Information & Security

V. Bontchev & V. Yosifova
vol. 43, no. 2 (2019): 264-282
https://doi.org/10.11610/isij.4320

Published since 1998 ISSN 0861-5160 (print), ISSN 1314-2119 (online)

Research Article

Analysis of the Global Attack Landscape Using Data from a Telnet Honeypot

Vesselin Bontchev* (✉), *Veneta Yosifova

National Laboratory of Computer Virology,
Bulgarian Academy of Sciences, Sofia, Bulgaria,
https://nlcv.bas.bg

ABSTRACT:

After the Mirai botnet was discovered in 2016, we decided to set up a honeypot for it and see how widespread it really was. In the process we discovered that many other malicious attackers were using similar attack vectors. This paper outlines the process we went through to pick the right honeypot and the supporting infrastructure (backend database, visualization). This article presents the statistics we have collected from this honeypot, the conclusions we have drawn from these statistics, as well as the tools we have developed to share the data.

ARTICLE INFO:

RECEIVED: 21 AUG 2019
REVISED: 13 SEP 2019
ONLINE: 22 SEP 2019

KEYWORDS:

honeypot, malware, Mirai, botnet, Telnet

Introduction

In August 2016, the malware research group MalwareMustDie discovered a new botnet, infecting various Internet-connected devices (mostly home routers, surveillance cameras, and DVRs) – the so-called "Internet of Things" (IoT) – and used for distributed denial-of-service (DDoS) attacks.[1] On 20th of September, 2016, this botnet was used in a massive DDoS attack against the site of the investigative journalist Brian Krebs.[2] On September 30, 2016, the source code of the bot was released on HackForums as open source. Since then, it has been

✉ Corresponding Author: Tel.: +359 87 6355968 Fax: +359 2 9713710;
E-mail: vesselin.bontchev@nlcv.bas.bg

used and modified by many attackers, spawning hundreds of different variants, and has been responsible for many other massive DDoS attacks (e.g., against the French cloud provider OVH, against the DNS provider Dyn, against the ISPs of Liberia, and so on).Reference literature sources as needed (see also the section on the reference style below).

The bot is spread via TCP port 23 (Telnet) by exploiting weak or default credentials of various IoT devices. It generates random IP addresses (taking care to avoid certain IP ranges, like the one belonging to the US Department of Defense) and tries to connect to port 23 at that address. If the connection is successful and it sees a login prompt, it attempts to log in, using a table of widely used username and password pairs that it carries within itself. If the login is successful, the vulnerable IP is reported to the command-and-control (C2) server of the botnet, which then proceeds to log in, perform some additional checks, upload a copy of the bot for the CPU architecture found on the vulnerable device, and execute it. The C2 server also keeps track of the successfully infected devices and can send them commands to perform some kind of DDoS attack (ten different kinds of such attacks are supported) for a specified time and against a specified target. Antonakakis *et al* provide detailed description of it and analysis of the botnet.[3]

Once the botnet gained notoriety in the press and its source code was made publicly available, many threat actors started using it, or modified copies of it, to create their own botnets. We became interested in how widespread the problem really was. In order to determine this, we decided to set up a honeypot listening to port 23, emulating a vulnerable device, and monitoring the attempts to infect it.

First Attempts

It took us several unsuccessful attempts until we found the right honeypot for our purposes.

Our Requirements

We had several requirements for any eventual honeypot candidate:

1. *It had to be free, open source.* Our organization is on a government budget and our finances are rather limited, so we could not afford to buy a commercial product. Furthermore, we almost certainly would have to tinker with the honeypot, in order to make it produce the kind of results we were interested in, so source code availability was a highly desired property.
2. *It had to be able to capture samples.* We wanted not only to record who was attacking us but also to capture samples of the malware for further analysis.
3. *It had to have some kind of visualization.* We wanted to be able to present our results in easily understandable form. For this, we needed some kind

of visualization software and we wanted to concentrate our efforts on analyzing the observed data, instead of on writing such a software on our own.

MTPot

Our first attempt was MTPot [4] – a honeypot created by the security company Cymmetria and designed to monitor only attacks from the original Mirai bot. Unfortunately, this honeypot had several flaws that made us eventually give up on it:

1. It was monitoring only the attack pattern of the original Mirai bot. Since the source code was released, various miscreants have made “their” version of the bot by modifying various strings in it – many of which resulted in changes of the attack patterns and were, therefore, not detected by this honeypot.
2. Not only this, but it was looking only for the attack pattern of the bot (whose function is only to detect vulnerable devices; not to infect them). It wasn’t monitoring the attempts of the C2 server to infect the honeypot.
3. The honeypot used a Telnet Python library (`telnetsrv`), which is essentially abandonware – buggy, not supported, and not developed. For instance, we discovered that it crashed if the attacker sent a backspace over Telnet.

The honeypot logs the credentials (username and password) used by the attacking bots – but does not attempt to capture samples of them for analysis. The credentials used by the original Mirai are well-known, so this did not bring us any interesting new information, and it was not clear whether any new captured credentials were from new Mirai variants or from different bots.

Creating Our Own

Our next attempt was to write our own honeypot, initially along the lines of MTPot, but using a state-of-the-art Python communication library, `twisted`. Unfortunately, using this library is somewhat non-trivial and its documentation is not easy to understand.

Furthermore the Mirai bot uses the Telnet protocol in a somewhat peculiar way. It expects the Telnet server to provide a specific set of communication options and fails to connect, if they are not present. This is a design flaw, not an intentional protection, but it means that the bot is unable to connect to computers running the standard `telnetd` server, like Linux desktops. Clearly, the author of the bot had tested it only with the Telnet server provided by the `busybox telnetd` command available on IoT devices. However, we did not know this at the time and were rather frustrated when our Telnet server worked just fine if tested manually with the `telnet` client command but the bots wouldn’t connect to it. So, we had to eventually abandon the idea of creating the honeypot ourselves.

