Creation of a Collaborative Model of Honeypots

Asif Al Ferdous Khan

Resumen—Hoy en día todo gira entorno a Internet y la ciberseguridad es un asunto muy importante tanto para la economía como nuestras vidas personales. La constante evolución de los métodos de ataque empleados para vulnerar la seguridad de redes y máquinas privadas provoca la necesidad de nuevos modelos de recopilación de datos y estudios de los mismos. Los Honeypots son herramientas empleadas para esta tarea por su capacidad de simulación de servicios y entornos para engañar a los atacantes y recopilar datos sobre su modus operandi. Es por ello que en este artículo se propone un modelo de red distribuida de honeypots para la recopilación de inteligencia sobre ciberataques y el desarrollo de una plataforma para centralizar los datos generados y gestionar dicha red.

Palabras clave—Honeypot, Ciberseguridad, Inteligencia de Amenazas, Modern Honey Network, Dionaea, Kippo

Abstract—Nowadays everything is connected to the Internet and cybersecurity has become a really important topic for both the economy and our personal lives. The constant evolution of the methods employed to undermine the security of private networks and machines has caused the necessity to bring new models to collect data in order to study them. Honeypots are instruments employed for this task because of their capacity to simulate services and environments to lure attackers and to collect data about their modus operandi. That is why in this paper we propose a model of a distributed network of honeypots dedicated to the collection of intelligence about cyberattacks and the development of a platform to centralize the acquired data and to manage said network.

Index Terms—Honeypot, Cybersecurity, Threat Intelligence, Modern Honey Network, Dionaea, Kippo

1 Introduction

GIVEN the rapid growth of the Internet over the last decades and its increasing presence in the economy and our daily lives, security against cyberthreats is vital.

Cybercrime is an ongoing and critical issue to face at the present. It is mainly dedicated to information and data theft, particularly to the ones with potential market value such as: login credentials, passwords, PIN codes, banking information or credit cards.

The information is generally stolen by misleading the victim through phising[1] methods or malware infection, both on client computers and servers. Many of these infected computers tend to remain under the control of the attackers forming part of a botnet[2]. The botnet receives its instruction from one or more criminal's servers, where the stolen information is also sent. These servers are known as C&C (Command-and-Control). [1]

But nowadays it is becoming standard practice to use legit servers from third parties, which have been compromised by security breaches and infected in order to avoid detection.

The infection of those servers is done by scanning the

- *Contact E-mail: asif.alferdous@e-campus.uab.cat*
- *Specialization: Information Technologies*
- *Tutored by:*
  Ramon Vicens (Blueliv)
  Guillermo Navarro Arribas (DEIC)
- *Course 2014/15*

[1] Phishing refers to the acquisition of sensitive information for malicious reasons by supplanting the identity or masquerading as a trustworthy entity. It is often done using fake web pages or e-mails.
[2] A botnet is a distributed network of infected computers known as bots, remotely controlled by a malicious entity.

Internet looking for open ports of services with vulnerabilities, to later exploit them to gain unauthorized access.

There is an incipient necessity to gather intelligence about the methodology used for these breaches in order to detect and prevent them. The study of the modus operandi of the attackers is a key factor to improve the security of the information and to guarantee the integrity of the systems.

One of the tools used for this purpose are honeypots, which simulate services and environments to lure attackers to believe they are penetrating a real system. The interaction between the attacker and the honeypot is logged for posterior analysis and study.

A large scale networked implementation of these tools could provide an important amount of data about the methodology used by the attackers, although the main limitation of honeypot technology is the high level of maintenance it requires.

Moreover, all the gathered data remains locally in the machine that hosts the honeypot software, with no inherent scheme of compilation of the data from the host and neither a correlation of the data between the different honeypots in the network.

1.1 Objectives

This work is done in cooperation with Blueliv, a targeted cyberthreat intelligence provider for enterprises, aiming to provide new sources of threat intelligence and counter-intelligence data.

The main objective of this project is to develop a honeypot management platform and deploy a network of honeypots in different geographical locations with a centralized server to process and store all the gathered data. Moreover, this platform will seek to streamline the process of deployment of new honeypots and to streamline the later maintenance. The Fig. 1 shows a model of the platform's expected functioning.

Other objectives, in a personal level, are to expand our knowledge about information security, since it is a field that interests us, and also to acquire more specific knowledge about honeypot technology.

The specific goals to reach for the fulfilment of the project are:

1. Develop a honeypot management platform for the centralized server. This platform should have, at least, the following capabilities:
    a. Ability to see data generated from a network of honeypots.
    b. Ability to deploy new honeypot sensors into the network from the platform.
    c. Improve the maintenance with the ability to launch fixes and updates from the platform.
    d. Ability to edit the configuration files of the honeypots from the platform.
    e. Implement a large scale usage of the previous capabilities (i.e. queue the deployment of multiple honeypots automatically)

2. Deploy a network of sensors with honeypot technology to acquire intelligence about the methodology employed by the attackers.

3. Implement a centralized client-server paradigm for the data generated from the sensors (clients) sending it to the server with the platform.

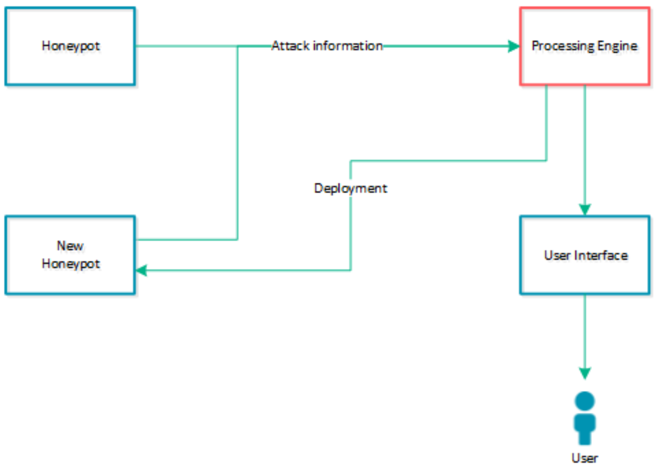


Fig. 1. Model of the honeypot management platform. Deployed honeypots send information about attacks to the processing engine, which are accessed through the user interface. The engine also deploys new honeypots.

