Analyzing conti-leaks without speaking russian — only methodology

medium.com/@arnozobec/analyzing-conti-leaks-without-speaking-russian-only-methodology-f5aecc594d1b

Arnaud Zobec February 28, 2022

Arnaud Zobec

Feb 28

.

5 min read

If you're like me and you don't speak russian, and you have a conti leak to analyze, here is some tricks for you.

Disclaimer : I will not do the analysis in depth of the files here. It's just a blogpost to show methodology in such case. The audience for this blogpost can be students, or people interested in CTI without big budget. This is NOT an analysis of Conti-leaks. This is NOT a TODO list in every case. It's my methodology for json files.

I will talk about how I modified the file to load it easily with Python, and how I used some libraries to translate the text, and how I used other softwares, like Gephi, or command-lines like egrep to have informations quickly.

First look at the files

When you look at the files first, it appears to be in json. Awesome, we love JSON, it's very easy to use it.

```
{
  "ts": "2021-03-24T15:36:48.724506",
  "from": "veron@q3mcco35auwcstmt.onion",
  "to": "defender@q3mcco35auwcstmt.onion",
  "body": "[Ошибка: сообщение зашифровано, и невозможно его расшифровать.]"
}
{
  "ts": "2021-03-24T15:37:56.147132",
  "from": "terry@q3mcco35auwcstmt.onion",
  "to": "stern@q3mcco35auwcstmt.onion",
  "body": "Привет"
}
{
  "ts": "2021-03-24T15:37:56.838446",
  "from": "terry@q3mcco35auwcstmt.onion",
  "to": "stern@q3mcco35auwcstmt.onion",
  "body": "https://privnote.com/oY0Eb4Bq#aEKOIQTVm"
}
{
  "ts": "2021-03-24T15:37:58.678752",
  "from": "terry@q3mcco35auwcstmt.onion",
  "to": "stern@q3mcco35auwcstmt.onion",
  "body": "По трику"
}
```


First look at the content of the json files contained in the leak.
You have several ways to load the file into Python, and I'll show you two different methods under:

First method : transform the files a little bit and load it via JSON libraries

```
#To make one filecat *.json > big.json#To remove the first \nsed -i -e
':a;N;$!ba;s/{\n/{/g' big.json#Remove the \n after the commassed -i -e
':a;N;$!ba;s/,\n/,/g' big.json#Remove the \n before {sed -i -e
':a;N;$!ba;s/\"\n/\"/g' big.json
```

Your file should now look like this :

big.json content

But you know, there is a WAY simpler trick if you use jq :) . It was just to forced you to use sed to make a little bit of file manipulation ;)

```
cat *.json | jq -cr > big.json
```

It will make a one-line for each json line it can read.

And now that I have a clean file, what I want to do is to load every line in a list of dictionnaries in python (and print it for the example).

```
import jsonchatList = []with open('onebig.json') as f:    for jsonObj in f:
_Dict = json.loads(jsonObj)        chatList.append(_Dict)for line in chatList:
print(line['body'])#print each body
```

Easy peasy lemon squeezy

Remember ? I don't speak russian, but I want to read it, and I have no money to pay a professionnal translator. But my data is inside a python dictionnary, so I can do whatever I want with it.

Translation via python

I use a free library that is called deep-translator (https://github.com/nidhaloff/deep-translator)

(to install it : pip install -U deep-translator)

What I will do is to use the library on the "body" key in the json file, for each line, and translate it into english into a new key "LANG-EN". And if there is some fail, I want the message to be "Error during Translation"

And finally, I want to print the result of the line as a JSON line.

```
import jsonfrom deep_translator import GoogleTranslatorchatList = []with
open('onebig.json') as f:    for jsonObj in f:        _Dict = json.loads(jsonObj)
chatList.append(_Dict)for line in chatList:    try:        translation =
GoogleTranslator(source='auto', target='en').translate(line["body"])
line["LANG-EN"] = translation    except Exception as e:        line["LANG-EN"] =
"Error during Translation"    print(json.dumps(line, ensure_ascii =
False).encode('utf8').decode())
```

As you can see, I had to use ensure_ascii = False and encode('utf-8') because I still want to print russian characters.

