Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Measurement

Identifying Groups of Attackers Using Minimal Honeypots

Bc. Miroslav Hanák

Supervisor: Ing. Pavel Píša, Ph.D.
Supervisor–specialist: Ing. Karel Kočí
Field of study: Open Informatics
Subfield: Computer Engineering
August 2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Hanák**	Jméno: **Miroslav**	Osobní číslo: **434716**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra měření**		
Studijní program:	**Otevřená informatika**		
Studijní obor:	**Počítačové inženýrství**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Identifikace skupin útočníků pomocí minimálních honeypotů

Název diplomové práce anglicky:

Identifying Groups of Attackers Using Minimal Honeypots

Pokyny pro vypracování:

1. Get familiar with HTTP, FTP, and SMTP (submission) protocols in the scope of connection establishment, authentication, and subsequent disconnect.
2. Study and modify an existing implementation of Telnet minimal honeypot to support HTTP, FTP, and SMTP (submission) as well. Log an attacker's IP address together with protocol-specific selected metadata and authentication data. Furthermore, expand server-side software to
store new data to a database.
3. Analyze the collected logs in various combinations with different clustering algorithms. Consider that an attacker can be infected with more than one malicious program so it can be part of multiple groups.
4. Evaluate results and provide a suggestion for an application of the algorithms on collected data in regards to identifying new records as attackers and new groups of attackers.

Seznam doporučené literatury:

[1] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei ; Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M. (eds.): A density-based algorithm for discovering clusters in large spatial databases with noise (1996).
[2] Achtert, E.; Bohm, C.; Kriegel, H. P.; Kröger, P.; Zimek, A.: On Exploring Complex Relationships of Correlation Clusters. 19th International Conference on Scientific and Statistical Database Management (2007)
[3] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and T. Berners-Lee: Hypertext Transfer Protocol -- HTTP/1.1, RFC 2068, DOI 10.17487/RFC2068, January 1997
[4] Postel, J. and J. Reynolds: File Transfer Protocol, STD 9, RFC 959, DOI
10.17487/RFC0959, October 1985
[5] Gellens, R. and J. Klensin: Message Submission for Mail, RFC 4409, DOI 10.17487/RFC4409, April 2006

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Pavel Píša, Ph.D., katedra řídicí techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Ing. Karel Kočí, katedra řídicí techniky FEL

Datum zadání diplomové práce: **08.01.2020** Termín odevzdání diplomové práce: **14.08.2020**

Platnost zadání diplomové práce:
do konce zimního semestru 2021/2022

Ing. Pavel Píša, Ph.D.		prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to express my greatest gratitude to Ing. Karel Kočí for his guidance through the entire work process on this thesis. Furthermore, I would like to thank Ing. Pavel Píša Ph.D. for his valuable comments and remarks regarding this report. Another acknowledgment belongs to all awsome people developing the Turris routers, especially to Ing. Vojtěch Myslivec, Ing. Martin Prudek and Ing. Lukáš Jelínek. Lastly, I would like to thank my family who has supported me during my whole studies.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses,

Prague, August 14, 2020

Abstract

This thesis deals with the design and implementation of minimal honeypots and the following analyses of data collected by them. A honeypot is a tool for recording computer network attacks. The main goal of the analyses is to explore and choose the right algorithms for identifying groups of attackers and identifying new records as attackers. The minimal honeypots extend the data collecting system Sentinel deployed on Turris routers.

Keywords: honeypot, Turris, Sentinel, unsupervised machine learnig, clustering

Supervisor: Ing. Pavel Píša, Ph.D.

Abstrakt

Tato práce se zabývá návrhem a implementací minimálních honeypotů a následujícími analýzami dat z nich získaných. Honeypot je nástroj pro záznam útoků v počítačové síti. Cílem analýz je prozkoumat a vybrat vhodné algoritmy pro idnetifikaci skupin útočníků a identifikaci nových dat jako útočníků. Minimální honeypoty rozšiřují systém pro sběr dat Sentinel nasazený na síťových zařízeních Turris.

Klíčová slova: honeypot, Turris, Sentinel, strojové učení bez učitele, shlukování

Překlad názvu: Identifikace skupin útočníků pomocí minimálních honeypotů

Contents

Figures

Tables

Chapter 1

Introduction

This thesis aims to design and implement HTTP, FTP, and SMTP minimal honeypots and perform analyses of data collected by them. Minimal honeypots are honeypots that collect only connection and login attempts. They are added to Turris:Sentinel [43]; data collecting system deployed on Turris [1] routers and servers. It is data source of Distribited adaptive firewall [2]. The new minimal honeypots are going to increase data volume provided to the Distributed adaptive firewall and thus help it to block more malicious IP addresses. The main goal of the data analyses is to find methods and algorithms for identifying attackers and groups of attackers among captured IP addresses in regards to future enhancements of the Distributed adaptive firewall by those methods.

Turris routers, nowadays available to a wide public as fully open-source network device focused on security, came from not-for-profit research Project:Turris [3] of CZ.NIC, z. s. p. o. [4]. CZ.NIC is a not-for-profit association operating the Czech national top-level domain .CZ. The goal of the Project:Turris was to reveal how many network attacks focus on average Internet home users who do not run any services intended and advertised for public access. The first version of the Turris router was designed and developed for that purpose.

After some time of data collection and processing, an idea to automatically detect and prevent attacks raised. When someone on the Internet, connected to one of the routers and had malicious behavior, his IP address was automatically added on the list of blocked IPs called Greylist [5]. The list is automatically distributed to all the routers, which as follows, update their firewalls and block all the malicious IP addresses. Distributed adaptive firewall was born. [42] [49]

In the early stages of the project, the system capturing malicious IP addresses and information about them was designed and implemented with a specific purpose for a smaller specific amount of devices. Over time, the number of routers increased, and also demands on the collected data changed. The requirement for a new modular, extensible, and scalable system came up.

[1]https://www.turris.cz/en/

[2]https://project.turris.cz/en/security

[3]https://project.turris.cz/en/

[4]https://www.nic.cz/

[5]https://view.sentinel.turris.cz/

The new data collecting system - Sentinel, was designed and implemented from scratch to fulfill those needs.

The starting point for the work done in this thesis's scope is a study of the state-of-the-art honeypot background together with HTTP, FTP, and SMTP protocols, which serve as a base for the new minimal honeypot design. The protocols were studied only in a scope needed for understanding required for the minimal honeypots design. The protocols' standards leave enough space for creating unique and easily finger printable honeypot, thus real servers' behaviors were analyzed to provide a better base for the new minimal honeypots designs. Also, Sentinel's architecture and its component's functionality were studied for the following addition of the new minimal honeypots to it.

Sentinel's component running on Turris routers and initially implementing only Telnet minimal honeypot was redesigned and refactored to accommodate implementations of the three new minimal honeypots. This component was also thoroughly tested and validated. Mini integration testing framework was designed and implemented for testing purposes. Three Sentinel's server components were enhanced to process a new type of data introduced and collected by the new minimal honeypots.

After the redeployment of the updated Sentinel's components and a few days of initial data collection, the data's introductory statistics and observations were done. Based on the observations, the methods for attackers and groups of attackers identification were established. Various unsupervised machine learning algorithms were studied, and a couple of them were selected as promising candidates for the task of the identification. The algorithms were initially performed and tested with the collected data.

1.1 Thesis Layout Overview

The first part I, is intended to present fundamental background for later minimal honeypot design. Chapter 2 introduces what honeypot is together with various purpose, design, and implementation aspects. Chapters 3, 4 an 5 are dedicated to protocols of services which emulations are designed and implemented.

A purpose of the second part II is to introduce design and the original state of Sentinel system, which is extended by new minimal honeypots. Chapter 6 describes the Sentinel system as a whole and used technologies for its realization. Chapter 7 gives a more detailed look at Minipot component, where the new minimal honeypots' implementations will be added.

