# Honeypot Documentation

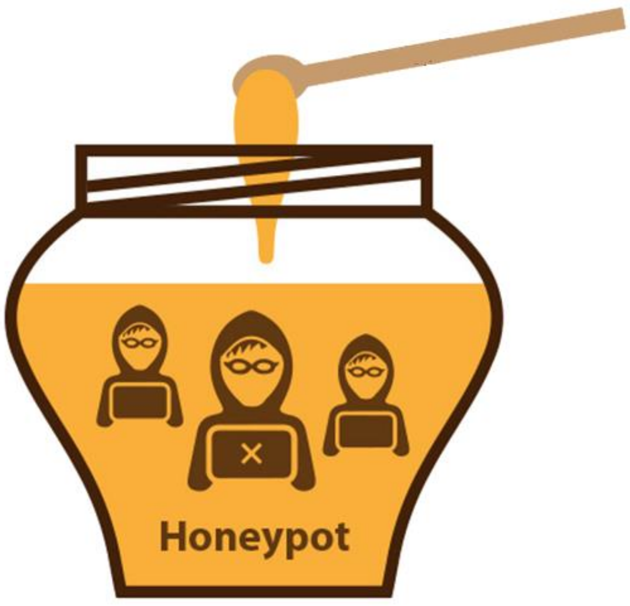
## Table of contents

# 1. Introduction

This documentation is written for Alex Coulon's internship for the Thomas Bata University in Zlin (Czech Republic). This document will elaborate on the chosen technologies for the Linux honeypot solution. The research on different solutions together with a WRM for the chosen solution will be explained thoroughly and motivated.

Then this document will dive deeper into the chosen technologies and technical analysis of the installation and maintenance of the honeypot. As a big integration of this project I included more information about the security part.

As requested by the supervisor this documentation is not in-depth or explaining the installation. He should be able to understand and maintain the system, change basic configuration with the help of this document.



# 2. Research

Setting up a basic Linux honeypot is just some clicks and it's up & running. Finding the right honeypot solution that suits your wishes and expectations will take some time. During the internship I spend some time on researching the ideal solution for the UTB. I found hundreds of options for honeypots and I decided to pick a top 4 who are looking the most interesting for my expectations. These 4 we're weighted via a Weighted Ranking Method to decide which one will suit the expectations and demand of the UTB the best.

## 2.2 Honeypot Differences

### 2.2.1 Production Honeypots

Production honeypots are simplified honeypots widely used to gather basic information about incoming attacks. The possibilities and in-depth options on these types of honeypots are mostly limited to the basics. Production honeypots are mostly in 1 place running on 1 specific service in organizations.

### 2.2.2 Research Honeypots

Research honeypots are full fledged honeypots. These honeypots are used to gather all possible information on attacks, they are focusing on specific hacking strategies and techniques. Research honeypots are mostly divided on all services and locations of an organization.

## 2.3 Honeypot Types

### 2.3.1 Pure honeypots

Pure honeypots are the real deal made in honeypots, a physical server is build like a real part of the company with fake confidential data and specific security issues that are monitored closely. They are looking way more realistic and valuable target for attackers.

### 2.3.2 Low-Interaction

Low interaction honeypots are deployed in production environments in a virtual machine between the real virtual machines. They mostly run a limited set of services following the most common attack vectors or where the organization is most interested in. The interaction part is fairly small, this mostly contains the possibility to setup up a connection via ftp, ssh, telnet but they are failing whenever the necessary data is retrieved from the attack.

### 2.3.3 Mid-Interaction

Mid interaction honeypots are deployed in a production environment in a virtual machine between the real virtual machines. These honeypots are in place to confuse attackers on whether this is a real system or a honeypot, the scope is mostly a specific set of honeypots. The interaction part is bigger then the low-interaction honeypots. Hackers will be able to connect to a server, run scripts, and try to exploit the system they got into. This is mostly run in virtual environments so after each attack the environment is logged, deleted & redeployed.

### 2.3.4 High-Interaction

High interaction honeypots are deployed in a production environment in a virtual machine between the real virtual machines. These honeypots provide a full set of one or more specific services with full access to it via a known exploit. These services are disconnected from the organization data so hackers have full access and can do whatever they want to try in the honeypot. The amount of data gathered is massively higher since they have full control of the service. This is mostly used to follow and monitor the steps a hacker is taking whenever he got into the system and rooted a high privilege user.

## 2.4 Honeypot benefits

1) Low false positive in contrast to IDS/IPS
2) Automated monitoring
3) Runs on low budget/older hardware
4) Expose vulnerabilities
5) Insights into attackers & newest trends
6) Find hackers KPI's in your system
7) Fully customizable to own needs
8) Possible to visualize data
9) Training employees on handling data streams with no impact on production

## 2.5 WRM

The weighing has been done on 5 KPI's (Key Points of Interest).

1) Opensource – 30% - The UTB made clear they are looking for opensource solutions

2) Documentation – 10% - Having well documented and written documentation makes setting up and maintaining the solution much easier.

3) Active Maintained – 20% - A solution that isn't being patched, updated or upgraded is not a good one. We prefer a solution that is being actively maintained.

4) Technical Usage – 30%- We want a solution that is technical not too easy nor hard. We are looking for something what can be maintained by someone without thorough knowledge. We prefer a solution with tools and software we have some previous experience in.

5) Extensibility – 20% - A hardcoded non-customizable solution won't fit any company. The expansibility and configurability of the solution should be broad enough to customize it for our expectations.

| | **Opensource** | **Documentation** | **Active maintained** | **Technical Usage** | **Extensibility** | **Final Score** |
|---|---|---|---|---|---|---|
| *Weight* | *20* | *10* | *20* | *30* | *20* | *100* |
| Dionaea | 20 | 6 | 5 | 20 | 15 | 66 |
| Cowrie | 20 | 8 | 18 | 20 | 10 | 76 |
| Heralding | 20 | 4 | 12 | 15 | 5 | 56 |
| MTPot | 20 | 4 | 0 | 10 | 5 | 39 |

## 2.6 Decision

Following my preference and the result of the WRM method I have decided to continue on using Cowrie for our Linux Honeypot solution. Cowrie is an opensource well maintained honeypot for Linux.

Official description states the following:

"Cowrie is a medium to high interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker. In medium interaction mode (shell) it emulates a UNIX system in Python, in high interaction mode (proxy) it functions as an SSH and telnet proxy to observe attacker behavior to another system.

Cowrie is maintained by Michel Oosterhoff."

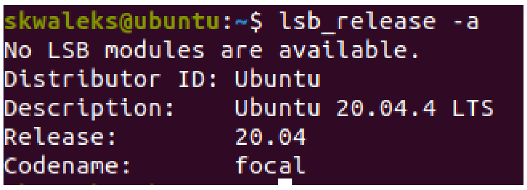
## 3. Environment

Our working environment is an ESXI 7 virtual environment hosted on an Asus desktop located in our network. We have a gigabit line coming from the UTB network on which we created our own subnet with a router and switch to connect our machines & ESXI to it. We have received a gigabit connection that is unmonitored and tagged as test traffic to make sure we aren't being blocked by existing ids/ips or the ISP.

