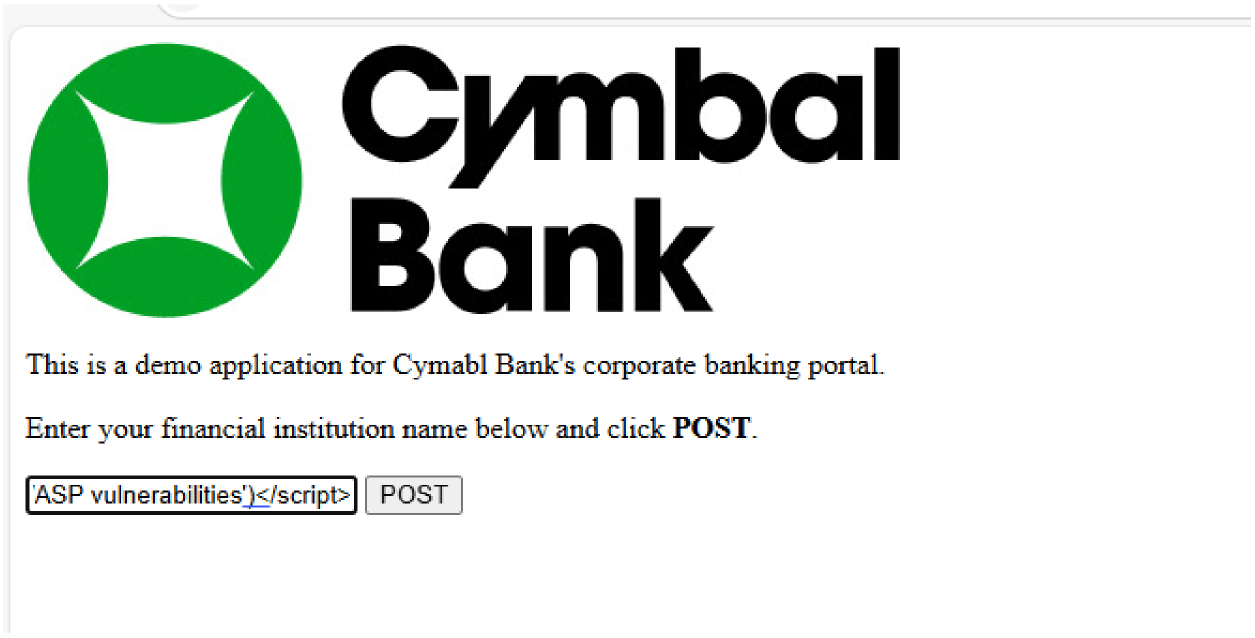
Setting up and running the vulnerable application involves several steps. First, create a firewall rule using Google Cloud Shell to allow Web Security Scanner access. Next, connect to the VM instance via SSH through the Google Cloud console. Once connected, download and extract the web application files using the provided gsutil command. Finally, start the application by running the Python script. It's crucial to note that this setup is for development and testing purposes only, as the application may contain vulnerabilities. The firewall rule allows access from any IP address, which is necessary for scanning but would be a security risk in a production environment. Remember to keep the SSH connection open while performing subsequent tasks to ensure the application continues running. This process enables you to deploy a vulnerable web application for security testing, but it's essential to exercise caution and not use this configuration in any public-facing or production scenarios without proper security measures.

Access the vulnerable application

Procedure

1. Access the Application

Open a new browser window while the application is running. Navigate to http://<YOUR_EXTERNAL_IP>:8080, replacing <YOUR_EXTERNAL_IP> with the static IP address of your VM from the previous task. Verify that the Cymbal Bank corporate banking portal with a web form appears.



This is a demo application for Cymabl Bank's corporate banking portal.

Enter your financial institution name below and click **POST**.

'ASP vulnerabilities')</script>

2. Inject Test Script

The following HTML code snippet could be used: <script>alert('This is an XSS Injection to demonstrate one of OWASP vulnerabilities')</script>. When injected into a susceptible web page, this JavaScript code executes within the user's browser, potentially triggering actions ranging from simple alert boxes (as in this example) to more malicious activities such as data theft, session hijacking, or website defacement. It's crucial to emphasize that this code should only be used in controlled, authorized environments and never for exploiting vulnerabilities on live systems.

