SILESIAN UNIVERSITY OF TECHNOLOGY

FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

CONTROL, ELECTRONIC, AND INFORMATION ENGINEERING

Final Project

Design and implementation of a honeypot

Author: Dawid Czyrny

Supervisor: dr inż. Błażej Adamczyk

Gliwice, January 2018

Table of contents

1. INTRODUCTION

Modern world is firmly based on straightforward Internet access. The amount of data exchanged within the world network is tremendous. People are online incessantly, continuously checking their emails and social media activities, making transactions and placing online orders. The fact that our lives take place as much as in real world as in the online world creates opportunities for development as well as poses new threats to our privacy and security.

The constant struggle between the white hats – cyber security professionals trying to ensure our safety in the network and the black hats – hackers trying to put their hands on sensitive information, is ongoing. The activity of hacker groups has increased drastically through recent years and their attack vectors are getting more and more sophisticated. From 2009 over 7.1 billion identities have been exposed in data breaches [1].

The Figure 1 represents the industry sectors most commonly attacked by cyber criminals as of September 2017. Around 25% of energy, healthcare and retail industries worldwide were targeted for attacks leading to possible sensitive information leakage or even denial of service. The case is known where in a result of infamous *WannaCry* ransomware attack a hospital was unable to provide the help the patients needed [2].

Computer security professionals puzzle to stay ahead of the hackers and invent new ways of introducing higher security levels to the systems as well as prevent the existing vectors of attack. In order to do that one need to have access to the information about how do the hackers act and what kind of security flaws has the software being currently exposed to the world web.

One way to do that is to gather such data yourself using a so called honey pot. Honeypot is a piece of software designed solely for interaction with hackers. It is a trap that simulates an activity of a real world server. Any connections and interactions with the server are logged for further analysis creating an invaluable source of intelligence about the hacker's activities and attack methods. It is a matter of discussion if such approach is actually a proper defence method as honeypot itself invites the hackers to attack it providing new means for intrusion [3].

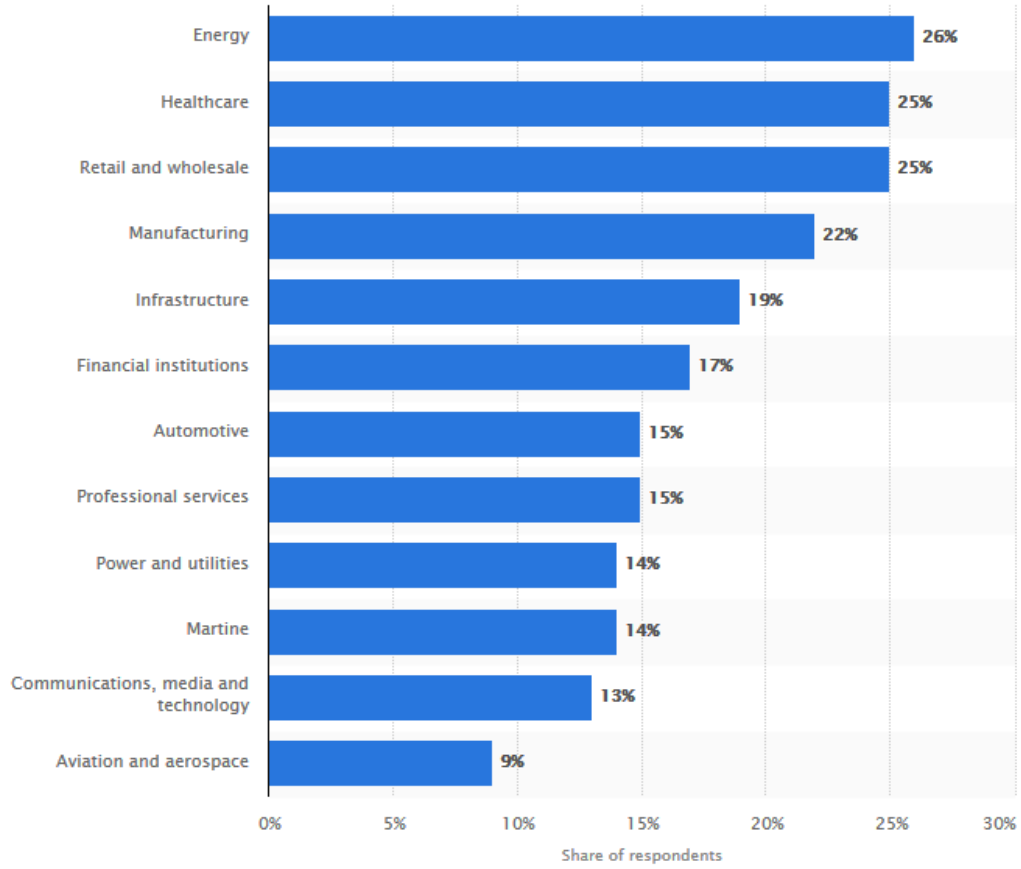

Figure 1 Industries impacted by cyber attacks worldwide as of September 2017 [4]

The author's motivation for this project was the urge and hunger for IT security knowledge. Nowadays cyber security is put on par with physical or political security as lots of sensitive data is being processed by the computers. Designing such system as honeypot allowed to increase author's proficiency in the field of network security as well as to broaden his software engineering skills. Moreover this kind of project has favourable potential for further development and extension.

The purpose of this work was to implement a low interaction* honeypot, capable of emulating services and logging the incoming data. The application was developed with the aim for modularity and stealth.

The chapter 2 brings closer the overall idea of honeypots and their purpose in computer security in general. In the chapter 3 the main assumptions of the project

* Low interaction – basic service emulation, more in chapter 2.2 Classification by level of interactivity

are enclosed with the software design details. The implementation and the application workflow are discussed in chapter 4. The results of extensive testing are presented in chapter 5. The last chapter number 6 concludes the whole work in a brief summary.

2. IDEA OF HONEYPOT

One of the first documented publicly available honeypot could be found in 'Deception ToolKit' by Fred Cohen in 1998 [5]. Introduced as system appearing to have lots of known vulnerabilities - it served as a deception to mislead the hackers and gather data about their activity.

They can be used either for research or defence purposes. Honeypot exposed to the Internet will be most likely attacked by all kind of bots and scanners searching for vulnerable hosts with open ports – probing its services and gathering data for possible further, more sophisticated attacks.

Honeypot deployed in a local network can work as a guardian alarming about any unusual activity in the network – more or less acting as an Intrusion Detection System (IDS) of sorts. Due to the fact that honeypot is not involved in a regular network traffic any connection towards it can be treated as an act of encroachment, alarming the administrators about undesired activity in the network, exposing the presence of an infiltrator.

Through years honeypots evolved and branched out according to their specific service emulation, level of interactivity or the environment they work in.

2.1. CLASSIFICATION BY SERVICE EMULATION

This classification refers to the ability of the honeypot to emulate single or multiple services. The system can be tailored to perfectly mimic the activity of e.g.. email server or could be designed to emulate several services like HTTP server and SSH server. Usually honeypots focused on single services imitate them indistinguishably from the real ones allowing to fool the attacker and gather highly valuable data and the ones mirroring several network activities are usually of low-interaction level providing more diverse data but at the cost of lower level of insight.

