2021

Notes Magazine #05

by Cody Sixteen

1/11/2021

Hello World

Hi ;]

„New Year – New Me" – isn't it? ;) So below let's jump directly to 2021 with the quick summary of the 'noted-articles' I prepared for you (and for a *„future me*" of course ;))

Are you ready...?

First part is related to my adventures related to routers, IoT and other „network appliances" we can find online during our pentest activities. Few similar you can find already published on the blog.

Second part is related to the introduction I made for my self when I was learning Rust for a very first time.

Third part is dedicated to the Rosie. We'll try to talk in the same language.

I hope you'll find it useful. ;)

Here we go...

Contents

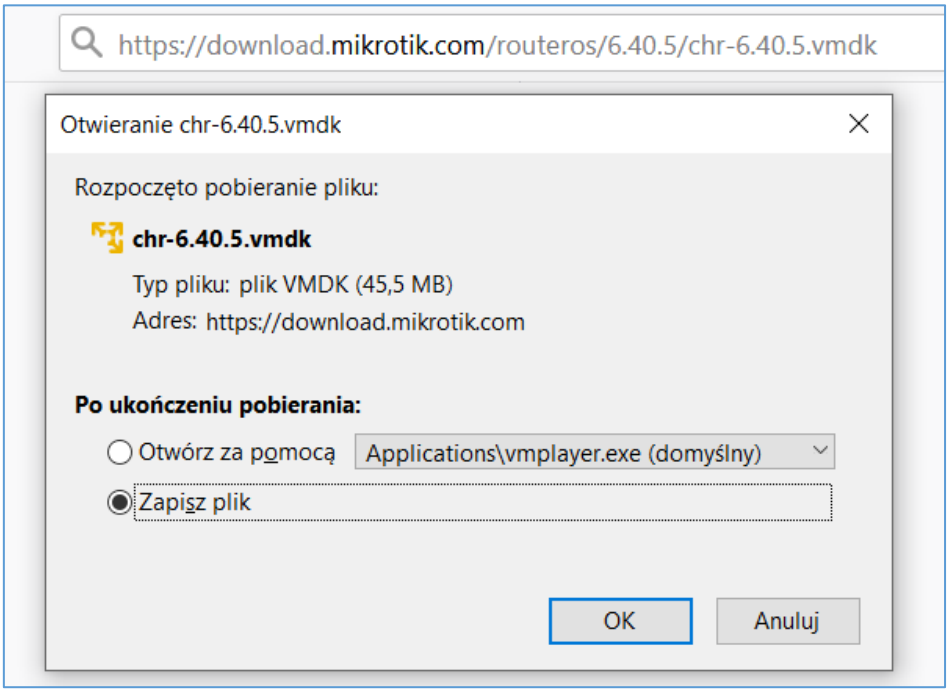
Automated Route(r) Learning

Intro

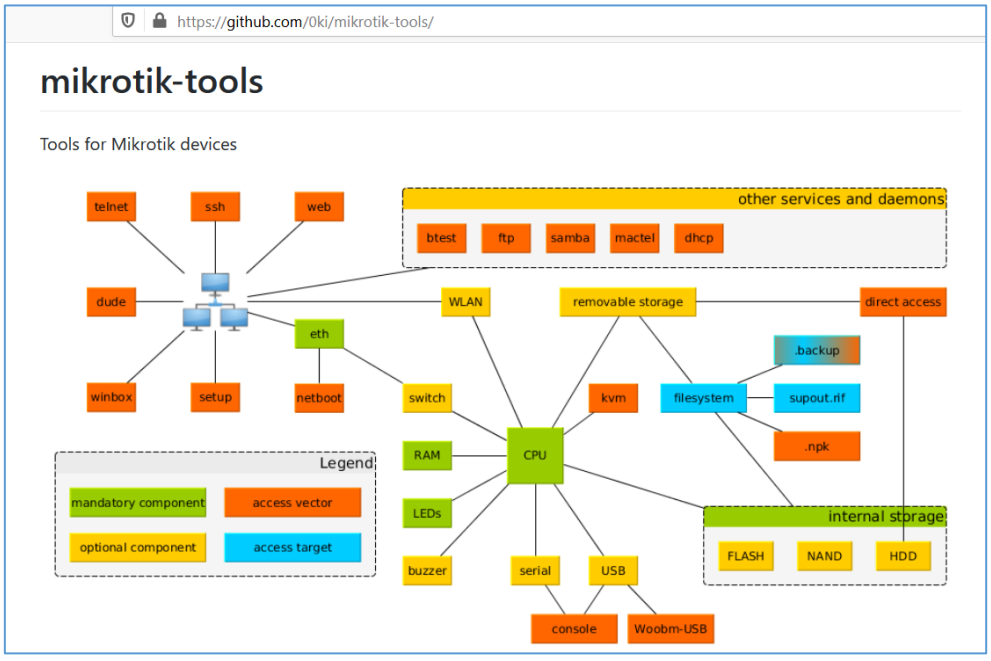
Some time ago I started learning about the routers[1, 2] and other network appliances[3, 4]. Last week I found a very interesting article[5] about „routers security" so I decided to check it. Below we'll try to recreate the „lab" described in the article. It should help us to start doing our own research „related to the routers". Let's go...

Recreating the Lab for MikroTik

According to the article[5] – we should start here[6]:



Next „few minutes" I spent on preparing my Ubuntu 18 VM as well as the Kali VM (I used 2019 version). When all is ready we can continue here[7] (or here[8]):

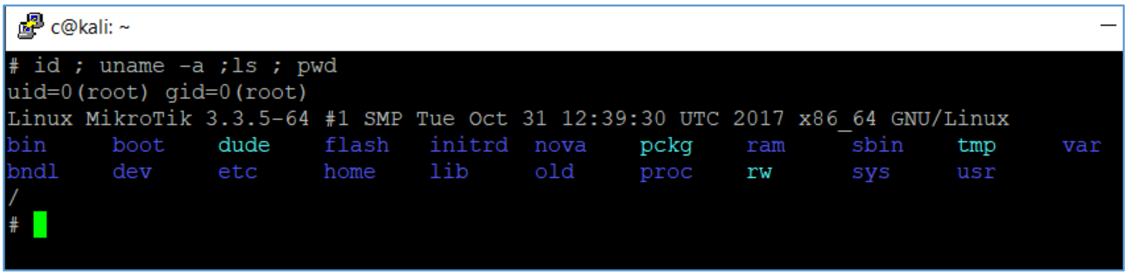


(I used the first tool mentioned above.) Your results should be pretty similar to the one presented below – achieving root shell on our MikroTik VM:

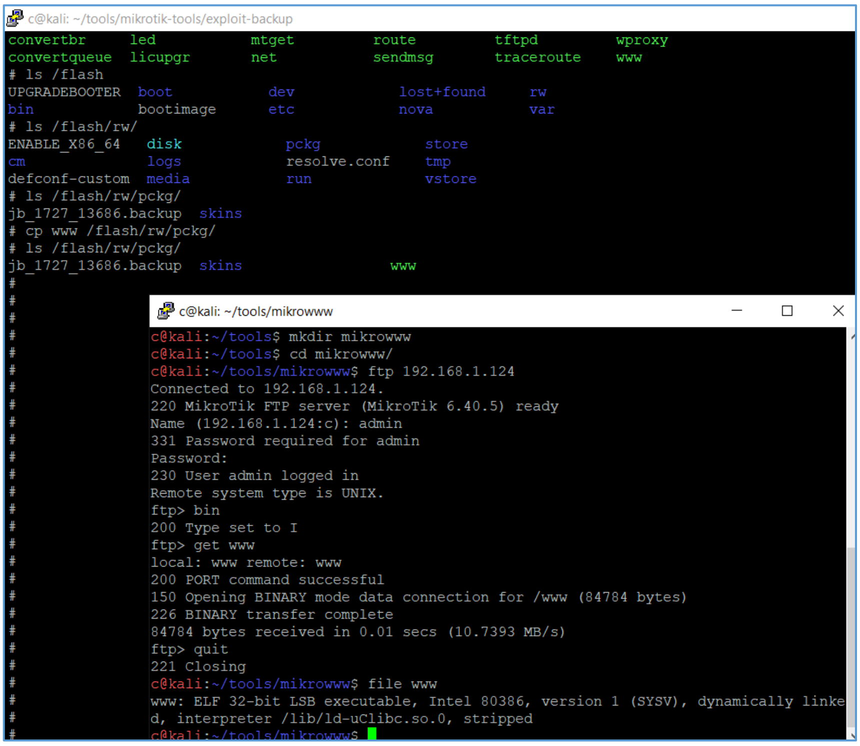
```
c@kali: ~/tools/mikrotik-tools/exploit-backup

We got admin@192.168.1.124 with password ''.
Is this correct? (y/N) y

Let's begin.
Verifying version...
Downloading current configuration...
Patching...
Uploading exploit...

 * * * * * * * *
Congrats! Jailbreak was (likely) successfull. Device will now reboot.
 * * * * * * * *
Linux mode can be accessed via telnet using user 'devel' with admin's password.

Device is now rebooting...
You may opt to install additional utilities to make using the shell easier.
Please note that this will enable telnet service on port 23/tcp,
send YOUR PASSWORD AND USERNAME UNENCRYPTED over the network, and
   may REMOVE YOUR ABILITY TO UPDATE software on smaller devices.

Would you like some additional utilities with your jailbreak? (y/N) y
Waiting for device to reboot...
```

Ok... Ok?