3. Execute the Test

- Click the "POST" button to submit the form.

Section

Content

Expected Result

An alert window should appear with the message: “This is an XSS Injection to demonstrate one of OWASP vulnerabilities”

Analysis

The appearance of the alert demonstrates that the application is vulnerable to XSS attacks. While this particular injection is harmless, it illustrates how an attacker could potentially execute arbitrary JavaScript in the context of other users’ sessions.

Security Implications

* Data Theft: Attackers could use similar techniques to steal sensitive information from users.<br>* Session Hijacking: XSS vulnerabilities can be exploited to hijack user sessions.<br>* Malware Distribution: Malicious payloads could be injected to infect users’ devices.

Recommendation

Implement proper input validation and output encoding to prevent XSS vulnerabilities. Follow OWASP (Open Web Application Security Project) guidelines for secure web application development.

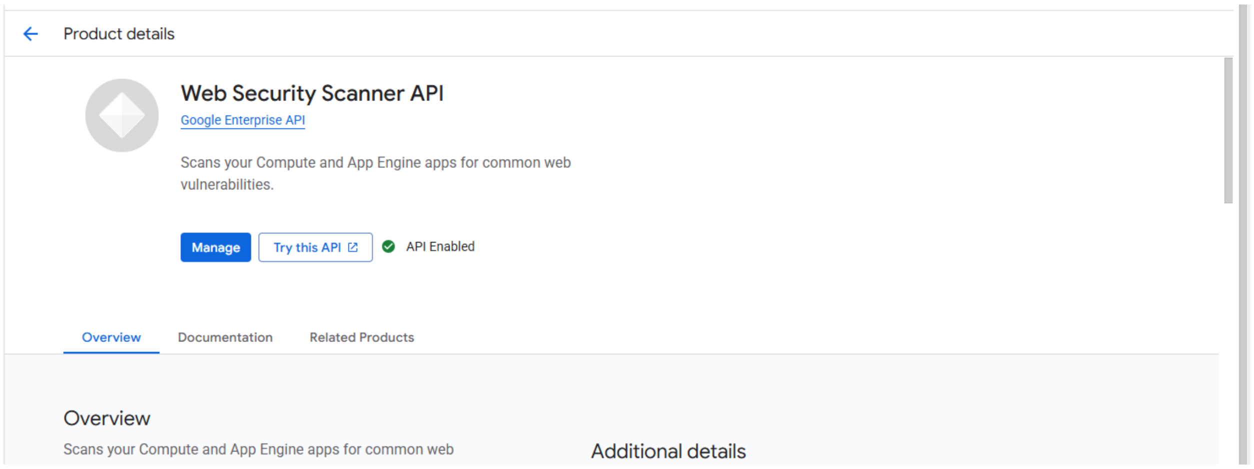
Web Security Scanner: Setup and Application Scanning Procedure

Now to remediate this issue, enable the Web Security Scanner API and conduct a vulnerability scan on the deployed application.

Procedure

Enabling Web Security Scanner API

To enable the Web Security Scanner API, start by navigating to the Google Cloud console. Next, access APIs & Services > Enabled APIs and services. Then, click + Enable APIs and services. Search for “Web Security Scanner” and select Web Security Scanner API. Finally, click Enable to complete the process.


Product details

Web Security Scanner API

Google Enterprise API

Scans your Compute and App Engine apps for common web vulnerabilities.