2.2. CLASSIFICATION BY LEVEL OF INTERACTIVITY

Such classification distinguishes how thoroughly the service is emulated. It could be either of low-interactivity – allowing the users to connect to the service but not to interact with it or of high-interactivity – utilizing full functionality of the service and recording the session of the user. There can be distinguished three types of interactions [6].

Low-interaction honeypots are easier to implement and easier to provide reliable level of security. On the other hand they do not provide extensive data about the intruder's attack techniques.

Medium-interaction level is not as common as low or high interaction levels as they tend to emulate the services only partially, providing decent layer of security and a fair amount of data for analysis. Partial emulation however can be a red flag for an attacker, making him realise that he is not in fact interacting with a real server.

High-interaction honeypots are harder in design especially from the point of view of security and isolation of the attacker from the rest of the system and network. Also the simulation of the real fully functional service like shell is not trivial. The advantage of such approach is the amount and variety of data logged during the hacker's interaction. Such data can be later on analysed at an angle of attack surfaces, even providing information about new unknown exploits and used tools.

2.3. CLASSIFICATION BY THE ENVIRONMENT

Honeypots can also be described by the environment they work in – it could be either a real life server machine devoted only to run the deception tools or it could be a virtual machine or a set of several virtual machines simulating the entire network.

Advantage of real servers is the possibility to isolate them on a physical level, but they need a separate machine that consumes energy and resources. Virtual machines on the other hand are capable of imitating multiple servers on a single computer generating convincing production network activeness.

The virtual environment by definition should be very well isolated from the host machine however there are known vulnerabilities disclosed in past years that

would allow the attacker for escaping the virtual machine [7] and interaction with the operating system underneath. Also one need to take into account the ability of the hacker to realize that he operates within the virtual machine [8], so precaution should be made to ensure that the virtual environment's indicators are hidden.

2.4. KNOWN SOLUTIONS

The concept of honeypot is nothing new, therefore there are many existing solution available for both commercial and non-commercial use. Some of the threat detection solutions are not honeypots as such but may act as one, i.e. sandboxes like FortiSandbox. Their purposes is to mitigate the existing attack vectors as well as to protect the critical systems from zero-day exploits* and unknown malware. They serve as active defence mechanism and the honeypot acts as intelligence gatherer.

Probably the best known honeypot is the *Honeyd* project [9] – its main strength is the use of virtualization. It is an open source software designed for Unix platforms and developed mainly by Niels Provos. The application is capable of emulating thousands of hosts running arbitrary services and providing the ability to hide themselves under various operating systems. The *Honeyd* is a low-interaction level honeypot, able to simulate an entire network topology – this creation is often called a honeynet. That way an attacker does not deal with single server but with what appears to be a real, alive network. Such enormous environment requires additional attention to security as the bigger the simulation the easier it is for an unnoticed flaw responsible for a vulnerability of the whole system.

Another example is *HoneyBOT* [10] - a medium-interaction honeypot designed primarily for Windows systems. It is a commercial software developed by Atomic Software Solutions. The creators claim that it is an easy to use application ideal for research purposes or as a part of an Intrusion Detection System. It is capable of safe capturing malicious payloads and intercepting raw data send by the potential hackers.

High-interaction honeypots are rather hard to implement, but there is an example of Linux distribution which is designed to act as fully interactive

* Zero-day – newly discovered vulnerability, unknown until recently

deception. The *Honeywall* [11] is able of collecting highly accurate data regarding the cyber threats and thanks to support of Sebek – kernel module devoted to extensive data gathering – is capable of logging the keystrokes on the system, even in encrypted environments. It is released by the *Honeynet Project* under the General Public Licence.

There are a lot of honeypots developed by cyber security enthusiasts regarding single emulations like i.e. MySQL database server or SSH server recording the whole interaction session However none of these solutions appear to be the leading one and some of these projects have been in fact abandoned by their creators.

3. DESIGN OF THE HONEYPOT

The main goal of the project was the implementation of easily deployable honeypot of low-interactivity level with main emphasis put on the modularity of the application as well as on its indistinguishability from an actual production server. Furthermore the application was designed to not rely heavily on outer frameworks in order for it to be portable and to assure low-dependency level.

3.1. PROJECT ASSUMPTIONS

As a result of finite time dedicated for this project the system was thought to provide the handling of three essential protocols at the end of this project - namely the Hypertext Transfer Protocol (HTTP), the Simple Mail Transfer Protocol (SMTP) and the Secure Shell (SSH) cryptographic network protocol.

3.1.1. STEALTH

The easiest way to obtain the information about running services is to use the so called banner grabbing technique [12]. For the honeypot to remain undetectable it needs to introduce itself as real world application. In order to do that the honeypot needs to substitute the banners of emulated services with the ones of the real programs – for the sake of that low-level networking needs to be involved.

3.1.2. LOW-LEVEL DEPENDENCY

The usage of native libraries is favourable in this project for two purposes. Firstly it fulfils the requirement for no use of heavy frameworks making the application light-weight in manner and secondly it allows for fairly easy access and modification of the data the honeypot is going to use to introduce himself to the attacker.

3.1.3. MODULARITY

For the program to be modular and easily extensible it has to be divided into the core part and the service emulation part [13]. The core engine manages the spawning and termination of separate services. Emulation part handle services as modules - isolated and independent of each other. Individual logger is also assigned to particular emulated application. A single emulation is enclosed within one process. Such isolation level provides additional reliability features ensuring stability of the whole honeypot even if one of its services would fail due to unforeseen attack scenarios.

3.2. CLASS DIAGRAM

The Figure 2 represents the diagram class of the Honeypot application. *HoneypotEngine* class is the core of the program. It is composed of simple console user interface (*HoneypotUI*) and a *ServiceController* operating on the services themselves. The engine instantiates a single controller for the whole program loading the settings of the honeypot at the very beginning.

The user interface provides simplified set of information about current status of the honeypot. It reports to the user any failure that could occur during the initialization of the program.

The controller is composed of multiple services loaded dynamically at its initialization. It handles proper spawning of the services as processes and loading of their settings. It is also responsible for a shutdown and cleanup of child processes brought to life.

Each service derives from *Service* class providing interface for managing the initialization, functioning of the server and proper shutdown. The base class implements also the function responsible for setup of the individual logger. Particular service has its own dedicated settings file in which one can define to

what addresses should the server bind and on what ports to listen for incoming connections.