Now, your output should look like this :


```
{"ts": "2021-01-29T04:04:39.308133", "from": "mango@q3mcco35auwcstmt.onion", "to": "stern@q3mcco35auwcstmt.onion", "body"
: "привет", "LANG-EN": "Hey"}
{"ts": "2021-01-29T04:04:43.474243", "from": "mango@q3mcco35auwcstmt.onion", "to": "stern@q3mcco35auwcstmt.onion", "body"
: "битков не хватит на все..", "LANG-EN": "bits are not enough for everything .."}
{"ts": "2021-01-29T04:32:02.648304", "from": "price@q3mcco35auwcstmt.onion", "to": "green@q3mcco35auwcstmt.onion", "body"
: "привет!!!", "LANG-EN": "Hey!!!"}
{"ts": "2021-01-29T04:32:16.858754", "from": "price@q3mcco35auwcstmt.onion", "to": "green@q3mcco35auwcstmt.onion", "body"
: "опять прокладки сменились??? нет связи!", "LANG-EN": "have the pads changed again? no connection!"}
{"ts": "2021-01-29T04:33:01.808125", "from": "green@q3mcco35auwcstmt.onion", "to": "price@q3mcco35auwcstmt.onion", "body"
: "Привет", "LANG-EN": "Hey"}
```

output of the translation script in python

Second method : transform the files a little bit and load it via pandas

I will transform the first big.json file a little bit, to make it like one big JSON file.

To do it, I'll put every json line into a json tab:

```
#add a "," between "}" and "{"sed -i -e ':a;N;$!ba;s/}/},/g' big.jsonThen I add this character "[" at the beginning of the file and this character "]" at the end of the file
```

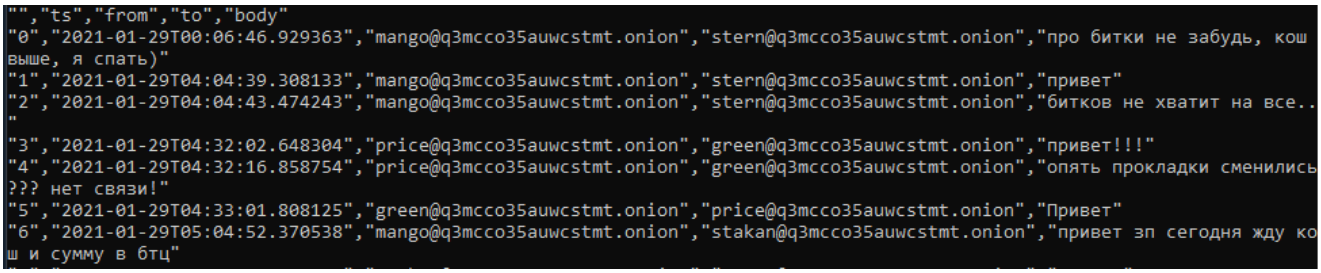
And now, I can load it into a Pandas DataFrame very easily !

```
import pandas as pddf = pd.read_json('big.json')#Yes, it's that easy
```

And why using pandas dataframe ?
Well we can sort it by dates very easily, and transform it into CSV to export to use with other tools that do not deal with JSON easily.

```
import pandas as pddf = pd.read_json('big.json')sorted_df = df.sort_values(by="ts")sorted_df.to_csv('onebig.csv', doublequote=True, quoting=1, escapechar="\\")
```

This code above will create a file called "onebig.csv" sorted by dates.



onbig.csv output

And now what ?