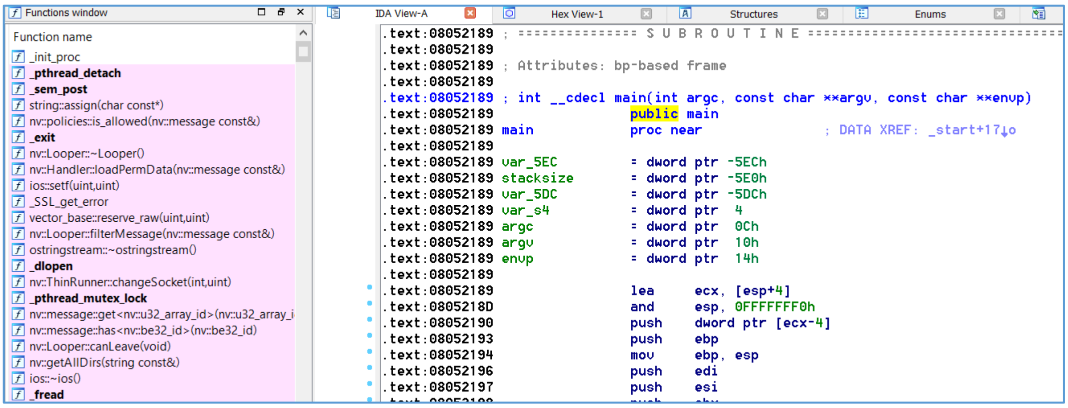
;*



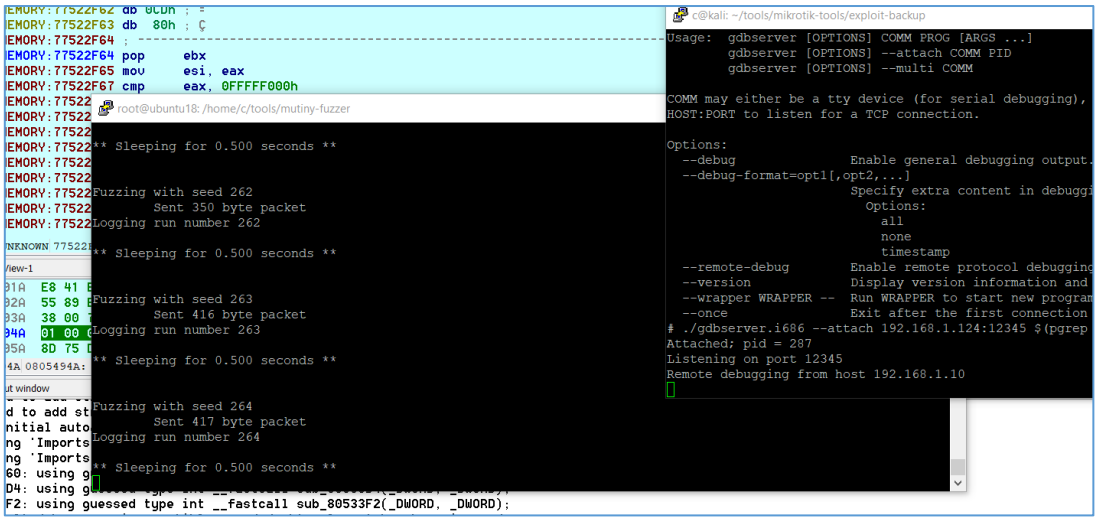
So far, so good. Next step according to the article [5] should be uploading „our tools" to the target host. Following the article – as I did – we can use FTP to do that. Let's continue below then:



As you can see I copied *www* binary as well. I was wondering what I'll be able to get when I'll put that binary in Ida ;)



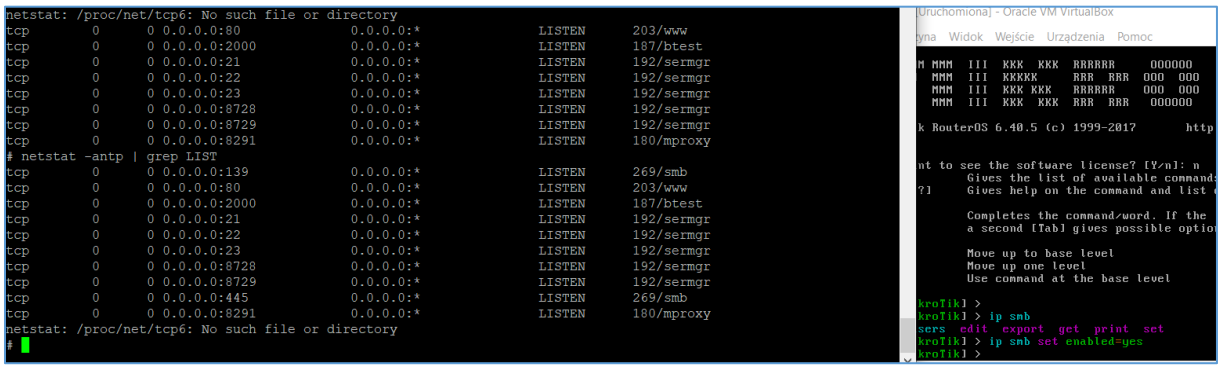
It doesn't look like an obfuscated code... We'll get back to that later... ;)



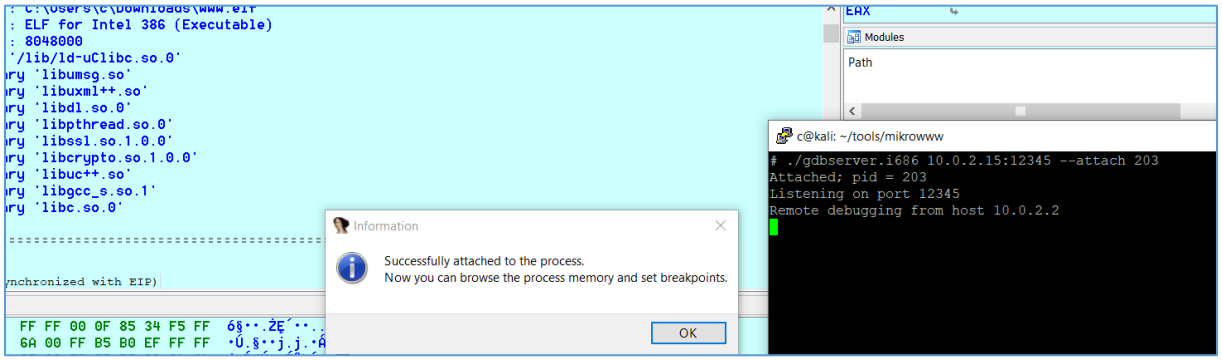
My next step was to upload the tools (lik gdb, etc) to the target VM. To do that I used ftp just like before. At this stage when all is prepared properly – let's take a snapshot (just to save some time in the future ;)). Now we can continue...

Similarity

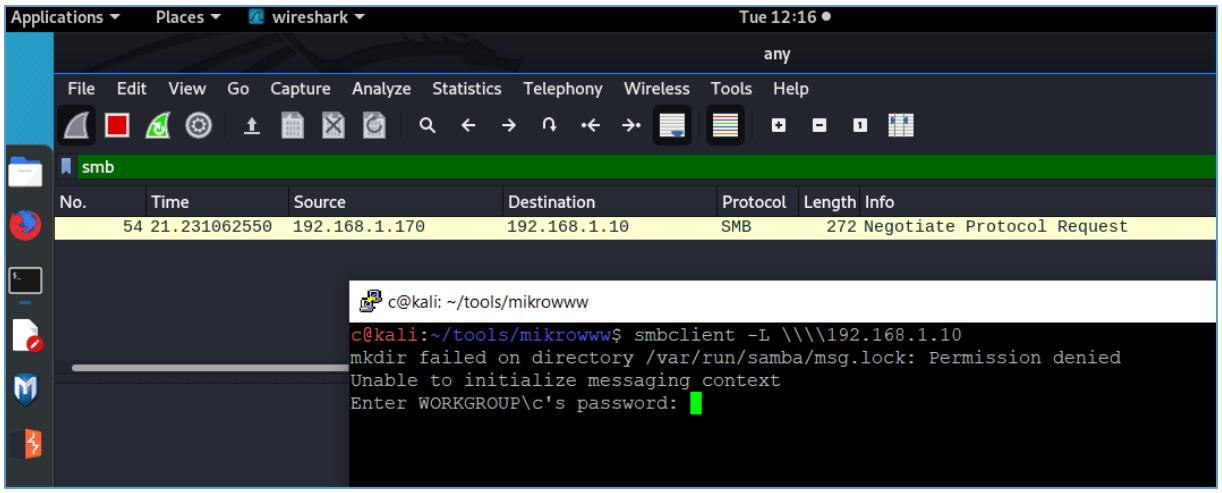
For now we should be here:



Cool. As you can see we have a multiple 'targets' ready to „fuzz(me if you can" scenario described in *Notes Magazine 03, page 79*[9]). We can continue below, checking all the binary we'd like to in Ida:



Next I was recreating the steps from the article[5]:



Next (according to the fuzzing notes mentioned above[9]) we can continue with *Mutiny* fuzzer[10] – so we should be somewhere here:

https://github.com/Cisco-Talos/mutiny-fuzzer

Written by James Spadaro (jaspadar@cisco.com) and Lilith Wyatt (liwyatt@cisco.com)

root@kali: /home/c/tools/mutiny-fuzzer

```
root@kali:/home/c/tools/mutiny-fuzzer# ^C
root@kali:/home/c/tools/mutiny-fuzzer# cp ../smb01.pcap ./
root@kali:/home/c/tools/mutiny-fuzzer# ./mutiny_prep.py smb01.pcap
Processing smb01.pcap...
Which port is the server listening on? (445/36412)
Default 445: 445

        Message #0 - Processed 216 bytes outbound
Processed input file smb01.pcap

How many times should a test case causing a crash or error be repeated?
Default 3: 2

When the test case is repeated above, how many seconds should it wait between tests?
Default 5: 3

Which protocol? (tcp/udp/layer3)
Default tcp: tcp

What port should the fuzzer connect to?
Default 445: 445


Would you like to auto-generate a .fuzzer for each client message? (y/n)
Default n: y


Wrote .fuzzer file: smb01-0.fuzzer

All files have been written.
root@kali:/home/c/tools/mutiny-fuzzer#
```