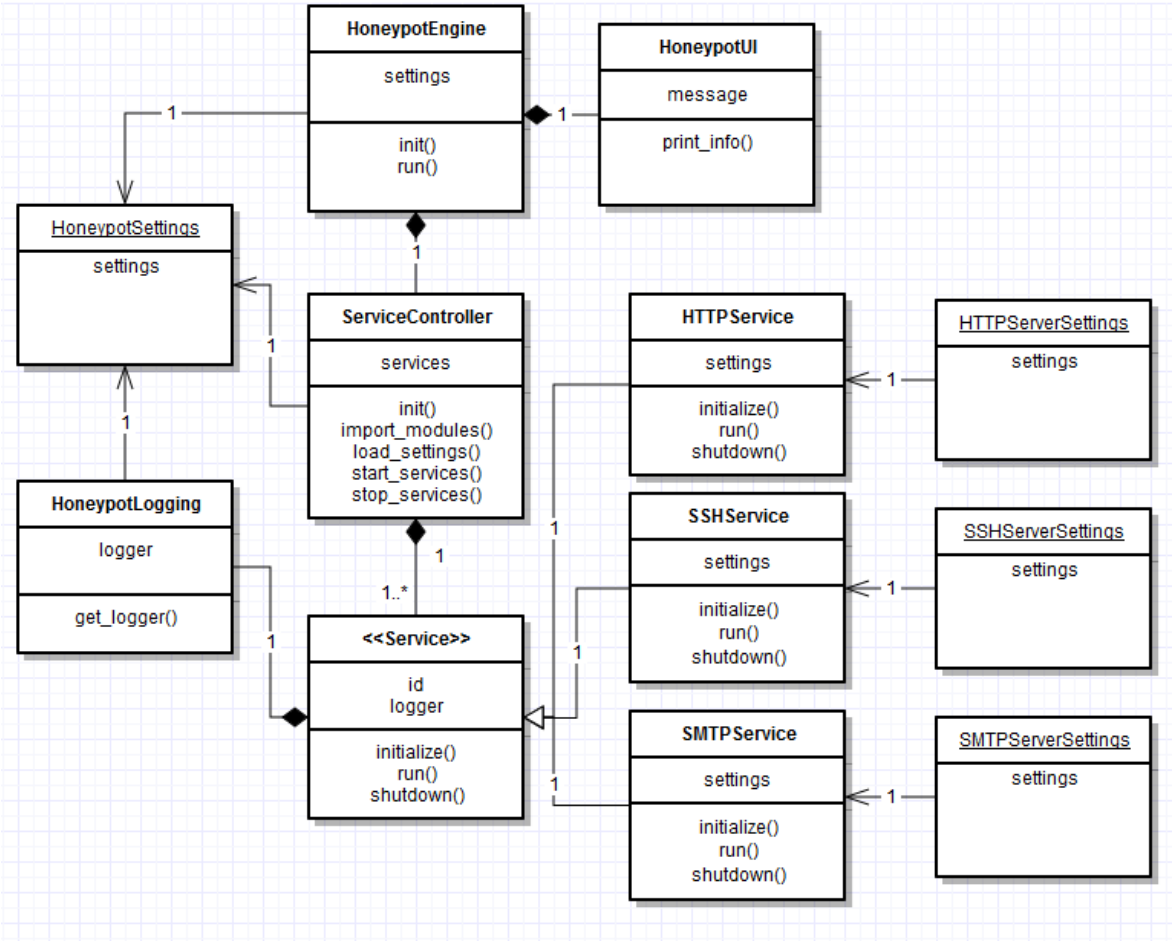

Figure 2 Honeypot class diagram

3.3. WORKFLOW

The below chart (Figure 3) brings closer the general workflow of the application. At the starting point a single instance of *HoneypotEngine* class is brought to life. After the initialization - a *run()* method is called which handles loading all of the subsystems of the honeypot and instantiation of used objects. The controller and user interface objects have the same lifetime as the engine itself as they are inseparable parts of it.

UI class handles the messages delivered by the *ServiceController* and prints them formatted to the screen. Controller is loaded with burden of handling the

whole service initialization and instantiation as well as their termination and proper cleanup after that.

The engine instantiates single *HoneypotUI* object and one *ServiceController* object. Later on when the *run()* method of the engine is called the controller imports all of the given services. Settings of all services are loaded into proper objects. Instantiation of services causes bringing to life their individual loggers. Each logger creates a new log file or appends to an existing one. New log files are generated daily. The type of data gathered in the logs depends on the given service implementation.

When the initialization is completed the services' main functions are spawned as separate processes together with their loggers. They serve their functionalities as long as the service controller sends the terminate signal which may occur either by the interaction of the user or by the set timeout.

The loggers and services close their opened files and listening sockets respectively. When the shutdown is completed, the cleanup is performed and the honeypot exits.

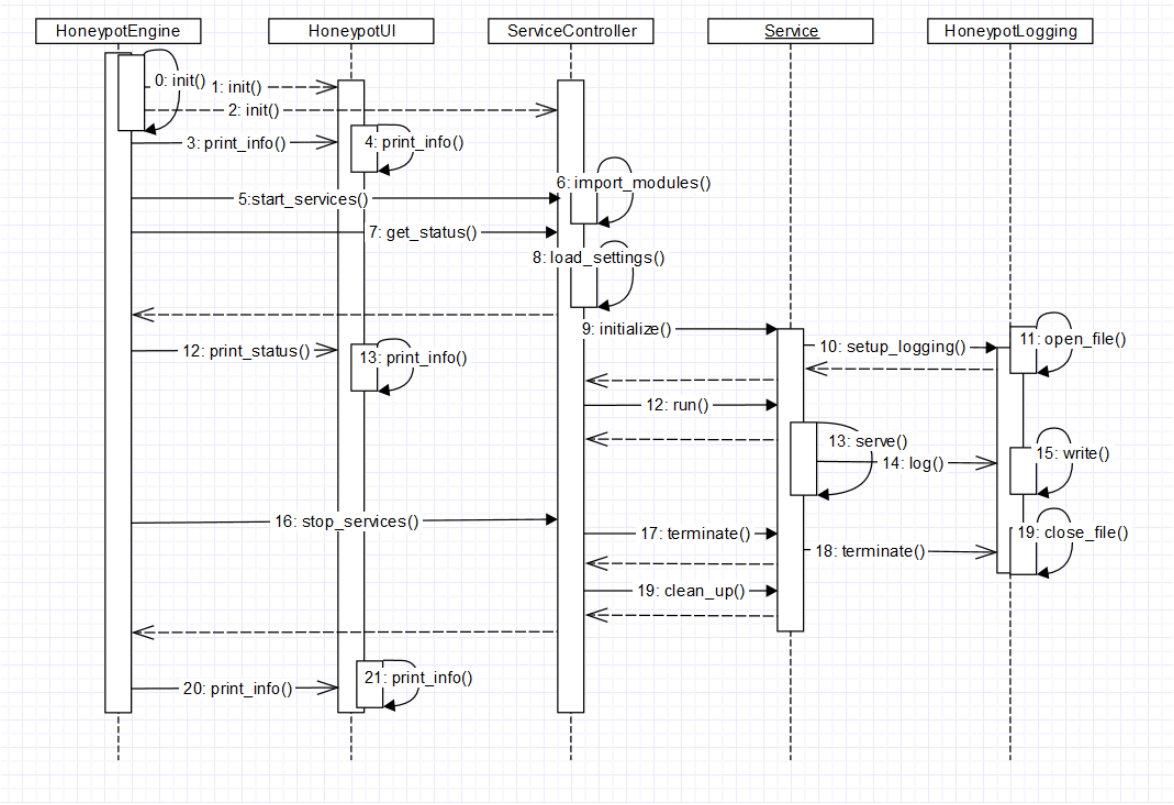

Figure 3 Honeypot workflow

4. IMPLEMENTATION

Considering implementation of the Honeypot a few determinants were taken into examination regarding the choosing of appropriate tools and libraries. Firstly how rich the availability of native components was in given solution. Secondly how well documented were the provided libraries. Thirdly, how active and reliable was the community of considered technology.