# 4. Cowrie

## 4.2 Linux environment

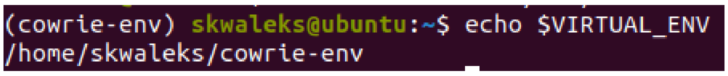
For running my honeypot I decided to use Ubuntu 20.4.4 LTS release as the operating system. This OS gives me the assurance that it won't be discontinued in the near future and (security) updates are still pushed. If there are any updates released in this version they will also be installed ASAP.



```
skwaleks@ubuntu:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.4 LTS
Release:        20.04
Codename:       focal
```

## 4.3 Honeypot virtual environment

For security reasons and optimal implementation of the honeypot solution I build a python virtual environment inside the ubuntu machine in which the honeypot will run.



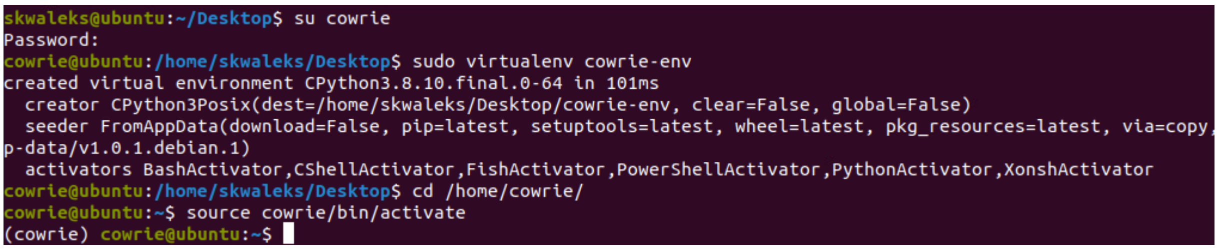
```
(cowrie-env) skwaleks@ubuntu:~$ echo $VIRTUAL_ENV
/home/skwaleks/cowrie-env
```

## 4.4 Starting the virtual-environment

The environment can be started by applying these steps into a bash shell:

- su cowrie (cowrie/XXX)
- sudo virtualenv cowrie-env
- cd /home/cowrie
- source cowrie/bin/activate

After following this procedure you will have a shell into the virtual environment for the honeypot.



```
skwaleks@ubuntu:~/Desktop$ su cowrie
Password:
cowrie@ubuntu:/home/skwaleks/Desktop$ sudo virtualenv cowrie-env
created virtual environment CPython3.8.10.final.0-64 in 101ms
  creator CPython3Posix(dest=/home/skwaleks/Desktop/cowrie-env, clear=False, global=False)
  seeder FromAppData(download=False, pip=latest, setuptools=latest, wheel=latest, pkg_resources=latest, via=copy,
p-data/v1.0.1.debian.1)
  activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator
cowrie@ubuntu:/home/skwaleks/Desktop$ cd /home/cowrie/
cowrie@ubuntu:~$ source cowrie/bin/activate
(cowrie) cowrie@ubuntu:~$ 
```

## 4.5 Running the honeypot

The honeypot can be booted from inside the virtual environment we activated earlier.

| | |
|---|---|
| Bin/cowrie start | This will boot the honeypot |
| Bin/cowrie stop | This will stop the honeypot |
| Bin/cowrie force-stop | This will force-stop the honeypot without saving data |
| Bin/cowrie restart | This will restart the honeypot |
| Bin/cowrie status | This will provide the latest status of the honeypot |
| Bin/cowrie shell | This will get you into the shell of the honeypot without having to connect yourself. |

## 4.6 Configuration

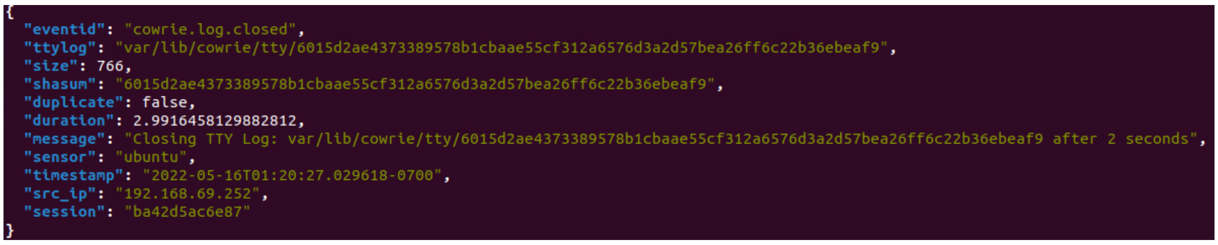
The cowrie honeypot can be customized via the following configuration files:

- /etc/cowrie.cfg  (configuration file for cowrie)
- /share/cowrie.fs.pickle (fake filesystem for the environment)
- /etc/userdb.txt (accepted credentials for telnet/ssh connections)
- Honeyfs/ (file contents for the fake filesystem)
- Honeyfs/etc/issue.net && honeyfs/etc/motd (pre & post login banners)
- Share/cowrie/txtcmds/ (create fake commands)

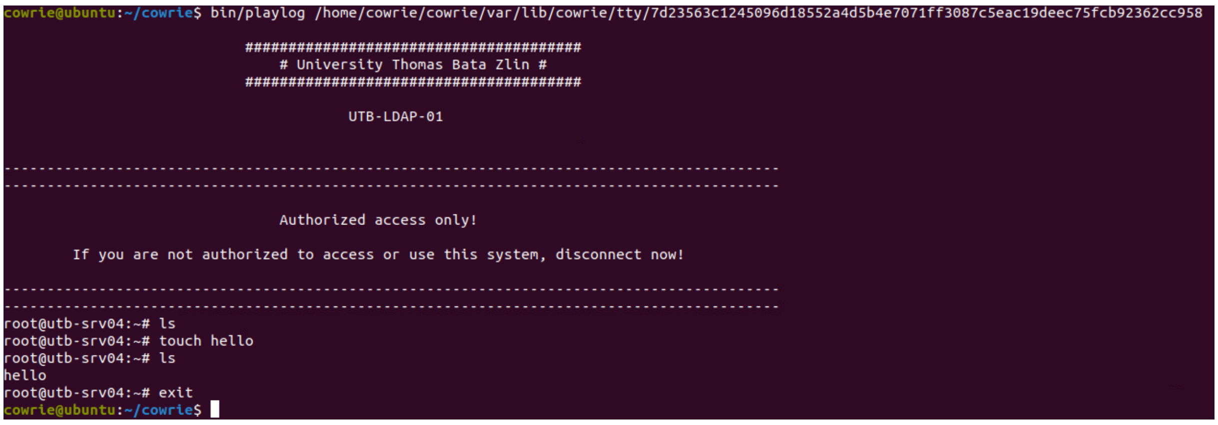
## 4.7 Logging

All attacks are logged by the honeypot and split up in different logging types.

In /home/cowrie/var/log/cowrie there is a cowrie.log & cowrie.json file which are 1 big logging file with everything combined. There are also logs split up by day for more in depth analysis of specific days.  Besides that every unique connection is logged in a tty log which can be replayed with the 'bin/playlog' utility. This will redo the whole attack session in you're gui exactly. Same commands, same timeframes and all.



```
{
  "eventid": "cowrie.log.closed",
  "ttylog": "var/lib/cowrie/tty/6015d2ae4373389578b1cbaae55cf312a6576d3a2d57bea26ff6c22b36ebeaf9",
  "size": 766,
  "shasum": "6015d2ae4373389578b1cbaae55cf312a6576d3a2d57bea26ff6c22b36ebeaf9",
  "duplicate": false,
  "duration": 2.9916458129882812,
  "message": "Closing TTY Log: var/lib/cowrie/tty/6015d2ae4373389578b1cbaae55cf312a6576d3a2d57bea26ff6c22b36ebeaf9 after 2 seconds",
  "sensor": "ubuntu",
  "timestamp": "2022-05-16T01:20:27.029618-0700",
  "src_ip": "192.168.69.252",
  "session": "ba42d5ac6e87"
}
```



```
cowrie@ubuntu:~/cowrie$ bin/playlog /home/cowrie/cowrie/var/lib/cowrie/tty/7d23563c1245096d18552a4d5b4e7071ff3087c5eac19deec75fcb92362cc958

                            #######################################
                                # University Thomas Bata Zlin #
                            #######################################

                                        UTB-LDAP-01


------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------

                                Authorized access only!

        If you are not authorized to access or use this system, disconnect now!

------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
root@utb-srv04:~# ls
root@utb-srv04:~# touch hello
root@utb-srv04:~# ls
hello
root@utb-srv04:~# exit
cowrie@ubuntu:~/cowrie$ 
```

## 4.8 Honeynet

For the Linux honeypots I have 2 honeypots running, each spoofing another service. Both data streams are going to the same data index from which the ELK stack will handle the data. Both honeypot logs are tagged with an tag to differentiate which data is coming from which honeypot.