The third part III deals with the new minimal honeypots design and corresponding extension of Sentinel. Chapter 8 brings brief analyses and comparison of Sentinel with theory and best practices in chapter 2. Also, various aspects of new minimal honeypots design are presented in this chapter. Chapter 9 is Minipot component redesign and the new minimal honeypots implementation. Testing of the new minimal honeypots - Minipot component is described in chapter 10. Extensions to server components are stated in chapter 11.

In the final fourth part IV, analyses of data collected by new minimal honeypots are discussed. Chapter 12 presents basic statistics of the data collected by new minimal honeypots. The methods for attacker and attackers group identification are established in chapter 13 Clustering algorithms used for the identification are elaborated in chapter 14. Chapter 15 summarizes the results of this thesis.

Part I

Theoretical Background

Chapter 2
Honeypot

There is not any standardized honeypot definition. Honeypots are different than most of the security tools. Most of the tools address specific problems. For example, a firewall is used as a control of traffic flow. Intrusion detection systems are designed to detect attacks by monitoring network activity. Honeypots are different because they are not limited to solve a single specific problem. They can achieve so many various goals and can come in a variety of forms. Some of the goals are the following: detect, defer, capture, analyze attacks, analyze activities of the black hat community. We can see them as deception tools, weapons to lure attackers, intrusion detection tools, vulnerability emulation, jail for attackers, and controlled production system attackers can break into. That is why the honeypot definition is vague. However, all the goals and points of view share the same manifestation. Honeypot is a security resource whose value lies in being probed, attacked, or compromised.

It must be attacked to capture any information. There is no value if the honeypot is not attacked. Of course, one must first deploy a honeypot in a real network to be attacked. Honeypot should be separated from the rest of the network using firewalls and other defense mechanisms to safeguard other devices from attackers. It also adds fake value to the honeypot. A selection of the defense system plays a role in what one wants to achieve. To know how attackers compromise current defense mechanisms, the honeypot should be secured with it. To track an attacker and get information on how it causes damage, weak or no defense system at all is supposed to be used.

It is crucial to inform all people in an organization which deployed a honeypot, they must not interact with it, because any activity with a honeypot is suspected by nature. Any interaction initiated by honeypot means that system has been most like compromised, and the attacker is making outbound connections. Honeypot should not contain any valuable data. It has no production value. [1]

2.1 Types

Honeypots can be categorized according to the following schemes:

2.1.1 Purpose

Production

Production honeypots are placed inside of a production network with other production servers. Primarily companies and corporations use production honeypots to improve the overall security of their network and help mitigate risk. Production honeypots capture only limited data and give less information than research honeypots. They supposed to be simple, easy to use, build, and deploy. They generally have less risk. [1]

Research

Their main goal is to gather information about the motives, tactics of the black hat community targeting different networks. They are used to analyze threats that an organization faces and learn and how to better protect against the threats. Mostly research, military, and government organization use them. Research honeypots are complex to deploy and maintain and have a greater risk than production honeypots. They require extensive resources. [1]

2.1.2 Interaction Level

The more an attacker can do to a honeypot, the more information can be derived from it. However, at the same time, more an attacker can do, the more potential damage can occur - the more risk is taken. [1]

Low-interaction

They are the simplest of honeypots in design, functionality, implementation, configuration, deployment, and maintenance. They usually emulate some service. The emulation is usually done by software running on a host operating system. Thus the interaction is limited. Activities of an attacker are naturally sandboxed within the boundaries of the software, so he cannot harm other systems in a network where a honeypot is deployed. The level of risk is low. The software is usually installed on a host system and configured to offer whatever services an administrator wants. It makes deployment and maintenance easy. They log only limited information and capture only prior known activities. Their weak point is that they won't respond to any type of attack, which is not prior known and supported. This behavior can identify them. The main function is detection, especially unauthorized scans and connection attempts. [1]

High-interaction

They utilize actual operating systems rather than software emulations, so an attacker gets a more realistic experience, and more information can be gathered. It is advantageous to capture zero-days attacks and detect not yet known vulnerabilities. Attackers can use high-interaction honeypot to harm other systems in a network. They have high risk. A variety of technologies, such as firewalls and intrusion detection systems, are combined to implement high interaction honeypot. This architecture is complex and expensive to deploy, configure, and maintain, especially not letting an attacker know that he is monitored and controlled. Once deployed correctly, they can provide valuable insights that no other honeypot can do. [1]

Medium-interaction

They try to combine the benefits of low and high interaction honeypots while removing their shortcomings by utilizing application-layer virtualization. The aim is not to fully emulate a whole operating system nor to implement all application layer protocol details. They provide sufficient responses to known exploits on certain ports. Once the payload has been downloaded, it is analyzed, and a medium-interaction honeypot emulates action that would be performed to download malware. The malware can be downloaded and analyzed. [1]

2.1.3 Implementation

Commercial

It is seen as a software product, and usually, it must be purchased. A manufacturer should offer support and regular updates or adding new features. They come with GUI for easier usage, configuration, and maintenance. They are not that customizable for specific requirements. [1]

Homemade

They are fully customizable, and their cost is mostly cheaper than in the case of commercials ones. They require a great level of knowledge and skills for development, implementation, deployment, configuration, and maintenance. There is no official support. [1]

2.2 Value

The value of honeypot depends on what its goal is. There are advantages and disadvantages which affect the honeypot value.

2.2.1 Advantages

Data value Honeypots collect a low volume of data, but usually with high value, because every log is most probably a scan, a probe, or an attack. On the other hand, usually gigabytes of data - firewall logs, system logs, intrusion detection systems alerts are typically collected, but it can be difficult to derive any value from it.

Resources Honeypots don't require the latest technologies, a lot of memory, chip speed, or a large storage. They can be deployed on a relatively cheap computer. Resource limitations or resource exhaustion can be severe problems for firewalls, intrusion detection systems, log servers, which can generally process and analyze vast amounts of data.

Simplicity It is the main advantage. In general, honeypots don't require any complicated and advanced configurations. One must deploy it somewhere and wait.

Fewer false positives As mentioned above, any activity within a honeypot is suspicious. It is a minimal probability it is an accidental connection attempt.

Do not require known attack signatures Because all activities are suspicious; no attack signatures are needed to detect and attack. [1]

2.2.2 Disadvantages

Only direct interactions are monitored It is the greatest of honeypots disadvantage. If an intruder breaks into a network and attacks other systems than honeypot. The honeypot won't detect anything.

Risk An expert attacker can use them for more attacks.

Fingerprinting A honeypot can be identified if it has specific expected characteristics and behaviors, or the emulation of service is incorrectly implemented. It is a problem, especially in commercial honeypots. If a honeypot is identified, an attacker can use it to create fake attack alerts to cover the real attack. In a case of research honeypot, wrong information can be feed by an attacker to create a fake attack signature and avoid later attack detection. [1]

2.3 Roles in Network Security

2.3.1 Prevention

Honeypots add little value in preventing an attacker from getting into a network. But they can help to prevent an attacker from targeting an organization by deception and deterrence. Deception makes attackers waste of time and resources by attacking a honeypot. Deterrence is if attackers know an organization deployed honeypots, they can be scared to attack it. But deterrence and deception do not prevent from most common targets-of-opportunity attacks, because automated tools are used to compromise as many systems possible without prior analyses of the target systems. [1]

2.3.2 Detection

The detection means the act of identifying and alerting unauthorized activity and honeypots add great value to it. Usually, the task of the detections is done by intrusion detection systems - IDS. Still, they face three common challenges, which honeypots reduce quite well and thus make a good detection environment.

False positives They happen when IDS falsely alerts some activity as an attack typically because of wrong traffic modeling, bad rules, and signatures, etc. A few false positives are not a problem. But many of them are a big problem because an administrator should check all of them if they are truly false positives. Honeypots have no production traffic, so there is a minimal possibility of generating false positives.

False negatives They are when IDS fails to detect an attack. All activity within honeypots is captured and most likely an attack attempt, so false negatives are almost impossible.