2 Methodology

The methodology employed in this project consisted in five phases:

1. Market research
2. Analysis
3. Design
4. Development
5. Deployment

2.1 Market research

An initial market research was done in order to discover the state of the art in the field of honeypots. The goal of this phase was to search for technology that could suite the needs of this project and could serve as basis for further development.

It was stablished that in the case that no available technology suited to the needs of the project, developing own technology would have been considered.

2.2 Analysis

An analysis of the available technologies led to decide which of them accomplished better the requirements established in the objectives.

After a comparison of their features, one solution was selected as the foundation of the project and used to develop the required capabilities. The chosen solution was deployed in a small scale for further analysis and testing. To familiarize ourselves with the technology a technical analysis of its architecture and functions was conducted.

Given the case that none of the technologies had served as basis, a viability study about developing own technology from scrap would have been done during this phase.

2.3 Design

In order to optimize the development phase, an extensive planning was necessary. After the analysis of the architecture of the chosen solution, the new features to develop were stablished. From the data obtained during the analysis, both the initial and the new architecture and functions were represented into several documents: model diagrams, sequence diagrams and database diagrams; and the concrete elements to develop were established.

2.4 Development

After the planning and designing of the functionalities to introduce, they were developed over the basis technology. The testing of the new features and their overall effect on the platform was conducted as constant functional tests.

2.5 Deployment

After reaching the fulfilment of the requirements in the development, a large scale deployment was done to start the recollection of intelligence data. This was done by deploying several honeypot sensors in virtual servers located in various geographical locations; and the generated data was centralized in the developed platform.

3 State of the art

To understand the state of the art, below is a more detailed definition of what a honeypot is, followed by a brief description of the main honeypot management frameworks found during the market research.

3.1 Honeypots

Honeypots are security resources that simulate and pretend to be services of a computing system, or the whole system, with vulnerabilities or weak points in order to be probed, attacked, compromised, used or accessed in any unauthorized way. [2]

Their objective is to gather all the possible information from these attacks or unauthorized accesses in order to study the methodology used from the attackers and prevent them in the future. Since they only simulate the environment and responses of these services, no sensitive information is compromised from the attacks. [3]

In the deployment phase two honeypots were used as sensors: Dionaea and Kippo. Dionaea is a low-interaction honeypot specialized in trapping and collecting malware samples for further analysis. It is written in Python and can listen to ports and emulate protocols (such as HTTP, HTTPS, FTP, MySQL, SMB…) to interact with the malware. Kippo is another low-interaction honeypot but it emulates the SSH service. It is also implemented in Python and stores information about brute-force login attacks against the service. [2] [4]

3.2 Honeywall CDROM

The Honeywall CDROM is a project developed by The Honeynet Project. It's a bootable installation CDROM designed to simplify the process of creation of a honeynet[3] by automating the deployment of the Honeywall[4]. The purpose of the Honeywall is to capture, control, and analyze all inbound and outbound activity interacting with the honeynet. [5]

3.3 SURFcert IDS

SURFcert IDS (formerly known as SURFids) is a distributed intrusion detection system and an early warning system developed by SURFnet. It is a free and open source software that employs sensors as proxies to redirect the traffic from a monitored network, which serves as a bait, to a centralized one where the actual honeypots are located and the interaction takes place. [2]

3.4 Modern Honey Network

Modern Honey Network (MHN to abbreviate) is an open source software developed by ThreatStream. It provides deployment and events aggregation capabilities for several of the current open source honeypot software available. [6]

4 Analysis

The market research provided a view of the state of the art in honeypot technology and three honeypot management platforms were found: Honeywall CDROM, SURFcert IDS and MHN.

These were possible candidates to use as basis to reach the fulfilment of the objectives, but further analysis was required to stablish the actual compatibility. Table 1 shows a comparison of the main features.

[3] A honeynet is the network formed by two or more honeypots.
[4] Term used by The Honeynet Project to refer to the gateway of a honeynet.

TABLE 1
COMPARISON OF FEATURES

	Honeywall CDROM	MHN	SURFcert IDS
Monitoring	Local network	Remote hosts	Remote hosts
Monitoring method	Network's gateway & local sensors	Remote sensors & Central server	Remote sensors & Central network
Honeypot location	Internal network	External host	Internal host/network
Distributed	✗	✓	✓
Distribution paradigm	Not distributed	Distributed sensors send monitored data to central server	Distributed sensors act as proxies and redirect traffic to central network where it is monitored
Damage scope in case of security breach	High	Low	High
Open source	✓	✓	✓
Free	✓	✓	✓
Active maintenance	✗	✓	✓

It was easy to conclude that Honeywall CDROM had low compatibility with the objectives of this project, since its approach was focused on monitoring a local network, it had no large scale distributed solution and it also had become partially outdated with no active maintenance.

Both MHN and SURFcert IDS had better compatibility and could serve as basis for this project. They were similar in some features but they were totally different in the distribution paradigm. In one hand, MHN allocated the honeypots in the distributed sensors and sent the collected data to the main server, which acted as processing engine and database. On the other hand, SURFcert IDS used its sensors similarly to a proxy to redirect the traffic to its centralized network where the actual honeypots and the database were.

Even though SURFcert's distribution simplified notably the installation and maintenance processes, in case of a security breach the damage scope, as possible leakage/destruction of information or system's integrity loss, would be considerably higher in comparison to MHN.