T-Pot

Our next consideration was T-Pot – a system of various honeypots developed by Deutsche Telecom.[5] It has many advantages but also many disadvantages as well.

Advantages of T-Pot

1. It is very extensive and is actively developed.
2. *It is very easy to install.* The installation is designed for inexperienced users. Basically, you need a dedicated machine with 4-12 Gb RAM and 64 Gb free disk space, you download an ISO image, burn it on a DVD, boot the machine from this DVD and it proceeds to install everything necessary.
3. *It has many honeypots.* Besides a Telnet honeypot, it also has many others – for SSH, SMB, ADB, ElasticSearch, MySQL, HTTP, SIP, MQTT, FTP, UPNP, RDP, industrial controllers, etc.
4. *It works great.* It is very well-refined and well-tested, has built-in visualization to observe pictures of the gathered data and so on.
5. *By using it, one is helping a community.* T-Pot is designed to be used by a distributed system of installations that, by default, forwards the gathered data to a community that specializes in threat intelligence.

Unfortunately, T-Pot also has some disadvantages, which resulted in our eventual decision not to use it.

Disadvantages of T-Pot

1. *It is rather difficult to tinker with.* While T-Pot makes it easy for inexperienced users to install-and-forget it, it is not very easy to modify and reconfigure its many parts, in order to get what you actually need – if it is something different from the default.
2. *It keeps only 24 hours' worth of data.* At the end of this period all gathered data is discarded and the various honeypots are reinitialized. While this is useful to get a picture of what is happening *right now*, we wanted to keep long-term logs and other data in order to spot trends and to conduct further long-term analysis.
3. *It needs a dedicated Linux machine.* While we eventually ended up getting that, we didn't have one when we started our quest for a honeypot. All we had was a rather limited virtual machine running on a Windows 10 host, which itself wasn't very powerful and couldn't afford to run a VM with 12 Gb RAM on it.
4. *It relies heavily on the ELK (ElasticSearch, LogStash, Kibana) stack.* As we later discovered, for our conditions this was somehow unreliable and better-looking alternatives existed.

The Solution – Cowrie

Eventually, we managed to discover a solution that fit our needs almost perfectly. This was Cowrie – a Telnet and SSH honeypot, developed by Michel Oosterhof.[6]

The Advantages of Cowrie

Cowrie has many advantages, which suited our needs almost perfectly. In particular:

1. *It is under active development.* The author makes improvements and adds new features all the time. He is very helpful, answers questions quickly, and was instrumental with getting our instance of the honeypot up and running.
2. *It has excellent documentation.* While other open-source products often leave the user digging into the code in order to understand how to use the product or, at best, have just installation and basic usage instructions that leave you scratching your head every time you want to modify something or do something non-standard with them, the documentation of Cowrie is simply excellent. It explains in details how to install and configure every aspect of it and when occasionally something was unclear, the author modified the documentation to clarify it.
3. *It works great.* Despite being a relatively complex product, Cowrie works almost flawlessly, does not crash, masquerades as a real machine very well (although a dedicated human attacker can still detect it), has extensive logging, captures samples for later analysis, and, of course, works not just for Mirai but for any attacker that tries to log in via Telnet or SSH.
4. *It offers an emulated shell to the attacker.* For security reasons, Cowrie never runs untrusted code. However, it emulates dozens of Linux commands, so that the attacker is led to believe that they are indeed working on a real Linux machine that they have managed to log into – while every single step of theirs is logged thoroughly.
5. *It can store the logged data into many kinds of databases.* This is achieved via output plugins – a system that is very extensible. Cowrie has a large set of such plugins – for JSON logs, ElasticSearch, HPFeeds, InfluxDB, Kafka, syslog, MongoDb, MySQL, Redis, Slack, Splunk, and others. Some output plugins can send captured samples for analysis and scanning to Cuckoo Sandbox or VirusTotal. If necessary, an expert user can write their own output plugin in the unlikely case that they need to log into some kind of database that Cowrie doesn't support yet.
6. *It is not just a Telnet honeypot.* It also contains an SSH honeypot, which benefits from the same overall design (emulated commands, simulated file system, output plugins, etc.). While we weren't originally looking for an SSH honeypot (we started by being interested only by Mirai), after experimenting with the one in Cowrie and seeing the picture of the ongoing attacks over this protocol, we decided to use this kind of honeypot, too.

The Problems with Cowrie

However, in the process of setting up and using Cowrie for a while, we discovered that it (or parts of it), too, had its set of problems. None of them turned out being a showstopper, but some of them caused us to spend considerable efforts to circumvent them. We'll share our results here in the hope that they might help others who decide to use this honeypot.

ElasticSearch

Initially, we configured Cowrie to log all acquired data into an ElasticSearch database. This decision was spurred by the fact that this is the configuration used by the author himself and the documentation about how to set up and how to configure this part of the honeypot was most extensive. Unfortunately, we lived to regret this decision.

One minor problem for us was that the data query language of ElasticSearch is somewhat unusual and we had problems figuring out the exact queries necessary to visualize what we were interested in.

But the biggest problem was that after a power failure, the whole database became corrupted. It wasn't just some portion of data that could be discarded as lost – the whole database simply stopped working. We couldn't find anything in it and we couldn't write new data to it – we were getting various strange errors all the time. After several unsuccessful attempts to repair the database, we ended up scrapping it completely and recreating it from the data in the JSON logs (we wrote a small Python script for extracting it from there and recording it in ElasticSearch). The process took more than a week, since we had collected several months' worth of data.

To our dreadful disappointment, just two days after we finished recreating the database, we had another power failure, which again resulted in a total corruption of the database, making it unusable. At this point we decided that the situation was intolerable.

We consulted a database specialist, who expressed a rather negative opinion of ElasticSearch and advised us to switch to a robust, commercial database instead. He really meant Oracle, but we could not afford that, so we decided to go for MySQL instead.