Visualisations : with gephi

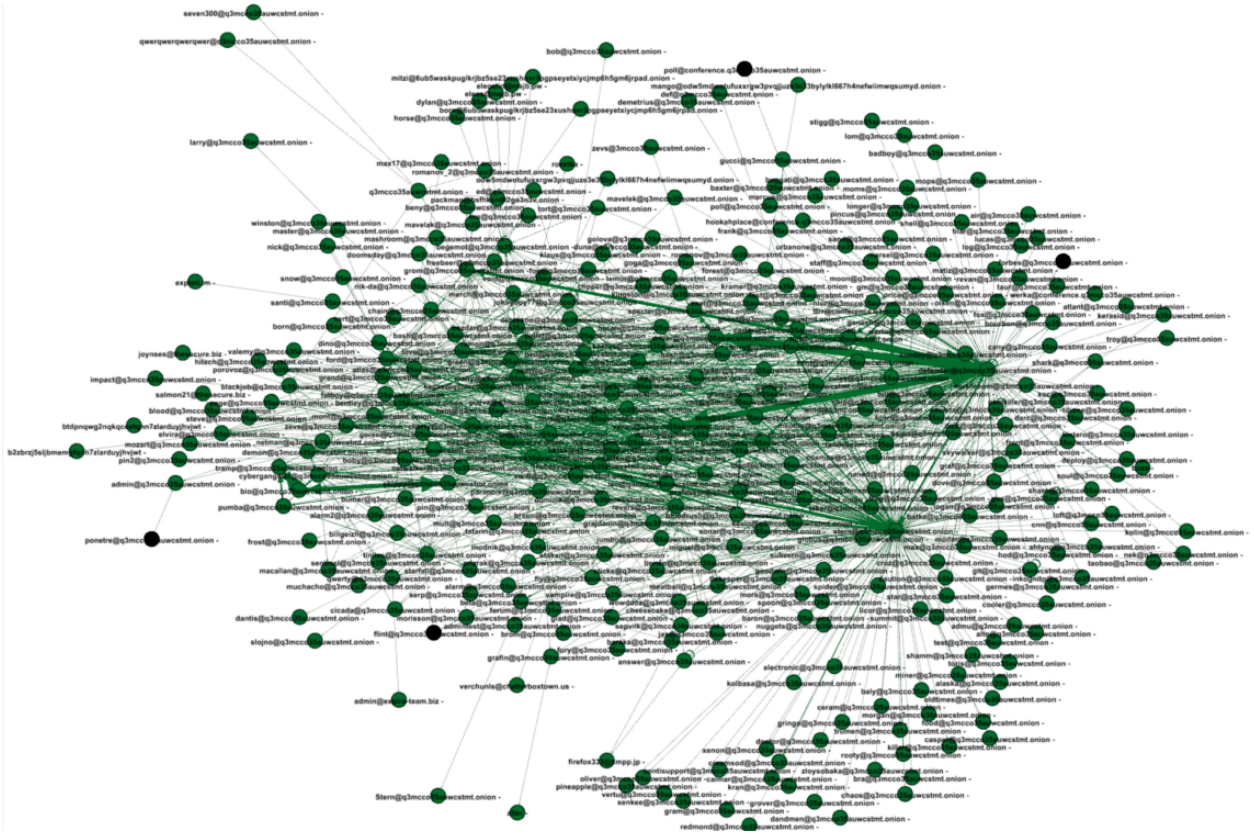
Gephi is an Open Graph Viz Platform - https://gephi.org/

You can use gephi, and a Yifan Hu spacialisation to see the interactions between people , by applying a ponderation on links (for example).

The bigger is the arrow, the bigger is the weight of the link. It means those at each side of the arrow are two people that are often talking together.

We can easily identify people of interest using gephi with this methodology.

Oh. You may want to have a graphic card to use it, it's very power consumptive.