Finally, MHN provides several installation scripts in bash to automate the installation of honeypots and its configuration with the platform.

After considering all these matters, MHN was selected as technology to use as basis, especially because of its approach of the distribution paradigm and its lower damage scope.

4.1 Modern Honey Network's architecture

From MHN there are two key functionalities required to mention: event aggregation capabilities and honeypot installation and configuration scripts. Even though, both of them are important additions for the project, they present some lack of features as explained next. The development phase will partially consist on improving these features among including others.

The installation and configurations scripts are written in bash language and include support for several types of honeypots, such as: Dionaea, Kippo, Wordpot, Snort and many others. But these scripts require the external access to the remote host to manually run the script.

The data aggregation capabilities include the centralization of some data generated by the network of sensors. These data includes source IP, attacked port, associated protocol and geolocation. Some honeypots have additional support on the gathered information, such as Kippo's most used users and passwords. It also provides some data analysis as daily attack counts, a real-time map of attacks and some filtering options of the stored attacks. But there is a lack of detailed information about the attack, such as the data of the logs. To understand how the data aggregation works, below is an explanation of the architecture and its representation can be seen in Fig. 2.

MHN's architecture is structured in six layers:

- Honeypot sensors
- Hpfeeds
- Mnemosyne
- Databases
- Rest API
- Web application

In the first layer there is the network of honeypots deployed in remote hosts to act as sensors. When these sensors are attacked, some data about the attack is sent to the MHN's server. This is done using the next layer.

To stablish a communication channel through which the attack event data is sent, the Hpfeeds protocol is used. Hpfeeds is a lightweight authenticated publish-subscribe protocol largely used with honeypot technology. [7]

Once the data has been received into the platform, Mnemosyne is the engine that processes it. It is responsible of the next three layers: the normalization and adequacy of the heterogeneous data generated from different types of honeypots, to provide persistence for hpfeeds using a mongoDB database and to expose the normalized data through a RESTful API. [8]

Both Hpfeeds and Mnemosyne are implemented in Python; though, currently there are implementations in other languages.

Finally, the last layer consists in a web application that serves as user interface to consult the data. It is implemented with Flask, which is a microframkework that combines a frontend implemented with HTML templates and a backend in Python which does the main work and renders the templates. [9]

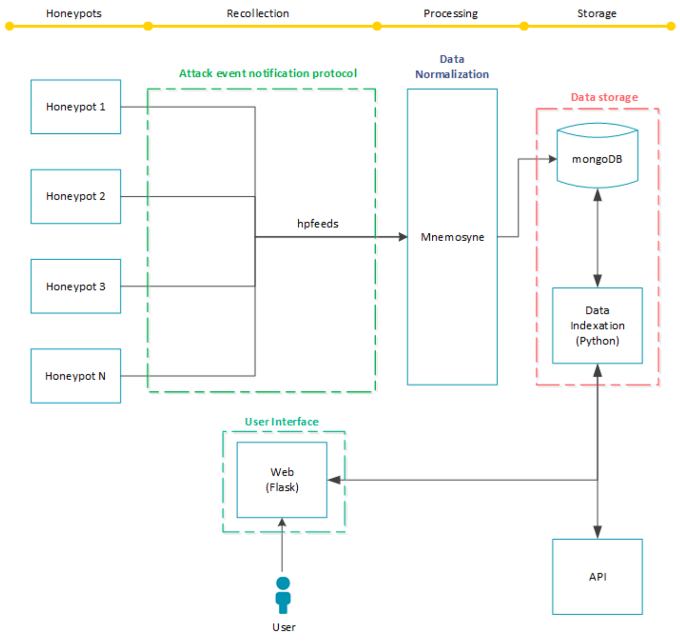


Fig. 2. Architecture of Modern Honey Network. The honeypot layer generates data, which is recollected, processed and stored as explained previously. Finally, the access to the stored data is done through the user interface or the RESTful API.

5 Design

After analyzing the architecture of MHN and its capabilities, the remaining features to develop and implement in order to achieve the objectives were stablished:

1. Ability to deploy new honeypot sensors from the platform using the installation scripts. This would be done through:
   a. Implementation of SSH capabilities into the backend of the platform.
   b. Modification of the frontend.
   c. Implementation of task queuing (background asynchronous execution) to avoid timeouts in the web interface.
   d. Enhance task queuing capabilities to implement multiple deployment instructions.

2. Maintenance capabilities for the deployed honeypots through:
   a. Persistence of the SSH credentials to maintain the access to the remote hosts.
   b. Development and implementation of update scripts.
   c. Modification of the frontend.

3. Ability to edit configuration files of the deployed honeypots from the platform.

The previous features to develop were the most critical to the fulfillment of the project, but next are some less prioritized features stablished to develop in case that the time frame allowed it:

4. Improve the data recollection including logs and other files from the honeypots.

5. Improve data analysis through:
   a. Implementation of a dashboard view.
   b. Implementation of a detailed view of an attack.

5.1 Views

As stated earlier, the web interface is built on Flask microframework. Flask is implemented in Python and uses Jinja2[5] templates to create the HTML views for the web application. [10]

The basic implementation of Flask is simple. Suppose a view that renders the template for the deployment page. This view, when called, will receive some data through a POST method, will execute some determined Python instructions using the received data, use a variable to indicate the success or failure of the operation and finally render the template. The implementation of the previous view, omitting library imports and instances' creation, would be as stated next:

```
@route('/deploy/', methods=['GET', 'POST'])
def deploy():
     # Deployment operations would go here with
     # a variable 'res' indicating the result
     # of the execution.
     return render_template('deploy.html',
                                    result=res)
```

The `@route` decorator declares the associated URL that will trigger the function. It also declares which methods are allowed, by default GET if nothing is declared. In this example the POST method is necessary to receive the data used for the later execution and the GET method to deliver template with the result of the execution.