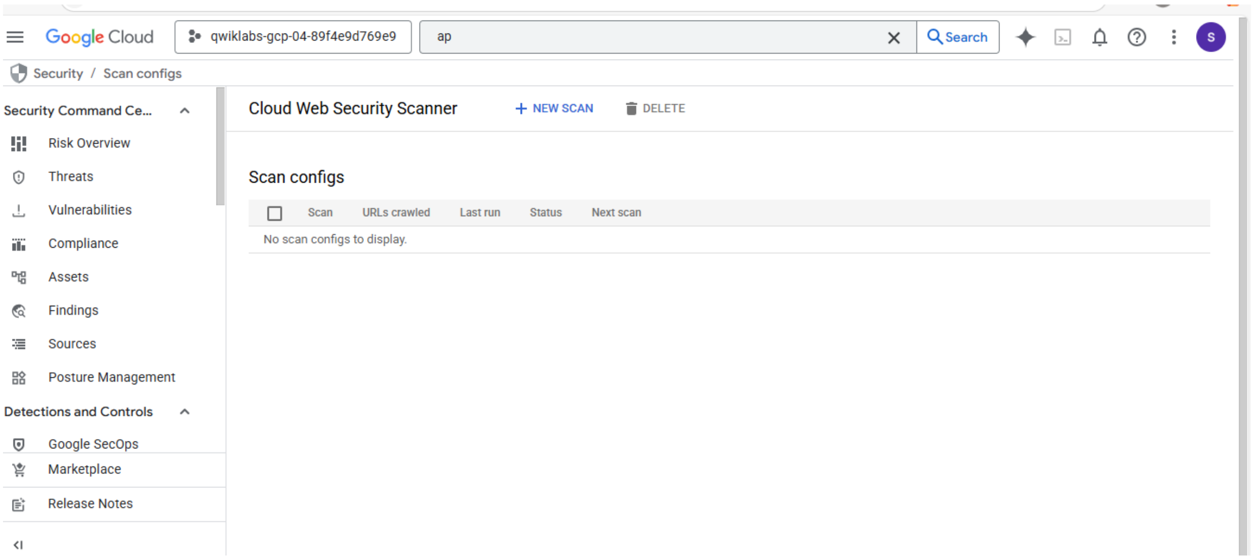
Overview

Scans your Compute and App Engine apps for common web

Additional details

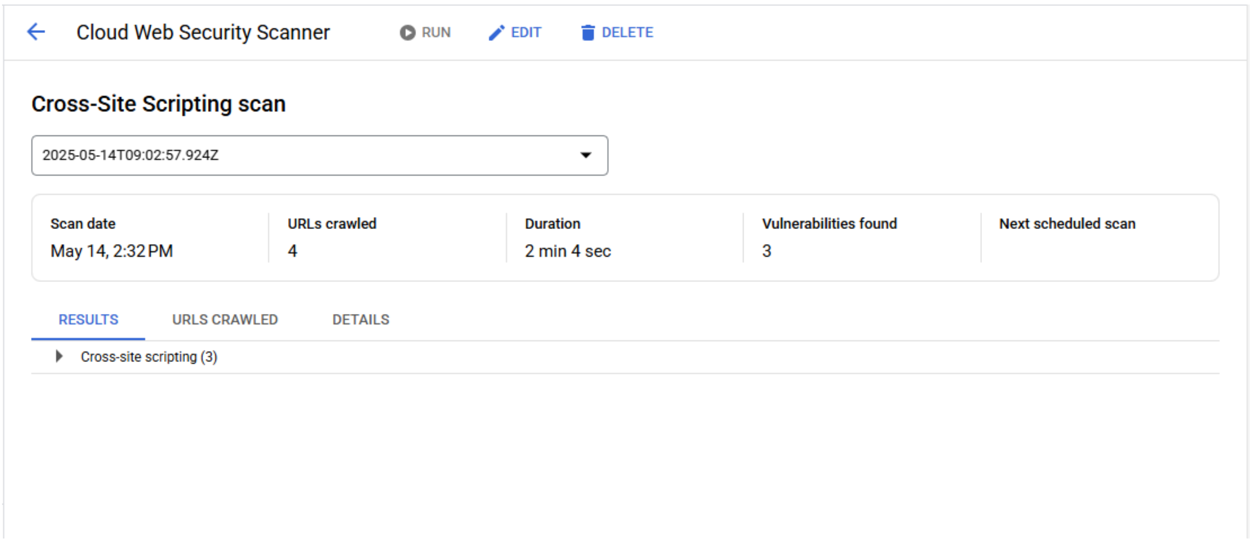
Configuring and Running the Scan

In the Google Cloud console, navigate to Security > Web Security Scanner, click "+ New scan" in the toolbar, and configure the scan as follows: Name: "Cross-Site Scripting scan", Starting URL: `http://<EXTERNAL_IP>:8080` (Replace `<EXTERNAL_IP>` with your VM's static IP), Authentication: None, Schedule: Never. Click "Save" to create the scan configuration, then click "Run Scan" to initiate the scanning process.