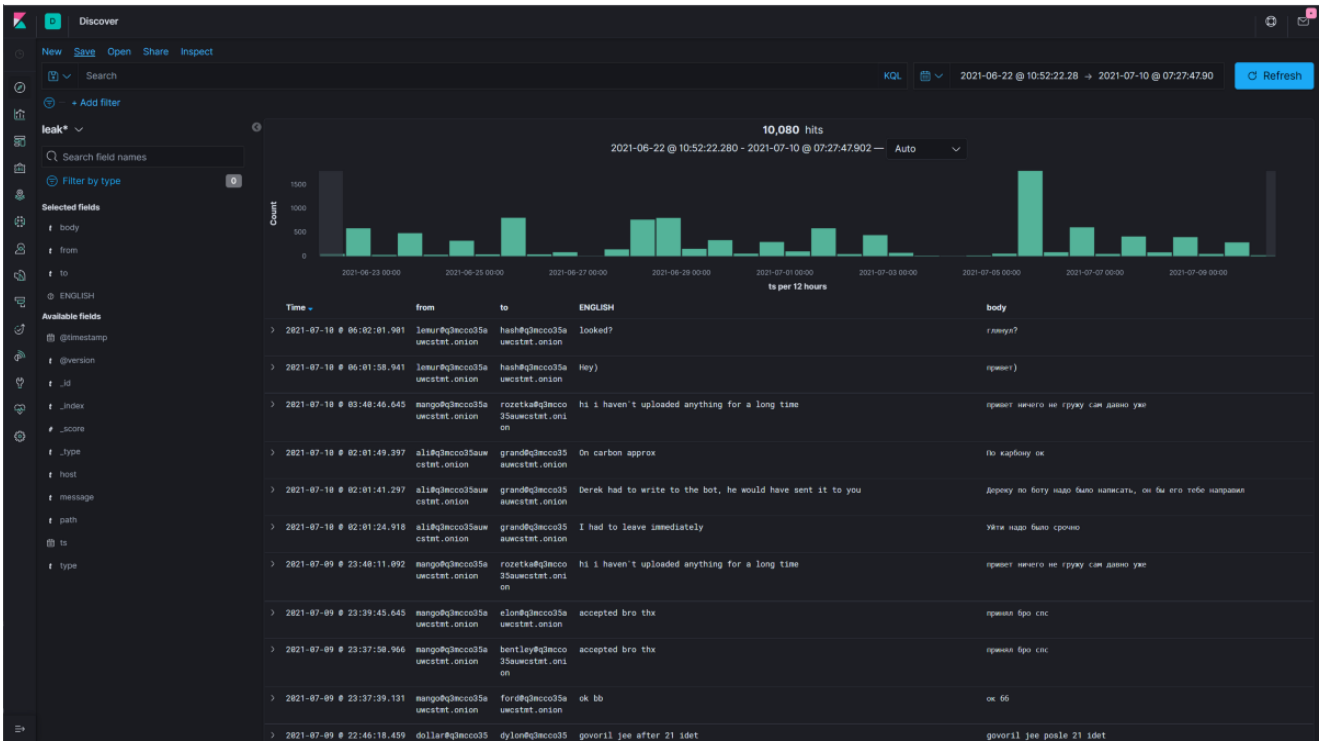


Yifan Hu spacialisation using Gephi

Visualisations : with elasticsearch and kibana

With a very simple configuration, you can load your data into an elasticsearch/kibana cluster, and read things, request it, etc.

```
#content of /etc/logstash/conf.d/00-leak-analysis.confinput {       # this is the
actual live log file to monitor       file {              path =>
"/myfolder/leak/*.json"              type => "leak"              #codec => json
start_position => ["beginning"]       }}filter{ if [type] == "leak" {         json {
source => message         } }}output {    if [type] == "leak" {        elasticsearch
{            hosts => ["localhost:9200"]            index => "leak-%{+yyyy-MM-dd}"
}    }}
```



read messages in Kibana
Then , while using kibana, you can sort by users , or search for specific things.

To go further :

Maybe you want to extract quickly the url contained in the big.json file ?