[5] Jinja2 is a template engine for Python. It is a text-based template language that allows calling functions on objects and sandboxed execution.

Once called the URL, the function is executed and the `res` variable stores its result. The function returns the rendered template including the result.

The Jinja2 template would be:

```
<html>
     <head>
            <title>Deployment result</title>
     </head>
     <body>
            The result is: {{ result }}
     </body>
</html>
```

During the rendering, the `{{ result }}` variable is substituted with the value given by `res` in the previous code.

Below is the list of views and templates developed to introduce the new features:

1. Deployment layer:
   a. Deployment page (deploy.html) – A form to introduce the deployment information (honeypot, IP, auth. credentials…)
   b. Execution page (ssh.html) – Uses the given information to authenticate the access to the remote host and launches the deployment using SSH.
   c. List of deployments (deploy_list.html) – A historic of the launched deployments and access to the execution details.
   d. Details of execution (status.html) – To check the execution output and errors.
2. Maintenance layer
   a. Maintenance page (maintain.html) – To select the honeypot and the maintenance task.
   b. Execution page (update.html) – To launch an update to a previously deployed sensor.
3. Configuration edit layer
   a. Edit configuration (config.html) – To edit the configuration file of a previously deployed honeypot.

5.2 Database

MHN relays on two databases:

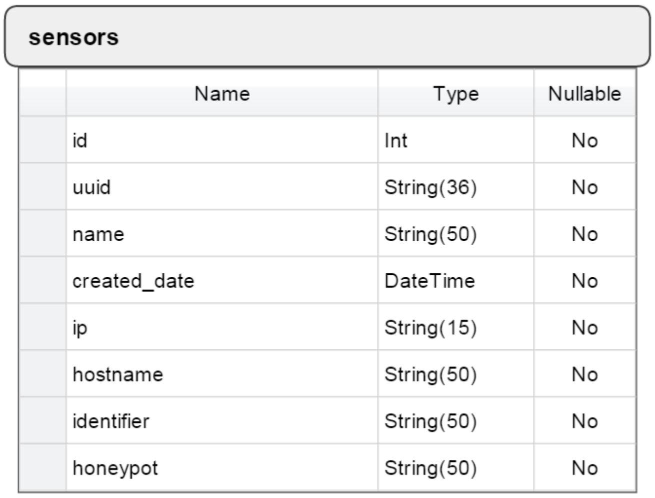
- A non-relational mongoDB database employed by Mnemosyne to maintain all the information related to hpfeeds and attack events.
- A relational SQLite database which maintains all the information regarding the platform, as for example the users and the sensors' data.

A diagram of the mongoDB database is available in the section A1 of the annex. For the introduced functionalities, it wasn't necessary to modify this database.

To integrate the new functionalities it was necessary to modify the SQLite database, concretely the 'sensors' table.

This table was modified to introduce the SSH authentication credentials of the remote host: a username and a password or a private key file. Paths to locate the external files regarding the configurations of the honeypots were also introduced. These paths include both the local location and the remote host's location.

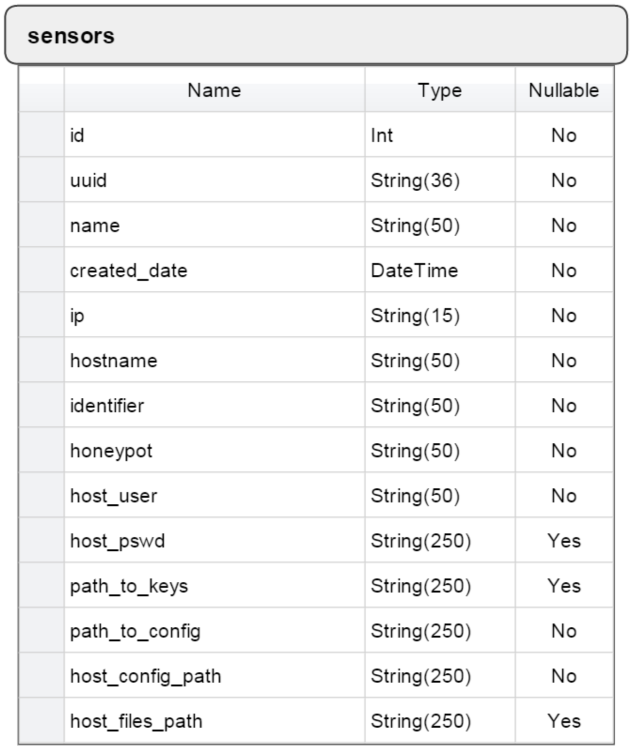
Fig. 2 shows the original state of this table, while Fig. 3 shows it after the modifications done.



sensors

Name	Type	Nullable
id	Int	No
uuid	String(36)	No
name	String(50)	No
created_date	DateTime	No
ip	String(15)	No
hostname	String(50)	No
identifier	String(50)	No
honeypot	String(50)	No

Fig. 3. Table for the sensors' persistence, which was modified in order to persist the new functionalities' data.



sensors

Name	Type	Nullable
id	Int	No
uuid	String(36)	No
name	String(50)	No
created_date	DateTime	No
ip	String(15)	No
hostname	String(50)	No
identifier	String(50)	No
honeypot	String(50)	No
host_user	String(50)	No
host_pswd	String(250)	Yes
path_to_keys	String(250)	Yes
path_to_config	String(250)	No
host_config_path	String(250)	No
host_files_path	String(250)	Yes

Fig. 4. Sensors' table modified to persist the SSH authentication credentials for the remote host and the honeypot's configuration files.

6 DEVELOPMENT

Below is the summary of the work done during the development phase.