Scan configs

	Scan	URLs crawled	Last run	Status	Next scan
No scan configs to display.					



Cross-Site Scripting scan

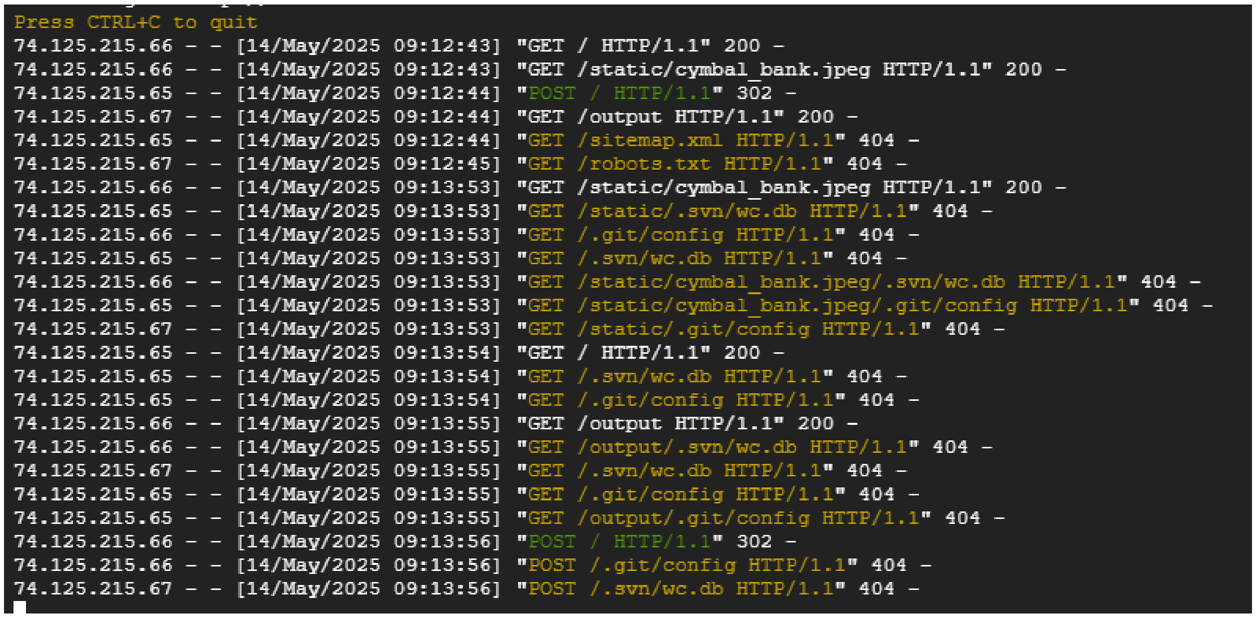
2025-05-14T09:02:57.924Z

Scan date	URLs crawled	Duration	Vulnerabilities found	Next scheduled scan
May 14, 2:32 PM	4	2 min 4 sec	3	

Cross-site scripting (3)