Next:

```
Fuzzing with seed 66
        Sent 218 byte packet
Logging run number 66

** Sleeping for 0.500 seconds **


Fuzzing with seed 67
        Sent 333 byte packet
Logging run number 67

** Sleeping for 0.500 seconds **


Fuzzing with seed 68
        Sent 224 byte packet
Logging run number 68

** Sleeping for 0.500 seconds **


Fuzzing with seed 69
```

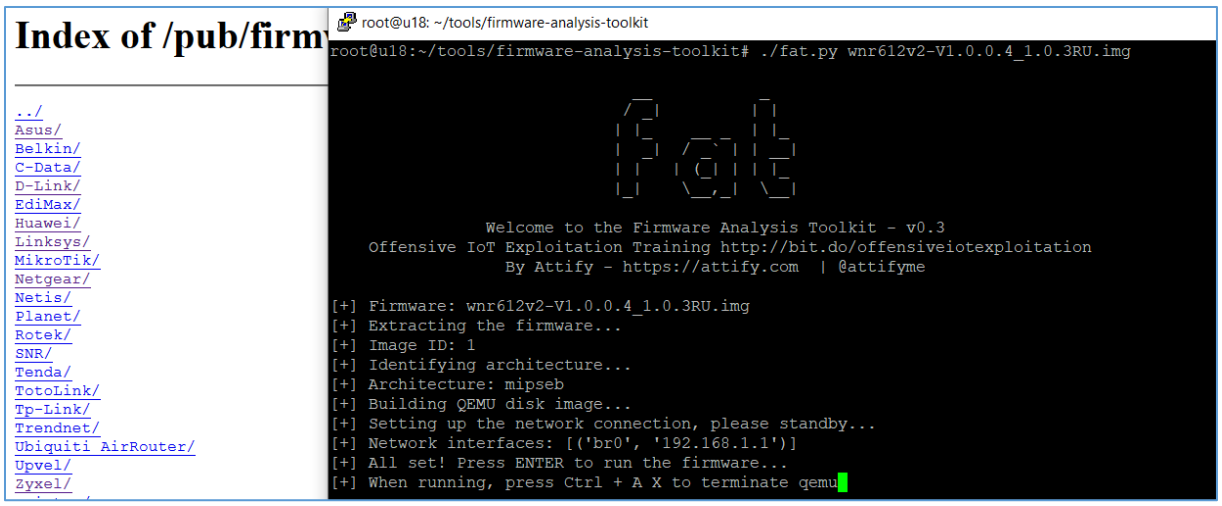
c@kali: ~/tools/mikrowww

```
# netstat -antp | grep LIST
tcp        0      0 0.0.0.0:139             0.0.0.0:*               LISTEN      269/smb
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      203/www
tcp        0      0 0.0.0.0:2000            0.0.0.0:*               LISTEN      187/btest
tcp        0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      192/sermgr
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      192/sermgr
tcp        0      0 0.0.0.0:23              0.0.0.0:*               LISTEN      192/sermgr
tcp        0      0 0.0.0.0:8728            0.0.0.0:*               LISTEN      192/sermgr
tcp        0      0 0.0.0.0:8729            0.0.0.0:*               LISTEN      192/sermgr
tcp        0      0 0.0.0.0:445             0.0.0.0:*               LISTEN      269/smb
tcp        0      0 0.0.0.0:8291            0.0.0.0:*               LISTEN      180/mproxy
netstat: /proc/net/tcp6: No such file or directory
# pkill smb
# /nova/bin/smb
Main start
Socket: created socket: 15, on :139
Socket: created socket: 16, on :445
Socket: created socket: 17, on :137
Socket: created socket: 18, on :138
NBServ: Reg name on ip: 10.0.2.15
```

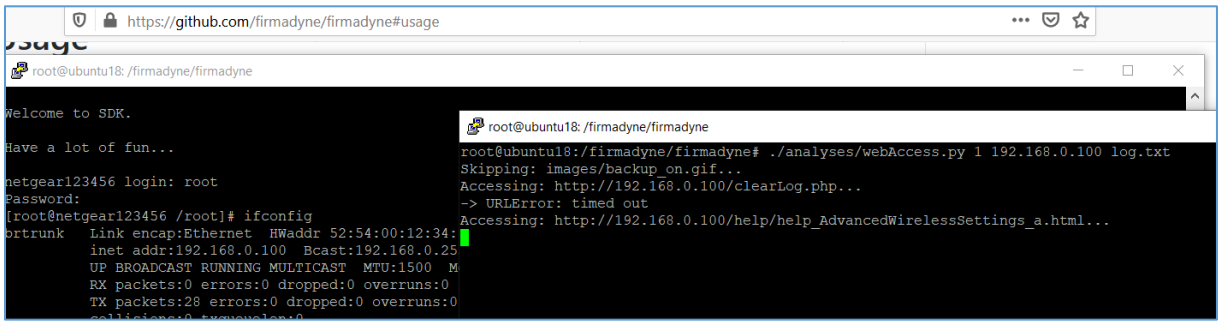
So far so good. Now I believe the *time* is our 'enemy' so feel free to run the fuzzing process on your own lab to get some crashes and have fun. ;)

Going down?

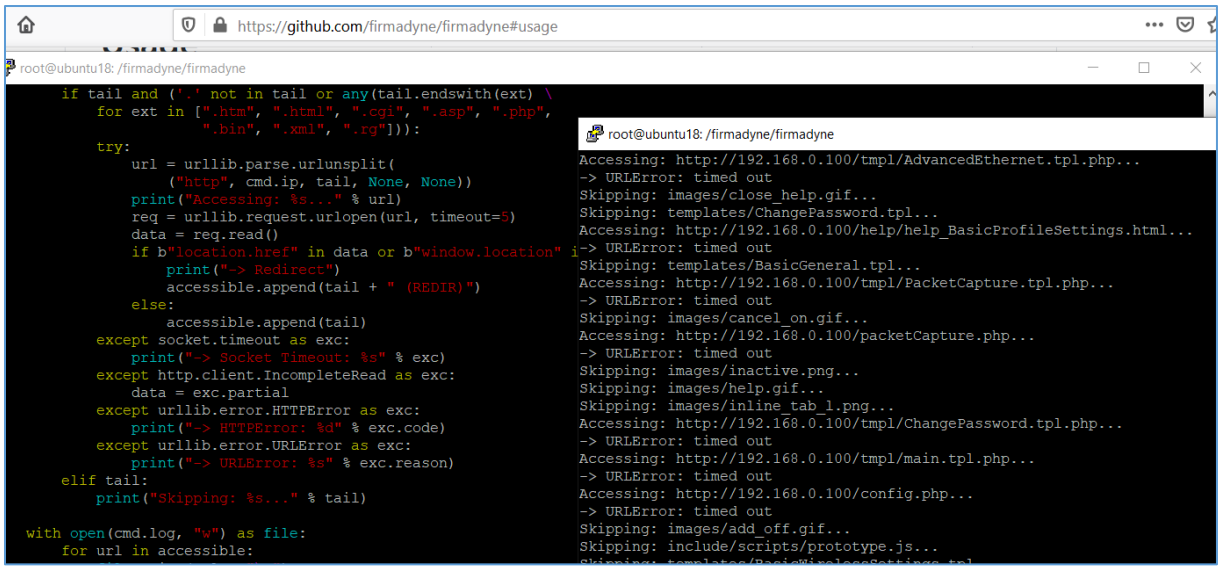
After a while the idea evolved to something similar I described for Wordpress plugins[9]. When our 'lab' is prepared, everything works fine – what else should we do now to „try to find more bugs"? Well – my first guess was to create some kind of an *automated* environment now. We'll use the same *base* we did before – so you can easily stay with your Ubuntu 18 VM. Let's start here:



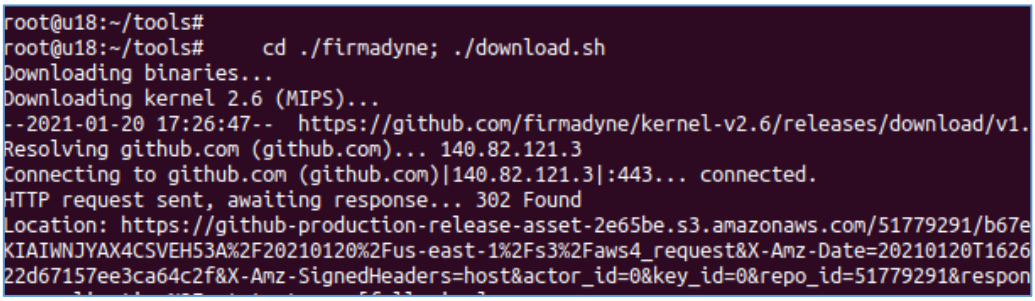
Finding an 'example firmware of the router' should be the easy part here ;) let's move forward then. As we can see – for this router firmware we can get an IP (it will be the most often error message you'll see during the research: somehow 'firmadyne'-related scripts can not obtain an IP sometime for some appliances – well, „nobody's perfect". ;)) Let's jump here:



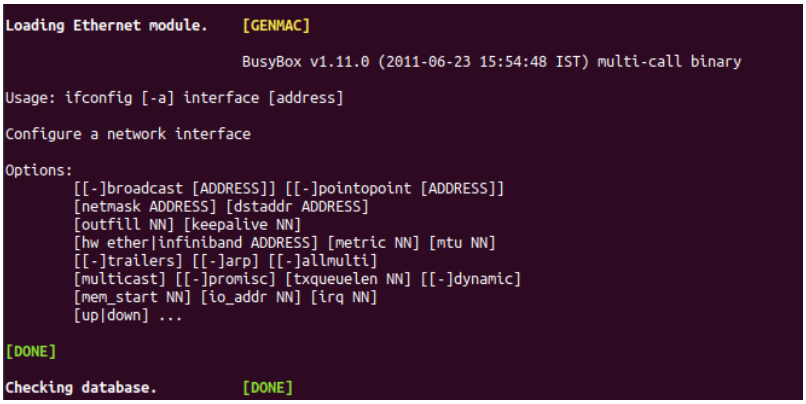
As you can see script collection[11] is prepared mostly for the same scenario that we're using during our adventures[13]. In the content of the directory we can find some 'example steps' (coded in small scripts) that we'll also try to take when we'll find an applicane „like the one presented on the screen" (ex. during our pentest or CTF), so: nmap, nikto, gobuster/dirb and so on should be your friend at this stage. We can also use *sslscan* or Metasploit Framework but it depends on you what you'll now like to add to 'your' (or simply: modified) scripts from the *firmadyne* catalog[11]. In my case we are here:



As you can see I tried to understand the script by reading it ;] So let's continue below:



I restarted the *installation* process multiple times (after restoring the snapshot – also multiple times ;)). I was wondering what I do not understand (read as: why this-or-that firmware is not getting an IP). So I landed here:



We'll not fix it here – for more hints what's going on – try this page[14]. For now we'll jump to the next section. Here we go...