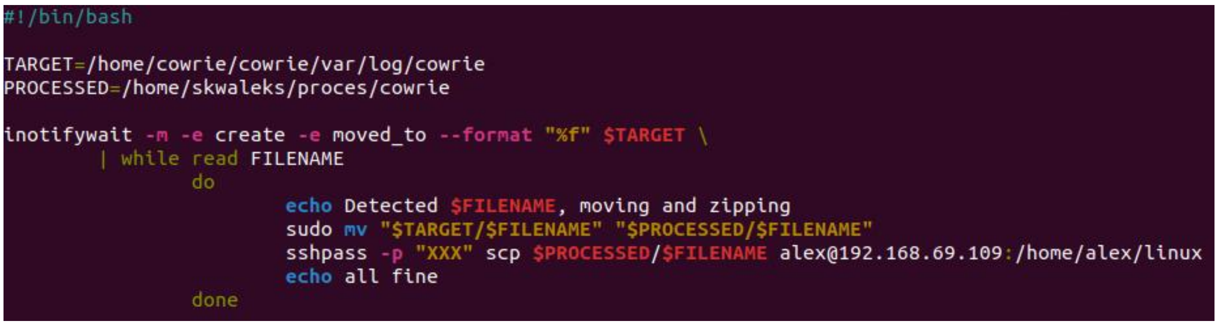
# 5. Backups

## 5.2 Snapshots

To prevent any data loss or broken machines because of crashes our virtual machines are daily snapshotted so we can reinstate to a previous version easy whenever we need to.

## 5.3 SCP

All data coming from the honeypot hackers are pushed to the ELK stack for further processing. I setup automated backups of the honeypot logs. Whenever a hacker quitted the session on the honeypot a file watcher script will automatically copy the log file and push it to an FTP server over SCP with a script that is running in the background 24/7.

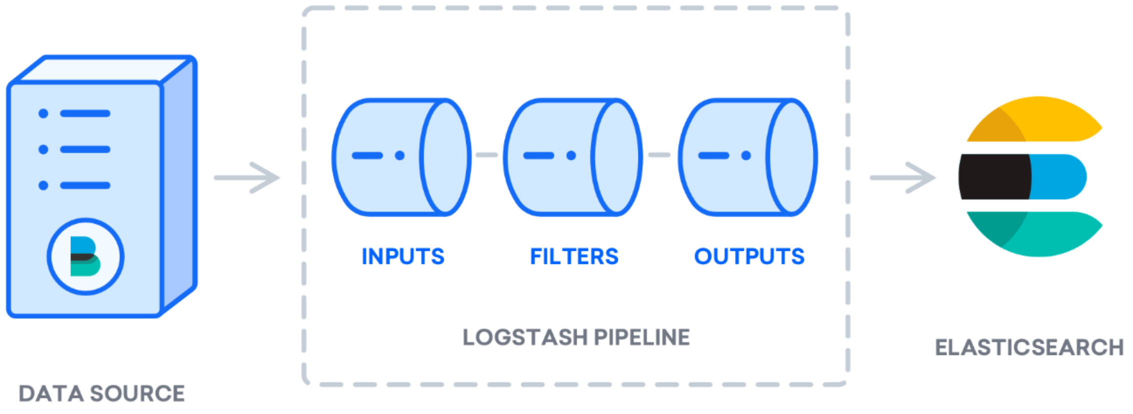


```
#!/bin/bash

TARGET=/home/cowrie/cowrie/var/log/cowrie
PROCESSED=/home/skwaleks/proces/cowrie

inotifywait -m -e create -e moved_to --format "%f" $TARGET \
        | while read FILENAME
                do
                        echo Detected $FILENAME, moving and zipping
                        sudo mv "$TARGET/$FILENAME" "$PROCESSED/$FILENAME"
                        sshpass -p "XXX" scp $PROCESSED/$FILENAME alex@192.168.69.109:/home/alex/linux
                        echo all fine
                done
```

# 6. Logstash

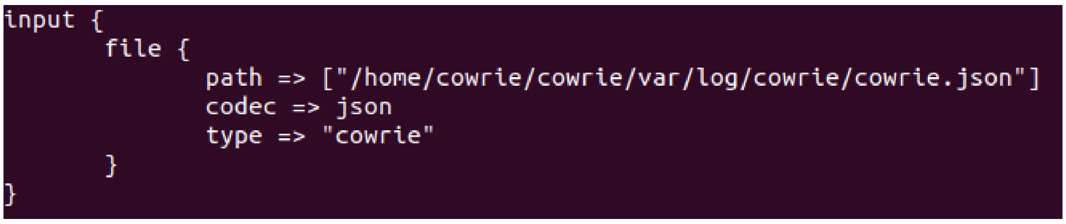
## 6.2 Information

For all of our honeypots we have a Logstash installation and configuration set up to extract data from our logs and push it to Kibana over Elasticsearch. This script is divided in 3 pieces inputs, filters & outputs. We want the data monitored by the honeypots to be pushed to, our Kibana dashboards over our data search engine Elasticsearch. Logstash can connect to our Elasticsearch by making use of the unique generated certificate from Elasticsearch. Logstash can be started via 'systemctl start logstash'. The logstash config can be found in /home/skwaleks/etc/logstash/conf.d/logstash-cowrie.conf