Monitoring the Scan

During an SSH-in-browser scan, the scanner analyses different URLs to identify potential vulnerabilities. As the scan progresses, you can observe log generation in the SSH-in-browser window. The scanning process usually takes around 5 to 10 minutes to complete, providing valuable insights into the security posture of the web application being tested.



```
Press CTRL+C to quit
74.125.215.66 - - [14/May/2025 09:12:43] "GET / HTTP/1.1" 200 -
74.125.215.66 - - [14/May/2025 09:12:43] "GET /static/cymbal_bank.jpeg HTTP/1.1" 200 -
74.125.215.65 - - [14/May/2025 09:12:44] "POST / HTTP/1.1" 302 -
74.125.215.67 - - [14/May/2025 09:12:44] "GET /output HTTP/1.1" 200 -
74.125.215.65 - - [14/May/2025 09:12:44] "GET /sitemap.xml HTTP/1.1" 404 -
74.125.215.67 - - [14/May/2025 09:12:45] "GET /robots.txt HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:53] "GET /static/cymbal_bank.jpeg HTTP/1.1" 200 -
74.125.215.65 - - [14/May/2025 09:13:53] "GET /static/.svn/wc.db HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:53] "GET /.git/config HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:53] "GET /.svn/wc.db HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:53] "GET /static/cymbal_bank.jpeg/.svn/wc.db HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:53] "GET /static/cymbal_bank.jpeg/.git/config HTTP/1.1" 404 -
74.125.215.67 - - [14/May/2025 09:13:53] "GET /static/.git/config HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:54] "GET / HTTP/1.1" 200 -
74.125.215.65 - - [14/May/2025 09:13:54] "GET /.svn/wc.db HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:54] "GET /.git/config HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:54] "GET /output HTTP/1.1" 200 -
74.125.215.66 - - [14/May/2025 09:13:55] "GET /output/.svn/wc.db HTTP/1.1" 404 -
74.125.215.67 - - [14/May/2025 09:13:55] "GET /.svn/wc.db HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:55] "GET /.git/config HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:55] "GET /output/.git/config HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:56] "POST / HTTP/1.1" 302 -
74.125.215.66 - - [14/May/2025 09:13:56] "POST /.git/config HTTP/1.1" 404 -
74.125.215.67 - - [14/May/2025 09:13:56] "POST /.svn/wc.db HTTP/1.1" 404 -
```

Reviewing Results

1. Once complete, return to the Google Cloud console.

2. Review the **Results** tab for identified cross-site vulnerabilities.

3. Additional vulnerability details can be found in the **Vulnerabilities** tab under the **Security Command Centre**.

Expected Outcome

The scan should detect and report cross-site scripting (XSS) vulnerabilities in the application.

Notes

- Ensure the application remains running in the SSH-in-browser window throughout the scanning process.

- The scanning process may take 5–10 minutes to complete.

- This procedure demonstrates how Web Security Scanner can effectively identify XSS vulnerabilities in web applications.

Security Implications

Identifying vulnerabilities through automated scanning is a crucial step in maintaining application security. However, it's important to note that while automated tools are valuable, they should be complemented with manual testing and code review for comprehensive security assessment.

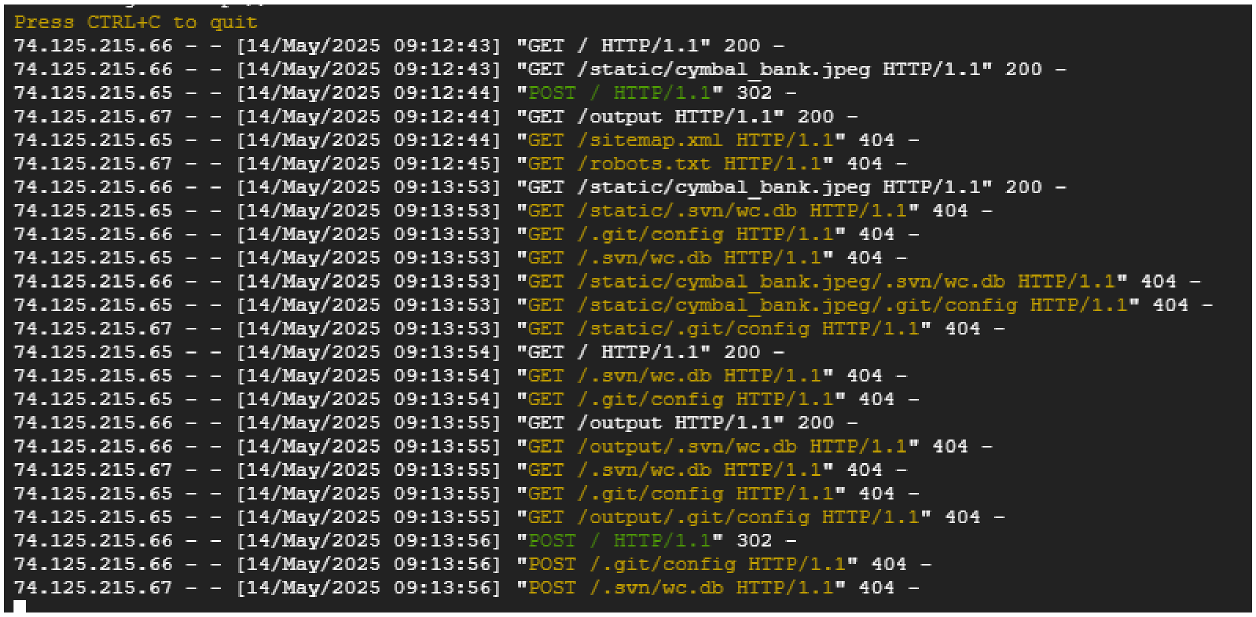
Remediating Cross-Site Scripting (XSS) Vulnerability