4.1. TECHNOLOGY

Having a vast, alive community and a rich, well-documented library of components ready to use a Python language was chosen as the environment for the implementation of the honeypot.

Python is a script language founded in 1991 by Guido Van Rossum. It is a heavily object-oriented programming language actively developed by its devoted community [14]. It forces the readability of the code, is free and cross-platform – available for all major operating system: Windows, Linux and Mac OS X. Moreover Python is secure as it does not operate on the memory directly like in case of C family language, making it more reliable. Any errors or exceptions occurring during the interpretation of the code are traced back to its source unless they are explicitly silenced.

Main disadvantage of this language is its performance. Due to being high-level interpreted language it appears to be slower than the compiled alternatives. However there is a possibility of conversion and compilation of the scripts into a standalone executable using e.g. *PyInstaller* [15], thus eliminating this minor flaw.

The Python's current version is designated as 3.6, released in December 2016. The language was chosen also due to the author's previous experience with it and because of the security professionals adoration towards it. [16]

4.2. LIBRARIES

Python [17] is famous for its massive collection of standard library modules which cover almost every aspect that is relevant in computer programming – from graphical user interface through web application modules down to process management and operations on single bits. It has also broad range of frameworks

available for the programmers to suit their needs – encapsulating basic functionalities and providing higher-level interfaces for it.

4.2.1. STANDARD LIBRARY

In order for the honeypot to be able to dynamically load all the services an import function was needed. The module *importlib* serves such purposes. The *ServiceController* makes use of it importing all of the provided services' modules on the fly making the designed honeypot highly extensible.

The other issue was the isolation of emulated services. It could have been done using *threading* module however threads share common memory which was not necessary in this case, in fact even unwanted because of security. To provide true remoteness a *multiprocessing* module was involved allowing for spawning the simulated servers as distinct processes. The module provides pipe and queue functionalities for sharing the data between spawned children, nevertheless there is no use for that as mentioned earlier. There are three supported ways of starting new processes[*] in Python – the honeypot uses spawning as it makes a copy of the data needed for the process to run and no unnecessary file descriptors and handles from the parent process are shared. A fresh python interpreter session is also started which is rather slow in comparison to other ways of starting processes however it provides the desired isolation.

Each spawned service uses Python's native *logging* module – it implements flexible event logging system its advantage is the ability to log not only the programmer's explicit messages but also any exceptions that would occur during the lifetime of the application it relates also to messages from third-party modules. Loggers support filtering and formatting of the stored information in the desired manner. A separate log file is created for each specific service. The *logging* module is additionally intended to be thread-safe using threading locks.

Due to the project's network nature a lion's share was based on Python's standard *socket* handling library and on *socketserver* – a module that provides an interface for writing web servers. The whole communication was build on top of the *TCPServer* class from the mentioned *socket* module supporting various network system calls.

---

[*] spawn, fork, forkserver - https://docs.python.org/3/library/multiprocessing.html

The HTTP service uses custom request handler and custom HTTP server derived respectively from the *BaseHTTPRequestHandler* and the *HTTPServer* supplied by *http.server* module. The handler's request processing methods were overridden and the introductory banners were substituted. Right now the service introduces itself as a Debian Apache2 server – through the banners and through the index page, capturing all the requests coming to it and storing the headers, client's IP address and the raw request as well. The server is composed of the tailored handler and a *ThreadingMixIn* class from *socketserver* to provide asynchronous event handling. The configuration of the service can be changed through the modification of the *HTTPServerSettings*.

The SMTP service is based on the *smtpd* module from Python's Standard Library. Similarly to HTTP service it is also based on a custom server deriving from already implemented class *SMTPServer.* Overloading the functions with self tailored ones allows for capturing the data from the sender. Replacing the standard *SMTPChannel* class with a custom one provides the ability of disguising the server as a legitimate one. It introduces itself as Postfix – Linux SMTP server. The service is also able of collecting raw data in case of some non-ASCII characters which would cause the process to crash. Multiple sessions can be handled at the same time due to involvement of *asyncore* and *asynchat* modules which provide such functionality. Simple interaction with the server is possible using commands like: *HELO/EHLO, MAIL FROM, RCPT TO* or *DATA*. The mail server introductory banners and binding settings can be changed within the *SMTPServerSettings.*

The last service emulated in accordance to the project's expectations is the handling of Secure Shell protocol. In its nature this protocol relies heavily on the cryptographic aspect of security. As implementation of such functionalities does not belong to trivial ones an outer library was used for this purpose – namely the *Paramiko* project discussed in the next paragraph.

4.2.2. PARAMIKO

One of the assumptions of the project was its low-dependency level and the reliance on standard Python libraries. However as SSH protocol is quite complex in its postulates therefore an exception was made and *Paramiko* [18] library was involved. It is a pure Python interface build around Secure Shell networking

concepts. The cryptographic aspect is provided by the Python C extension handling low-level encryption. It provides both server and client functionalities. In this project the server side was used.

The basis for the SSH service is Paramiko's *ServerInterface*. It provides handling of various authorization methods including Generic Security Services Application Program Interface, Digital Signature Standard, usage of public/private key authentication or simple verification with login and password. The project being a low-interaction level honeypot implements the SSH service simply using login and password.

Overriding the function responsible for checking the username and password allows for capturing of the used combinations. The handshake data is also gathered for further investigation along with the connecting client's version and its IP address.

The server opens a socket and listens for incoming connections after successful negotiations using Transmission Control Protocol handshake a secured session is established asking for authentication. The access to the server can never be granted as it is designed to reject any kind of username and password combination.

The SSH simulation can introduce itself as any real server, right now it is an OpenSSH server. To prevent maintenance of idle connections a simple timeout mechanism is implemented. These settings can be adjusted within the *SSHServerSettings* class.

In the future an implementation of SSH protocol handling based entirely on Python's Standard Library would be preferable due to project's main assumptions. External libraries based on additional C extensions can be potentially vulnerable which may lead to Remote Code Execution (RCE) on the server in the worst case scenario. Furthermore Paramiko itself supports only major system distributions.

4.3. GATHERED DATA AND SECURITY ASPECTS

The data logged into the files depends only on the implementation of a given service (module). Writing new service emulation forces the programmer to explicitly set the data written into the logs.

As for now the HTTP server logs all of the incoming requests, including headers, user agent. Currently unhandled methods like HEAD and DELETE are also logged. For more in depth analysis the data is also stored in raw form (Table 1).

The SMTP server is capable of logging the entire session with the user including incoming and outgoing commands. The data that cannot be interpreted as string is stored in hexadecimal format (Table 2).