## 6.3 Input

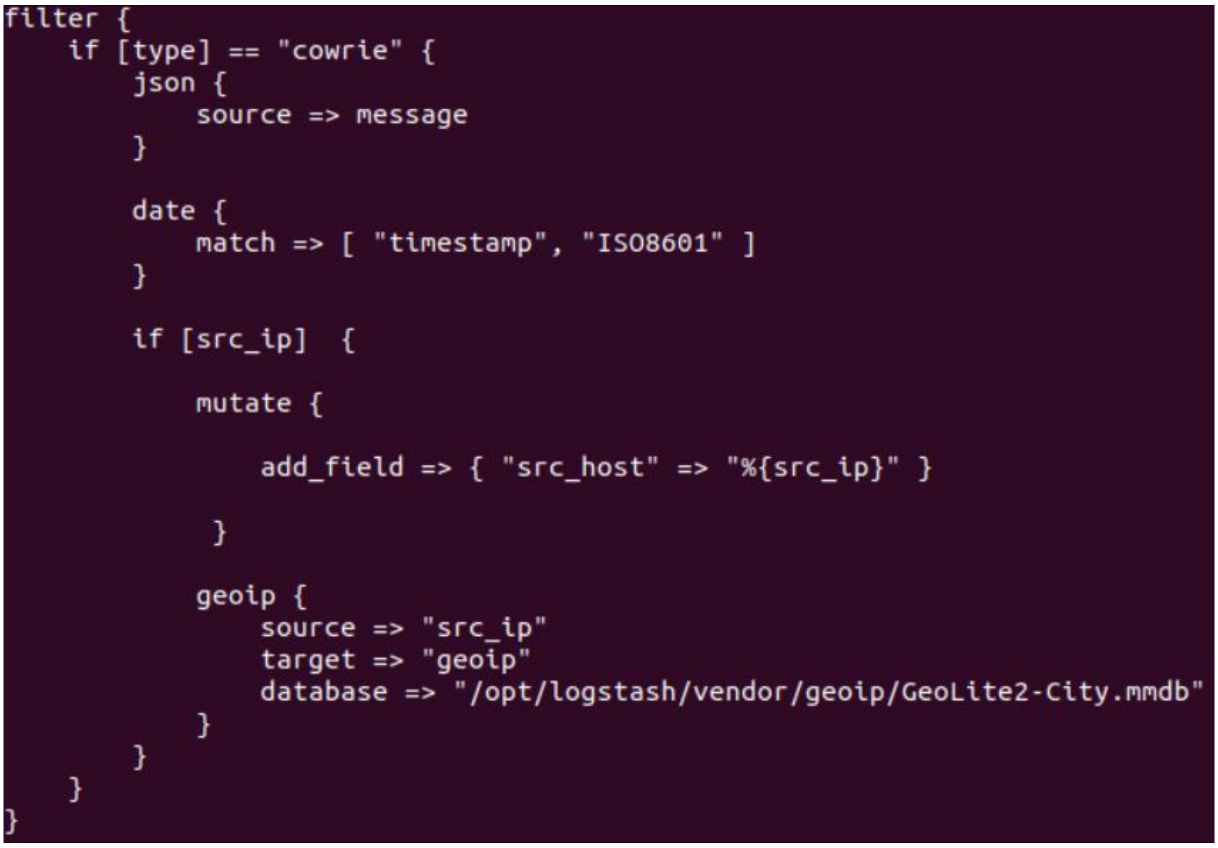
The input part of the script will take live data from the given logfile from the honeypot, this will be given in a JSON format and a type as a name to use for further scripting. The codec will create a great possibility for Logstash to decode your data.



```
input {
        file {
                path => ["/home/cowrie/cowrie/var/log/cowrie/cowrie.json"]
                codec => json
                type => "cowrie"
        }
}
```

## 6.4 Filters

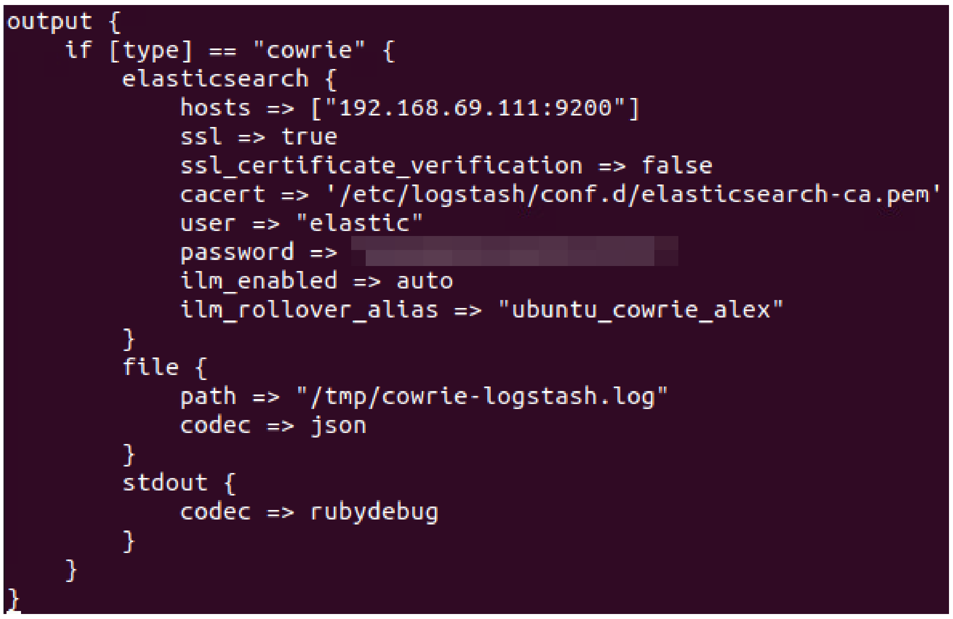
The filters can be used to change, add, modify or interact with the data. The filter field will run automatically when the type is our previously setup cowrie type. As the source of our data log we use the 'message' field from our honeypot log data. To have datetime in the correct format we set our timestamp field to the correct ISO code. The honeypot doesn't add it's own ip (host ip) to the logs by default, because we have 2 honeypots we need to be able to differentiate from which host they are coming, that's why we add another field with the src_host ip in it. For visualizing where the attacks come from we added the geoip2-lite city database so we can automatically point  out cities where the attacks are coming from on a map.



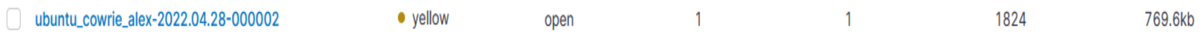
```
filter {
    if [type] == "cowrie" {
        json {
            source => message
        }

        date {
            match => [ "timestamp", "ISO8601" ]
        }

        if [src_ip]  {

            mutate {

                add_field => { "src_host" => "%{src_ip}" }

             }

            geoip {
                source => "src_ip"
                target => "geoip"
                database => "/opt/logstash/vendor/geoip/GeoLite2-City.mmdb"
            }
        }
    }
}
```

## 6.5 Output

The output field will take the data from the input field that pas been parsed trough the filters and will push this to our data search engine called Elasticsearch. Whenever the type is cowrie, this config is executed. We have an Elasticsearch service running on its default port (9200) so that’s where we are trying to push too. For security reasons we implemented SSL certification. We use the elasticsearch-ca.pem file to connect to our Elasticsearch instance. To be able to login on it we provide the username and password to connect. Since all honeypots are pushing to the same Elasticsearch instance we are giving ILM aliases to our data streams to be able to recognize them. All traffic from my honeypots will be found as “ubuntu_cowrie_alex” in Elasticsearch.



```
output {
    if [type] == "cowrie" {
        elasticsearch {
            hosts => ["192.168.69.111:9200"]
            ssl => true
            ssl_certificate_verification => false
            cacert => '/etc/logstash/conf.d/elasticsearch-ca.pem'
            user => "elastic"
            password => [redacted]
            ilm_enabled => auto
            ilm_rollover_alias => "ubuntu_cowrie_alex"
        }
        file {
            path => "/tmp/cowrie-logstash.log"
            codec => json
        }
        stdout {
            codec => rubydebug
        }
    }
}
```