6.1 New honeypot deployment

Starting the development phase, the first step was to integrate SSH capabilities to the backend in order to deploy a new sensor in a remote host given the authentication credentials. This was done using Paramiko, a Python library for the SSHv2 protocol implementation. [11] A simplified version of Paramiko's usage is shown below:

```
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAdd
                                Policy())
ssh.connect(ip, username=user, password=pw)
stdin,stdout,stderr = ssh.exec_command(command)
ssh.close()
```

First of all, an instance of the SSH client is created and its policy regarding unknown hosts is declared as to accept all. Next, a connection is stablished using the instance previously created passing the authentication credentials. The execution is launched and its input, output and errors are saved in stdin, stdout and stderr respectively. Finally, the connection is closed.

Also, as explained in the section 5.1, some new views with their respective templates were necessary. The first view renders a page with a form to input the remote host's information and the honeypot to deploy. This information is sent to the second view through POST method, which stablishes the SSH connection with the given information.

The first version of this implementation used to stablish the SSH connection, executed the command and waited for the result before rendering the page to show the result. This provoked some timeout issues with the web server, which also made the SSH execution to fail and, so, the honeypot wouldn't be successfully installed.

In order to solve this issue, a different approach was applied. The execution of the SSH commands is handled by Celery, an external asynchronous task queuing service. A function independent to the view stablishes the SSH connection and executes the commands. This function is declared as a Celery task using the following decorator above it:

```
@celery.task
def ssh_exec(ip, user, pw, pkey, command,
             honeypot):
```

Before adding the task to the queue, the view tests the given authentication credentials in order to validate them. If the authentication is successful, the task is queued and the view renders the page indicating that the deployment has been launched. In case that the authentication fails or any other error happens, the page is rendered informing about the error and the task is not queued.

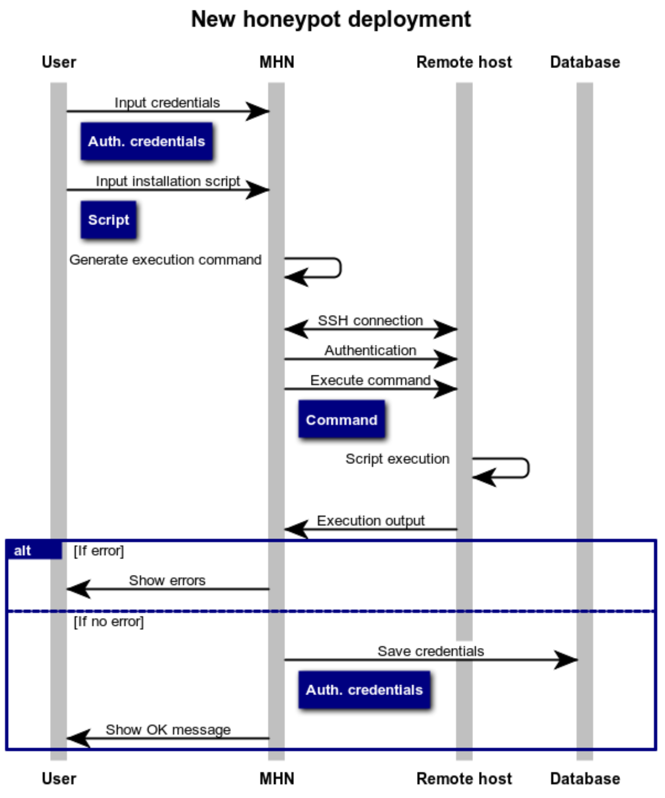


Fig. 5. Sequence diagram of the process of a new honeypot deployment.

To check the status of the queued tasks a page has been added. In this page, all the tasks are listed, and if finished, the user can access to the details of the execution. All the information regarding the tasks is saved as JSON data files.

Finally, a page to show the details and result of a determined execution was added.

Fig. 5 shows a summary of the process followed for a new honeypot deployment.

6.2 Honeypot update

To implement the ability to launch updates from the platform, as represented in Fig. 6, the first task was to adapt the SQLite database to persist the new information as explained in the section 5.2.

To update the honeypots, new scripts in bash language were created. This scripts work in the same way as the original scripts of MHN but they execute the instructions required to update a previously deployed honeypot as provided by their creators. Currently only scripts for Dionaea and Kippo have been developed.

Finally, the views and pages indicated in the section 5.1 were created to add this feature to the frontend.

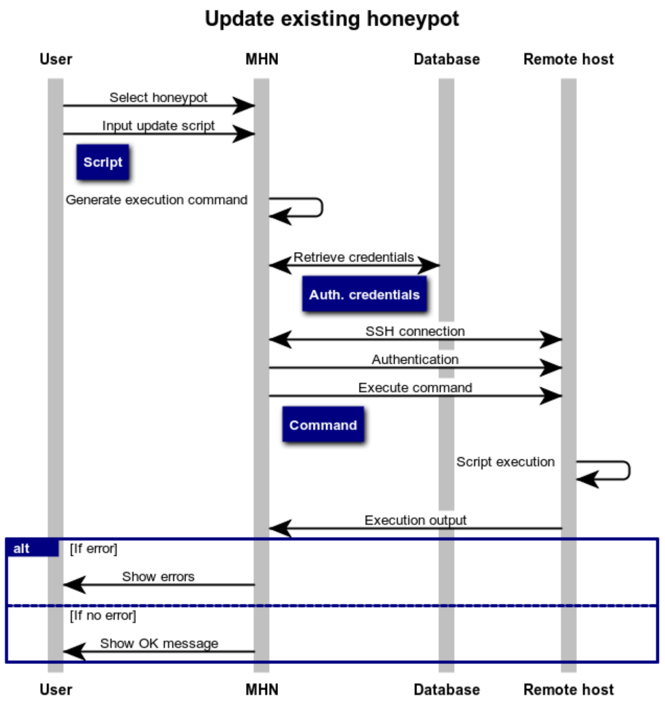


Fig. 6. Sequence diagram of the process of updating of a previously deployed honeypot.