Secure Shell protocol handles simple username and password authentication. The logger saves all of the incoming combinations of logins and passwords that the attacker is using for authentication. Moreover the hex dump of the negotiation phase is also stored in case if the hacker would try to tweak with the standard handshake (Table 3).

```
[...]
2018-01-05 14:17:04,626 - ERROR - ('HEAD / HTTP/1.1', 'ERROR', '-')
2018-01-05 14:17:04,647 - ERROR - ('HEAD / HTTP/1.1', 'ERROR', '-')
2018-01-05 14:17:04,943 - ERROR - ('GET / HTTP/1.1', '200', '-')
2018-01-05 14:17:04,944 - INFO - --- REQUEST DATA START ---
2018-01-05 14:17:04,944 - WARNING - client_address: ('192.168.56.101', 35838)
2018-01-05 14:17:04,944 - INFO - request: GET / HTTP/1.1
2018-01-05 14:17:04,944 - INFO - command: GET
2018-01-05 14:17:04,944 - INFO - path: /
2018-01-05 14:17:04,945 - INFO - request_version: HTTP/1.1
2018-01-05 14:17:05,003 - INFO - headers:
Host: 192.168.56.1
Connection: Keep-Alive
User-Agent: Mozilla/5.00 (Nikto/2.1.6) (Evasions:None) (Test:Port Check)
2018-01-05 14:17:05,004 - INFO - raw_request: b'GET / HTTP/1.1\r\n'
2018-01-05 14:17:05,004 - INFO - --- REQUEST DATA END ---
[...]
```

Table 1 Sample HTTP server log

[…]

2018-01-04 12:42:05,575 - WARNING - Incoming connection from: ('192.168.56.101', 50590)

2018-01-04 12:42:05,576 - INFO - 220 DB6PR0022.mydomain.com ESMTP Postfix (Debian/GNU)

2018-01-04 12:42:10,562 - INFO - b'ehlo name'

2018-01-04 12:42:10,762 - INFO - 250-DB6PR0022.mydomain.com

2018-01-04 12:42:10,763 - INFO - 250-SIZE 33554432

2018-01-04 12:42:10,763 - INFO - 250-8BITMIME

2018-01-04 12:42:10,763 - INFO - 250 HELP

2018-01-04 12:42:24,639 - INFO - b'\x0f\x06\x0e\x0f\x06\t\x06\x0e\x06\x0f\x05\x06'

2018-01-04 12:42:24,839 - INFO - 500 Error: command "   " not recognized

2018-01-04 12:42:25,061 - INFO - b'\x06\x0f\x05'

2018-01-04 12:42:25,261 - INFO - 500 Error: command "   " not recognized

2018-01-04 12:42:26,680 - INFO - b'\t\x0f\x06\t\x05\x06\x05\x0f\x05\x06\t'

[…]

Table 2 Sample SMTP server log

2018-01-05 14:40:33,633 - WARNING - Incoming connection from: ('192.168.56.101', 59770)

2018-01-05 14:40:33,634 - DEBUG - starting thread (server mode): 0x3ad7128

2018-01-05 14:40:33,634 - DEBUG - Local version/idstring: SSH-2.0-OpenSSH_7.6p1 Debian-2

2018-01-05 14:40:33,634 - DEBUG - Remote version/idstring: SSH-2.0-OpenSSH_7.6p1 Debian-2

2018-01-05 14:40:33,634 - INFO - Connected (version 2.0, client OpenSSH_7.6p1)

2018-01-05 14:40:33,635 - DEBUG - Write packet <kexinit>, length 504

2018-01-05 14:40:33,635 - DEBUG - OUT: 00 00 02 04 0B 14 AC AC 47 84 19 B3 54 FE 51 A6 ........G...T.Q.

2018-01-05 14:40:33,635 - DEBUG - OUT: B5 4F DB C2 0A 91 00 00 00 6F 65 63 64 68 2D 73 .O.......oecdh-s

2018-01-05 14:40:33,636 - DEBUG - OUT: 68 61 32 2D 6E 69 73 74 70 32 35 36 2C 65 63 64 ha2-nistp256,ecd

[…]

2018-01-05 14:40:35,502 - DEBUG - IN: 9E 53 97 A1 1C 4E 24 D9 E1 14 6A AD 81 0D F0 C0 .S...N$...j.....

2018-01-05 14:40:35,502 - DEBUG - IN: 55 82 26 E2 39 2D B8 A1 79 3C 99 DF 91 CE A1 95 U.&.9-..y<.....

2018-01-05 14:40:35,502 - DEBUG - Got payload (124 bytes, 75 padding)

2018-01-05 14:40:35,502 - DEBUG - Read packet <userauth-request>, length 48

2018-01-05 14:40:35,502 - DEBUG - Auth request (type=password) service=ssh-connection, username=root

2018-01-05 14:40:35,502 - INFO - Username: root

2018-01-05 14:40:35,502 - INFO - Password: helo

2018-01-05 14:40:35,502 - INFO - Auth rejected (password).

[…]

Table 3 Sample SSH server log

The Honeypot was designed to endure all kind of unknown attacks, however it is not safe to assume that it cannot be breached in some way. To maximize the secure behaviour of the application all of the services run by it are isolated and independent of each other. The application should never be run with administrator privileges, furthermore to ensure even higher level of security it should be run

preferably in virtual environment or some kind of container having no access to sensitive data. If the Honeypot fails in its task due to unknown attack there is a mechanism ensuring termination of all of the child processes emulating the services. It is triggered whenever an unhandled exception appears. This way any occurrence of memory leak that would lead to Denial of Service is highly unlikely.

5. TESTING

Regarding the honeypots their testing process takes a huge part in assessing their usefulness. For the honeypot to act as an effective trap it cannot be disclosed as one. It needs to faithfully mirror the activity of a real production server along with its introductory banners and fingerprints.

There is not one strict technique for testing the honeypots that would allow to estimate the effectiveness of the system and its undetectability for tools used by the attackers. However there are some suitable papers regarding this issue.[8], [19]–[21].

Due to the tests being of penetration nature a *Kali Linux* [22] distribution was used for this purpose. It is the most popular distribution for penetration testing tailored especially to meet the needs of pentesters. It comes with plethora of tools build into the system ready to perform all sorts of vulnerability and detection tests for both applications and servers.

Tests were performed using *Kali GNU/Linux Rolling* and the examined Honeypot was running on 64 bit *Windows 7 Service Pack 1* with usage of *Python* interpreter in version 3.6. The target machine had the following IP address: 192.168.56.1 and the attacking machine was identified by the address: 192.168.56.102. The services HTTP, SMTP and SSH were listening on their default ports, namely 80, 25 and 22 respectively.

5.1. GENERAL TESTS

The first tool and the most versatile used for testing purposes of the honeypot was the *Nmap: the Network Mapper* [23] in version 7.50. It is a very robust tool capable of scanning the entire network or a single particular host searching and probing for open ports. It is also competent of assessing the versions of listening applications as well as the operating system of the examined machine.

Launching a simple service discovery scan targeted towards the emulated services running on their default ports gives the following results:

nmap –sV –p 22,25,80 192.168.56.1

```
[...]
PORT   STATE SERVICE VERSION
22/tcp   open   ssh      OpenSSH 7.6p1 Debian 2 (protocol 2.0)
25/tcp   open   smtp     Postfix smtpd
80/tcp   open   http     Apache httpd 2.4.27 (Debian)
MAC Address: 0A:00:27:00:00:13 (Unknown)
Service Info: Host:  DB6PR0022.mydomain.com; OS: Linux; CPE: cpe:/o:linux:linux_kernel
[...]
```

Table 4 Default nmap service scan results

From the logs in Table 4 the Honeypot appears to be working as expected. It introduces itself with fake banners, making the attackers believe they are targeting an actual server with real services. On this stage the scanner assumes the server is being hosted by some Linux distribution.