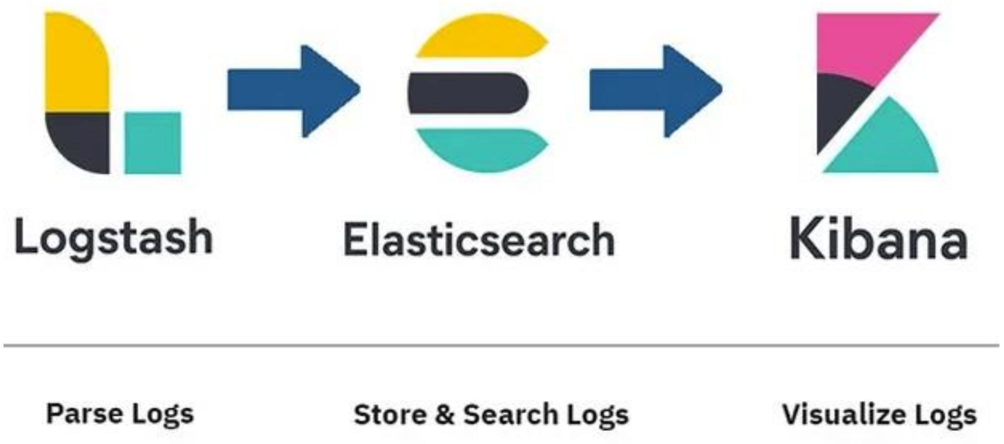


| | | | | | | |
|---|---|---|---|---|---|---|
| ubuntu_cowrie_alex-2022.04.28-000002 | ● yellow | open | 1 | 1 | 1824 | 769.6kb |

# 7. Elasticsearch

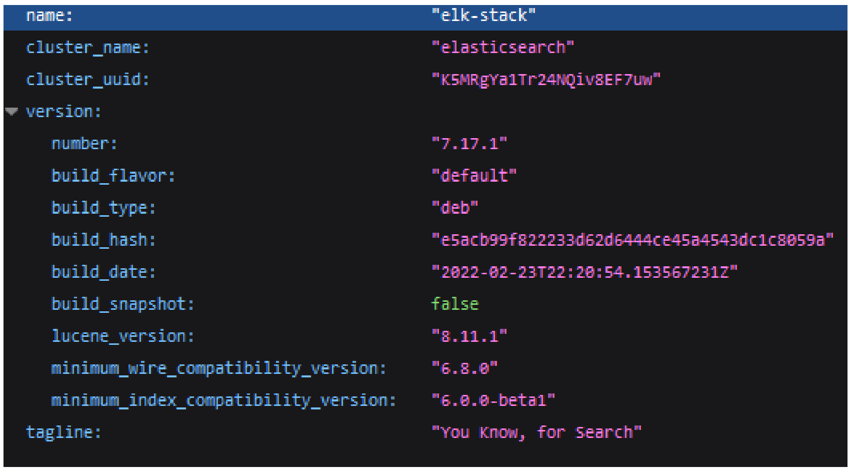
## 7.2 Information

Elasticsearch is our data search engine, analytics & storing place. Elasticsearch can be connected to over ‘https://192.168.69.111:9200’. This instance is called ‘elk-stack’ with a single node cluster “elasticsearch”. Data in Elasticsearch is retrieved from Logstash. Our Kibana instance can pull data from Elasticsearch to visualize and analyze. The Elasticsearch configuration can be found at ‘/etc/elasticsearch’



## 7.3 Security

To connect to Elasticsearch you need a special certificate “elasticsearch-ca.pem”. We implemented password protection on the interface and for the connection. Elasticsearch is running only on the https port.



```
  name:                                 "elk-stack"
  cluster_name:                         "elasticsearch"
  cluster_uuid:                         "K5MRgYa1Tr24NQiv8EF7uw"
▾ version:
    number:                             "7.17.1"
    build_flavor:                       "default"
    build_type:                         "deb"
    build_hash:                         "e5acb99f822233d62d6444ce45a4543dc1c8059a"
    build_date:                         "2022-02-23T22:20:54.153567231Z"
    build_snapshot:                     false
    lucene_version:                     "8.11.1"
    minimum_wire_compatibility_version: "6.8.0"
    minimum_index_compatibility_version: "6.0.0-beta1"
  tagline:                              "You Know, for Search"
```

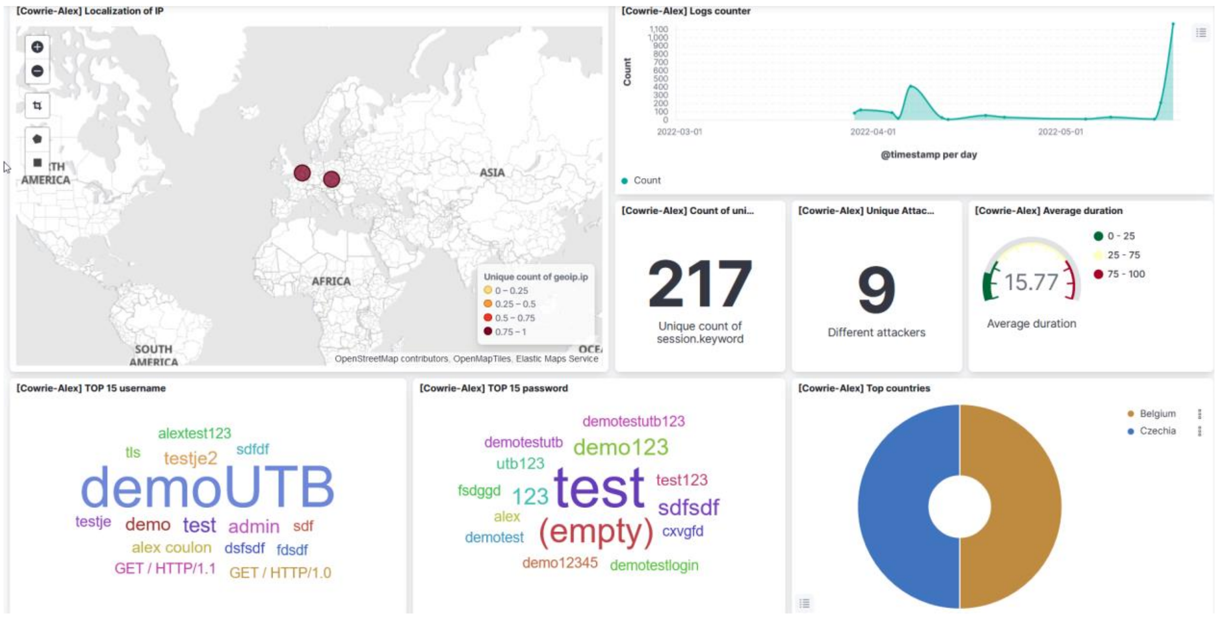
# 8. Kibana

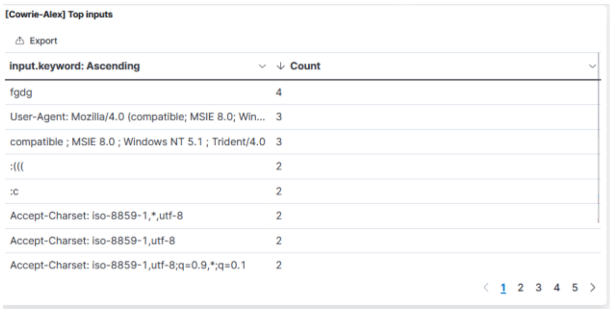
## 8.2 Intro

As the last step in the story of our ELK-stack we have Kibana. Kibana is being used as our visualization and analysing tool. The Kibana dashboard can be found on https://192.168.69.111:5601 . Credentials are custom generated. Kibana will grab the date from our Elasticsearch instance running on port 9200.

## 8.3 DB's

In the Kibana Analytics section we build some dashboards for our honeypots. We created a dashboard for every OS and one general dashboard. For the Linux dashboard I implemented different types of visualization. There are visualisations showing the location of the attack and city. The amount of attacks and unique attackers are shown with an average duration of the attack. The most used usernames & passwords are shown. To get more insight what hackers are trying to exploit on our system there is a table with the most used commands/input in our honeypot. The top 10 host and source ip's are ordered by usage in a table. The most attacked IP and the most cowrie event logs are visualized in a piechart. Besides that there is a table with all used passwords checked if they comply with the password policy from the companies honeypot.