Lazy Monitoring

My initial goal was to create a new additional script to:

- loop to check if qemu (so ‘our target firmware’) is started

- check if firmware started in qemu has a valid IP address (ping it or something like that ;))

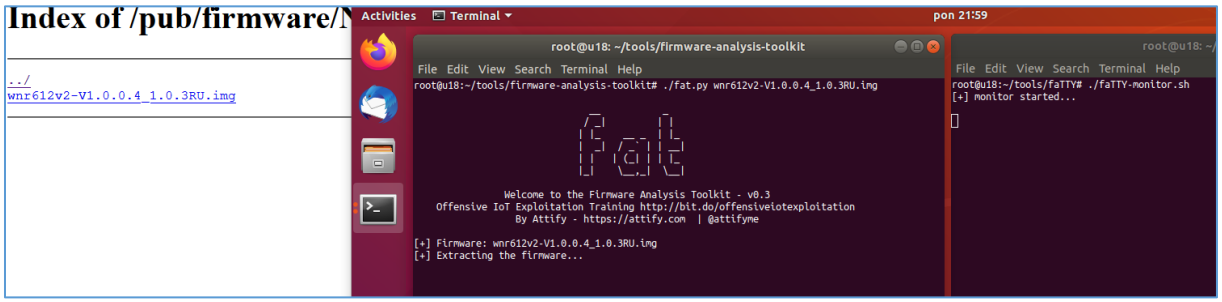
- run ‘our scan scenarios’ (so mentioned nmap, dirbuster(s) and so on... to grab some details about the target host).

In other window we should run a ‘loop’ to:

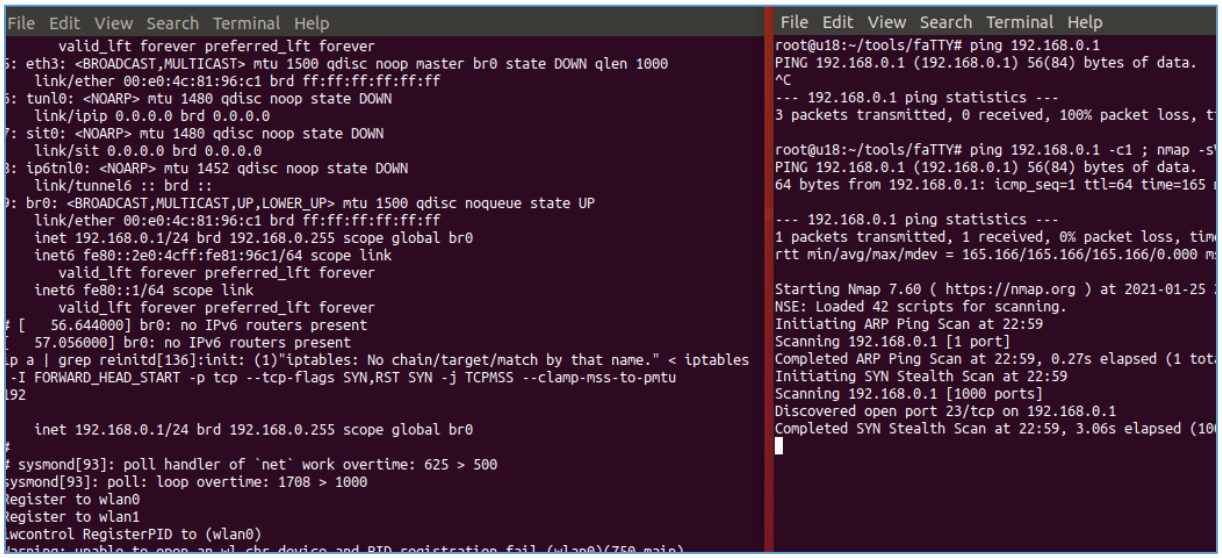
- download ‘example firmwares’ we’re looking for to check/scan/test

- try to *automatically* run it. ;]

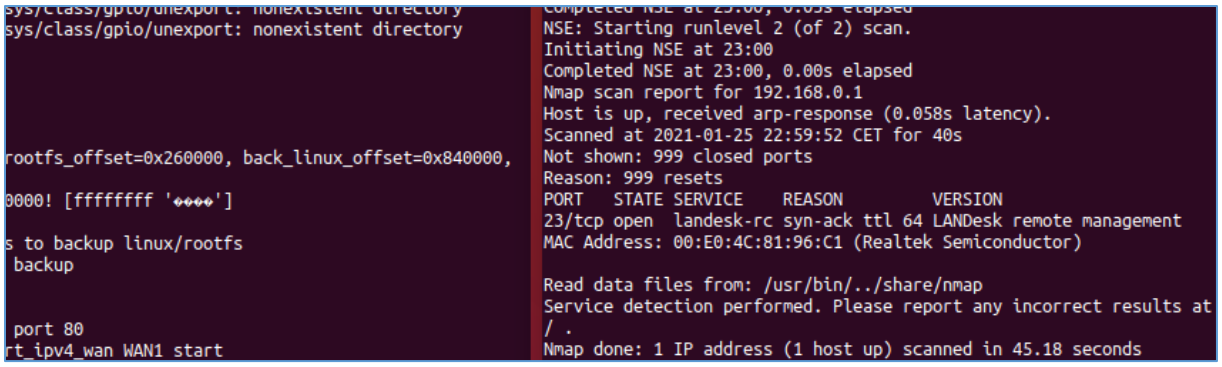
So far we should be somewhere here:



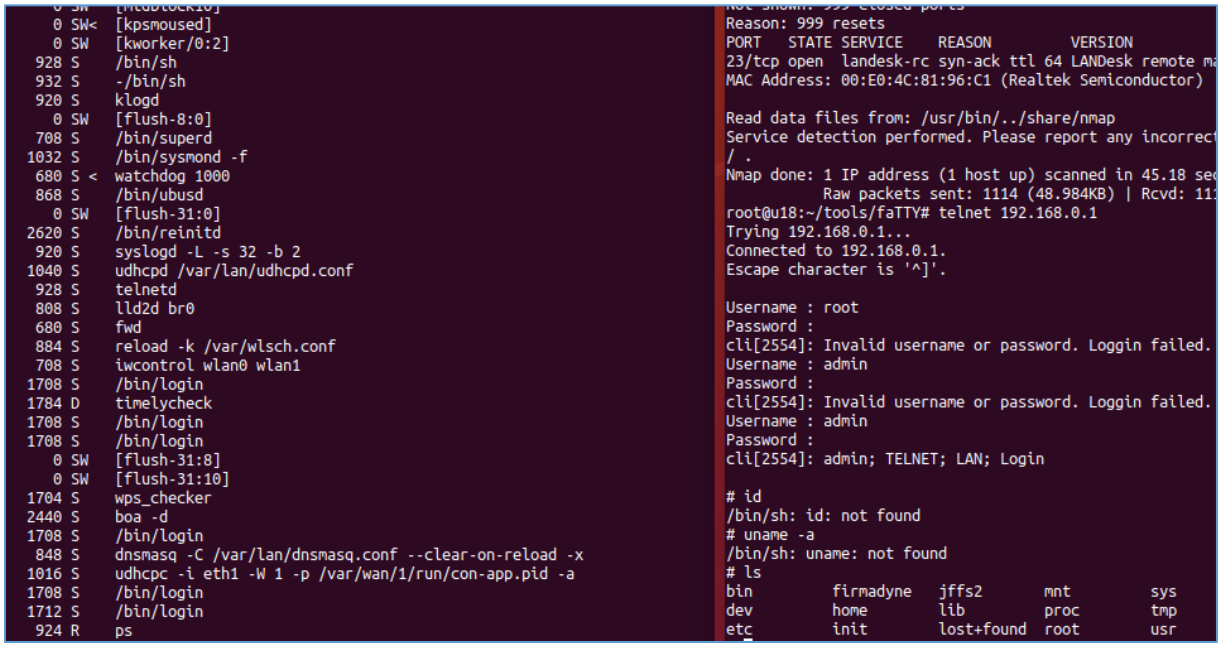
Next the firmare I tried started properly so I was able to ping/access it:



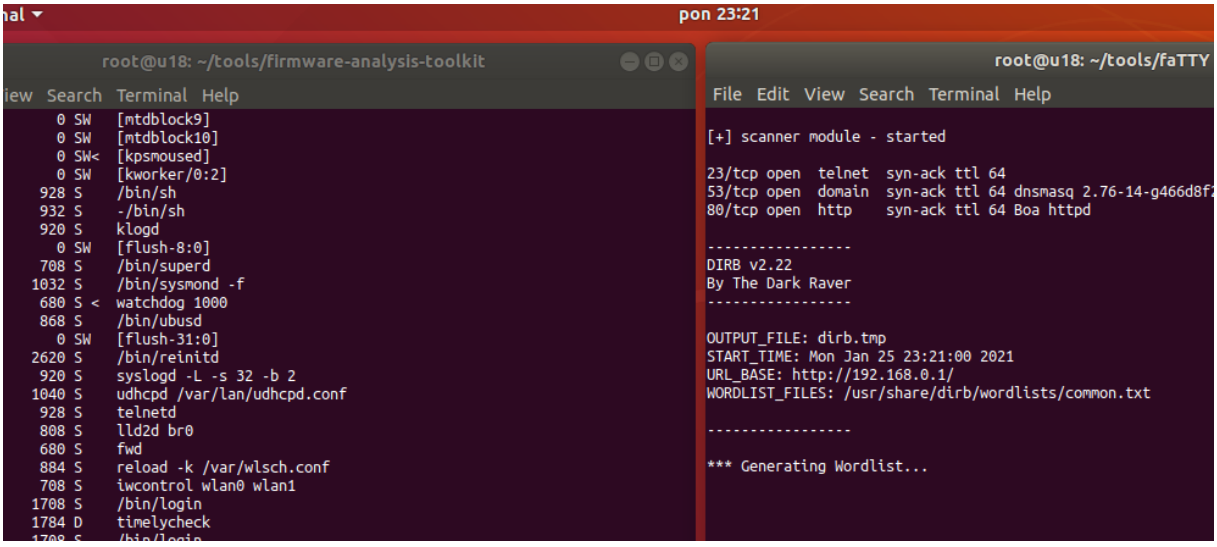
Next – sure thing, portscan (*nmap –sV –F –n host –oN logfile*):



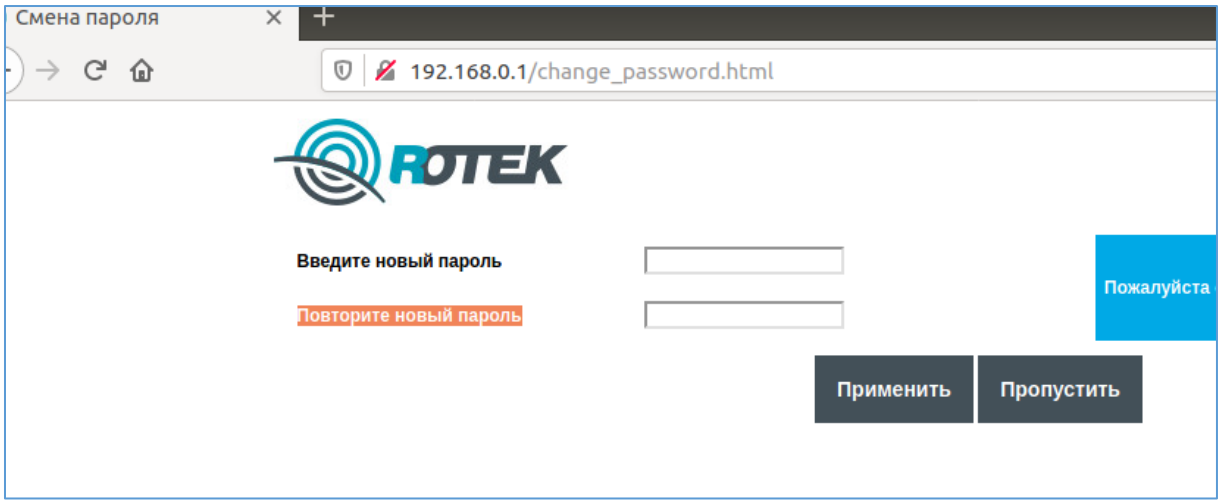
In the meantime I tried to log in to the appliance (via telnet as you can see):



(At this stage you can try to create some quick script to bruteforce telnet service or simply check few ‘default passwords’ like root, admin, and so on... Next:



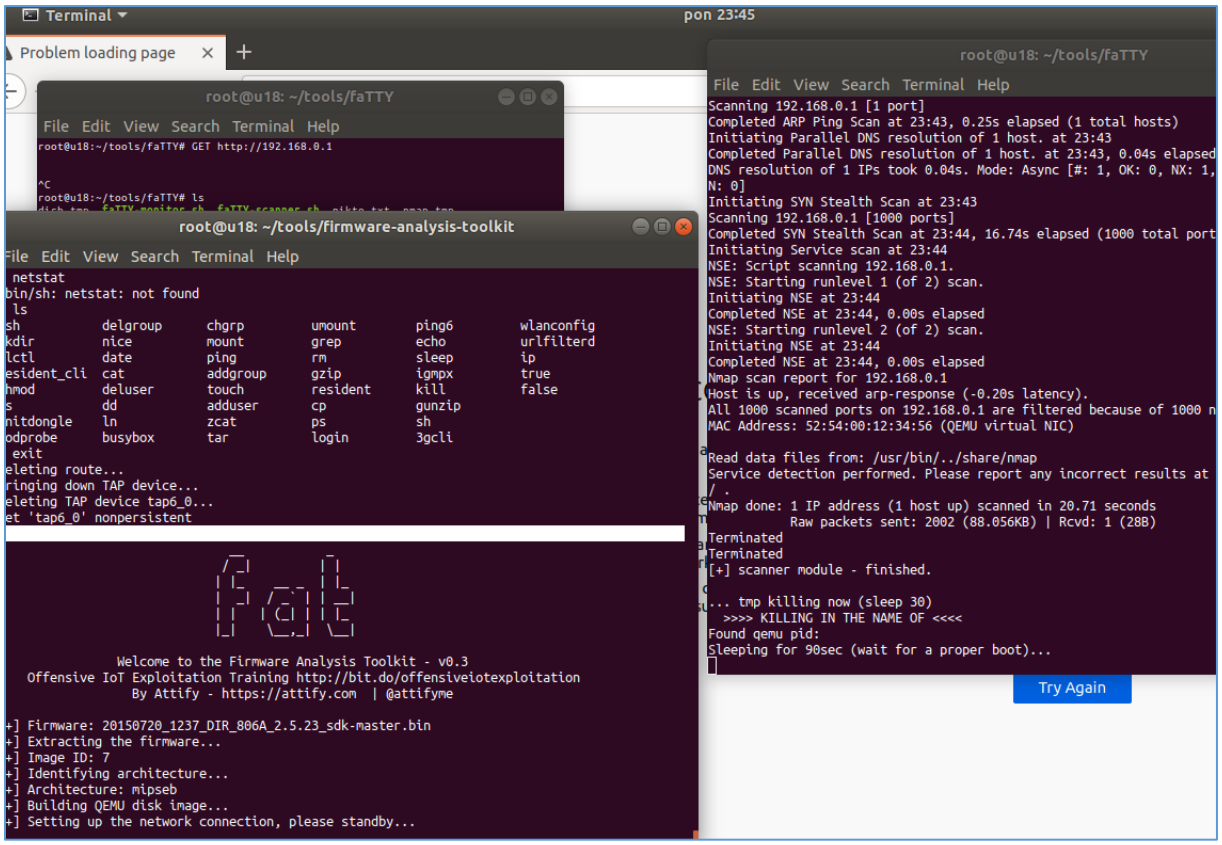
Feel free to use any other tool to enumerate WWW. I used *dirb*. Next:



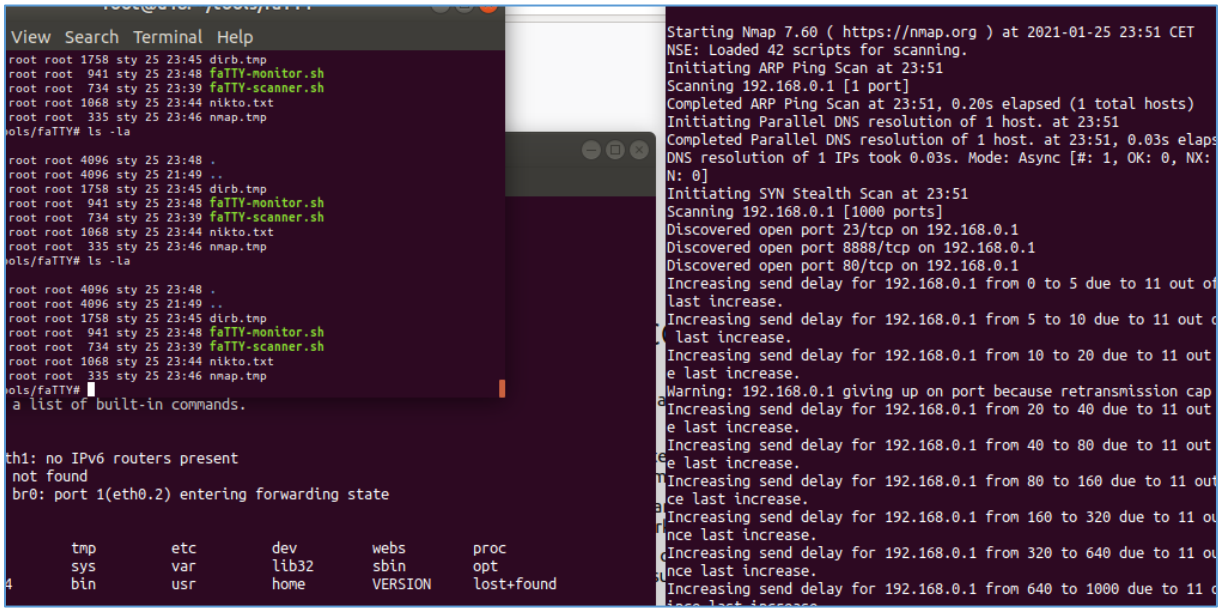
So far, so good. As you can see below I downloaded „few more" ;] firmwares. Next stage was to run *fat.py* script in a loop (remember to add *./reset.py* to the chain):

```
DIR880A1_FW108WWb03.bin
DIR890LA1_FW110b07.bin
fat.config
fat.py
firmadyne
FW_WDRT-731U_V2.0.1.5_EN_PLA01.bin
'FW-WNRT617v2_v3.16.5(130508).bin'
'Keenetic-Ultra-V2.06(AAGJ.1)C2.bin'
LICENSE
N300-1.1.0.32_101.img
'netis(MW5230_EU)-V2.2.39550.bin'
'netis(WF2409D)-V1.4.37280.trx'
'netis(WF2409E_RU)-V2.6.40535.bin'
'netis(WF2419E)-V2.2.36411.bin'
OLDFW
qemu-builds
README.md
reset.py
ROTEK-RX22101-V1.23.87.4-UFN.bin
setup.sh
TEW651BRV2_FW204B01.bin
US_FH456V2.0BR_V11.13.01.20_multi_TDEGYH01.bin
US_N301V1.0V1.0BR_V5.07.64.4_ru_RU01.bin
V5.07.46_en.bin
'WNAP320 Firmware Version 2.0.3.zip'
wnr2200-100-V1.0.2.24.img
wnr612v2-V1.0.0.4_1.0.3RU.img
root@u18:~/tools/firmware-analysis-toolkit# for i in `ls *.bin` ; do ./fat.py $i ; done
```