It is perfectly possible that for others, ElasticSearch would work well as a database for storing the data logged by the honeypot. For instance, if the database server is in the cloud, there is no danger of power failures. Furthermore, we were advised that ElasticSearch should really be used in a cluster, so that a failure in one of the nodes can be circumvented by the others. Unfortunately, we could not afford either of these solutions.

Geolocation

We were interested in where the attacks were coming from – not just what IP addresses they came from. In order to obtain this information, the IP addresses have to be geolocated.

Cowrie itself does not do IP geolocation. However, when ElasticSearch is used to store the gathered data, the documentation explains how to use FileBeat and LogStash (parts of the ELK stack) to “enrich” the logs with geolocation data for every recorded IP address. Unfortunately, since we decided not to use the ELK stack, this approach was off-limits to us.

Our first approach aimed at circumventing this problem tried to perform geolocation from within MySQL. We downloaded the free geolocation databases from MaxMind,[7] converted them into MySQL tables in the database used by Cowrie, and modified the queries performed by the visualization to also query the geolocation data of every visualized IP address. Unfortunately, this turned out being prohibitively slow and we had to give up on that idea.

Or next solution was to modify both Cowrie and the database schema used by it, in order to record geolocation data obtained in real-time, as every IP address was recorded in the database. Basically, our thinking was that as the ELK stack adds geolocation data to the database in real-time, so should the MySQL output plugin.

MySQL

There are a lot of attackers on the Internet, attacking randomly selected IP addresses via the Telnet and SSH protocols. Approximately 2.5 attacks occur every second, as we shall demonstrate later in this paper. As a result, Cowrie collects a humongous amount of information. Some tables in the MySQL database quickly expanded to tens of millions of rows and became tens of gigabytes in size after just a few months of constant monitoring of these two protocols. What is worse, the database schema used by Cowrie is not very efficient. For instance, every time an attacker issues a command, this command is recorded in its entirety in a table without checking whether it has been seen before. This flaw results in this particular table expanding very quickly and the MySQL server starts suffocating after a certain time.

Initially, we used the same MySQL server that is normally used to drive the web site of our Lab. After a few months of running the honeypot, we noticed that our site started failing in bizarre ways. It turned out that the MySQL server had begun spontaneously to stop responding. Initially we weren’t certain what exactly the problem was and kept restarting it after each failure – until one day it refused to restart. Investigation showed that the server had become simply unable to handle the very large and frequently updated database of the honeypot.

In order to address all these problems, we had to make several changes in Cowrie and in the MySQL database schema used by it. First, as explained in the previous section, we added geolocation data. Second, we made it store only unique versions of the commands entered by the attackers. Third, we made the storing of such commands in the database optional and controlled from an option in Cowrie’s config file.

Unfortunately, the author refused to merge our modifications with the main project. His objection was that they modified significantly the database schema, making it incompatible with a visualization tool named Kippo-Graph.[8] This tool was designed to visualize the data gathered by Kippo, which was the Telnet honeypot that Cowrie is based on – and the author had kept compatibility of the database used by it and its visualization tool. We attempted to contact the author of Kippo-Graph, in order to ask him to implement support for both database schemas – something, which is perfectly possible and even relatively easy. Unfortunately, it turned out that Kippo-Graph is basically abandonware, nobody has been developing it for years, and there was nobody available to implement our request.

Therefore, we were forced to go with our own fork of Cowrie, where all these changes of ours were implemented.[9] Unfortunately, this has the disadvantage that our version is getting out-of-sync with the original and the latest developments haven't been ported to it yet.

Visualization

Since we decided to forego dependence on the ELK stack, and since the Kippo-Graph visualizer did not satisfy us, we were forced to start looking for a new data visualization tool. Fortunately, somebody on Twitter suggested us to try Grafana [10] and we immediately fell in love with it.

Grafana has many advantages over Kibana (the visualizer of the ELK stack). To begin with, it is much more beautiful. While Kibana can handle only ElasticSearch as a data source, Grafana supports many kinds of data sources via plugins. It does support ElasticSearch too but, most importantly for us, it supports MySQL.

So this is what we ended up using.

Our Final Setup

To summarize, here is the final setup we for our Telnet and SSH honeypot:

- Honeypot: our modified fork of Cowrie
- Backend: MySQL database
- Visualization: Grafana
- Web server: Nginx (needed to run Grafana)
- HTTPS certificate authority for the web server: Let's Encrypt (provides free web site certificates)

The visualization of our honeypots can be accessed via the URL

https://pandora.nlcv.bas.bg/grafana

Accessing it requires credentials (username and password). A demo account, capable of only viewing the data visualization but not of creating new panels or of otherwise modifying the dashboard can be accessed by using the username guest and the password guest.

The same site is also available over Tor via the URL

http://nlcv2zqfqzd2qiwr.onion/grafana

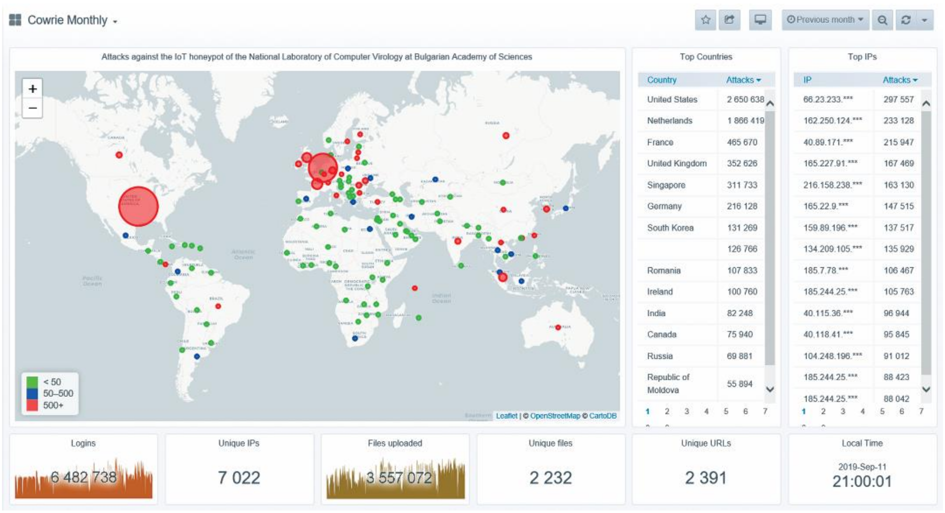
This was done mostly as an intellectual exercise (we wanted to learn how to set up onion sites) but it also has the benefit that the site is hard to block or otherwise censor, should some country's government decide to do so for whatever reason.