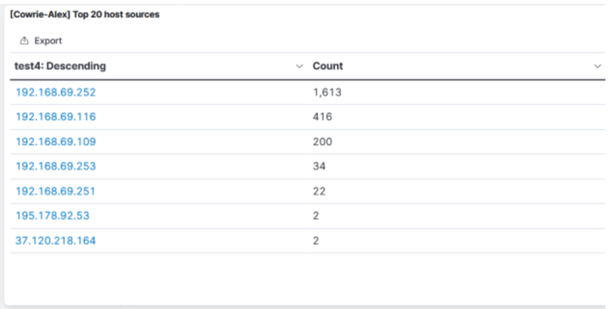



[Cowrie-Alex] Top inputs

Export

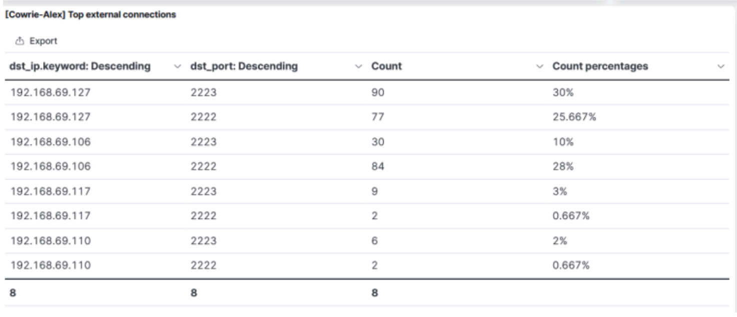
| input.keyword: Ascending | Count |
|---|---|
| fgdg | 4 |
| User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Win... | 3 |
| compatible ; MSIE 8.0 ; Windows NT 5.1 ; Trident/4.0 | 3 |
| :((( | 2 |
| :c | 2 |
| Accept-Charset: iso-8859-1,*,utf-8 | 2 |
| Accept-Charset: iso-8859-1,utf-8 | 2 |
| Accept-Charset: iso-8859-1,utf-8;q=0.9,*;q=0.1 | 2 |

< 1 2 3 4 5 >


[Cowrie-Alex] Top 20 host sources

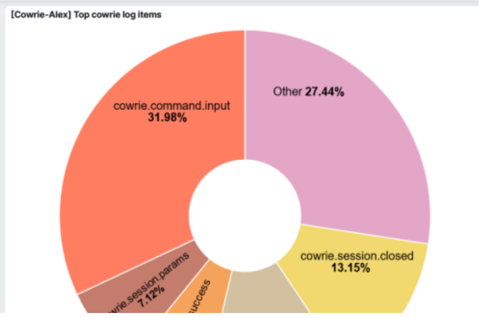
Export

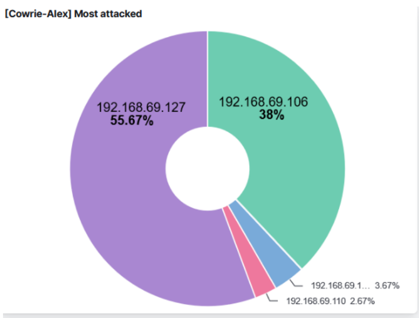
| test4: Descending | Count |
|---|---|
| 192.168.69.252 | 1,613 |
| 192.168.69.116 | 416 |
| 192.168.69.109 | 200 |
| 192.168.69.253 | 34 |
| 192.168.69.251 | 22 |
| 195.178.92.53 | 2 |
| 37.120.218.164 | 2 |

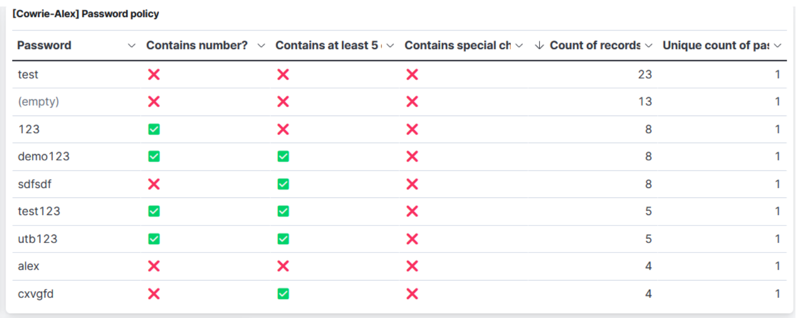

[Cowrie-Alex] Top external connections

Export

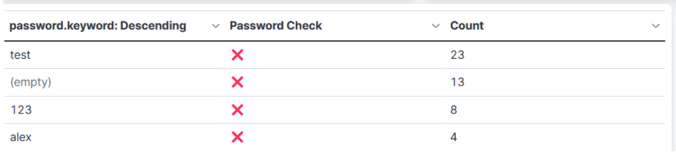
| dst_ip.keyword: Descending | dst_port: Descending | Count | Count percentages |
|---|---|---|---|
| 192.168.69.127 | 2223 | 90 | 30% |
| 192.168.69.127 | 2222 | 77 | 25.667% |
| 192.168.69.106 | 2223 | 30 | 10% |
| 192.168.69.106 | 2222 | 84 | 28% |
| 192.168.69.117 | 2223 | 9 | 3% |
| 192.168.69.117 | 2222 | 2 | 0.667% |
| 192.168.69.110 | 2223 | 6 | 2% |
| 192.168.69.110 | 2222 | 2 | 0.667% |
| 8 | 8 | 8 | |






[Cowrie-Alex] Password policy

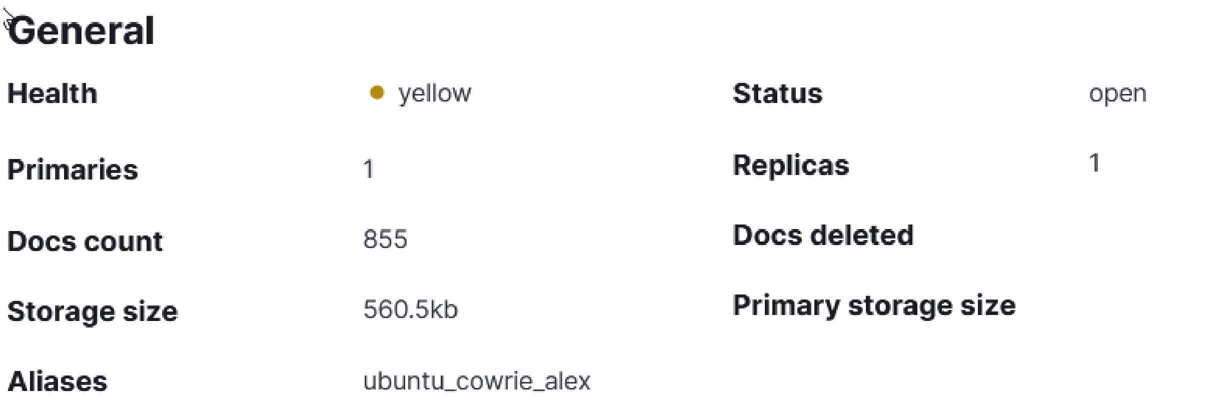
| Password | Contains number? | Contains at least 5 | Contains special ch | Count of records | Unique count of pa |
|---|---|---|---|---|---|
| test | ✗ | ✗ | ✗ | 23 | 1 |
| (empty) | ✗ | ✗ | ✗ | 13 | 1 |
| 123 | ✓ | ✗ | ✗ | 8 | 1 |
| demo123 | ✓ | ✓ | ✗ | 8 | 1 |
| sdfsdf | ✗ | ✓ | ✗ | 8 | 1 |
| test123 | ✓ | ✓ | ✗ | 5 | 1 |
| utb123 | ✓ | ✓ | ✗ | 5 | 1 |
| alex | ✗ | ✗ | ✗ | 4 | 1 |
| cxvgfd | ✗ | ✓ | ✗ | 4 | 1 |



| password.keyword: Descending | Password Check | Count |
|---|---|---|
| test | ✗ | 23 |
| (empty) | ✗ | 13 |
| 123 | ✗ | 8 |
| alex | ✗ | 4 |