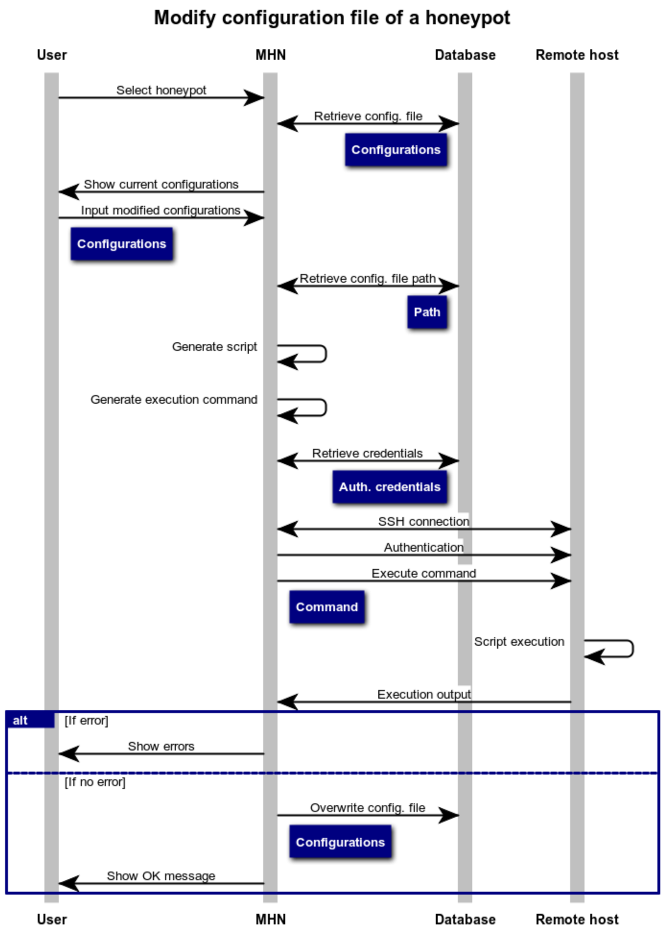


Fig. 7. Sequence diagram representing the process followed to edit a remote honeypot's configuration file.

6.3 Edit honeypot's configurations

Using the resources of the features previously developed, it was easy to implement the editing of the remote configuration of a honeypot as shown in Fig. 7. Concretely through the database modifications explained in the section 5.2 and the some of the views for the maintenance capabilities.

There is a local copy of each honeypot's configuration file in the platform's server which the user can edit. When the editing is done, these configurations are sent through the usage of SSH and SCP to the remote host where the honeypot is located and overwrites its configuration file.

Once this process is done successfully the local configuration file also gets overwritten by new one to reflect these changes for future usage and consistency.

7 Deployment

The final phase of this project was the deployment of a distributed network of sensors with the platform centralizing the information. In order to do so, eight virtual machines located in various geographical points were created to host the honeypot sensors. The honeypots deployed were Kippo and Dionaea. Initially, there was the same amount of sensors of both of them but since Kippo honeypots received more interaction and provided more data, the deployment was focused on them.

The virtual machines employed, using Ubuntu 14.04 distribution with 32-bit architecture, were hosted in DigitalOcean. The reason behind using this provider resides on the speed and simplicity of creating a virtual machine they offer, which settles to the rapid-and-easy deployment philosophy that this project tries to stablish. And the reason behind using Ubuntu 14.04 x32 is the stability and compatibility that offers, since most honeypots are tested with this configuration. Table 2 shows the basic information about the deployed sensors.

The central platform was also deployed in a DigitalOcean virtual machine located in Amsterdam.

TABLE 2
Deployed Honeypots

Host Name	Honeypot	Location
Node1	Dionaea	New York
Node2	Kippo	New York
Node3	Kippo	San Francisco
Node4	Dionaea	Amsterdam
Node5	Kippo	Amsterdam
Node6	Kippo	London
Node7	Kippo	Frankfurt
Node8	Kippo	Singapore
Node9	Kippo	Singapore
Node10	Dionaea	Singapore

8 RESULTS

After ten days of deployment, the sensors already had gathered a considerable amount of data produced by numerous port scans and attacks as shown in the Table 3.

TABLE 3
ATTACK EVENTS

Name-Honeypot (Location)	Scans/Attacks
Node1-Dionaea (NY)	4801
Node2-Kippo (NY)	2478
Node3-Kippo (SF)	3266
Node4-Dionaea (AMS)	2929
Node5-Kippo (AMS)	991
Node6-Kippo (LON)	27036
Node7-Kippo (FRA)	5097
Node8-Kippo1 (SNG)	15482
Node9-Kippo2 (SNG)	1949
Node10-Dionaea (SGP)	1744

The average amount of the total daily interaction received is between 7500-9000.

Note that many of these interactions are not specifically attack events, many of them are port scans which usually are reconnaissance actions for future attacks but not attacks by themselves.

The top attackers' IP were mainly from China, Russia and Japan, but many attacks were also received from IP addresses located in Poland, Italy, United Kingdom, France and United States.

The most attacked port is the 22, associated to the SSH service. This is partly due to the fact that Kippo works exclusively with this service while Dionaea works with various, such as: HTTP, HTTPS, FTP and MySQL among others. Still, SSH protocols' interaction numbers are significantly higher to the other services' and, so, are worthy of a more detailed review.