To address the identified XSS vulnerability in the application by implementing input validation and output encoding.

Procedure

Stopping the Current Application

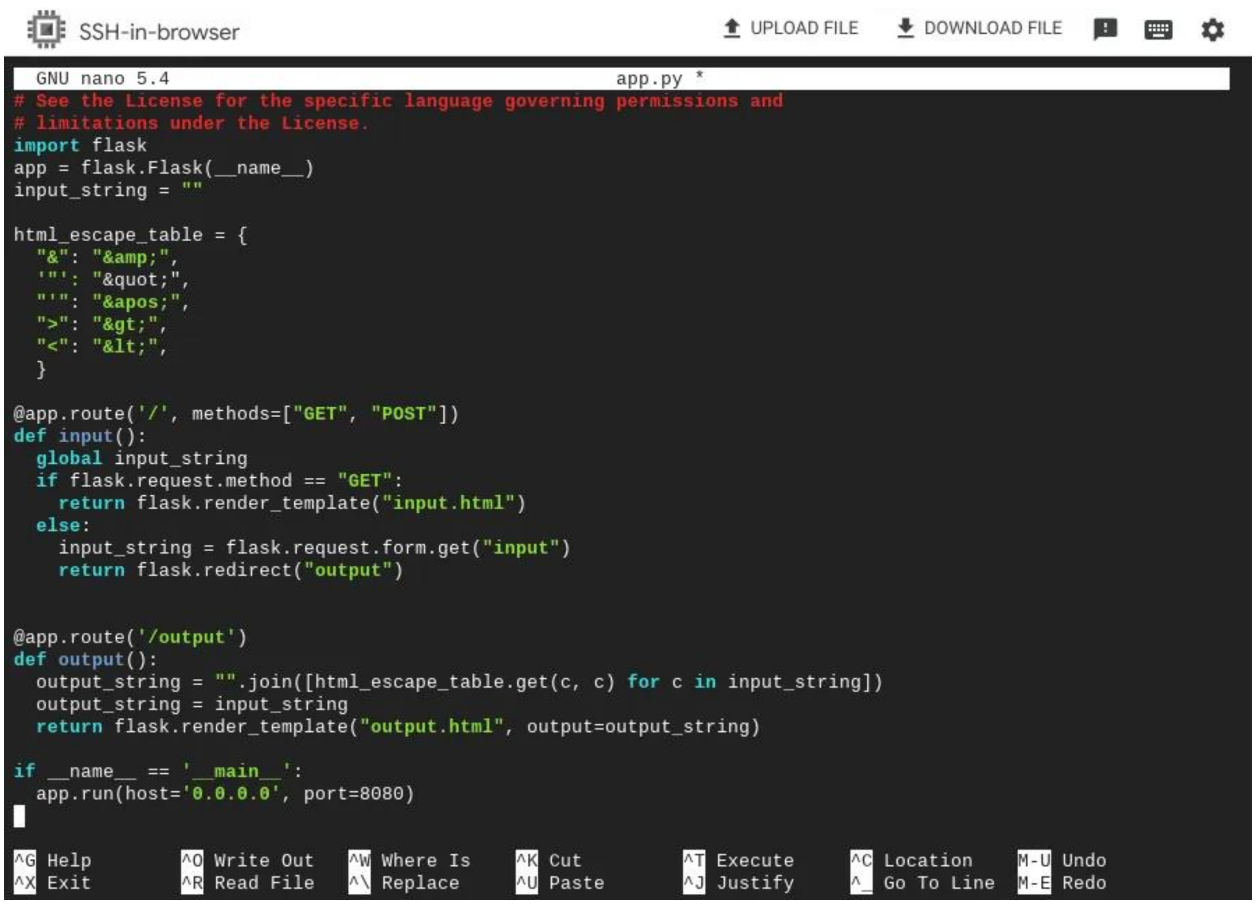
After connecting to your VM instance through the SSH-in-browser page, you can stop the currently running application. To do this, you have two options. You can either press the CTRL and C keys simultaneously on your keyboard, or you can use the "Send key combination" icon to input the CTRL and C key combination. Once you have stopped the application, you can proceed with the next steps of your task.



```
Press CTRL+C to quit
74.125.215.66 - - [14/May/2025 09:12:43] "GET / HTTP/1.1" 200 -
74.125.215.66 - - [14/May/2025 09:12:43] "GET /static/cymbal_bank.jpeg HTTP/1.1" 200 -
74.125.215.65 - - [14/May/2025 09:12:44] "POST / HTTP/1.1" 302 -
74.125.215.67 - - [14/May/2025 09:12:44] "GET /output HTTP/1.1" 200 -
74.125.215.65 - - [14/May/2025 09:12:44] "GET /sitemap.xml HTTP/1.1" 404 -
74.125.215.67 - - [14/May/2025 09:12:45] "GET /robots.txt HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:53] "GET /static/cymbal_bank.jpeg HTTP/1.1" 200 -