Data agregation The challenge here is collecting all data used for the detection and corroborating that data into valuable information. Modern technologies are extremely effective at capturing an extensive amount of data, which makes valuable information retrieval even more difficult. Honeypots, in general, capture a small amount of high-value data. Complex honeypots can also detect a zero-day attack. [1]

2.3.3 Response

Once an attack is detected, a response to it is needed. Detailed information about an attacker's activities is critical. Often the compromised system is a production system running essential services. Hence it isn't easy to shut it down. When the system is offline, it is also difficult to distinguish an attacker's activities from normal activities only based on various logs. Honeypot can be quickly and easily taken offline for full forensic analyses. Also, all the retrieved data are most likely related to an attacker. [1]

2.4 Implementation Matters

2.4.1 Fingerprinting Mitigation

The mitigation of a honeypot signature is critical when we want to collect information or track an attacker. In the case of honeypot use as a deterrence tool, fingerprinting mitigation is not needed. The following measures help to mitigate the fingerprinting.

Behaviour modification It applies mainly to commercial honeypot solutions because they have the potential for signatures like standard behaviors or default configurations. So the default configuration should be changed before the deployment. Custommade honeypots don't have this issue, because everyone is unique.

Blending with environment It is good that the deployed honeypot is similar to the real systems as possible e.g, when mostly Windows OS is used, the right choice is Windows-like honeypot.

Developing realism Honeypots can be easily detected if they don't apply real systems, especially those that emulate operating systems. In case of usage of advanced operating system detection techniques such as IP stack analyses, differences between emulated OS and OS running the honeypot can be spotted. High-interaction honeypots running real OS can reduce this problem. [1]

2.4.2 Data Capture and Management

One of the main functions of a honeypot is data capturing. In general, research honeypots collect more detailed data than simpler production honeypots. Research honeypots can collect advanced information like used toolkits, attacker's keystrokes, and packet's payloads. Production honeypots usually collect just IP address of an attacker, a timestamp of an incident and attacked service. A place also influences types of captured data. Network sniffers, firewall, and router logs can be used to enrich data from honeypots.

Kinds of collected data are tightly coupled with data management because we will get the desired goals of a honeypot from collected data. If only honeypots in order of units were deployed at the same geographical location, i.e., building, data can be logged to the local system and retrieved manually. If there are multiple honeypots in different networks spread in various geographical locations centralized system to manage and aggregate all the information is needed. Centralized architecture has several advantages in this context. There is one point of data retrieval, backups, and archiving. Combining all the data in one place can increase their value. The data can be used for further analysis and mining.

A good practice is to have separate honeypot administration and logging interfaces, especially for a low-interaction honeypot. Enough precautions should be taken not to allow an attacker to use the interfaces to compromise devices in an internal network. Different honeypots can collect different types of data in various formats. The best way to approach it is a database system. [1]

2.5 Deployement Concerns

2.5.1 Number and Location Selection

Research honeypots should be placed in general outside of an organization's perimeter firewall because the main goal is to detect all the activities to which an organization is exposed to i.e., to collect as much information as possible.

Also, it is enough to deploy one honeypot at the network level and another honeypot on the host level. Placing more honeypots on the same level won't help to collect new information. Doing so can only increase the probability of an attacker interaction with the honeypot.

Production honeypots are mostly placed inside an organization's perimeter because the main task here is to interact with everything that has passed the perimeter's security. In the case of a large organization with multiple networks, several honeypots are needed, because different networks may require different honeypots to help to secure them. More honeypots mean more cost in terms of resources and time. [1]

2.5.2 Risk Mitigation

No system can guarantee 100% security, but a good security system should reduce the risk as much as possible. Honeypot interaction level plays the main role in reducing the risk. The greater the interaction level, the greater the complexity and risk are. As little as necessary, the interaction level should be selected for an expected value of a honeypot. Also, honeypot testing reduces the risk.

High-interaction A network with a high-interaction honeypot should be separated from other networks by router or firewall, allowing only inbound connections to the honeypot network. An outbound connection from the honeypot network should be blocked not to allow any harm to the production network.

Low-interaction The risk here is not coming from a honeypot itself, because there are no real operating system and applications, but from the base of the operating system on which the honeypot is installed. Best security practices and requirements should be followed and applied for the operating system before deploying a honeypot. [1]

Chapter 3

Hypertext Transfer Protocol 1.1

Designed in the early 1990s, HTTP was created to transmit the World Wide Web resources between web clients and web servers and has evolved to support the scalability needs of Web. A resource's nature isn't defined further; it can be a document, a photo, or anything else. A Uniform Resource Identifier identifies each resource and relationship between resources. HTTP adheres to a classical client-server model. A client program establishes a connection to a server and sends a request to a server. The server program accepts the connection and processes the request and sends one or more response messages back to the client.

HTTP does not include an identifier for associating a given request message with its corresponding response messages. It relies on that response arrives precisely in the order in which requests were made on the same connection. More than one response message for one request only occurs when one or more informational responses precede a final answer to the same request. It is a stateless protocol, meaning that the server does not keep any state between two requests. [12]

3.1 Components

User Agent refers to any of the client programs that initiate a request, including (but not limited to) browsers, spiders (web-based robots), command-line tools, custom applications, and mobile apps. [12]

Origin server is a program that can originate authoritative responses for a given target resource e.g., home automation, units, configurable networking components, office machines, autonomous robots, news feeds, traffic cameras, ad selectors, and video delivery platforms. [12]

3.2 Message Syntax

All HTTP 1.1 messages consist of a start-line followed by zero or more header fields, an empty line indicating the end of the header section, and an optional message body. A receiver of an HTTP message must interpret received bytes in an encoding, which a superset of ASCII encoding. HTTP does not have specific length limitations for many of its protocol elements because the lengths that might be appropriate will vary widely, depending on the deployment context and purpose of the implementation. [12]

3.2.1 Start Line

An HTTP message can be either a request or a response. These two types of message syntactically differ only in the start-line, which is either a request-line (for requests) or a status-line (for responses). [12]

3.3 Message Syntax

All HTTP 1.1 messages consist of a start-line followed by zero or more header fields. An empty line indicates the end of the header section, and an optional message body. A receiver of an HTTP message must interpret received bytes in an encoding, which a superset of ASCII encoding. HTTP does not have specific length limitations for many of its protocol elements because the appropriate leng will vary widely, depending on the deployment context and purpose of the implementation. [12]

3.3.1 Start Line

An HTTP message can be either a request or a response. These two types of message syntactically differ only in the start-line, which is either a request-line (for requests) or a status-line (for responses). [12]

Status Line

It consists of the protocol version, a space SP, the status code, another space, a possibly empty textual phrase describing the status code, and ending with CRLF. The 3-digit status-code indicates the result of the server's attempt to understand and satisfy the client's corresponding request. The rest of the response message is interpreted in light of the semantics defined for that status code. The reason-phrase element exists to provide a textual description associated with the numeric status code. [12]

3.3.2 Header Section

Each header field consists of a case-insensitive field name followed by a colon, optional leading whitespace, the field value, and optional trailing whitespace. The field name labels the corresponding field value as having the semantics defined by that header field. There is no limit on the header field's length or a header section's length as a whole, nor is there no limit on the header's count. The order in which header fields are received is not significant. A server must not apply a request to the target resource until the entire request headers section is received. The fields can contain conditionals, authentication credentials, or deliberately misleading duplicate header fields that would impact request processing. [12]

3.3.3 Body

A body of an HTTP message carries its payload. Its presence in the message is optional The body is identical to the payload body unless a transfer-coding has been applied. The rules for when a message-body is allowed in a message differ for requests and responses. Content-Length or Transfer-Encoding headers indicate the presence of the body in a request. The presence of the body in a response depends on both the the request method to which it is responding and the response status code. [12]

Transfer Codings

Transfer coding determines a transformation that has been applied to a payload body to ensure safe transport through the network.

