Testing with OWASP ZAP Python API

This tutorial will cover using the ZAP Python API to spider and scan an application.

Spidering involves ZAP visiting a known URL, crawlng that page for links to other pages and visiting them. ZAP repeats the same process on the newly found pages.

Active Scanning progresses from spidering and involves ZAP visiting all the URLs found through spidering and actively trying to exploit vulnerabilities.

The ZAP Python API is a useful tool for automating security testing with ZAP.

The Python implementation to access the OWASP ZAP API. For more information about OWASP ZAP consult the OWASP ZAP project.

In this tutorial

In this tutorial you will use the ZAP Python API to spider and scan an application.

1. Obtain an example script from the ZAP script library
2. Run a spider and scanning script
3. Report the results

Prerequisites

- ZAP Python API To install do either of the following:
- Download the latest Python implementation from PyPi

NOTE: the 2.4 package name has been kept to make it easier to upgrade)

- Run the following command:

```
pip install python-owasp-zap-v2.4
```

Obtain an example python script

This script is ideal for standalone security testing or as a way to spider and scan after using ZAP as a proxy for your functional tests.

Obtain the script test_zap.py

The script is configured via the config file test_zap.config

Run a spider and scanning script

1. Start up ZAP with the following bash script:

```
/opt/zap/zap.sh –daemon –config api.key=“123456” –port $port & while $(!
netstat –anp | grep $port | grep LISTEN ); do    if [[ $counter = 300 ]];
then        exit 1;    fi;    echo “sleeping $counter”;
counter=$((counter+1));    sleep 1s; done echo “done sleeping”;
```
2. Start by loading the required modules:

```
#!/usr/bin/env python

import time
from pprint import pprint
from zapv2 import ZAPv2
```

1. Define the target to scan

```
target = 'http://127.0.0.1'
```

1. Change the API key to match the one set in ZAP, or use None if the API key is disabled.

2. Now, instantiate the zap instance as follows.

```
zap = ZAPv2(apikey=apikey)
```

By default ZAP API client will connect to port 8080

If ZAP is listening to a non-default port, for example, if ZAP is listening on port 8090 then use the following:

```
zap = ZAPv2(apikey=apikey, proxies={'http': 'http://127.0.0.1:8090', 'https':
'http://127.0.0.1:8090'})
```

1. Set the target and start a session

```
print 'Accessing target %s' % target
zap.urlopen(target)
```

1. Give the sites tree a while to get updated

```
time.sleep(2)
print 'Spidering target %s' % target
scanid = zap.spider.scan(target)
```

1. The Spider takes a few moments to start

```
time.sleep(2)
while (int(zap.spider.status(scanid)) < 100):
    print 'Spider progress %: ' + zap.spider.status(scanid)
    time.sleep(2)

    print 'Spider completed'
```

The Spider is completed and the Active Scan begins.

1. The Active Scan takes a few moments to finish

```
time.sleep(5)
print 'Scanning target %s' % target
scanid = zap.ascan.scan(target)
while (int(zap.ascan.status(scanid)) < 100):
    print 'Scan progress %: ' + zap.ascan.status(scanid)
    time.sleep(5)

print 'Scan completed'
```

The Spider and the Active Scan are completed.

Report the results

A report can be generated of alerts associated with each identified vulnerability.

```
print 'Hosts: ' + ', '.join(zap.core.hosts)
print 'Alerts: '
pprint (zap.core.alerts())
```

Conclusion

Use the report to identify and fix vulnerabilities then perform the test again to check the fix. Bear in mind that spidering and scanning can take a long time if you have a large application. For this reason, it can be ideal to configure testing to run overnight.

Resources

For help using OWASP ZAP API refer to:

Examples - collection of examples using the library

Wiki

OWASP ZAP User Group - for asking questions