74.125.215.65 - - [14/May/2025 09:13:53] "GET /static/.svn/wc.db HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:53] "GET /.git/config HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:53] "GET /.svn/wc.db HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:53] "GET /static/cymbal_bank.jpeg/.svn/wc.db HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:53] "GET /static/cymbal_bank.jpeg/.git/config HTTP/1.1" 404 -
74.125.215.67 - - [14/May/2025 09:13:53] "GET /static/.git/config HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:54] "GET / HTTP/1.1" 200 -
74.125.215.65 - - [14/May/2025 09:13:54] "GET /.svn/wc.db HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:54] "GET /.git/config HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:55] "GET /output HTTP/1.1" 200 -
74.125.215.66 - - [14/May/2025 09:13:55] "GET /output/.svn/wc.db HTTP/1.1" 404 -
74.125.215.67 - - [14/May/2025 09:13:55] "GET /.svn/wc.db HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:55] "GET /.git/config HTTP/1.1" 404 -
74.125.215.65 - - [14/May/2025 09:13:55] "GET /output/.git/config HTTP/1.1" 404 -
74.125.215.66 - - [14/May/2025 09:13:56] "POST / HTTP/1.1" 302 -
74.125.215.66 - - [14/May/2025 09:13:56] "POST /.git/config HTTP/1.1" 404 -
74.125.215.67 - - [14/May/2025 09:13:56] "POST /.svn/wc.db HTTP/1.1" 404 -
```