Chunked The chunked transfer coding splits the payload body to a series of chunks, each containing its size indicator, followed by an optional trailer header fields. It allows data stream of unknown size to be transferred as a sequence of length-delimited buffers. This makes it possible for the sender to retain connection persistence and the recipient to know when it has received the entire message. [12]

Compress It is an adaptive Lempel-Ziv-Welch (LZW) coding. [12]

Deflate It is zlib data format containing a deflate compressed data stream that envolves a combination of the Lempel-Ziv (LZ77) compression algorithm and Huffman coding. [12]

Gzip It is an LZ77 coding with a 32-bit Cyclic Redundancy Check (CRC) that is commonly produced by the gzip file compression program. [12]

3.4 Semantics

3.4.1 Methods

The request method token indicates the purpose of the request and what the client expects as a successful result. The method's semantics might be further specialized by the the semantics of some header fields when present in a request if those additional semantics do not conflict with the method. [13]

GET method requests the transfer of a current selected representation for the target resource. It is the primary mechanism of information retrieval. [13]

HEAD method is identical to GET except that the server must not send a message body in the response. This method is usually used for obtaining metadata about the request-target. [13]

POST method requests that the target resource process the representation in the request according to the resource's own specific semantics. [13]

PUT method requests that the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. [13]

DELETE method requests the origin server to remove the the target resoource. [13]

CONNECT method requests that the recipient establish a tunnel to the destination origin server and if successful, after that restrict its behavior to forwarding of packets, in both directions, until the tunnel is closed. [13]

OPTIONS It allows a client to determine the options and requirements associated with a resource, or the capabilities of a server, without implying a resource action. [13]

TRACE method requests a remote, application-level loop-back of the request message. [13]

3.4.2 Version

HTTP uses a major.minor numbering scheme to indicate versions of the protocol. The protocol version indicates the sender's conformance with the set of requirements laid out in that version's corresponding specification of HTTP. The first digit - major version indicates the HTTP messaging syntax, whereas the second digit - minor version shows the highest minor version within that major version to which the sender is conformant and able to understand for future communication. [12]

3.4.3 Response Status Codes

The three-digit status code indicates the result of a request. The first digit of the status-code determines the type of response. The last two digits do not have any category. Status codes are extensible. HTTP clients are don't need to understand the meaning of all registered status codes However, a client must understand the class of any status code and treat an unrecognized status code as equivalent to the x00 status code of that class [13]

3.4.4 Headers

Header is a request modifier. A client can use it to provide more information about the request context, make the request conditional based on the target resource state, suggest preferred formats for the response, supply authentication credentials, or modify the expected request processing. Header fields are fully extensible. There is no limit on the introduction of new field names and their semantic All defined header fields ought to be registered with IANA in the Message Headers registry [37]. There are request message headers and response message headers. Some headers can be in both types of messages [12]

3.5 Connection Management

HTTP messaging is independent of the underlying transport or session controls. It only needs a reliable transport with in-order delivery of requests and responses. HTTP implementations are expected to maintain the state of current connections, establish a new connection or reuse an existing connection, process messages received on a connection, detect connection failures and close each connection. Most clients maintain multiple connections in parallel, even to one server. Most servers are designed to maintain many of concurrent connections and control requests to enable fair use and detect denial-of-service attacks.

HTTP 1.1 uses persistent connections, allowing multiple requests and responses to be carried over a single connection. The close connection option signals that the connection will not persist after the current request/response. A client supporting persistent connections can send multiple requests without waiting for each response. This is called requests pipelining. Servers usually have some timeout value beyond which they close an inactive connection. There are no length requirements on the timeouts when a persistent connection is used. A client or server that wants to close the connection due to a time out should do it gracefully. A connection can be closed at any time. A server should sustain persistent connections, when possible, and let the underlying transport's flow-control mechanisms solve overloads, rather than terminate connections with the expectation that clients will retry as it can increase network congestion. [12]

3.6 Authentication

An extensible set of challenge-response authentication schemes by which a server challenges a client to provide authentication information provides a general framework for access control and authentication The IANA Authentication Scheme Registry [36] lists registered authentication schemes and their corresponding specifications.

A case-insensitive token describes the authentication scheme. A comma-separated list of parameters or a single sequence of base64-encoded information for authentication via the scheme follows. The realm authentication parameter indicates a scope of protection of a given resource. It allows resources on a server to be partitioned into a set of protection spaces, each with its authentication scheme and authorization database. A response can contain multiple challenges with the same scheme but with different realms.

A server sends 401 Unauthorized response with WWW-Authenticate header field containing at least one challenge for accessing a resource by a client. A user agent that wishes to authenticate itself sends to a server a request with Authorization header containing the response(s) for a server's challenges.

Each authentication scheme defines only how the credentials are encoded before transmission. No mechanisms for assuring credentials' confidentiality are not part of the HTTP specification. HTTP depends on the security of the underlying transport- or session-level services to provide confidential data transmission. Transmission of credentials in the header fields implies significant risk and security considerations regarding the confidentiality of the underlying transport service. [14]

Chapter 4
File Transfer Protocol

FTP was designed for transferring any files between two computers, over originally any network. It is a very old protocol, and there were released so many extensions adding new functionalities or changing current ones. FTP is an application layer, Request-Response, Client-Server, based protocol. At the Transport layer, it relies entirely on TCP [33] to provide reliability across the unreliable best-effort IP based networks. FTP is an example of an out-of-band signaling protocol. There are separate sessions - TCP connections for data transfers and control commands. [16]

4.1 Communication Flow

First, a client initiates one control connection from an arbitrary port to a server and sends commands - requests to it. The well-known port number assigned for the FTP control connection at a server is 21. The control connection complies with the Telnet protocol. A server sends to a client response for each command. The FTP commands specify the data connection (transfer mode, data port, representation type, and structure) and the file system operation (retrieve, store, delete, append, etc.). A new TCP connection is established for each data transfer and closed after the file transfer gets completed. A data connection may be used for simultaneous sending and receiving. If a server opens a data connection (known as Active mode), it doesn't work well across firewalls and Network Address Translation (NAT) gateways. To overcome these issues, clients use information from a server to initiate the data connection. This is known as Passive mode. The well-known port number assigned for the FTP data connection at a server is 20. The data doesn't need to be necessarily transferred to the same client, which initiated the control connection. It is also possible for a client to manage data transfer between two servers [16]

4.2 Commands and Replies Syntax

The File Transfer Protocol follows the specifications of the Telnet protocol for communication on the control connection. Network Virtual Terminal

ASCII character set [18] is default encoding used by FTP. However, another encoding can be negotiated via the Telnet protocol. [16]

4.2.1 Commands

The commands begin with a command code optionally followed by arguments separated with one or more Spaces. CRLF terminates a command. Upper and lower case alphabetic characters should be treated identically. [16]

4.2.2 Replies

A single line reply is defined to contain the 3-digit code, followed by Space, followed by one line of text and terminated by the CRLF sequence. The 3-digit code is intended for use by machine to determine the exact state of communication. The text is intended for the human user. [16]

The multi-line reply begins with the required reply code, followed immediately by a Hyphen known as Minus, followed by text. The last line will start with the same code, followed by Space, optionally some text, and the CRLF. In rare cases where the text contains three digits and a Space at the beginning of any line, the beginning of each text line should be offset by some neutral text, like Space. This scheme ensures that multi-line replies may not be nested. [16]

4.3 Commands and Replies Semantics

There has been released several FTP standard's extensions that introduced new commands and their new replies. All the extensions are tracked in [17].

4.3.1 Commands

Commands defined in [16] are divided into access control, transfer parameters, and service commands. The access control commands are USER, PASV, ACCT, CWD, CDUP, SMNT, REIN, QUIT. The transfer parameters commands are PORT, PASS, TYPE, STRU, MODE. The rest are service commands. Access control and transfer parameter commands and their brief descriptions are listed in the table 4.1.