The next few figures illustrate the various information panels on the dashboard associated with our Telnet and SSH honeypots. Normally, the dashboard shows information gathered during the past 24 hours, but this is highly configurable. In the next figures we have used data for the whole month of August, 2019.

Fig. 1 shows the main panel of the dashboard associated with Cowrie. It contains a geographical world map, showing where the attacks are coming from. Each circle represents a country from which IP addresses have attacked the honeypot. The circle is not at the particular coordinates of the IP addresses but roughly at the geographical center of the corresponding country. The size of the circles is proportional to the number of attacks from this country and they are color-coded.

To the right of the map, there are two tables. The first shows the names of the countries attacking the honeypot, with the number of attacks coming from each one of them, sorted in decreasing order of attacks. The other table contains the IP addresses that are most actively attacking the honeypot, again sorted in reverse order of the number of attacks. Since in the terms of the EU GDPR rules, an IP address is personally identifiable information, we are anonymizing the last octet of each IP address for privacy reasons.

At the bottom of the image, there are several indicators: the total number of login attempts (successful or not) for the specified period, the number of unique IP addresses they are coming from, the total number of files, uploaded by the attackers, the number of unique files, the number of unique URLs from which


Cowrie Monthly
Previous month
Attacks against the IoT honeypot of the National Laboratory of Computer Virology at Bulgarian Academy of Sciences
< 50
50–500
500+
Leaflet | © OpenStreetMap © CartoDB

Top Countries

Country	Attacks
United States	2 650 638
Netherlands	1 866 419
France	465 670
United Kingdom	352 626
Singapore	311 733
Germany	216 128
South Korea	131 269
	126 766
Romania	107 833
Ireland	100 760
India	82 248
Canada	75 940
Russia	69 881
Republic of Moldova	55 894

1 2 3 4 5 6 7

Top IPs

IP	Attacks
66.23.233.***	297 557
162.250.124.***	233 128
40.89.171.***	215 947
165.227.91.***	167 469
216.158.238.***	163 130
165.22.9.***	147 515
159.89.196.***	137 517
134.209.105.***	135 929
185.7.78.***	106 467
185.244.25.***	105 763
40.115.36.***	96 944
40.118.41.***	95 845
104.248.196.***	91 012
185.244.25.***	88 423
185.244.25.***	88 042

1 2 3 4 5 6 7

Logins	Unique IPs	Files uploaded	Unique files	Unique URLs	Local Time
6 482 738	7 022	3 557 072	2 232	2 391	2019-Sep-11 21:00:01

Figure 1: The main panel of the Cowrie dashboard.

they are being fetched, and the local time of the visitor (not necessarily of the honeypot).

In addition to this main panel, the dashboard contains several secondary panels.

Fig. 2 shows a bar chart of the number of login attempts for every hour of the specified period. Fig. 3 shows a pie chart, illustrating what percentage of the attacks come via the Telnet protocol and what – via the SSH protocol. Fig. 4 shows the top five URLs, from which the uploaded files are most often fetched, sorted by decreasing popularity.

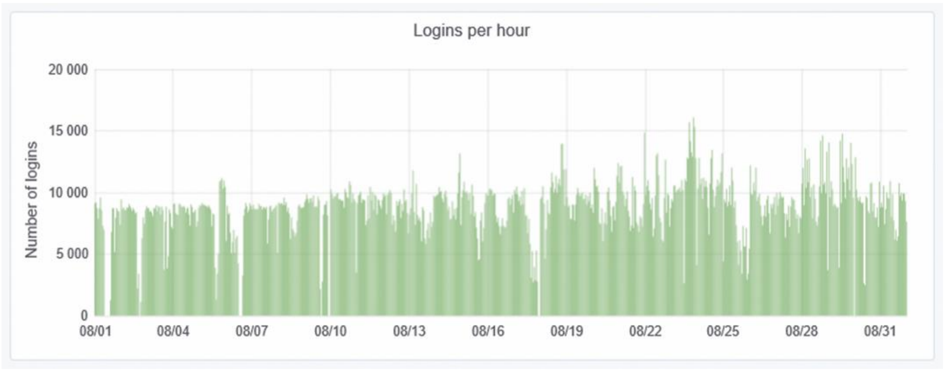


Figure 2: Hourly chart of the number of login attempts.