The nmap tool is also equipped with its own scripting engine called *NSE* (Nmap Scripting Engine). Majority of those scripts are written by the community of this tool. They can be categorized by the level of intrusion, bandwidth exhaustion or by their purpose – vulnerability exposure, authentication bypassing, etc.

Using the whole set of *discovery* scripts which allow for more in depth view into the services' offerings we obtain the following results:

nmap -p80,22,25 --script=discovery 192.168.56.1

```
[...]
PORT   STATE SERVICE
22/tcp   open   ssh
|_banner: SSH-2.0-OpenSSH_7.6p1 Debian-2
| ssh-hostkey:
|_  1024 60:73:38:44:cb:51:86:65:7f:de:da:a2:2b:5a:57:d5 (RSA)
| ssh2-enum-algos:
|   kex_algorithms: (5)
|   server_host_key_algorithms: (1)
|   encryption_algorithms: (8)
|   mac_algorithms: (6)
|_  compression_algorithms: (1)
25/tcp open  smtp
|_banner: 220 DB6PR0022.mydomain.com ESMTP Postfix (Debian/GNU)
|_smtp-commands: DB6PR0022.mydomain.com, SIZE 33554432, 8BITMIME, HELP,
|_smtp-open-relay: Server is an open relay (14/16 tests)
80/tcp open  http
|_http-aspnet-debug: ERROR: Script execution failed (use -d to debug)
|_http-chrono: Request times for /; avg: 12486.10ms; min: 152.45ms; max: 22173.03ms
[...]
```

Table 5 Discovery scripts results

Report provides banners of services except for the HTTP server – the script failed in that case. Moreover it hands over the SSH host key fingerprint, the used domain and informs that the SMTP server is an open relay.

The fact that the SMTP server appears to be an open relay could be alarming for an attacker as such behaviour of this kind of services is rather uncommon. Some limitations should be introduced in the future regarding the accepted recipient's addresses and especially the sender ones.

Another issue is the capability of nmap to estimate the targets operating system more precisely by probing characteristic ports of it. As the Honeypot itself does not provide the option to masquerade the operating system it runs on, it has to be kept in mind to either hide the system's fingerprints using external tools or make sure to introduce the services according to the owned system.

nmap –sV –O 192.168.56.1

[...] *Remote Operating System Detection* *Used port: 25/tcp (open)* *OS match: Microsoft Windows Server 2008 R2 or Windows 8.1 (100%)* *OS match: Microsoft Windows 7 Professional or Windows 8 (100%)* *OS match: Microsoft Windows Embedded Standard 7 (100%)* *OS match: Microsoft Windows Phone 7.5 or 8.0 (100%)* *OS match: Microsoft Windows Vista SP0 or SP1, Windows Server 2008 SP1, or Windows 7 (100%)* *OS match: Microsoft Windows Vista SP2, Windows 7 SP1, or Windows Server 2008 (100%)* [...]

Table 6 Operating system detection results

5.2. HTTP SERVICE TESTS

A well-known tool used for HTTP server scanning purposes is *Nikto*. It is an open-source software capable of recognising the configuration of the most popular web servers. Searching for index files, interpreting response headers and looking for typical misconfigurations and some types of vulnerabilities.

The Table 7 features the results of Nikto scan which points out the lack of some of the security headers protecting against Cross-Site Scriprintg (XSS) and clickjacking (interception of user's mouse clicks). This can be both tempting and alarming for an attacker as absence of these headers can be a cause of possible vulnerabilities on the website leading to exploitation. On the other hand these headers are rather standard in modern web server communications, that is why it can be alarming.

Another instrument for HTTP server testing is the *Uniscan* included in Kali distribution. It is able to scan the target in search of Remote File Inclusion, Local File Inclusion and Remote Code Execution vulnerabilities.

As for now the Honeypot's web server hosts only default Apache's website the results of the scan are rather laconic (Table 8).

nikto –host 192.168.56.1

[...]
+ Server: Apache 2.4.27 Debian
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Scan terminated: 10 error(s) and 3 item(s) reported on remote host
+ End Time: 2017-12-19 10:11:26 (GMT1) (4 seconds)
[...]

Table 7 Nikto scan results

uniscan –jg –u http://192.168.56.1

[...]

Type Error:

http://192.168.56.1/;R48P1iq}=.Hx5Une.php responded with code:200 the server might just responde with this code even when the dir, file, or Extention: .php doesn't exist! any results from this server may be void

http://192.168.56.1/+fW=;c(B7._Gg+YyqY+W.html responded with code:200 the server might just responde with this code even when the dir, file, or Extention: .html doesn't exist! any results from this server may be void

[...]

Table 8 Uniscan results

The server is designed to log any kind of attempt of directory traversal and to respond to it with the code of 200 – OK. In order for the Honeypot web server to be more credible there could be added some other fake websites with possible fake admin panels and proper error response website.

Previously used nmap has also set of scripts designed for HTTP server scanning. Some of them were used against the honeypot. First one was checking the handling of different HTTP methods and the other returned full set of used response headers. Table 9 and Table 10 respectively.

nmap -p80 --script http-methods 192.168.56.1

```
[...]
PORT   STATE SERVICE
80/tcp     open      http
| http-methods:
|_  Supported Methods: GET POST
MAC Address: 0A:00:27:00:00:13 (Unknown)
[...]
```

Table 9 Nmap HTTP methods

nmap -p80 --script http-headers 192.168.56.1

```
[...]
PORT   STATE SERVICE
80/tcp open  http
| http-headers:
|   Server: Apache 2.4.27 Debian
|   Date: Mon, 18 Dec 2017 22:40:41 GMT
|   Content-type: text/html
|   Content-length: 10701
|
|_  (Request type: GET)
MAC Address: 0A:00:27:00:00:13 (Unknown)
[...]
```

Table 10 Nmap HTTP headers

The web server currently implements methods *GET* and *POST.* There could be added also handling of other commands like *PUT, HEAD* or *DELETE* for higher interaction level. Such changes would enlarge the set of gathered data and would increase the veracity of the decoy. Usage of additional headers would also benefit the operation of the Honeypot as mentioned above.

5.3. SMTP SERVICE TESTS

For testing purposes of the SMTP server connection a simple computer networking utility called *netcat* was used. It is build into majority of Unix-based systems, capable of establishing connections on the level of Transmission Control Protocol.

Table 11 *Netcat SMTP session* presents exemplary communication with the Honeypot's SMTP server. The connection was established successfully and the interpretation of commands was correct. Sending some non-printable characters to the server may cause lack of feedback answer however the server is still working and all the input is logged as raw data.

nc –C 192.168.56.1 25

```
[...]
220 DB6PR0022.mydomain.com ESMTP Postfix (Debian/GNU)
HELO mail@mymail.com
250 DB6PR0022.mydomain.com
HELP
250 Supported commands: EHLO HELO MAIL RCPT DATA RSET NOOP QUIT VRFY
MAIL FROM: my@mail.com
250 OK
RCPT TO: your@email.com
250 OK
DATA
354 End data with <CR><LF>.<CR><LF>
Hello there
how are you?
.
250 OK
[...]
```

Table 11 Netcat SMTP session

Using *Metasploit* framework – popular computer security exploiter equipped with modules aiding in exposing known vulnerabilities – some tests against open relay SMTP server were performed.