4.3.2 Responses

The three digits of a reply code each have a special meaning. This allows a range from simple to sophisticated responses and actions by a client process. The first digit denotes whether the response is positive, negative, or incomplete. A simple client process will be able to determine its next action by merely examining the first digit. A client process that wants to know approximately what kind of error occurred can examine the second digit. The third digit gives a finer gradation of meaning in each function category specified by the

Command	**description**
USER	acces control user identification
PASV	acces control user authentication
ACCT	user account identification
CWD	change working directory
CDUP	change to parent directory
SMNT	mount file system data structure
REIN	user session termintion
QUIT	user session termination, control connection closure
PORT	specify data connection endpoint address
PASS	server listens for a data connection
TYPE	set data representantion type for transfer
STRU	set data file structure for transfer
MODE	set data transfer mode

Table 4.1: FTP commands overview

second digit. The reply codes must strictly follow the specifications. The text associated in reply is intended for a human user, and it may change according to associated command. [16]

Chapter 5
Simple Mail Transfer Protocol

SMTP serves for reliable and efficient mail transfer across a computer network. It is independent of the transmission system and requires only a reliable, ordered data stream channel. TCP [33] is usually used in the underlying layer to deliver it, but other options are possible. A mail can pass through several intermediate hosts in various networks and environments on its path from the sender to the ultimate recipient.

A process can transfer mail by SMTP to another process in another network via a relay or gateway process accessible to both networks. A relay SMTP system receives mail and transmits it, without modification to the message data other than adding trace information, for further relaying or delivery. A gateway SMTP system receives mail in one transport environment and transmits it to a system in another transport environment. The message can be delivered in a single connection between the original SMTP-sender and the final SMTP-recipient, or it can pass a series of hops through intermediary systems. An intermediate host is usually selected through the use of the domain name service (DNS) Mail eXchanger mechanism [26].

It is a client-server, request-response protocol. When a client wants to send a message, it establishes a two-way transmission channel and sends commands to the server. The server sends back responses to the commands. When the server has issued a success response at the end of the mail data. It accepts responsibility for either delivering the message or reporting the failure to do so.

Servers and clients provide a mail transport services they are called Mail Transfer Agents (MTAs). Mail User Agents (MUAs or UAs) are usually the sources and targets of mail. At the source, an MUA can collect mail to be transmitted from a user and send it to an MTA. The final MTA would hand the mail off to an MUA or at least transferring responsibility to it by depositing the message in a message store. For delivered messages, the receiving MUA obtains and process the the message according to local conventions. The MUAs usually submit messages for delivery by SMTP, but they retrieve delivered messages by IMAP protocol [28]. [19]

5.1 Submission

Although SMTP was defined as a message transfer protocol, that is, a means to route (if needed) and deliver finished (complete) messages. SMTP is now also widely used as a message submission protocol to introduce new messages into the transfer environment. SMTP, as specified in [19], is not ideally suited for this role. In some cases, the MUA cannot generate finished messages or local site policy may dictate that the message should be examined or modified. Completions or modifications of the messages are, in general, considered to be out of the scope of standardized downstream MTAs (MTAs after the first-hop submission MTA) functionality.

On route to its final destination, an email will often travel between multiple independent providers of email transmission services with no prior arrangement and different transmission rules. The messages' relays are often not authenticated to permit unconstrained communications. Therefore, it is hard to assign responsibility if undesired or malicious mail is injected into the mail infrastructure or debug problems occurring in mail transmissions.

In most email services, the weakest link in the responsibility chain is the initial submission of a message. Demand on submission servers to take responsibility for the message traffic they originate raised the importance of authentication and authorization of the initial submission Furthermore, the use of authentication technologies elsewhere in the service will provide limited benefits. To address these aspects, standardized mail submission protocol [25] has been developed that is gradually superseding practices based on original SMTP [19].

The four message-handling components distinguish Internet email architecture. MTA and MUA are already introduced in 5. At the origination end, an MUA works on behalf of end-users and performs initial submission into the transmission infrastructure, via a Mail Submission Agents (MSA). MSAs accept the message, perform any necessary preprocessing on the message, and relay the message to an MTA for transmission. MTA relays messages to other MTAs in a sequence reaching a destination Mail Delivery Agent (MDA) that, in turn, delivers the email to the recipient's inbox. Originally the MSA, MTA, and MDA were a single component.

This was reflected in the practice of performing all the transfers over TCP port 25. It is essential to distinguish MUA-to-MSA email submission, versus MTA relaying, versus the final MTA-to-MDA transition. MSAs and MDAs can take advantage of having prior relationships with users to constrain their transfer activities. An MSA can implement security policies and guard against unauthorized mail relaying or injection of unsolicited bulk mail. It also allows for the off-site submission by authorized users such as travelers. An MDA can reject all mail to recipients for which it has no arrangement to perform delivery.

The requirements for each of the components are becoming more sophisticated, so that their software and even physical platform separation is increasingly common. The separation allows the relevant software code differences, thereby different programs for relay, submission, and delivery.

Port 587 is reserved for email message submission. The message relay is unaffected and continues to use SMTP over port 25. Outbound traffic on the standard SMTP port 25 can be prohibited from avoiding spam generation by malicious software. But in some cases, it is not possible or convenient. An organization may choose to use port 25 for message submission as well. To promote the transition of initial message submission from port 25 to port 587, MSAs should listen on port 587 by default. But they should also be able to listen on other ports. [25] [29]

5.2 Authentication

MSAs must require authentication on port 587 and should require authentication on any other port used for submission. MSAs must perform authentication of all mail transactions on the submission port, even for a message having a recipient that would not cause the message to be relayed outside of the local administrative domain. Regardless of the submission port, MSAs must not relay unauthenticated messages to other domains. They must not be open relays. It helps in tracking and preventing unsolicited bulk email.

Transmitting user credentials in clear text over insecure networks should be avoided because anybody could eavesdrop and steal account data. MUAs and Mail Service Providers are discouraged from the use of cleartext protocols for mail access and mail submission without any security layer such as TLS [24]. [25] [29]

SMTP extension [21] allows for authentication mechanism selection by a client, an authentication protocol exchange, and optional negotiation of a security layer. The authentication mechanism is provided by Simple Authentication and Security Layer (SASL). A server should not use any configuration in which a plaintext password mechanism is used without protection against password eavesdropping. Client and server implementations of extension [21] must implement the PLAIN SASL mechanism [23] running over TLS to ensure interoperability [21] [25] [29]

5.2.1 Simple Authentication and Security Layer

The Simple Authentication and Security Layer (SASL) is a framework which provides authentication and data security services in connection-oriented protocols via replaceable mechanisms. It defines a structured interface between protocols and mechanisms and a protocol for securing the subsequent protocol exchanges within a data security layer. It allows new protocols to reuse existing mechanisms without requiring redesign of the mechanisms and allows existing protocols to make use of new mechanisms without redesign of protocols. This is achieved through the interfaces of an abstraction layer between the protocols and mechanisms. IANA maintains the SASL mechanisms registry [35]. [20]

5.2.2 Authentication process

It consists of a series of server challenges and client responses specific to the chosen SASL mechanism. The client initiates the authentication by sending the AUTH command 5.4.1 to the server. The first parameter of the AUTH command identifies the SASL mechanism to use.

A server performs the SASL exchange to authenticate the user, if it supports the requested authentication. Otherwise, the server rejects the AUTH command. Optionally, a security layer for subsequent protocol interactions during this session is negotiated. Some of the mechanisms can require it.

Client responses and server challanges are Base 64 [22] encoded strings. If any part cannot decode received Base 64 encoded challenge/response, it must cancel the authentication. The client can cancel authentication by sending a line with a single * character.

The optional initial response argument to the AUTH command can be used to save a round-trip when SASL mechanisms support it. After an AUTH command has been completed, no more AUTH commands may be issued in the same session. If an AUTH command fails, the client can continue in communication without authentication or try another mechanism or use different credentials by sending another AUTH command. The AUTH command is not permitted during a mail transaction 5.5. [21] [21] [29]

5.3 Commands and Replies Syntax

SMTP commands and replies are transmitted in lines. The line consists of zero or more ASCII characters terminated by the sequence character CRLF. These characters must not be transmitted separately in the content of a line. Then they must be transmitted only as CRLF line terminating sequence. The receiver should take no action until this sequence is received.