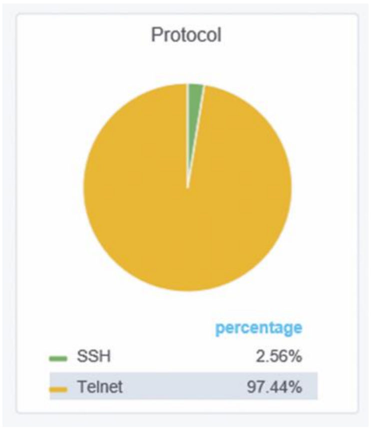
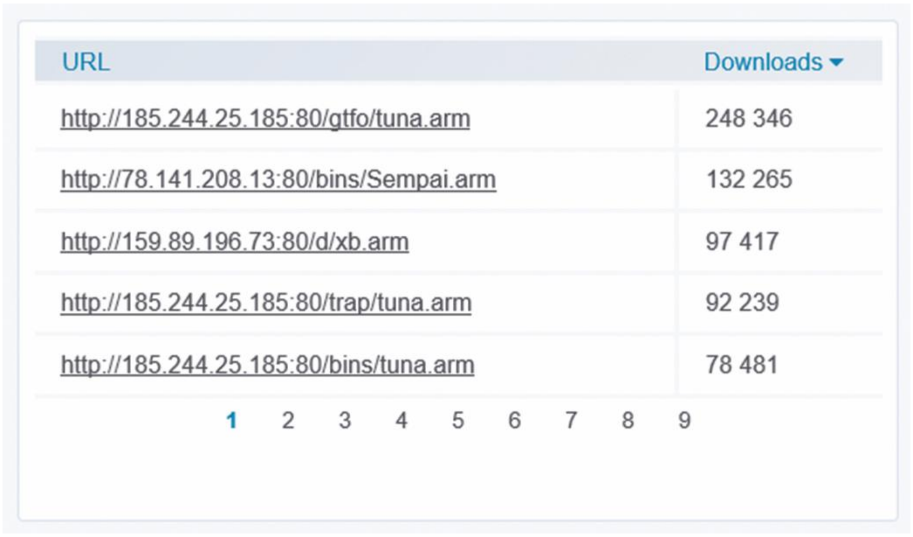


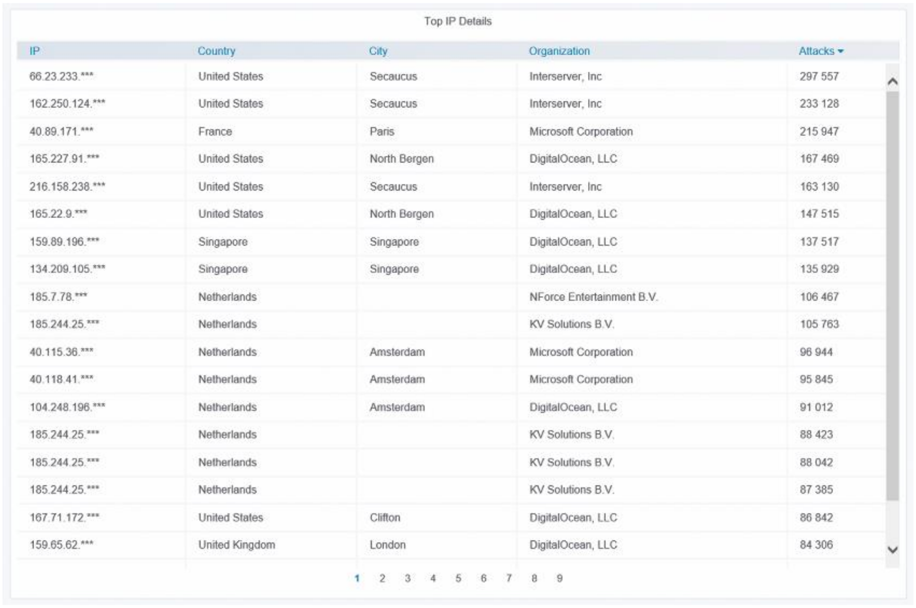
Figure 3: Relative percentage of attacks per protocol.



URL	Downloads ▾
http://185.244.25.185:80/gtfo/tuna.arm	248 346
http://78.141.208.13:80/bins/Sempai.arm	132 265
http://159.89.196.73:80/d/xb.arm	97 417
http://185.244.25.185:80/trap/tuna.arm	92 239
http://185.244.25.185:80/bins/tuna.arm	78 481

1 2 3 4 5 6 7 8 9

Figure 4: Top five URLs from which the uploaded files are fetched.



Top IP Details

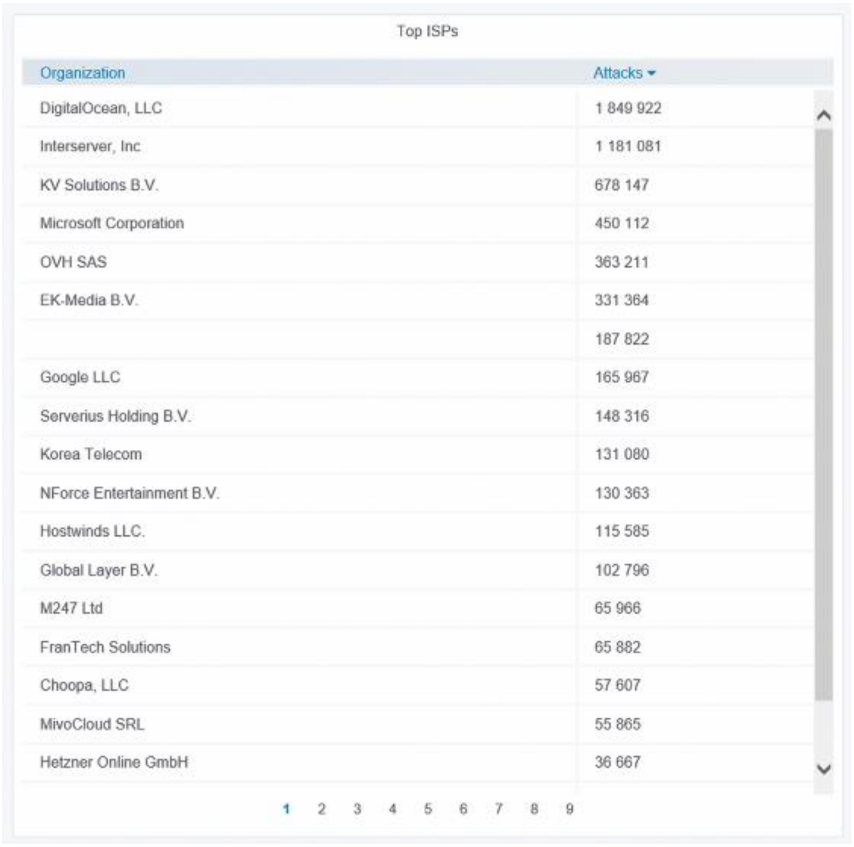
IP	Country	City	Organization	Attacks ▾
66.23.233.***	United States	Secaucus	Interserver, Inc	297 557
162.250.124.***	United States	Secaucus	Interserver, Inc	233 128
40.89.171.***	France	Paris	Microsoft Corporation	215 947
165.227.91.***	United States	North Bergen	DigitalOcean, LLC	167 469
216.158.238.***	United States	Secaucus	Interserver, Inc	163 130
165.22.9.***	United States	North Bergen	DigitalOcean, LLC	147 515
159.89.196.***	Singapore	Singapore	DigitalOcean, LLC	137 517
134.209.105.***	Singapore	Singapore	DigitalOcean, LLC	135 929
185.7.78.***	Netherlands		NForce Entertainment B.V.	106 467
185.244.25.***	Netherlands		KV Solutions B.V.	105 763
40.115.36.***	Netherlands	Amsterdam	Microsoft Corporation	96 944
40.118.41.***	Netherlands	Amsterdam	Microsoft Corporation	95 845
104.248.196.***	Netherlands	Amsterdam	DigitalOcean, LLC	91 012
185.244.25.***	Netherlands		KV Solutions B.V.	88 423
185.244.25.***	Netherlands		KV Solutions B.V.	88 042
185.244.25.***	Netherlands		KV Solutions B.V.	87 385
167.71.172.***	United States	Clifton	DigitalOcean, LLC	86 842
159.65.62.***	United Kingdom	London	DigitalOcean, LLC	84 306