## 8.4 Index management & patterns

My honeypot data is stored in an index named ‘ubuntu_cowrie_alex’ on the Elasticsearch instance. To be able to extract the data from this index I created an index pattern on Kibana called ‘ubuntu_cowrie_alex*” with the ‘*’ as a wildcard for all the data from this pattern. This way I can use the data stored in it for visualizations and analytics.


General

| | | | |
|---|---|---|---|
| Health | ● yellow | Status | open |
| Primaries | 1 | Replicas | 1 |
| Docs count | 855 | Docs deleted | |
| Storage size | 560.5kb | Primary storage size | |
| Aliases | ubuntu_cowrie_alex | | |

## 8.5 Custom labels

While having the data from our honeypot, our wishes are not satisfied. We'd like to have some information/visualization on the dashboard with data we can't get from our honeypots. That's why we scripted our own label field in Painless.

### 8.5.1 Private/public

To  make it easier to differentiate public and private ip's on our dashboard we made a script in painless which will check if the attacker's ip is in the private/public ip range. The script will return if it's public/private so we can use this data label in the Kibana visualization.

```
if (doc["src_ip"].size()!=0){
    def sourceip = doc['src_ip_voor_echt'].value;
    if (sourceip != null) {
        String start_range_172 = "172.16";
        String stop_range_172 = "172.32";
        int range_172_sub2 = 16;
        String start_range100 = "100.64";
        String stop_range100 = "100.128";
        int range_100_sub2 = 64;

        while (start_range_172 != stop_range_172) {
            start_range_172 = "172." + range_172_sub2;
            range_172_sub2++;
            if(sourceip.substring(0,6) == start_range_172){
                return
            }
        }
        while (start_range100 != stop_range100) {
            start_range100 = "100." + range_100_sub2;
            range_100_sub2++;
            if(sourceip.substring(0,6) == start_range100){
                return;
            }
        }
        if (sourceip.substring(0,7) == "192.168"){
            return;
        }
        else if(sourceip.substring(0,3) == "10."){
            return;
        }
        else{
            emit(sourceip);
        }
    }
    else {
        emit("Geen ip in src");
    }
}
else
{
    emit("Geen ip opgegeven");
}
```

### 8.5.2 Password check

Nowadays hackers get into systems by having thorough knowledge of the company they are attacking. If an attacker uses passwords that are complying with the company's custom password policies they might have inside information or someone helping him. We made a script that will check a password used by an hacker if it's following the companies password policy.

#### *8.5.2.1 Character Check*

This script is to check if the used password contains a special character or not.

```
if (doc['password.keyword'].size() != 0 ){
    def input = doc['password.keyword'].value;
    int lengte = input.length();
    def lenstring = lengte.toString();
    int teller = 0;
    String label = "❌";
    def array = new def[] {
          "+", "-
", "&&", "||", "!", "(", ")", "{", "}", "[", "]", "^", "~", "*", "?", ":", "%", ";", "."
, "²", "/", "<", "\\", ">", "?"};

    int arraylengte = array.length;
    for (int i=0;i<arraylengte;i++) {
        if (input.contains(array[i])) {
            label = "✅";
        }
    }
    emit(label);
}else {
    emit("❌");
}
```

#### *8.5.2.2 Number Check*

This script is made to check if the password contains a number.

```
if (doc['password.keyword'].size() != 0 ){
    def input = doc['password.keyword'].value;
    int lengte = input.length();
    def lengtestring = lengte.toString();
    int count = 0;
    String label  = "❌";
    def array = new def[] {
                "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};
    int arraylengte = array.length;
    for (int i=0;i<arraylengte;i++) {
        if (input.contains(array[i])) {
            label = "☑";
        }
    }
    emit(label);
}else {
    emit("❌");
}
```

#### *8.5.2.3 Length Check*

This script is made to check if the password follows the policy on length.

```
if (doc['password.keyword'].size() != 0 ){
    def input = doc['password.keyword'].value;
    int lengte = input.length();
    def lenstring = lengte.toString();
    if (lengte < 5) {
        emit("❌");
    }else if (lengte == 5){
        emit("☑");
    }
    else {
        emit("☑");
    }
}else {
    emit("❌");
}
```

### *8.5.2.4 Combined Check*

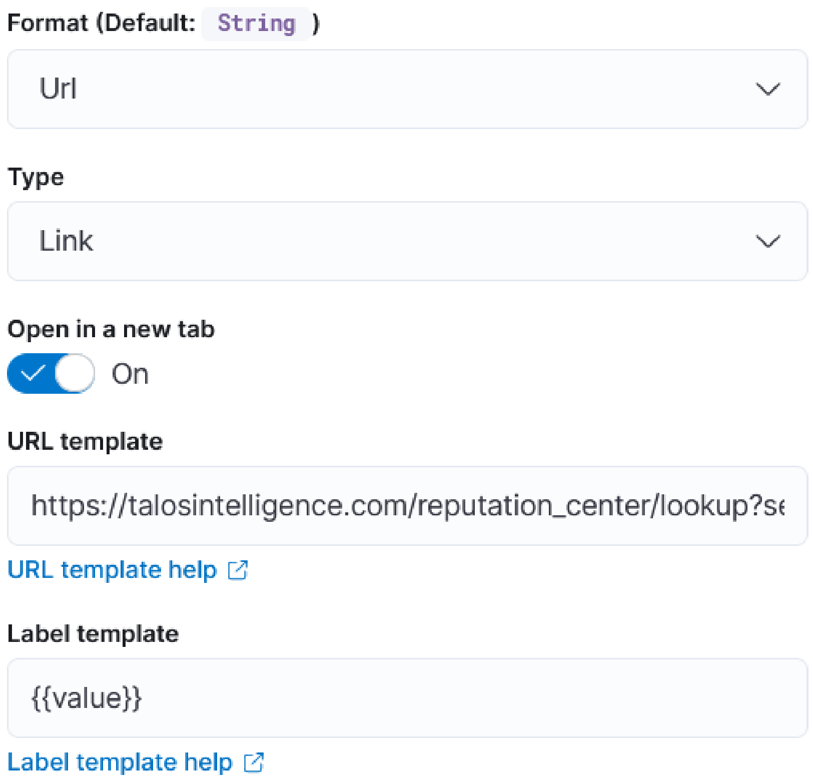
This script combines all of the previous ones to check if the password follows all 3 policy rules.

```
if (doc['password.keyword'].size() != 0 ){
    def input = doc['password.keyword'].value;
    int lengte = input.length();
    def lenstring = lengte.toString();
    int teller = 0;
    String label = "❌";
    def array1 = new def[] {"+", "-", "&&", "||", "!", "(", ")", "{", "}", "[", "]", "^", "~", "*", "?", ":", "%", ";", ".", "²", "/", "<", "\\", ">", "?"};
    def array2 = new def[] {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"};
    if (lengte < 5) {
        emit(label)
    }
    else if (lengte >= 5){
        int arraylengte = array1.length;
        for (int i=0;i<arraylengte;i++) {
            if (input.contains(array1[i])) {
                int arraylengte2 = array2.length;
                for (int l=0;l<arraylengte2;l++) {
                    if (input.contains(array2[l])) {
                        label = "☑";
                        emit(label);
                    }
                    else {
                        label = "❌";
                    }
                }
            }
            else {
                label = "❌";
            }
        }
    }
}
else {
    emit("❌");

}
```