When underlying transport service provides an 8-bit transmission channel, each 7-bit ASCII character is transmitted, right justified, with the highest-order bit cleared to zero. Unextended SMTP service provides 7-bit transport only. 8-bit message content transmission may be requested by a client using SMTP extensions.

SMTP client that has not negotiated an appropriate extension with a particular server must not send data with information in the highest order bit of byte. Receiving systems should reject such data. Command verbs, argument values other than a mailbox local-part, extension names keywords, and free form text are not case sensitive. The local-part of a mailbox must be treated as case sensitive. SMTP extensions may explicitly specify case-sensitive elements. [19]

5.3.1 Commands

All commands begin with a command verb, optionally one or more parameters follow. Parameters are separated by SP character from the command verb and each other if there are more parameters. Some commands don't permit any parameters. Clients must not issue such parameters and servers should reject commands containing them as having invalid syntax, if no extension allowing them was negotiated.

To improve interoperability, receivers should tolerate trailing white space before the terminating CRLF sequence. The maximum total length of a command line including the command word and the CRLF terminating sequence is 512 bytes. SMTP extensions may be used to increase this limit. [19]

5.3.2 Replies

In general, SMTP reply is a positive or negative acknowledgment of the previous command. An SMTP reply consists of a three-digit number, usually followed by some text. The number and text are separated by SP character. The three-digit number is for use by programs to determine which action to take next. The text is intended for a human user. Servers must not send reply codes starting with other digits than 2, 3, 4, or 5, if no extension allowing it, was negotiated before. A client should treat receiving such out-of-range codes as fatal errors and terminate the mail transaction.

Multiple-line replies may deliver more information. They require that every line, except the last, begin with the reply code, followed immediately by a hyphen (minus) followed by text. The last line starts with the reply code, followed by SP, optionally some text, and CRLF. Servers should send the SP even if there is no text, but clients must be prepared for it to be omitted. The reply code on each line must be the same. The maximum total length of a reply line including the reply code and the CRLF is 512 bytes. SMTP extensions can increase this limit. [19]

5.4 Commands and Replies Semantics

5.4.1 Commands

Commands and their brief description are listed in table 5.1.
There can be defined a local SMTP service extensions, which can introduce private-use commands. Keyword values and command verbs starting with an upper or lower case X refers to these extensions, which can be used exclusively through bilateral agreement. An SMTP server that does not recognize such a command or keyword is expected to reject it. An extended SMTP server may list the feature names associated with these private commands in response to the EHLO command. [19]

5.4.2 Replies

Replies serve to ensure the synchronization of requests and actions in the process of mail transfer and to guarantee that the client always knows the

Command	description
HELO	starts session
EHLO	extended helo - starts session
MAIL	initiates mail transaction
RCPT	identifies recepient(s) of mail data
DATA	mail data follows
RSET	aborts current mail transaction
VRFY	asks for confirmation of a mailbox identification
EXPN	asks for returning members of a mailing list
HELP	asks for sending helpfull information
NOOP	no operation
QUIT	close transmission channel
AUTH	authentication [21] , see 5.2
ETRN	asks server to process messages waiting for the client [31]

Table 5.1: SMTP commands overview

state of the server. Every command must generate exactly one reply. The reply contains a three-digit code and text. The three digits of the reply each have a special meaning. The reply codes must strictly follow the specifications in [19]. On the other hand, reply text is recommended rather than mandatory and may even change according to its associated command.

SMTP reply codes are based on the FTP reply model 4.3.2. The first digit says whether the response is positive, negative, or incomplete. An unsophisticated SMTP client, or one that received an unexpected code determines its next action by checking this first digit. The second digit approximately defines what kind of error occurred. The third digit is used for the finest gradation of information. [19]

5.5 Communication Flow

The communication between client and server is an alternating dialogue, controlled by the client. The client sends a command, and the server responds. The communication is purposely lock-step, one-at-a-time, although this can be modified by mutual agreement upon extension requests such as command pipelining

A client initiates a session by opening a transmission channel to a server. The server responds with a welcome message. The client introduces its identity to the server by sending the EHLO command. Also, the use of EHLO indicates that the client can process service extensions. The server provides a list of the extension it supports in reply to the EHLO command. Older clients not supporting the extensions may initiate mail session by HELO command.

After the transmission channel is established and initial handshaking is completed, a client usually starts a mail transaction. The MAIL command determining sender identification starts the mail transaction. It can be sent only when no mail transaction is in progress. A series of one or more RCPT commands follows, giving information about a receiver or receivers. The DATA command initiates the transfer of the mail data and is terminated by the end of mail data indicator, which confirms the transaction. When mail message has been transmitted, the client may either request the end of a session or initiate another mail transaction.

A client may also require ancillary services such as verification of email addresses or retrieval of mailing list subscriber addresses. A session is terminated when the client sends a QUIT command. The server responds with a positive reply code, after which it closes the transmission channel. [19]

Part II

Sentinel Design and Original Implementation

Chapter 6

Sentinel overview

The main purpose of the Sentinel system [43] [49] is to collect data used by Distributed adaptive firewall [1] to block malicious IP addresse on all Turris [2] router devices. Section 6.1 is about technologies used in Sentinel system. Section 6.2 introduces the message standard for data exchange between Sentinel's components. An architecture and brief functional desription of Sentinel's components are presented in section 6.3.

6.1 Used Technologies

6.1.1 MQTT

MQTT stands for MQ Telemetry Transport. It is a client-server publish/subscribe messaging protocol designed to be lightweight, open, simple, and easy to implement. It is ideal solution for centralised data collection. MQTT is used for sending data from Turris routers to Sentinel servers. Datastream from minimal honeypot component is published under topic *sentinel/collect/minipot*. [44].

Publish/Subscribe

The MQTT protocol is based on the principle of publishing messages and subscribing to topics. Multiple clients connect to a broker - server and publish messages to topics (publishers) and subscribe to topics (subscribers). Many clients can subscribe to the same topics and do with the information as they please. The broker act as a central node controlling information flow for particular topics. [45]

Topics

As stated above, MQTT messages are published on topics. There is no need to configure a topic; publishing on it is enough. They are treated as a hierarchy. Slash / serves as a separator. This allows data separation in the same way

[1]https://project.turris.cz/en/security
[2]https://www.turris.cz/en/

as files in a filesystem. Clients can receive messages by creating subscriptions. A subscription may be to a specific topic when only messages to the topic are received, or it may include wildcards. [45]

6.1.2 ZeroMQ

ZeroMQ [3] (also ØMQ, 0MQ or ZMQ) is a high-performance asynchronous messaging library, aimed for distributed or concurrent applications. It provides a message queue, but unlike message-oriented middleware, a ZeroMQ system can run without a dedicated message broker. It supports common messaging patterns (pub/sub, request/reply, client/server, and others) over various transports means (TCP [33], in-process, inter-process, multicast, WebSocket, and more), making inter-process messaging as simple as inter-thread messaging. This keeps code clear, modular, and extremely easy to scale. A large community of contributors develops ZeroMQ. There are third-party bindings for many popular programming languages. Sentinel components running on the same device (router/server) communicate through ZMQ.

6.1.3 MessagePack

MessagePack [4] is an efficient binary serialization format, supported by over 50 programming languages and environments. MessagePack aims to be as compact and simple as possible. All the Sentinel messages 6.2 are groups of data objects. The message thus needs to flatten into a one-dimensional stream of bytes before sending it to another component. After another component receives the message, it is deserialized back to its original structure.

6.2 Sentinel Messages Format

It is intended to define standard information format between Sentinel components communicating through ZMQ, usually running on the same device (router, server). ZMQ supports multipart messages when a message is divided into several parts. Sentinel standard message must contain exactly two parts; type of a message and its payload. ZMQ requires binary data. Thus the whole message must be first serialized before sending it. MessagePack 6.1.3 is used for that.