1 2 3 4 5 6 7 8 9

Figure 5: IP geolocation details.

Fig. 5 shows the next panel. It contains geolocation information about the 20 IP addresses that have attacked the honeypot the most often, sorted by decreasing number of attacks. The geolocation information includes the country and city where the IP address resides, as well as the organization that owns this IP address. As on the small panel with IP addresses (Fig. 1), the last octet of each address is anonymized.

Finally, the last panel (Fig. 6) shows the names of the organizations that own the IP addresses that are attacking the honeypot the most often, sorted by decreasing number of attacks. Almost all of them are either cloud hosting providers or major Internet service providers, with DigitalOcean, as a rule, being the source of a disproportionably large number of attacks.


Top ISPs

Organization	Attacks
DigitalOcean, LLC	1 849 922
Interserver, Inc	1 181 081
KV Solutions B.V.	678 147
Microsoft Corporation	450 112
OVH SAS	363 211
EK-Media B.V.	331 364
	187 822
Google LLC	165 967
Serverius Holding B.V.	148 316
Korea Telecom	131 080
NForce Entertainment B.V.	130 363
Hostwinds LLC.	115 585
Global Layer B.V.	102 796
M247 Ltd	65 966
FranTech Solutions	65 882
Choopa, LLC	57 607
MivoCloud SRL	55 865
Hetzner Online GmbH	36 667

Figure 6: Organizations that own the IP most often attacking addresses.

Additional Data Processing Tools

While collecting and examining all the information gathered by the honeypot is undoubtedly interesting, we try to be useful to the community by sharing it in various ways. This section outlines the main ways in which this sharing occurs.

Monthly Reports

One of us was asked to write monthly reports, summarizing the results seen by our honeypots during the previous month. Since this is a rather boring and uninteresting job, we decided to automate it completely. The result was a sophisticated bash script, which can accept various command line options that determine the contents and the format of the report that is produced. The script analyzes the JSON log files for a period, specified via a command line option (by default – covering the previous month), creates a human-readable text report, outlining the results observed by the honeypot during this period, and optionally sends it by e-mail to a specified e-mail address. The script is run automatically every month by a cron job and the report is send to the Director of our Lab, who then sends it to the Bulgarian Defense Institute.

The report includes such information as:

- the total number of attacks;
- the number of IP addresses which they have come from;
- the number of different countries where these IP addresses reside;
- the number of attacks coming from the territory of Bulgaria;
- the number of different attacking IP addresses residing there;
- a list with detailed information about these Bulgarian IP addresses;
- a list of the countries from which the attacks have originated, as well as the number of attacks originating from each country, sorted by decreasing number of attacks;
- a complete list of the URLs from which the files uploaded to the honeypot have been fetched from, together with the number of times they have been fetched from there, and sorted by decreasing order of popularity;
- a complete list of the distinct attacking IP addresses, with geolocation information about each one of them.

The Honeypot Information Sharing System

In order to handle requests for information obtained by our honeypot, which requests are more detailed and more frequent than once per month, one of us decided to implement a honeypot information sharing system (HISS) as a web application written in Django. It can be accessed at the following URL:

https://pandora.nlcv.bas.bg/hiss

The application requires login credentials; a demo account is available by using the username guest and the password *nlcv_guest*. The demo account has many restrictions, on order to prevent it from being abused for overloading our server. In particular, it is limited to showing information only for Bulgaria, only from the current year, the user is not allowed to change the password, the shown IP addresses are anonymized, and the attack times are not displayed (only the dates are). The latter is done in order to prevent an attacker from figuring out the IP address of our honeypot by correlating the attack times from our honeypot with a log of the times of his attacks against various IP addresses.

Once the user logs in, they can perform various searches – by country, by city, by organization, by date range and so on. Regular (non-demo) accounts can select the country for which they want the data displayed.

Regular accounts also have an API key, which can be used to query HISS and to obtain the relevant data automatically, using a REST API.

Unfortunately, since we currently to not have a load balancer, the web application probably will not be able to handle too many users logging in simul-taneously and issuing too many queries. In fact, it is rather slow even when a single user is using it. We are currently working on optimizing it. Eventually, and if we get the necessary funding, we might be able to acquire a load balancer.