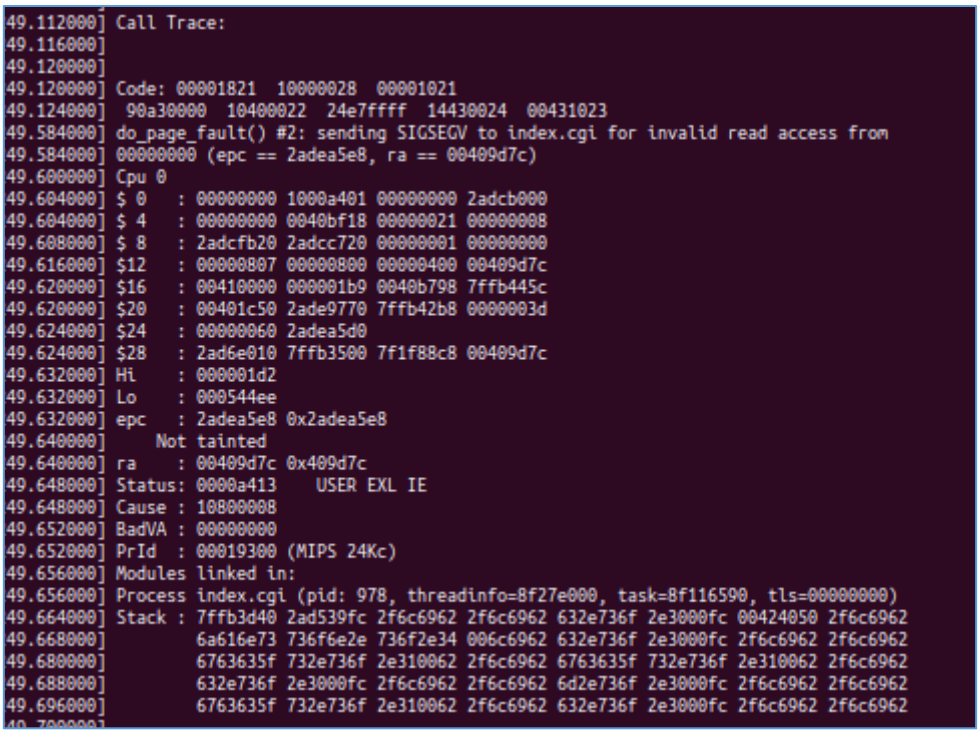
After a while:



Checking another and another firmware and collecting logs:



...when sudenly in the console1 I saw:



Well ;> I believe now you're ready to extend the *fat.py* script(s) and create your own 'small fuzzer' to find some new bugs in your routers/IoT firmwares. All described links and resources you'll find below.

In case of any questions – you'll know how to find me.

Have fun! ;)

References

Below you'll find the list of links/resources I found interesting:

1- https://code610.blogspot.com/2017/04/learning-routers.html
2- https://code610.blogspot.com/2018/11/learning-routers-part-2.html
3- https://code610.blogspot.com/2018/12/reading-firmware-fortigate-vm.html
4- https://code610.blogspot.com/2018/12/reading-firmware-foscam.html
5- https://medium.com/@maxi./finding-and-exploiting-cve-2018-7445-f3103f163cc1
6- https://download.mikrotik.com/routeros/6.40.5/chr-6.40.5.vmdk
7- https://github.com/0ki/mikrotik-tools/
8- https://github.com/tenable/routeros/tree/master/poc/bytheway
9- https://code610.blogspot.com/p/notes-magazine.html
10- https://github.com/Cisco-Talos/mutiny-fuzzer
11- https://github.com/firmadyne/firmadyne
12- https://github.com/attify/firmware-analysis-toolkit
13- https://code610.blogspot.com/p/mini-arts.html
14- https://github.com/firmadyne/firmadyne/issues

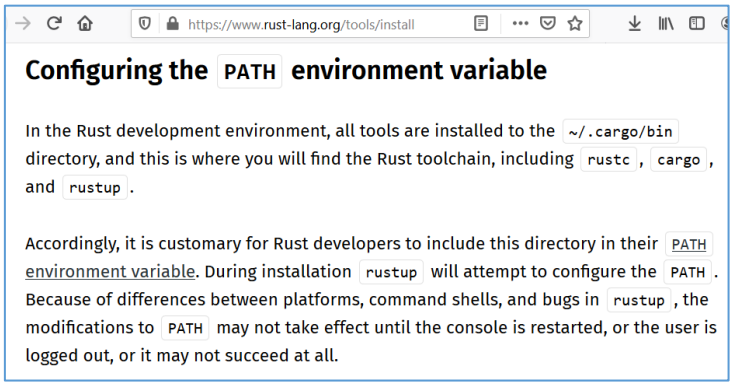
Rust In Pieces

Intro

Thanks to one of the topics mentioned in the video series called 'Random Topics'[1] once upon a time I decided to learn about the new language – Rust. Below we'll try to understand few basic rules of the language as well as the syntax (but keep in mind that similar to the [2] I will try to stick strictly with the documentation[3, 4]. As there is no point to rewrite the manual here – feel free to study it in your free time ;)). For now – we'll start here...

Environment

According to the docs[3, 4] and few other resources available online I decided to prepare an environment on Ubuntu 18 VM. Next we'll need to install few additional packages, see below:



You know I like to 'try harder'[5] ;) so I decided to install all of the mentioned apps manually (but feel free to use the *curl* command mentioned in the documentation[3]). In my case 'the basic environment' looks like this:

```
root@ubuntu18:~# uname -a ; lsb_release -a
Linux ubuntu18 5.3.0-28-generic #30~18.04.1-Ubuntu SMP Fri Jan 17 06:14:09 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.4 LTS
Release:        18.04
Codename:       bionic
root@ubuntu18:~# apt-get install cargo rustc rustup -y
Reading package lists... Done
Building dependency tree
Reading state information... Done

No apt package "rustup", but there is a snap with that name.
Try "snap install rustup"

E: Unable to locate package rustup
root@ubuntu18:~# snap install rustup
snap "rustup" is already installed, see 'snap help refresh'
root@ubuntu18:~#
```

If everything is ok so far and installation finished successfully we can move forward to the manual section. Ready...? ;]

Read the Manual

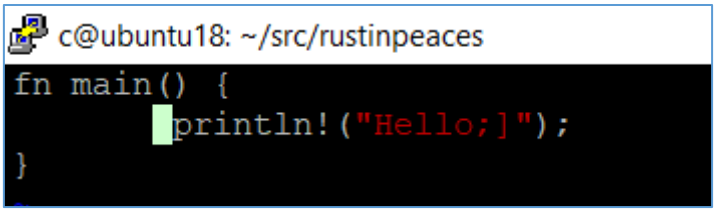
In my case, learning „new language" is done (ok... some about 50% of it ;)) by finding and/or understanding the analogy to the other language(s) I „already know" (or at least *understand*).

The „other 50%" is done by reading the documentation(s), resources available online and by searching (through mentioned above) for some (maybe) „similar cases" that I would like to do/solve/try (using that 'new language').

So that's why most of this section is related to few „examples" I found in the docs[3, 4]. Here we go...

Example 01

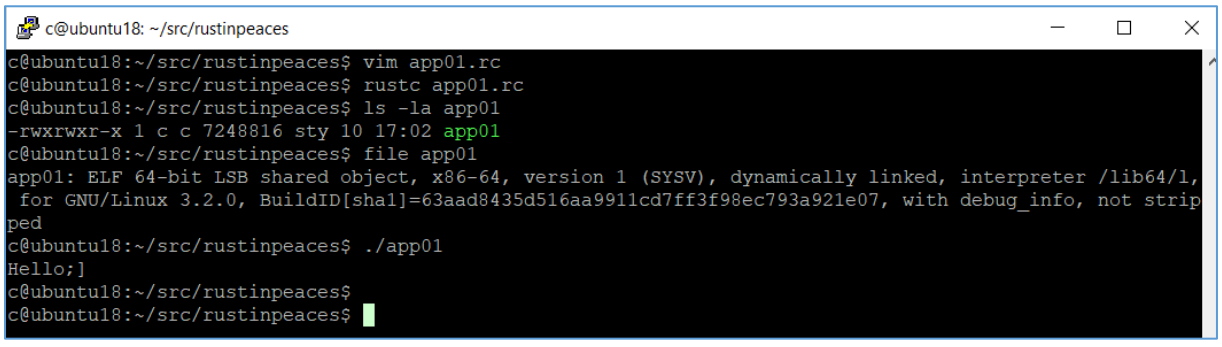
First of all I decided to create a 'default „simple program"' – so, yep, we'll start from „HelloWorld"(.rs for the Rust's extension). We are here in the Ubuntu (console):



That's it! Save your file in the favourite editor and go back to the console. According to the docs[6] now we need to compile our source code using (*man* for) *rustc* compiler:

```
NAME
    rustc - The Rust compiler

SYNOPSIS
    rustc [OPTIONS] INPUT

DESCRIPTION
    This program is a compiler for the Rust language, available at https://www.rust-lang.org.

OPTIONS
    -h, --help
        Display the help message.
```

So far, we should be here:



Great! Let's move forward.