6.2.1 Type

Type is a string identifying payload's data. It's hierarchically separated by slashes / in the same way as the MQTT topic 6.1.1. Type is used as the MQTT topic when sending standard messages from a router to the server.

[3] https://zeromq.org/get-started/
[4] https://messagepack.org/

6.2.2 Payload

Data in a payload are arranged in key-value pairs. Table 6.1 presents base of Sentinel messages payload. All the fields except *data* are the mandatory part of any Sentinel message. There can be any additional fields. The keys must be unique.

key	value
type	minimal honeypot type e.g. Telnet, HTTP, ...
action	type of captured event e.g. client connection, login attempt
ip	IP adress of a honeypot client - attacker
ts	time stamp of the message creation in unix time format
data	data related to captured event e.g. authentication data

Table 6.1: Minimal honeypot message payload content

6.3 Architecture

Sentinel data-collecting system is structured as microservices architecture. Each component is highly maintainable, testable, loosely coupled to other components and independently deployable [49]. The components can be divided into two groups according to where they are deployed - a server and a router components.

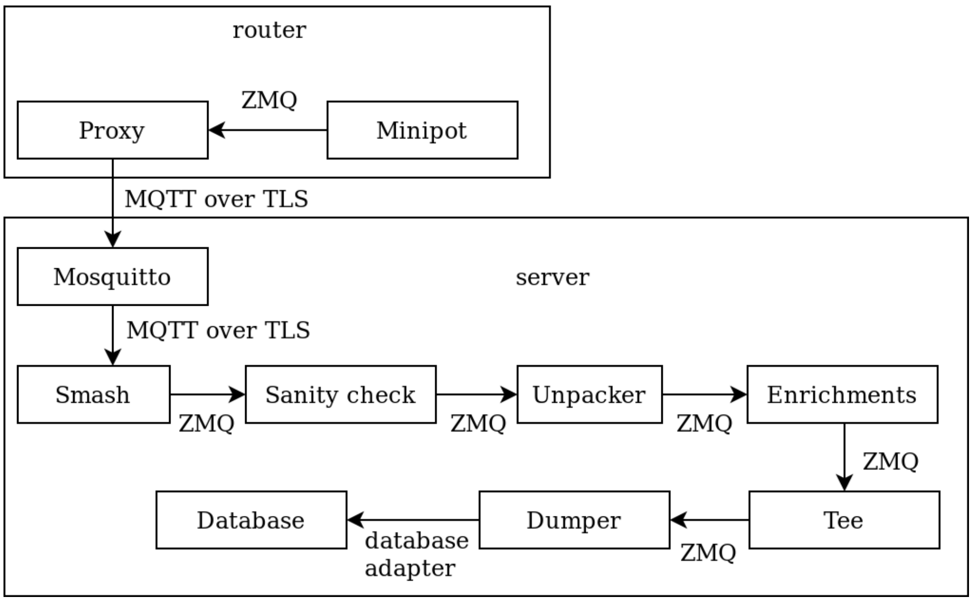


Figure 6.1: Components of minimal honeypot system

As represented in the figure 6.1, Proxy, and Smash communicate with Mosquitto over an encrypted channel, because the communication on lower layers is routed through the Internet. The authentication on both components is done by certificates. The certificates issuing and their distribution are managed by other Sentinel's components [49], which are not described here because it is not relevant for this thesis.

6.3.1 Router Components

Minipot [5] [6] collects authentication data by emulating various application layer services. Chapter 7 is dedicated to its more detailed description.

Proxy [7] relays messages received from Minipot (and also other Sentinel components running on a router) to Smash by MQTT protocol. It is an MQTT publisher client.

6.3.2 Server Components

Mosquitto [8] is an open-source MQTT broker - central node for router-server components communication.

Smash is server companion of router Proxy. It receives MQTT messages and forwards their payload - standard Sentinel messages over ZMQ to other components. It is MQTT subscriber client.

Sanity check checks incoming ZMQ messages payload on sanity according to given YAML [9] schema.

Unpacker Payload of minimal honeypot message contains field *type* 6.2 which describes type of minimal honeypot that generated the message. For several applications is more friendly to route messages directly according to minimal honeypot type. Unpacker reads *type* field in minimal honeypot message pyload and concatenates it to the message type field. Result of this is then the message with type field *sentinel/collect/minipot/type*.

Enrichments enrichs passing messages by country ISO code and autonomous system number from GEOIP MaxMind database based on IP address.

Tee duplicates Sentinel messages from input ZMQ socket to multiple output ZMQ sockets. It has name after Linux *tee* [10] command. Duplicated data stream is used as an input to Dynamic firewall system.

Dumper saves incomming minimal honeypot messages to a database.

Database Minimal honypot data are stored in PostgreSQL [11] open-source object-relational database management system.

[5]Minipot is in-house abbreviation for minimal honeypot.
[6]https://gitlab.labs.nic.cz/turris/sentinel/minipot
[7]https://gitlab.labs.nic.cz/turris/sentinel/proxy
[8]https://www.eclipse.org/mosquitto/download/
[9]https://yaml.org/
[10]http://man7.org/linux/man-pages/man1/tee.1.html
[11]https://www.postgresql.org/

Chapter 7

Minipot

Minipot [1] [2] component collects authentication data by emulating various application layer services and sends them through Proxy 6.3 to the Sentinel server. It accommodates the implementation of minimal honeypots. Originally, only Telnet minimal honeypot was implemented.

7.1 Architecture

Minipot consists of master process and child processes. [48]

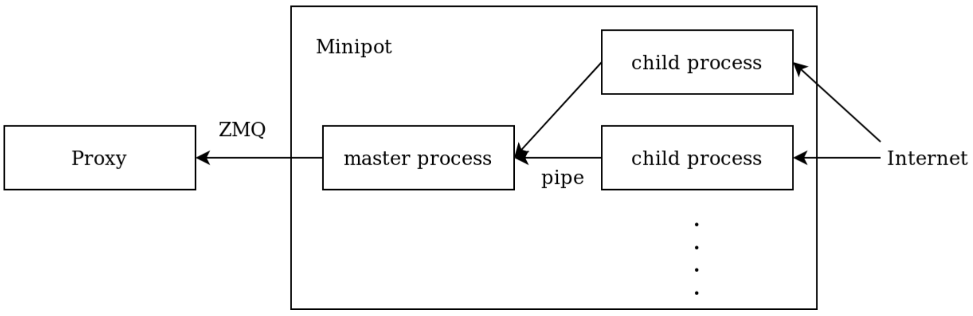


Figure 7.1: Minipot component architecture

Master process Master process creates child process, runs minimal honeypots, collects data from them and forwards the data to Proxy.

Child process Each child process runs particular minimal honeypot. There can be more than one child process running at the same time. It sends data to master process throught anonymous pipe.

7.2 Security Considerations

The master process is not directly exposed to the Internet. It does not process any input so that it can run under a privileged user. It can even run under

[1]Minipot is an abbreviation for minimal honepot.
[2]https://gitlab.nic.cz/turris/sentinel/minipot

a lower privileged user. It must be just ensured it has enough privileges to connect to the ZMQ socket provided by Proxy. Child processes are started under low privileged user (nobody by default), chrooted to */var/empty*, and dropped the ability to get new privileges before doing anything else. Thus, the process is not able to read any files (outside of */var/empty*, which should be empty) or do things reserved for high-privileged processes. [48]

7.3 Implementation

Minipot component is implemented in C language, intended to run in Turris OS [3] - GNU/Linux environment. It widely uses Linux system calls and *glibc* functions. It consists of the main process code and Telnet minimal honeypot code. The following subsections describe various implementation matters.

7.3.1 Program Arguments, Options and Parsing

Command-line options configure Minipot. Each, if used, must have one input argument. They don't have to be used. The corresponding variables have default values if not defined by an option argument.