Editing the Application Code

In order to implement input validation, the code in the nano editor needs to be modified. First, the “#” symbol needs to be removed from the first line and added to the beginning of the second line. Proper indentation must also be ensured. The modified code should look like this:



```
  GNU nano 5.4                                   app.py *
# See the License for the specific language governing permissions and
# limitations under the License.
import flask
app = flask.Flask(__name__)
input_string = ""

html_escape_table = {
  "&": "&amp;",
  '"': "&quot;",
  "'": "&apos;",
  ">": "&gt;",
  "<": "&lt;",
  }

@app.route('/', methods=["GET", "POST"])
def input():
  global input_string
  if flask.request.method == "GET":
    return flask.render_template("input.html")
  else:
    input_string = flask.request.form.get("input")
    return flask.redirect("output")


@app.route('/output')
def output():
  output_string = "".join([html_escape_table.get(c, c) for c in input_string])
  output_string = input_string
  return flask.render_template("output.html", output=output_string)

if __name__ == '__main__':
  app.run(host='0.0.0.0', port=8080)

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line M-E Redo
```

Once the code has been modified, the nano editor can be saved and exited by pressing “CTRL + X”, followed by “Y” to confirm saving changes, and then pressing “ENTER” to save.

Restarting the Application

To implement the latest modifications made to your application, you can restart it by running the command “python3 app.py” in your terminal or command prompt. This command will initiate the application’s execution with the updated code, allowing you to observe and test the effects of your changes. Executing this command ensures that the application incorporates the new features or bug fixes you’ve implemented, enabling you to evaluate their impact and functionality within the application’s environment.

Explanation of the Fix

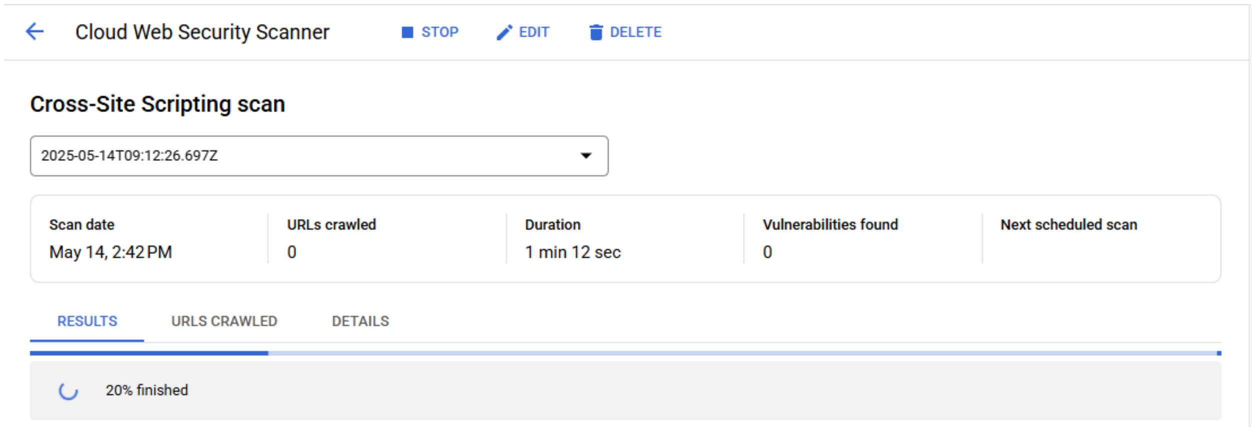
The implemented fix uses a technique called “output encoding” to prevent XSS attacks. The html_escape_table dictionary is used to replace special characters with their HTML-encoded equivalents. This ensures that any user input containing HTML or script tags will be rendered as plain text, rather than being interpreted as code by the browser.

Security Implications

The fix implemented aims to mitigate an immediate cross-site scripting (XSS) vulnerability. It prevents user-supplied HTML and JavaScript from being executed within other users' sessions, thus enhancing security. However, maintaining a secure application requires ongoing efforts. Regular security audits and updates are crucial to ensuring the application remains secure and protected against potential vulnerabilities or threats over time.

Best Practices

To safeguard your web application from potential security vulnerabilities, it is imperative to adhere to several key practices. Firstly, always validate and sanitise user input on the server-side to prevent malicious code or unwanted data from entering your system. Secondly, implement context-specific output encoding for all user-controlled data to thwart cross-site scripting and other injection attacks. Thirdly, follow the tenet of least privilege in application design, granting only the essential permissions and access levels to users and components. Finally, keep all libraries and frameworks up-to-date with the latest security patches to address newly discovered vulnerabilities and maintain a robust defence against cyber threats. After implementing this fix, it's recommended to re-run the Web Security Scanner to verify that the XSS vulnerability has been successfully addressed.



Cross-Site Scripting scan

2025-05-14T09:12:26.697Z

Scan date	URLs crawled	Duration	Vulnerabilities found	Next scheduled scan
May 14, 2:42 PM	0	1 min 12 sec	0	

20% finished

Conclusion

This hands-on lab provided a comprehensive overview of vulnerability management within Google Cloud Platform. By deploying a vulnerable application and leveraging the Web Security Scanner, we effectively identified and remediated a cross-site scripting (XSS) vulnerability. This exercise emphasized the critical role of proactive security measures in safeguarding web applications from potential threats. The lab underscored the importance of continuous vigilance in identifying and addressing vulnerabilities, highlighting the value of automated tools like the Web Security Scanner and the necessity of implementing secure coding practices such as input validation and output encoding. By combining these strategies, organizations can strengthen their security posture and protect sensitive data from unauthorized access.