According to the results (Table 12) only 6 out of 16 tests performed by Metasploit gave positive recognition regarding the presence of an open relay. On the other hand nmap's scripting engine using *smtp_open_relay* script resulted in 10 out of 16 positive results (Table 13).

msf module: auxiliary/scanner/smtp/smtp_relay; options: 192.168.56.1:25

```
[…]
[2017.12.23-10:07:55] 192.168.56.1:25   - SMTP 220 DB6PR0022.mydomain.com ESMTP Postfix (Debian/GNU)\x0d\x0a
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #1 - No relay detected
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #2 - Potential open SMTP relay detected: - MAIL FROM:<sender@example.com> -> RCPT TO:<target@example.com>
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #3 - No relay detected
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #4 - Potential open SMTP relay detected: - MAIL FROM:<sender@[192.168.56.1]> -> RCPT TO:<target@[192.168.56.1]>
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #5 - Potential open SMTP relay detected: - MAIL FROM:<sender@[192.168.56.1]> -> RCPT TO:<target%example.com@[192.168.56.1]>
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #6 - No relay detected
[...]
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #11 - No relay detected
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #12 - No relay detected
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #13 - No relay detected
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #14 - Potential open SMTP relay detected: - MAIL FROM:<sender@[192.168.56.1]> -> RCPT TO:<example.com!target>
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #15 - No relay detected
[2017.12.23-10:07:55] 192.168.56.1:25       - Test #16 - Potential open SMTP relay detected: - MAIL FROM:<sender@[192.168.56.1]> -> RCPT TO:<example.com!target@DB6PR0022.mydomain.com ESMTP Postfix>
[2017.12.23-10:07:55] Scanned 1 of 1 hosts (100% complete)
[…]
```

Table 12 Metasploit open relay test results

nmap --script smtp-open-relay -p 25 192.168.56.1

```
[...]
PORT   STATE SERVICE
25/tcp   open    smtp
|_smtp-open-relay: Server is an open relay (10/16 tests)
MAC Address: 0A:00:27:00:00:13 (Unknown)
[...]
```

Table 13 Nmap open relay test results

Another nmap script can be used to find admissible usernames for using the SMTP server. The script introduces itself to the server using list of names and returns the ones positively recognized.

nmap --script smtp-enum-users -p 25 192.168.56.1

```
[...]
PORT   STATE SERVICE
25/tcp open  smtp
| smtp-enum-users:
|   root
|   admin
|   administrator
|   webadmin
|   sysadmin
|   netadmin
|   guest
|   user
|   web
|_  test
MAC Address: 0A:00:27:00:00:13 (Unknown)
[...]
```

Table 14 Nmap smtp_enum_users script results

The table above (Table 14) represents the usernames with permission to use the server. This refers to the *RCPT TO* and *MAIL FROM* options as no authentication methods are introduced into the server. As for now the Honeypot accepts any username so the list can be infinite. Proper limitations should be introduced in the future as mentioned at the beginning of the tests.

5.4. SSH SERVICE TEST

The Honeypot's SSH server was tested using OpenSSH client as well as the modules available in Metasploit framework. A minor stress test was also performed regarding the handling of password brute force attacks using *Hydra* – Kali Linux build in tool designed among others for dictionary password attacks.

The connection established using the SSH client is presented in the Table 15. The connection is terminated after 3 unsuccessful login attempts or after timeout period set in the *SSHServerSettings* Honeypot module.

The Metasploit's module able to enumerate available session logins failed to confirm usefulness of any of the given usernames. Results in Table 16.

ssh 192.168.56.1

```
[...]
The authenticity of host '192.168.56.1 (192.168.56.1)' can't be established.
RSA key fingerprint is SHA256:OhNL391d/beeFnxxg18AwWVYTAHww+D4djEE7Co0Yng.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.56.1' (RSA) to the list of known hosts.

root@192.168.56.1's password:
Permission denied, please try again.

root@192.168.56.1's password:
Permission denied, please try again.

root@192.168.56.1's password:
root@192.168.56.1: Permission denied (password).
[...]
```

Table 15 SSH session

msf module: scanner/ssh/ssh_enumusers; options: 192.168.56.1:22

```
[...]
msf auxiliary(scanner/ssh/ssh_enumusers) > run
192.168.56.1:22 - SSH - Checking for false positives
192.168.56.1:22 - SSH - Starting scan
192.168.56.1:22 - SSH - User 'user' not found
192.168.56.1:22 - SSH - User 'root' not found
192.168.56.1:22 - SSH - User 'guest' not found
192.168.56.1:22 - SSH - User 'admin' not found
192.168.56.1:22 - SSH - User 'administrator' not found
192.168.56.1:22 - SSH - User 'test' not found
Scanned 1 of 1 hosts (100% complete)
Auxiliary module execution completed
[...]
```

Table 16 Metasploit SSH username enumeration

The password brute force attack was performed using prepared lists of usernames and passwords presented in the Table 17.

users.lst	pass.lst
user	*root*
root	*123qwe*
guest	*password*
admin	*admin*
administrator	*test*
test	*guest*

Table 17 Username and password lists

Hydra performed an attack of 42 combinations of given usernames and passwords including blank passwords. The Honeypot's SSH server handled all of them and logged all of the sent data. Hydra's output is visible in the Table 18.

hydra -vv -t 4 -L users.lst -P pass.lst 192.168.56.1 ssh

[...]
Hydra (http://www.thc.org/thc-hydra) starting at 2017-12-23 10:35:08

[DATA] max 4 tasks per 1 server, overall 4 tasks, 42 login tries (l:7/p:6), ~11 tries per task

[DATA] attacking ssh://192.168.56.1:22/

[VERBOSE] Resolving addresses ... [VERBOSE] resolving done

[INFO] Testing if password authentication is supported by ssh://user@192.168.56.1:22

[INFO] Successful, password authentication is supported by ssh://192.168.56.1:22

[STATUS] attack finished for 192.168.56.1 (waiting for children to complete tests)

1 of 1 target completed, 0 valid passwords found

[...]

Table 18 Hydra brute force login attack results

6. SUMMARY

The project's main assumptions has been met successfully. The designed Honeypot is capable of processing various sorts of unexpected data. The decoy is resistant to crash attempts and its modular design has been implemented making

its extension more than feasible. The emulation of the services is rather firm, however there is place for improvement.

6.1. TEST RESULTS

The results of the performed tests are in the majority conclusive. The Honeypot serves its purpose well. None of tests caused the honeypot to terminate unexpectedly – it is mainly a merit of Python's high level nature and its indirect memory handling.

The service emulation is satisfactory for a honeypot of this level of interaction. All of the services can be accessed as any of the real legitimate servers and they introduce themselves as ones. However the HTTP server activity could be more alive and the SMTP server's open relay appearance should be limited.

6.2. COMPARISON WITH EXISTING SOLUTIONS

On the field of low-level interaction honeypots the project's implementation with a little effort could make a competition for existing solutions. Known honeypots are rather tailored to mimic particular service and most likely are closed or compiled applications.