### 8.5.3 Talos Check

Incoming IP addresses may already be known as hackers ip's. That's why I implemented checks on all the public ip's to the talos IP filter. I wrote a small script that will get the IP for the index, put it in a value and make a request to the Talos website.

```
if (doc["src_ip"].size()==0){
    return
}else{
    def source = doc['src_ip'].value;
    if (source != null) {
        emit(source);
    }
    else {
        emit("None");
    }

}
```



Format (Default: String )

Url

Type

Link

Open in a new tab

On

URL template

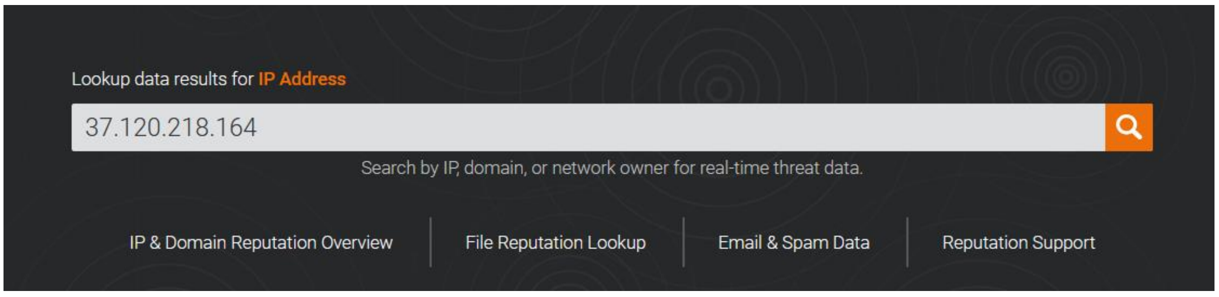
https://talosintelligence.com/reputation_center/lookup?s

URL template help

Label template

{{value}}

Label template help



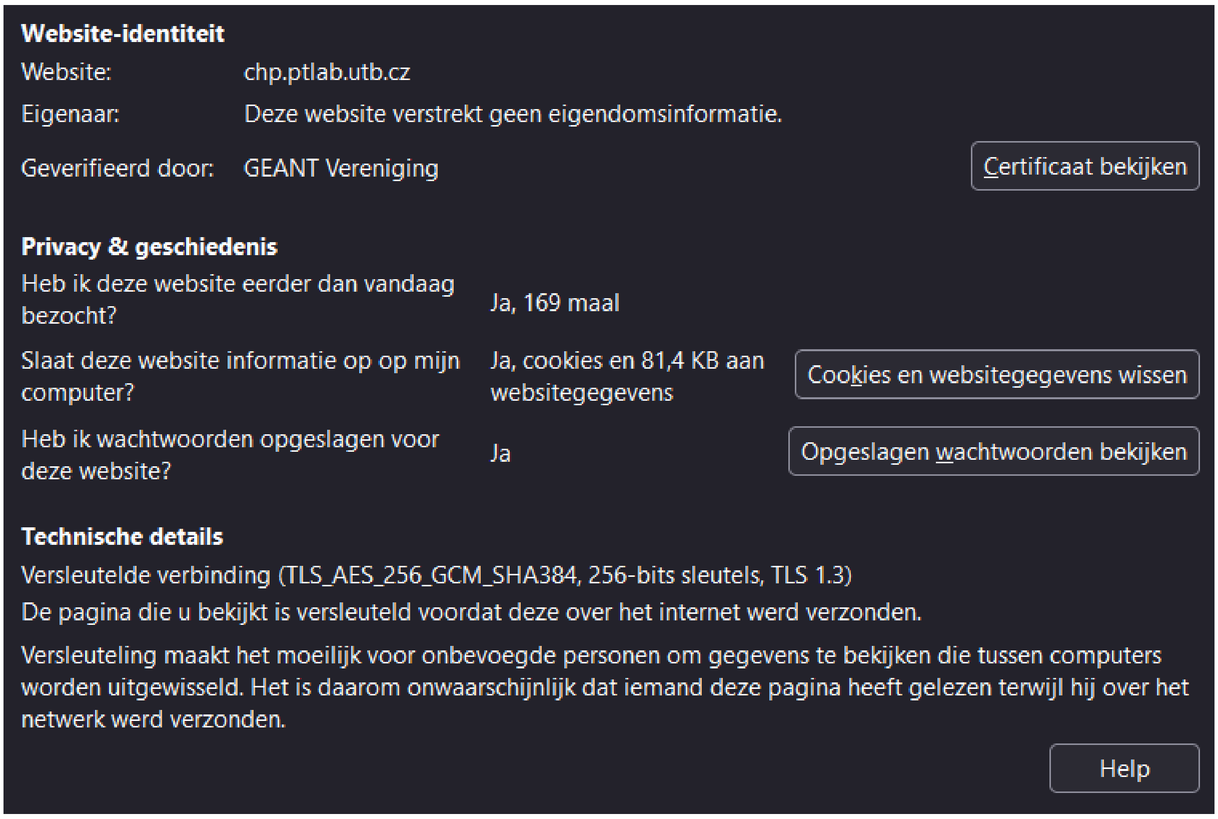
Lookup data results for IP Address

37.120.218.164

Search by IP, domain, or network owner for real-time threat data.

## 8.6 Security

For security reasons we limited the access to the Kibana dashboard by integrating user accounts with strong password policies. Kibana web dashboard has an SSL certificate and running on the PTLab domain of the UTB FAI (chp.ptlab.utb.cz – not accessible from outside the network). To get the domain up & running we setup a nginx site listener on the 443 port with the ssl certificate included in the configuration. All user passwords are stored in the kibana-keystore.



**Website-identiteit**

| | |
|---|---|
| Website: | chp.ptlab.utb.cz |
| Eigenaar: | Deze website verstrekt geen eigendomsinformatie. |
| Geverifieerd door: | GEANT Vereniging |

**Privacy & geschiedenis**

| | |
|---|---|
| Heb ik deze website eerder dan vandaag bezocht? | Ja, 169 maal |
| Slaat deze website informatie op op mijn computer? | Ja, cookies en 81,4 KB aan websitegegevens |
| Heb ik wachtwoorden opgeslagen voor deze website? | Ja |

Cookies en websitegegevens wissen
Opgeslagen wachtwoorden bekijken

**Technische details**

Versleutelde verbinding (TLS_AES_256_GCM_SHA384, 256-bits sleutels, TLS 1.3)

De pagina die u bekijkt is versleuteld voordat deze over het internet werd verzonden.

Versleuteling maakt het moeilijk voor onbevoegde personen om gegevens te bekijken die tussen computers worden uitgewisseld. Het is daarom onwaarschijnlijk dat iemand deze pagina heeft gelezen terwijl hij over het netwerk werd verzonden.