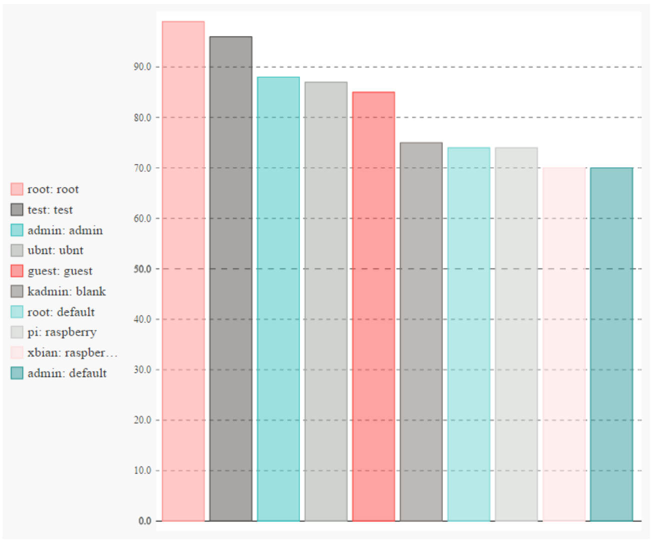


Fig. 8. Most used combinations of user-password in the attacks received to the Kippo honeypots.

Kippo simulates a fake SSH service with ability to interact with the attacker. The default user-password is 'root:123456', which attackers usually try to guess using brute-force or dictionary attacks. If they manage to successfully log in, Kippo tries to satisfy their demands in a controlled environment simulating some of the most usual commands (for instance: wget, ping, apt, ssh, adduser…) while logging all the activity and saving the downloaded files for further analysis.

The Fig. 8 shows a diagram with the most tried user and password combinations by the attackers in order to break in.

Once the intruders have successfully logged in, three patterns of behavior are observed:

1. No action is done right after the break in. This could be due to: the attack being the reconnaissance and data storage phase prior to a mass automated attack or the attacker noticing the honeypot environment.
2. Manually check information about the simulated host, such as the active processes, active users and other general information about the machine. Some of these cases lead to behavior 3 if the honeypot isn't detected. Others detect the honeypot due to the lack of activity and no further activity is done.
3. Download executable files from a remote host and leave them executing in the background. This happens when the intrusion is automated or the attacker doesn't notice the honeypot environment.

The most relevant behavior is the last one, since it provides intelligence about the attackers' methodology and their course of actions once the system has been penetrated.

The objective of these files is commonly to keep monitoring the machine and exploit it for further malicious actions, such as opening a backdoor to connect to a C&C server and receive orders. These orders are principally to make DDoS attacks to determined IP addresses. In some cases, the backdoor is used to sell the access (the authentication credentials) in the black market in order to be used as part of a botnet or employed for further distribution of malware.

9 CONCLUSIONS

Reached the deadline of the project, the main objectives stablished have been achieved. Even though the initial planning suffered a few setbacks and had to be readjusted, it was correctly followed. These readjustments were done in order to improve the employed methodology, such as the introduction of a design phase to optimize the development.

Currently, a functioning centralized platform with intelligence data gathering capabilities by means of a distributed network of honeypots has been deployed. The platform is able to not only centralize the acquired data, but also to deploy new honeypot sensors and configure them, reducing considerably the complexity of this process. It also simplifies the later maintenance, allowing launching updates and the editing of the configuration files from the platform.

The data gathered so far by the network of sensors already provided valuable information about behaviors, trends of attacks and, in general, intelligence about cyber-threats.

From a personal point of view, this project has served to learn about honeypot technology and to expand the previous knowledge about information security. It also helped to take contact with many technologies, such as Flask, mongoDB and Celery; to experience all the steps of a real development project; to understand the importance of an extensive design stage and how positively it affects to the development; and to be able to face unexpected problems and find suitable solutions.

The work done during this project could serve as basis for a corporative project. However, there are still many features and functionalities that could be developed and implemented.

10 Future work

The scheduled time has been the principal limitation on the features that could be developed. There's plenty of room for many other functionalities and improvements that could be implemented. Medium and low priority objectives, such as improving the data recollection, implementing a dashboard view or a more detailed view of an attack event, have not been achieved in the available time frame. These leave a path to follow in future projects.

There is a considerable amount of improvements that could be introduced in the platform, such as compatibility with more types of honeypots and its consequent maintenance requirements.

Regarding the sensors, there are also improvements that could be introduced. For example, Kippo could be improved to offer better responses to the attackers when prompted for information about the machine in order to resemble a more realistic environment. This would help to deceive intruders and convince them to pursue their activity.

Finally, a further and more exhaustive analysis of the methodology of the attackers could be done using the acquired data, such as the downloaded files, since there was not enough time to include it to the project.

Acknowledgment

I would like to thank both, Ramon Vicens from Blueliv and Guillermo Navarro from UAB, for their mentoring and help; to Ramon especially for making possible this project and giving me the opportunity to work in it.

I would also like to thank Victor Acin, Ramsés Pascual and Vicent Gil Esteve for their help, as well as to many other co-workers from Blueliv.

Finally, thanks to Celia Rupérez for listening always and for her unconditional support.

This work was supported by Blueliv.

The basis technology used in this project was originally developed by ThreatStream.

References

[1] "Botnets," Shadowserver Foundation, [Online]. Available: https://www.shadowserver.org/wiki/pmwiki.php/Information/Botnets. [Accessed 14 June 2015].

[2] T. Grudziecki, P. Jacewicz, Ł. Juszczyk, P. Kijewski and P. Pawliński, "Proactive Detection of Security Incidents II – Honeypots," 2012.

[3] A. Sardana and R. Joshi, Honeypots - A New Paradigm to Information Security, Science Publishers, 2011, pp. 27, 63-89.