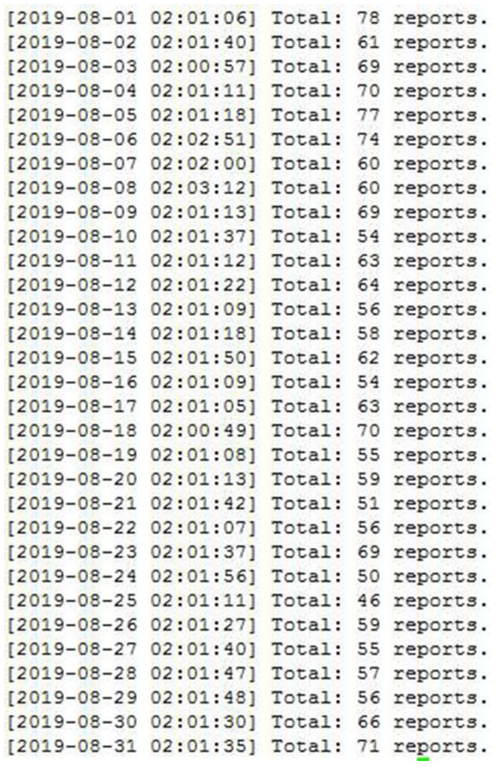
HISS is open source and can be found on GitLab.[11]

Monthly Abuse Reporting

As shown on Fig. 6, DigitalOcean is routinely the largest offender – i.e., attacks are coming the most often from IP addresses that belong to this company. In order to help them reduce the abuse of their services, we have implemented automatic abuse reporting.

Each night, at 02:00, a cron job runs a bash script, which analyzes the JSON log file for the previous day for any attacks coming from IP addresses that belong to DigitalOcean. For every distinct such address, an abuse report is sent by e-mail to the address that the company has designated for receiving of abuse reports. The e-mail is formatted in a special way – in the so-called X-ARF format.[12] It contains detailed information about the offending IP address, the time of the attack, and a log of the first attack as evidence.

As can be seen in Fig. 7, more than 60 such abuse reports are sent on average every day. The offending IP address is usually (but not always) taken down within 24 hours. Despite this, DigitalOcean remains the top source of attacking IPs, by far.



```
[2019-08-01 02:01:06] Total: 78 reports.
[2019-08-02 02:01:40] Total: 61 reports.
[2019-08-03 02:00:57] Total: 69 reports.
[2019-08-04 02:01:11] Total: 70 reports.
[2019-08-05 02:01:18] Total: 77 reports.
[2019-08-06 02:02:51] Total: 74 reports.
[2019-08-07 02:02:00] Total: 60 reports.
[2019-08-08 02:03:12] Total: 60 reports.
[2019-08-09 02:01:13] Total: 69 reports.
[2019-08-10 02:01:37] Total: 54 reports.
[2019-08-11 02:01:12] Total: 63 reports.
[2019-08-12 02:01:22] Total: 64 reports.
[2019-08-13 02:01:09] Total: 56 reports.
[2019-08-14 02:01:18] Total: 58 reports.
[2019-08-15 02:01:50] Total: 62 reports.
[2019-08-16 02:01:09] Total: 54 reports.
[2019-08-17 02:01:05] Total: 63 reports.
[2019-08-18 02:00:49] Total: 70 reports.
[2019-08-19 02:01:08] Total: 55 reports.
[2019-08-20 02:01:13] Total: 59 reports.
[2019-08-21 02:01:42] Total: 51 reports.
[2019-08-22 02:01:07] Total: 56 reports.
[2019-08-23 02:01:37] Total: 69 reports.
[2019-08-24 02:01:56] Total: 50 reports.
[2019-08-25 02:01:11] Total: 46 reports.
[2019-08-26 02:01:27] Total: 59 reports.
[2019-08-27 02:01:40] Total: 55 reports.
[2019-08-28 02:01:47] Total: 57 reports.
[2019-08-29 02:01:48] Total: 56 reports.
[2019-08-30 02:01:30] Total: 66 reports.
[2019-08-31 02:01:35] Total: 71 reports.
```

Figure 7: A month's worth of abuse reports sent to DigitalOcean.

Some Statistics

In this section we shall present some statistics of the data, collected by our honeypots over various time periods.

Cowrie has been running for nearly three years and we have collected a wealth of data from it. A summary of the data collected for the whole year 2018 is presented in Table 1.

Table 1. Summary of the data collected for the 2018.

Login attempts	**27,984,024**
File uploads	18,302,863
Unique IPs	76,815
Unique files	12,988
Unique URLs	40,130

The honeypot was attacked nearly once every second. One and the same file was uploaded averagely 1409 times. This is not only because one and the same bot attacks from many different sources but also because the honeypot never actually executes anything uploaded by the attacker. Mirai-like bots display a string (different for the different variants) when they are started. The uploader starts the bot and waits for this string to appear. Since, when the "victim" is a Cowrie honeypot, this never happens, the uploader assumes that some error has occurred during the upload and proceeds to re-upload (and restart) the bot hundreds of times.

Next, Table 2 shows the top 10 countries from which most of the attacks originate.

Table 2. Top 10 countries from which most of the attacks originate.

Country	Attacks
United States	9,859,501
Russia	3,000,089
The Netherlands	2,890,711
Italy	2,184,618
Germany	1,677,107
United Kingdom	1,567,544
Ireland	953,838
France	828,919
Romania	549,091
Canada	523,574

The USA is by far the most aggressive "offender" with more than three times the number of attacks than the next "contender", Russia. This is because of the abundance of cloud hosting services (like DigitalOcean) in the USA. Virtual machines created for free in the clouds of these hosting services are used to host the uploaders of Mirai-like botnets, each resulting in thousands of attacks against the honeypot. Please note that since our honeypot does not distinguish between Mirai bots (which come from infected IoT devices and only check the victim for vulnerabilities without infecting it) and Mirai uploaders (which receive reports from the bots and proceed to infect the vulnerable target), we do not know whether the infected IoT devices show a similar pattern and prevalence. Still, the conjecture that there are more IoT devices (including vulnerable ones) in the USA than, say, in Russia seems a plausible one to us.