Example 02

I lik to read the documentation prepared „by the vendor”. Reason(s?): beside learning „how things works” we can (also) try to *read between the lines* and (for example) find some bugs in the protocol design... but to not to go so far to (the past with DNS „by design bugs”;)) *future* – let’s continue with our „simple example 02”. Here we go...

This time my goal was read the documentation to understand how can I create a simple app that will:

- say hello

- get user’s name

- say hello <to-the-user’s-name>.

Simple, isn’t it? ;] Yes it is. It’s even pretty similar to the ‘scenarios’ for multiple *crackmes* available here[7]. (Un)fortunately I wasn’t able to find any example ‘crackme’[7] prepared for the (or „ created in the”) Rust language – so I decided to create a simple one for you. Quick results - below. ;)

(Full „source” of the challenge presented below you’ll find on my github, somewhere between the other ‘notes’ files[8];)). Let’s say we should be here:

```
File  Edit  View  Search  Terminal  Help
root@ubuntu18:/home/c/src/rustinpeaces/ex03# vim ex04.rs
root@ubuntu18:/home/c/src/rustinpeaces/ex03# head -n 4 ex04.rs
use std::io;

fn main() {
    let mut input = String::new();
root@ubuntu18:/home/c/src/rustinpeaces/ex03# ./ex04
What is the title?
asdfgh
I'm not so sure ;Z
root@ubuntu18:/home/c/src/rustinpeaces/ex03#
```

More details[9]:

```
root@ubuntu18:/home/c/src/rustinpeaces/ex03# base64 ex04 > crackme.ex04.txt
root@ubuntu18:/home/c/src/rustinpeaces/ex03# du -h crackme.ex04.txt
9,4M    crackme.ex04.txt
root@ubuntu18:/home/c/src/rustinpeaces/ex03# md5sum crackme.ex04.txt
36d658d40b6c7d76fa56912d88354208  crackme.ex04.txt
root@ubuntu18:/home/c/src/rustinpeaces/ex03#
```

Good luck! ;)

References

Below you'll find the list of links/resources I found interesting:

1 - https://www.youtube.com/watch?v=WaASAO3_WsY&ab_channel=GynvaelColdwind

2 - https://www.rust-lang.org/learn

3 - https://www.rust-lang.org/tools/install

4 - https://doc.rust-lang.org/rust-by-example/meta/doc.html

5 - https://code610.blogspot.com/2020/02/trying-harder.html

6 - https://doc.rust-lang.org/book/ch01-02-hello-world.html

7 - https://crackmes.one

8 - https://github.com/c610/free

9 - https://github.com/c610/free/blob/master/crackme.ex04.txt

Rosie Da Stoned

Intro

Maybe you know, maybe you don't – from time to time I like to take a break from the computer and (for example) read a book about some mithology, archeology or something else like the wor(l)ds created by Stephen King[1] or Neil Gaiman[2]. During one of those breaks (ok, to be honest: it was „many years ago" and I didn't even have a computer yet ;)) I found some story in one book in my house about so called Rosetta Stone[3].

After few years (during one of my 'breaks' ;)) I was wondering „maybe it'll be a good idea to create some software to *automatically* read *languageA* and translate it to *languageB"...* In the meantime - days goes by, I'm doing another pentest project for another company when suddenly I found an article[4, 5] that „someone already did it"[6]. ;>

I decided that there is no time for the break and that's how I started preparing „my own small Rosetta Stone". ;]

Here we go...

Watching the Tower of Babel

To continue with this *translator* idea we'll prepare a small environment. All examples presented in this text were prepared on Ubuntu 18 and Python language. If there are any other libs/resources needed to proceed – I'll mention it in the text below.

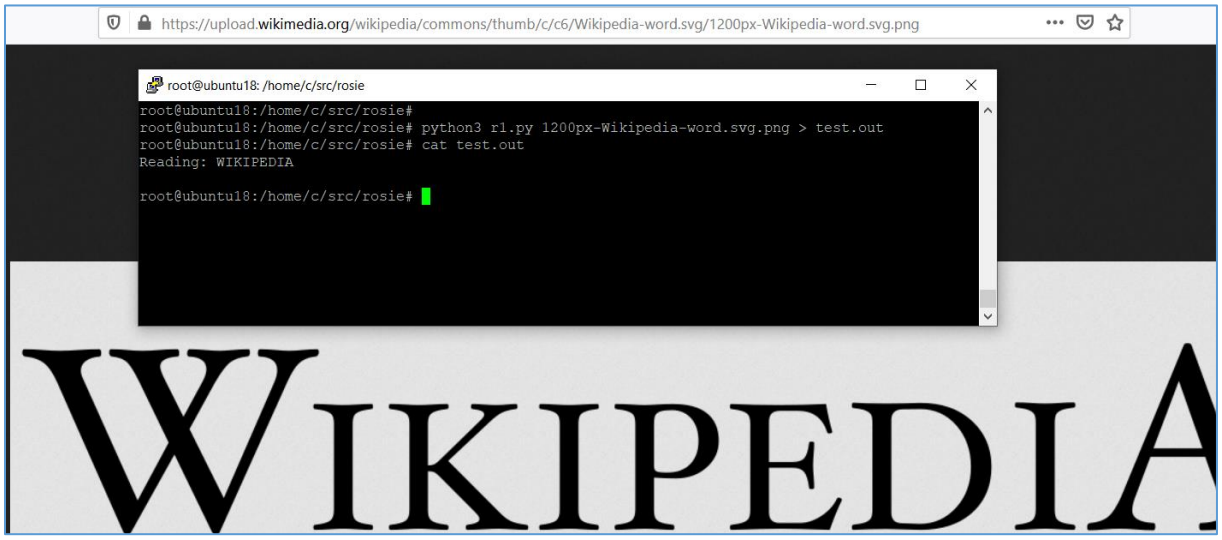
Let's start from the stage where we already have a picture of „some word" we'd like to translate. To proceed I found an example picture with the word. Then I asked my self: where should I start? What's next?

```
# apt install libopencv-dev python3-opencv tesseract-ocr -y; pip3 install
Image pytesseract
```

In my case there was a 500MB update – so after a while we should be somewhere here:

```
root@ubuntu18:/home/c/src/rosie# wget https://upload.wikimedia.org/wikipedia/commons/thumb/c/c6/Wikipedia-word.svg/1200px-
Wikipedia-word.svg.png
--2021-02-06 13:02:13--  https://upload.wikimedia.org/wikipedia/commons/thumb/c/c6/Wikipedia-word.svg/1200px-Wikipedia-wor
d.svg.png
Resolving upload.wikimedia.org (upload.wikimedia.org)... 91.198.174.208, 2620:0:862:ed1a::2:b
Connecting to upload.wikimedia.org (upload.wikimedia.org)|91.198.174.208|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18006 (18K) [image/png]
Saving to: '1200px-Wikipedia-word.svg.png'

1200px-Wikipedia-word.svg.png  100%[=================================================>]  17,58K  --.-KB/s    in 0,02s

2021-02-06 13:02:13 (716 KB/s) - '1200px-Wikipedia-word.svg.png' saved [18006/18006]

root@ubuntu18:/home/c/src/rosie# cat r1.py
from PIL import Image
import pytesseract
import sys

read_me = sys.argv[1]

out_txt = pytesseract.image_to_string(Image.open(read_me))

print ("Reading: %s" % (out_txt))
```

With this very basic example we should be able to read a 'sample image' (we grabbed from Wikipedia). Let's try:



So far – looks good ;) Let's continue then.

For now our new goal looks like this:

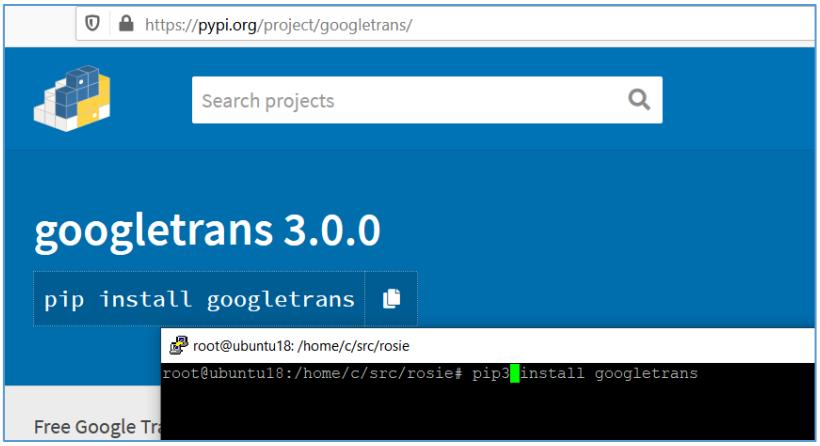
- open and read image

- save output to txt file

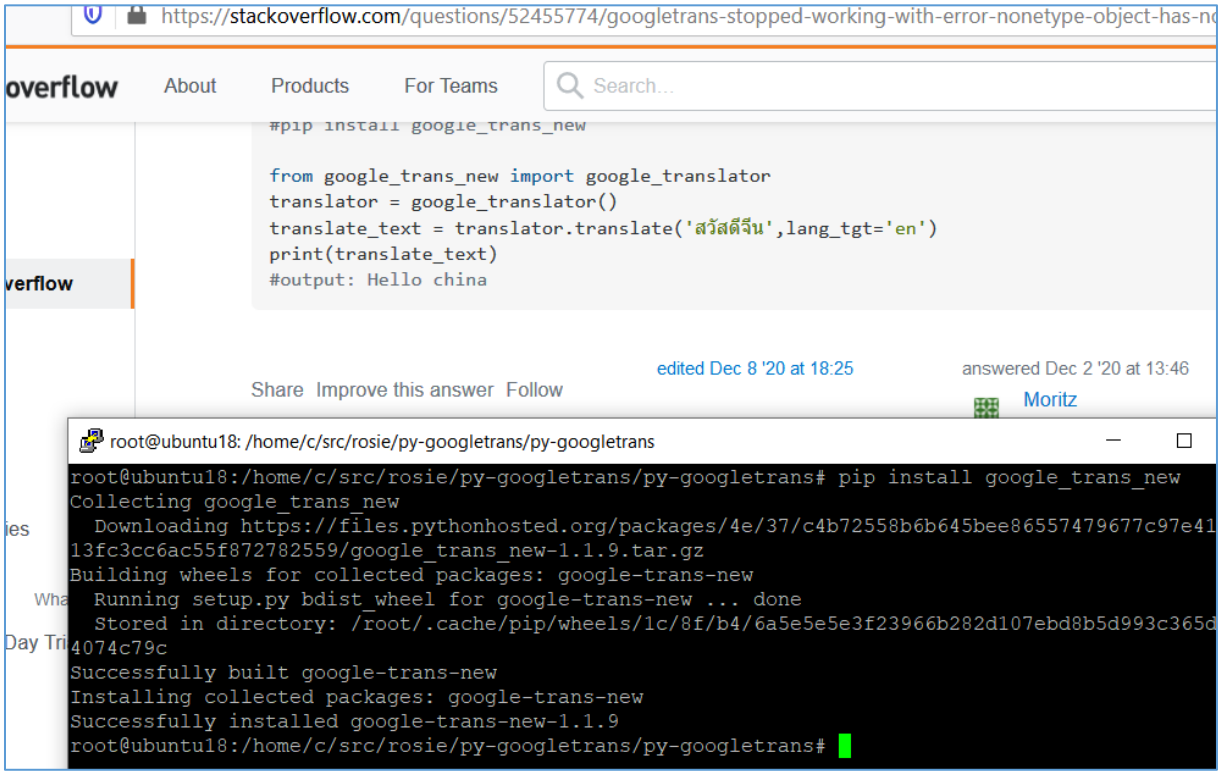
- connect to translator to check found word saved in txt file

- save translated word in new txt output file.

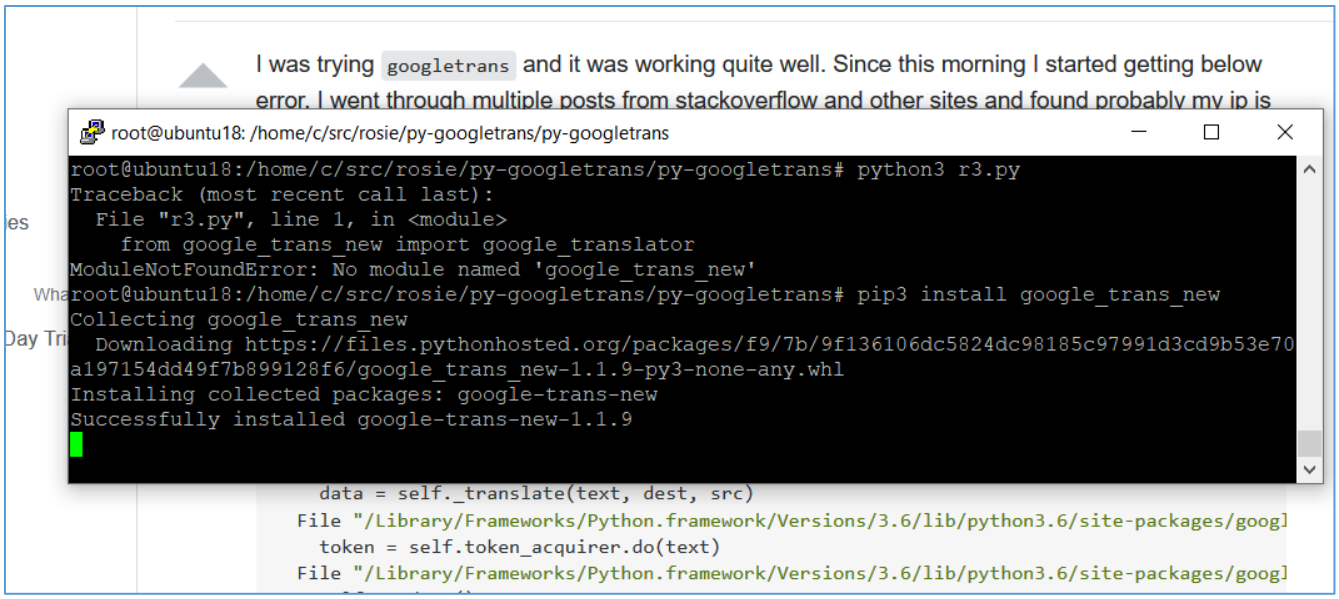
Let's try here:



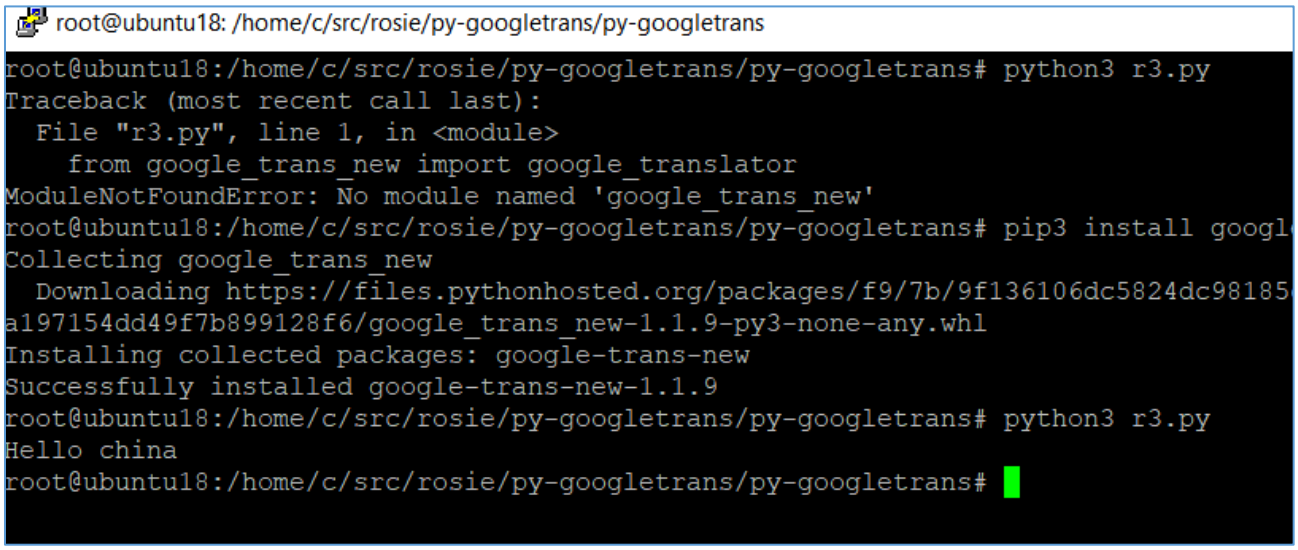
When all is ready we can check initial example – but after we'll fix that nasty error from the console:



And in case you're wondering why it's not working with python3... ;)



Yep ;D So one more time:



That's great!

It looks that now we are:

- able to read text from the image

- translate it to some other language (in our case let's stay with english).

So far, so good – continuing then. Now it's time to fuze both small scripts: reader and translator. We should be somewhere here:

```python
#!/usr/bin/env python
# rosie.py - small script to understand images in the stones ;]
#
# 06.02.2021 @ 13:31
#

from PIL import Image
import pytesseract, sys
from google_trans_new import google_translator

translator = google_translator()

# STEPS:
# 1] read image file
# 2] save text to file1
# 3] each line in file1 parse with translator()
# 4] save output to file2
#

# 1]
read_me = sys.argv[1]
out_txt = pytesseract.image_to_string(Image.open(read_me))

# 2]
fp1 = open('file1.txt','w')
fp1.write(out_txt)
fp1.close()
print("Image parsed, output text in file1.txt.")

# 3]
fp1 = open('file1.txt','r')
lines = fp1.read()

translate_text = translator.translate(lines, lang_tgt='pl')
print("Translation:\n----")
print(translate_text)
print("----\nEND.")
```

Quick results for the *History Channel* logo (found on Wikipedia):

```
HTTP request sent, awaiting response... 200 OK
Length: 467054 (456K) [image/png]
Saving to: ‘1200px-History_Logo.svg.png’

1200px-History_Logo.sv 100%[==========================>] 456,11K   322KB/s    i

2021-02-06 13:45:13 (322 KB/s) - ‘1200px-History_Logo.svg.png’ saved [467054/46

root@ubuntu18:/home/c/src/rosie# python3 rosie.py 1200px-History_Logo.svg.png
Image parsed, output text in file1.txt.
Translation:
----
HISTORIA
----
END.
root@ubuntu18:/home/c/src/rosie#
```

Cool! What’s next? I don’t know... maybe it’s time to buy a nice hat and join the "Raiders of The Lost Ark"...? but maybe you’d like to check the „Voynic Manuscript” first...? ;)

„Have fun & Good luck!” ;]

Cheers

References

Below you'll find the list of links/resources I found interesting:

1 - https://en.wikipedia.org/wiki/Stephen_King

2 - https://en.wikipedia.org/wiki/Neil_Gaiman

3 - https://en.wikipedia.org/wiki/Rosetta_Stone

4 - https://www.bbc.com/news/technology-53420320

5 - https://mashable.com/article/google-ai-ancient-translations-fabricius/

6 - https://wired.me/technology/artificial-intelligence/google-translate-for-egyptian-hieroglyphics/

7 - https://code610.blogspot.com/p/notes-magazine.html

Outro

Comments/questions – you'll know how to find me.

Thank you. I appreciate it.

Cheers