[4] A. Harper, S. Harris, J. Ness, C. Eagle, G. Lenkey and T. Williams, Gray Hat Hacking - The Ethical Hacker's Handbook, Third ed., McGraw-Hill, 2011, p. 639.

[5] Honeynet Project & Research Alliance, "Know Your Enemy: Honeywall CDROM Roo," 17 August 2005.

[6] ThreatStream, "Modern Honey Network," [Online]. Available: http://threatstream.github.io/mhn/. [Accessed 12 April 2015].

[7] M. Schloesser, "Hpfeeds," [Online]. Available: https://github.com/rep/hpfeeds. [Accessed 31 May 2015].

[8] J. Vestergaard, "Mnemosyne," [Online]. Available: https://github.com/johnnykv/mnemosyne. [Accessed 19 June 2015].

[9] A. Ronacher, "Flask "A Python Microframework"," [Online]. Available: http://flask.pocoo.org/. [Accessed 30 May 2015].

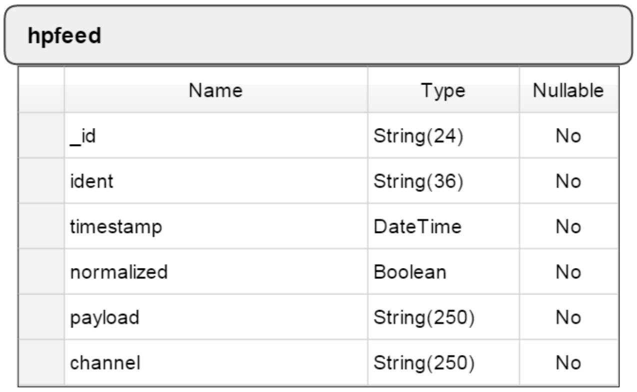
[10] A. Ronacher, "Jinja," [Online]. Available: http://jinja.pocoo.org/. [Accessed 19 June 2015].

[11] J. Forcier, "Paramiko," [Online]. Available: http://www.paramiko.org/. [Accessed 24 May 2015].

Annex

A1. MONGODB DATABASE DIAGRAM

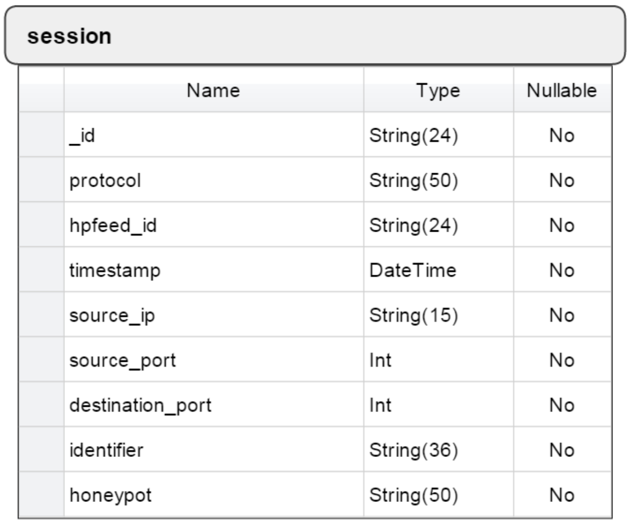
Fig. 1-4 show the main collections of the non-relational mongoDB database as mentioned in section 5.2. This database is managed by Mnemosyne to save all the incoming data from Hpfeeds as explained in section 4.1.



hpfeed

Name	Type	Nullable
_id	String(24)	No
ident	String(36)	No
timestamp	DateTime	No
normalized	Boolean	No
payload	String(250)	No
channel	String(250)	No

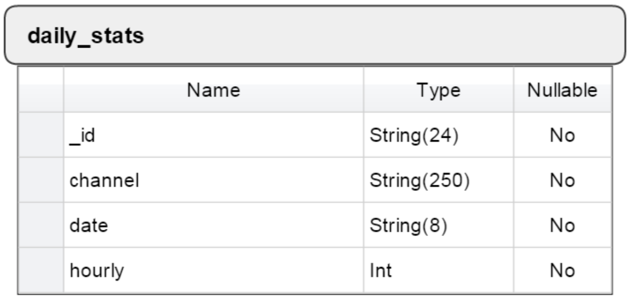
Fig. 1. The ‘hpfeed’ collection saves the raw messages received from the hpfeed channels. This data usually come in JSON format.



session

Name	Type	Nullable
_id	String(24)	No
protocol	String(50)	No
hpfeed_id	String(24)	No
timestamp	DateTime	No
source_ip	String(15)	No
source_port	Int	No
destination_port	Int	No
identifier	String(36)	No
honeypot	String(50)	No

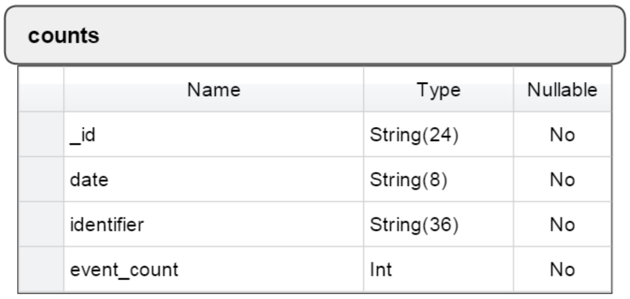
Fig. 2. The ‘session’ collection summarizes the interaction of a honeypot with the attackers.



daily_stats

Name	Type	Nullable
_id	String(24)	No
channel	String(250)	No
date	String(8)	No
hourly	Int	No

Fig. 3. The ‘daily_stats’ collection summarizes the data used to provide the statistics about the daily interactions of the honeypots.



counts

Name	Type	Nullable
_id	String(24)	No
date	String(8)	No
identifier	String(36)	No
event_count	Int	No

Fig. 4. The ‘counts’ collection summarizes various types of counts over time ranges in order to speed up aggregation queries.