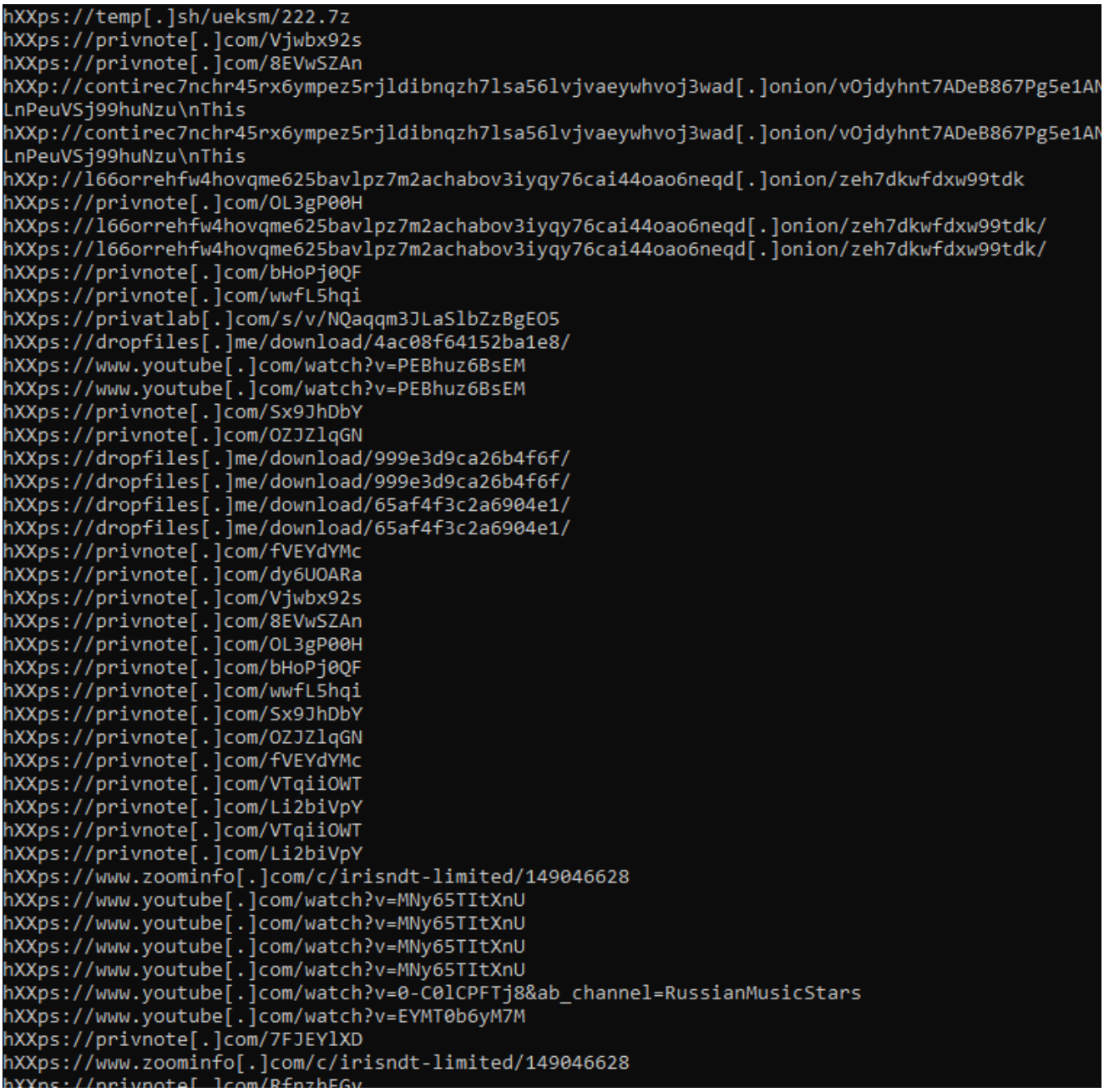
quick hint : use regexp via egrep

```
egrep '(http|https):\/\/[a-zA-Z0-9.\/?=_%&:-]*' -o big.json > url_output.txt
```

And there you are. Oh, and you can use defang (python tool) on your file to read it safely !

(to install defang : pip install defang)

```
defang -i url_output.txt -o url_output_defanged.txt
```



defanged URL observed in leak

It’s now your turn to be imaginative to read things inside this leak. Have fun :)