The honeyd is an immense project devoted to simulation of entire network. The application is written and compiled in C language. Introduction of any changes to the code can be very difficult and the deployment requires some preparations.

The Conpot honeypot is designed for low-interaction within production networks. Being written in Python it emulates variety of common industrial protocols and allows for collecting motives and methods used by adversaries targeting industrial control systems. It can be easily deployed using Docker[*].

Cowrie is a medium interaction honeypot handling Telnet and SSH connections. It derives from Kippo – medium interaction SSH honeypot also written in Python that has been abandoned in its development. It is capable of logging the entire user session. Moreover any files downloaded by the attacker are saved for further analysis. The application also supports Docker deployment.

[*]Docker – container (https://www.docker.com)

The honeypot presented in this project written in Python – a high level language is capable of emulating any kind of service. The implementations of services can be effortlessly modified or added using simple text editor. The application requires only the interpreter and the Paramiko library. The interaction level of the service is entirely up to the programmer's will. The presented piece of software can be used seamlessly for personal research purposes or for computer security education as the code of the application can be tweaked and modified to meet the desired behaviour.

With some extension and more extreme testing it could be used in a real production network serving its main goals – alarming about unknown network traffic and logging all of the crucial data.

6.3. FUTURE PERSPECTIVE

The potential of this project for further development is noticeable. At first the improvement of HTTP server would be required. The possibility of serving additional websites with some fake administration panels would be highly advantageous in terms of honeypot's misleading purposes. Adding some interesting files visible from the HTTP server level would also benefit its credibility. It should also deliver some erroneous messages whenever access is requested to prohibited location.

As mentioned previously the SMTP server's behaviour should be restricted so it would not accept any kind of sender's nor recipient's address. Perhaps the login functionality would be also beneficial as well as implementation of handling of additional SMTP server commands.

In the future the SSH server should be implemented using the native Python's Standard Library to meet the project's initial assumptions regarding low dependency level. Implementation of another authentication methods could be also desired in order to enrich the gathered data about login attempts and for more reliable appearance in the network.

The implemented Honeypot is prepared for further extension. Theoretically it can handle up to 65536 services as many as there are ports in computer systems. Additional feature introduced to the project could be spoofing of the operating

system of the host machine. This way the user would not need to worry to match the emulation of services with the system it is ran on.

A desirable functionality would be also a recovery of the emulated services. For example if due to some circumstances the service would terminate the *ServiceController* of the Honeypot would be able to restart is and continue the emulation of the decoy.

As an improvement there could be also introduced the feature of sending the logs to some outer database in order to separate the trap itself from the data it is collecting. It should be done using some highly secured connection, even if the honeypot would fail somehow in its task the logs are safe on another server.

6.4. CONCLUSION

During the realisation of the project a lot of research was performed regarding the honeypots and their role in network security. It contributed to the increment of author's knowledge about the general concepts of computer security as well as its network aspects.

The implementation of the project's assumptions committed to the development of author's programming and software design skills. The Python's language environment could also be deepened as well as the low-level networking aspects of socket programming.

The security of information technology is always a trending topic and any substantial research on that would be beneficial to all of the security enthusiasts and professionals. This project had given the author yet another perspective into the information security world.

LITERATURE

[1] 'Internet Security Threat Report'. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf. [Accessed: 10-Dec-2017].

[2] T. Fox-Brewster, 'Medical Devices Hit By Ransomware For The First Time In US Hospitals', *Forbes*. [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2017/05/17/wannacry-ransomware-hit-real-medical-devices/. [Accessed: 10-Dec-2017].

[3] B. Scottberg, W. Yurcik, and D. Doss, 'Internet honeypots: protection or entrapment?', in *2002 International Symposium on Technology and Society, 2002. (ISTAS'02)*, 2002, pp. 387–391.

[4] 'Global industry cyber attack victimization rate 2017 | Statistic', *Statista*. [Online]. Available: https://www.statista.com/statistics/784590/cyber-attacks-on-industries-worldwide-2017/. [Accessed: 10-Dec-2017].

[5] P. G. Neumann, 'The Risks Digest', *Risks Dig.*

[6] R. M. Campbell, K. Padayachee, and T. Masombuka, 'A survey of honeypot research: Trends and opportunities', in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2015, pp. 208–212.

[7] 'Black Hat ® Technical Security Conference: USA 2009 // Briefings'. [Online]. Available: https://www.blackhat.com/html/bh-usa-09/bh-usa-09-speakers.html. [Accessed: 11-Dec-2017].

[8] N. Krawetz, 'Anti-honeypot technology', *IEEE Secur. Priv.*, vol. 2, no. 1, pp. 76–79, Jan. 2004.

[9] 'Developments of the Honeyd Virtual Honeypot'. [Online]. Available: http://www.honeyd.org/. [Accessed: 17-Dec-2017].

[10] 'HoneyBOT - the windows honeypot'. [Online]. Available: http://www.atomicsoftwaresolutions.com/. [Accessed: 17-Dec-2017].

[11] 'DistroWatch.com: Honeywall CDROM'. [Online]. Available: https://distrowatch.com/table.php?distribution=honeywall. [Accessed: 19-Dec-2017].

[12] R. Spangler, 'Analysis of remote active operating system fingerprinting tools', *Univ. Wisconsin–Whitewater*, vol. 36, 2003.

[13] A. B, R. Rajendran, C. K, and S. K.c, 'Analyzing Design Patterns for Extensibility', in *Computer Networks and Intelligent Computing*, Springer, Berlin, Heidelberg, 2011, pp. 269–278.

[14] M. Lutz, *Programming Python: Powerful Object-Oriented Programming*. O'Reilly Media, Inc., 2010.

[15] 'Welcome to PyInstaller official website — PyInstaller'. [Online]. Available: http://www.pyinstaller.org/. [Accessed: 05-Jan-2018].

[16] T. J. O'Connor, *Violent Python: A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers*. Newnes, 2012.

[17] 'Python 3.6.4rc1 Documentation'. [Online]. Available: https://docs.python.org/3/. [Accessed: 18-Dec-2017].

[18] 'Welcome to Paramiko! — Paramiko documentation'. [Online]. Available: http://www.paramiko.org/. [Accessed: 18-Dec-2017].

[19] S. Mukkamala, K. Yendrapalli, R. Basnet, M. K. Shankarapani, and A. H. Sung, 'Detection of Virtual Environments and Low Interaction Honeypots', in

2007 IEEE SMC Information Assurance and Security Workshop, 2007, pp. 92–98.
[20] P. Defibaugh-Chavez, R. Veeraghattam, M. Kannappa, S. Mukkamala, and A. H. Sung, 'Network Based Detection of Virtual Environments and Low Interaction Honeypots', in *2006 IEEE Information Assurance Workshop*, 2006, pp. 283–289.
[21] T. Holz and F. Raynal, 'Detecting honeypots and other suspicious environments', in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, 2005, pp. 29–36.
[22] J. Muniz, *Web Penetration Testing with Kali Linux*. Packt Publishing Ltd, 2013.
[23] 'Nmap: the Network Mapper - Free Security Scanner'. [Online]. Available: https://nmap.org/. [Accessed: 19-Dec-2017].