OWASP ZAP Script Documentation

Purpose of the Scripts

OWASP ZAP (Zed Attack Proxy) is a widely used open-source web application security scanner. The provided scripts automate common tasks such as spidering a website, performing active scans, and exporting alerts. These scripts are designed to help security teams streamline their processes and identify vulnerabilities efficiently.

The first script, Spider Scan, initiates a crawl of a target URL to discover all linked pages and resources. The second script, Active Scan, conducts a vulnerability scan on a target URL to identify potential security weaknesses. The third script, Export Alerts to CSV, extracts identified vulnerabilities from OWASP ZAP and exports them into a CSV file for analysis and reporting.

Usage Instructions

General Setup

1. Ensure OWASP ZAP is installed and running. You can download it from OWASP ZAP's official site.
2. Start ZAP and set up its API by navigating to Tools > Options > API in the ZAP UI.
3. Obtain the API key from the ZAP settings.
4. Install Python and the requests library if not already installed:
5. pip install requests
6. Save each script as a .py file and run them from the command line or any Python IDE.

Script 1: Spider Scan

Command:

python spider_scan.py

Required Parameters:

- ✓ target_url: The URL of the website to spider (e.g., http://example.com).
- ✓ zap_api_url: The base URL for ZAP's API (e.g., http://localhost:8080).
- ✓ api_key: The API key for authenticating with ZAP.

Script 2: Active Scan

Command:

python active_scan.py

Required Parameters:

- ✓ target_url: The URL of the website to scan (e.g., http://example.com).
- ✓ zap_api_url: The base URL for ZAP's API (e.g., http://localhost:8080).
- ✓ api_key: The API key for authenticating with ZAP.

Script 3: Export Alerts to CSV

Command:

python export_alerts.py

Required Parameters:

- ✓ zap_api_url: The base URL for ZAP's API (e.g., http://localhost:8080).
- ✓ api_key: The API key for authenticating with ZAP.
- ✓ output_file: The path to save the CSV file (e.g., zap_alerts.csv).

Expected Outputs or Results

Spider Scan

- ✓ OWASP ZAP crawls the target URL and identifies all linked pages and resources.
- ✓ Progress is displayed in the terminal, and details can be viewed in the ZAP UI.

Active Scan

- ✓ A detailed vulnerability scan of the target URL.
- ✓ Progress is displayed in the terminal, and results can be viewed in the ZAP UI.

Export Alerts to CSV

- ✓ Extracts all alerts identified by ZAP and saves them in a structured CSV file, including details like alert name, risk level, and affected URL.

Dependencies Needed

- ✓ Python: Required to run the scripts.
- ✓ Requests Library: For making API calls to ZAP. Install it using:
- ✓ pip install requests

Line-by-Line Explanation

Spider Scan

```
import requests
import time
```

- requests: Used to interact with the ZAP API.
- time: Used for pausing between progress checks.

```
def start_spider_scan(target_url, zap_api_url, api_key):
```

- This function starts a spider scan for the target URL.

```
spider_url = f"{zap_api_url}/JSON/spider/action/scan/?url={target_url}&apikey={api_key}"
response = requests.get(spider_url)
```

- Constructs the API URL for initiating a spider scan and sends a GET request to start the scan.

```
scan_id = response.json().get('scan')
print(f"Spider scan started with ID: {scan_id}")
```

- Retrieves the scan ID from the response and prints it.

```
progress_url = f"{zap_api_url}/JSON/spider/view/status/?scanId={scan_id}&apikey={api_key}"
while True:
    progress_response = requests.get(progress_url)
    progress = int(progress_response.json().get('status'))
    print(f"Spider scan progress: {progress}%")
    if progress == 100:
        break
    time.sleep(5)
```

- Periodically checks the scan's progress and displays it. The loop exits when the scan reaches 100%.

Active Scan

```
def start_active_scan(target_url, zap_api_url, api_key):
```

- This function initiates an active scan for the target URL.

```
scan_url = f"{zap_api_url}/JSON/ascan/action/scan/?url={target_url}&apikey={api_key}"
response = requests.get(scan_url)
scan_id = response.json().get('scan')
```

- Constructs the API URL for starting an active scan and retrieves the scan ID from the response.

```
progress_url = f"{zap_api_url}/JSON/ascan/view/status/?scanId={scan_id}&apikey={api_key}"
while True:
    progress_response = requests.get(progress_url)
    progress = int(progress_response.json().get('status'))
    print(f"Active scan progress: {progress}%")
    if progress == 100:
        break
    time.sleep(5)
```

- Similar to the spider scan, this section monitors the progress of the active scan and displays it.

Export Alerts to CSV

```
import csv
```

- csv: Used to create and write to the CSV file.

```
def export_alerts_to_csv(zap_api_url, api_key, output_file):
```

- This function exports ZAP alerts to a CSV file.

```
alerts_url = f"{zap_api_url}/JSON/alert/view/alerts/?apikey={api_key}"
response = requests.get(alerts_url)
alerts = response.json()['alerts']
```

- Sends a request to fetch all alerts from ZAP and parses the JSON response.

```
with open(output_file, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Alert', 'Risk', 'URL', 'Description'])
    for alert in alerts:
        writer.writerow([alert['alert'], alert['risk'], alert['url'], alert['description']])
```

- Writes the alerts to a CSV file, including columns for alert name, risk, URL, and description.

```
print(f"Alerts exported to {output_file}")
```

- Confirms that the alerts have been successfully exported.

Why These Scripts Are Needed ?

Automating OWASP ZAP tasks ensures that web application security assessments are performed consistently and efficiently. The Spider Scan helps identify the full attack surface by discovering all linked pages and resources. The Active Scan identifies vulnerabilities within these pages, providing actionable insights to improve security. The Export Alerts to CSV script makes it easy to analyze and share findings with stakeholders, enhancing collaboration and reporting.