Finally, Table 3 shows the top 10 organizations that own the most actively attacking us IP addresses.

Table 3. Top 10 organizations that own the most actively attacking us IP addresses.

Organization	Attacks
DigitalOcean, LLC	10,159,569
FranTech Solutions	2,385,520
Aruba S.p.A.	2,207,162
Global Layer B.V.	1,498,374
Melbikomas UAB	824,015
LLC Baxet	690,718
3W Infra B.V.	603,213
Hostio Solutions B.V.	589,904
Online S.a.s.	566,011
Hostwinds LLC.	492,757

Every single one of them, without exception, is a cloud hosting provider. They are all suffering from abuse of their services by the botnet herders but Digital-Ocean is by far the worst hit. We see nearly five times as many attacks coming from their IP addresses than from the IP addresses of the next con-tender. The company clearly needs to do something to reduce the level of abuse of their services.

Future Work

In the future, we plan on expanding our set of honeypots – both in numbers (although this is highly dependent on whether we succeed in obtaining funding) and in kinds.

For instance, we have already set up SMB and ADB honeypots and are collecting data from them. We are actively working on setting up a Remote Desktop honeypot listening on port 3389, since there is strong evidence that it is a frequent avenue of attack, with the attackers trying to infect the victim machines with ransomware. Setting up such a honeypot is not an easy thing to do. On the one hand, we do not want to expose a real (or virtual) machine to the Internet via this protocol. Even if we take care to firewall the outgoing connections and even if we reimage the machine frequently, there is no guar-antee that the attackers who manage to log in will not use it to attack other machines on the Internet – and this is a possibility we are not willing to accept. On the other hand, if we only emulate successful login and play back a pre-recorded session to the attacker (this is the approach used in T-Pot), a human attacker will understand instantly that this is a honeypot and not a real machine.

We also plan to set up honeypots for other communication protocols, like FTP, memcache, MQTT, SIP, TFTP, UPNP (all of these are available in Dionaea, we just have to turn them on), as well as some protocols which we currently cannot attach a honeypot to, because the actual machine running the current honeypots needs to communicate through the corresponding ports. These include HTTP, MySQL and others. We also plan on setting up a network printer honeypot, a router honeypot for catching exploit attempts, and others.

If we manage to get the necessary funding, we plan on setting up a network of honeypots on various machines in the cloud, preferably residing physically in different parts of the world, in order to get a better picture of the various kinds of attacks that are ongoing on the Internet.

Finally, we intend to cooperate with the security, anti-virus, and threat intelligence companies that are active in this field, in order to improve our work and results.

Conclusion

Our experience shows that finding the right kind of honeypot for one's needs is far from a trivial task. In every single case we had to make significant modifications to the open source tools that are publicly available and the development effort often took several months.

Observing the results from running this honeypot over a considerable length of time shows without a doubt that of the covered communication protocols, Telnet is extensively used for attacks (usually by Mirai variants).

Acknowledgment

The preparation of this paper is supported by the National Scientific Program "Information and Communication Technologies for a Single Digital Market in Science, Education and Security (ICT in SES)," financed by the Ministry of Education and Science of the Republic of Bulgaria.

References

1 "MMD-0056-2016 - Linux/Mirai, how an old ELF malcode is recycled," *Malware Must Die*, August 31, 2016, http://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html.

2 Brian Krebs, "KrebsOnSecurity Hit With Record DDoS," *KrebsOnSecurity*, September 21, 2016, https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/.

3 Manos Antonakakis, et al., "Understanding the Mirai Botnet," *26th USENIX Security Symposium* (2017): 1093-1110, https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf.

4 GitHub - Cymmetria/MTPot, https://github.com/Cymmetria/MTPot.

5 T-Pot 16.10 - Multi-Honeypot Platform Redefined, Deutsche Telekom AG Honeypot Project, 31 Oct 2016, https://dtag-dev-sec.github.io/mediator/feature/2016/10/31/t-pot-16.10.html.

6 GitHub - Cowrie SSH/Telnet Honeypot, https://github.com/micheloosterhof/cowrie.

[7] "GeoLite2 Free Downloadable Databases," *MaxMind*, January 2, 2019, http://geolite.maxmind.com/download/geoip/database/.

[8] GitHub - Ikoniaris/Kippo-Graph, https://github.com/ikoniaris/kippo-graph.

[9] GitHub - Bontchev/Cowrie, https://github.com/bontchev/cowrie.

[10] Grafana - The open platform for beautiful analytics and monitoring, https://grafana.com/.

[11] GitLab - Honeypot information sharing site, https://gitlab.com/venetay/honeypot-information-sharing-site.

[12] GitHub - X-ARF Specification, https://github.com/xarf/xarf-specification.

About the Authors

Dr. Vesselin **Bontchev** has a M.Sc. degree in Computer Science from the Technical University of Sofia and a Ph.D. in Methodology of Computer Anti-Virus Research from the University of Hamburg. He has worked as a research fellow at the Technical University of Sofia, the Institute of Industrial Cybernetics and Robotics at the Bulgarian Academy of Sciences, as a director of the Laboratory of Computer Virology at the Bulgarian Academy of Sciences, and as a computer anti-virus researcher at FRISK Software International in Reykjavik. He is currently an Assistant Professor at the National Laboratory of Computer Virology at the Bulgarian Academy of Sciences.

Veneta **Yosifova** holds a Master's degree in Informatics from the University of Sofia "St. Kliment Ohridski" and Master's degree in International Relations from the Law Faculty of the same University. Currently she is a Ph.D. student in Machine Learning and Computer Security at the Technical University of Sofia. She has working experience as a programmer in various software companies and as security researcher at the National Laboratory of Computer Virology at the Bulgarian Academy of Sciences.