- *-u/--user* User to drop priviledges 7.2.
- *-t/--topic* Defines *type* field of Sentinel messages 6.2 generated by Minipot.
- *-s/--socket* ZMQ socket 6.1.2 for communication with Proxy.
- *-T/--telnet* Port to run Telnet minimal honeypot on.

7.3.2 Event-driven Programming Paradigm

The control flow in Minipot depends on events' occurrences such as data arrived from a pipe, data arrived from a socket, software interrupts, some time elapsed. The event handling is effectively solved using libevent [4] library. It provides a mechanism to execute a callback function when a specific event occurs.

First, event base and events must be declared. Events must be added to the base, together with their callbacks and data attached to it. Then the event loop is run, and since that point, the program execution is controlled by the loop. When an event occurs, its callback is executed. The loop runs until it is broken.

This approach is used in the master process to handle incoming data from its child processes and in child processes running minimal honeypots to manage communications on several connections simultaneously.

[3]https://project.turris.cz/en/software
[4]https://libevent.org/

7.3.3 Child Process Creation

A new process is created using *fork*. The processe's priviledges are dropped as described in 7.2 and Telnet minimal honeypot is run on given port.

7.3.4 Master-child Process Communication

The master process communicates with the child process through a pipe created during a child process creation procedure. The read-end of the pipe is closed in the child process. The write-end of the pipe is closed in the master process.

Message from child process - minimal honeypot always contains *action* and *ip* fields and optionaly *data* field. These corresponds to fields of standard Sentinel message 6.2 fields of the same names. Each part is preceded by 4-byte integer containing the part's size in bytes. In case there are no data, the *data* field length is 0.

7.3.5 Proxy Communication

After the *action*, *ip* and possibly *data* fields are received from child process, *type* and *ts* fields are added by the master process and standard Sentinel messages is complete after this stage. If *data* lenght received from child process has lenght 0, the *data* field is omitted. The complete standard message is serialized by MessagePack 6.1.3 and added to a complete messages buffer. The buffer's content is serialized again and sent to Proxy in regular time intervals or when it is full. This ensures load-balancing and effective handling of the messages. Libevent 7.3.2 is used for regular time intervals functionality. [48]

7.3.6 Telnet Minimal Honeypot

Protocol implementation

Telnet protocol is implemented by a state machine which determines the the current state of communication and intended actions. Each connection runs its state machine, so there can be more connections served simultaneously. The state machines are driven by particular events callbacks executed by the event loop 7.3.2. Since the event loop handles all I/O operations, they are done in non-blocking mode. The callbacks overview is in the table 7.1.

Callback name	Occurance
on_accept	new connection request on listening port
on_read	data arrived from a TCP socket
on_sigint	SIGINT [4] signal caught

Table 7.1: Telnet minimal honeypot callbacks

Memory management

There can be a theoretically infinite amount of connection requests. Also, the number of ongoing connections varies over time. Dynamic allocation of memory for data preserving communications states machine can lead from extensive memory allocations and frees to system resources exhaustion. The pooling method is incorporated to avoid these drawbacks. Data connection structure pool of predefined size is allocated during minimal honeypot start. When a new connection is accepted, a data structure from the pool is assigned to it, and when the connection is closed, the data structure is put back to the pool. This is done only by marking an appropriate flag. The allocated memory is freed once during the process teardown. This avoids the costs of possible intermediate data allocations and deallocations. If there is no free data structure in the pool, an incoming connection is accepted and immediately closed. The size of the pool thus naturally represents connection count, which can be handled simultaneously. Thus the resource exhaustion is not possible.

Collected Data

Telnet minimal honeypot is very simple. It asks first for username and then for a password after a client connection by just sending the appropriate text strings. The login attempt consists of sending username and password by client-attacker after corresponding minimal honeypot requests. Telnet minimal honeypot generates two types of Sentinel messages 6.2 with *type* field *telnet*. The *actions* of the generated messages together with the messages' data are depicted in tables 7.2 and 7.3.

Action	Meaning	Data
connect	connection of a new client	no
login	authentication attempt	yes

Table 7.2: Actions of messages generated by Telnet minipot

Key	Meaning
username	authentication data
password	authentication data

Table 7.3: Data structure of *telnet* Sentinel message with action *login*

[4]https://www.man7.org/linux/man-pages/man7/signal.7.html

Part III

Sentinel Enhancements

Chapter 8

New Minimal Honeypot Design

8.1 Sentinel Analyses

According to 2.1 and 7, the new minimal honeypots supposed to be research, low-interaction, homemade, easy to implement, and deploy. The information gathered by them is supposed to be used by Dynamic adaptive firewall [1] to block revealed attackers, so they indirectly help to secure production systems. They should be a simple and reliable indicator of that some IP address is having malicious behavior. Their primary role in network security 2.3 is to detect unauthorized activity in a network where they will be deployed. But again, they help indirectly in attack prevention. They represent a very low risk for such a system because the attacker won't interact with any real service or system and won't access anything.

Based on 2.4, Sentinel [43] can be seen as a centralized system for minimal honeypot gathered data management and aggregation. This approach is needed in case of honeypots are spread in various networks in different geographical locations, as stated in 2.4, which is the case of minimal honeypot system. The comand-line options 7.3.1 can be considered as very simple administration interface. Thus separation of data management and administartion interface is ensured as written in 2.4. Usage of database system 6.3 is also mentioned in 2.4.

After noting that Sentinel design and architectural facts 6.3 [49] corresponds to honeypot implementation best practices from [1] summarized in 2, promising future for Sentinel and minimal honeypots can be concluded. The fact that the minimal honeypots should be a standard part of Turri OS running on possibly many thousands of routers placed around the whole world in various kinds of environments makes them a very unique and exclusive source of information, which analyses are described in IV.

8.2 General Minimal Honeypot Design

Before honeypot design, first, the services for emulation must be chosen. HTTP 1.1 3, FTP 4 and SMTP submission 5 were selected after consultation

[1]https://project.turris.cz/en/security

with the Turris Project team for their wide usage across the Internet.

8.2.1 Fingerprinting Mitigation

The minimal honeypots should appear from the attacker's point of view as regular server implementation. The knowledge of client-server behavior and messages syntax is needed to implement a server and naturally, its emulation. A particular protocol standard provides this information. But exact maximal messages lengths - buffer sizes, timeouts, connection limits, reply syntax, valid and invalid characters, messages/commands separators need to be known for any server implementation. Unfortunately as seen on HTTP [12], FTP [16] and SMTP [19], the standards doesn't define that fine-grained implementation details. It gives anybody who wants to implement any particular server some degree of freedom to establish them. This naturally leads to a unique and easily recognizable server implementation.

Fingerprinting mitigation 2.4 is one of the main factors to be considered during a honeypot design and implementation. To avoid implementing unique and easily recognizable honeypot by defining buffer's sizes, connection limits, timeouts, replies on our own, already existing server implementations are emulated. The exact buffer sizes, timeouts, etc. are obtained partially from the official documentation of the server implementations, but from vast majority from interactions with deployed servers. A real server implementation usually offers various configuration options, which can significantly change its behavior and its recognizability. This also implies the recognizability of a minimal honeypot emulating the server since the minimal honeypot design is based on the results of deployed servers exploitations.

There are plenty of particular server implementations. The Internet offers access to a plethora of deployed and somehow configured instances of these servers implementations. The particular servers implementations on which exploitations are base for minimal honeypot design were selected after consultation with the DevOps team of Turris Project. The DevOps team also provided access to configured and deployed instances of the selected server's implementations.

8.2.2 Data Collection

Each protocol mentioned above's datagram (HTTP message, FTP/SMTP command) contains information about clients and servers. The purpose of the minimal honeypot is to collect information leading to identifying the clients - attackers. Authentication data in their nature are a good source of information for the identification. Minimal honeypots mainly collect authentication data. In some cases, a few more protocol-specific metadata are collected. Each application protocol usually supports more authentication protocols/mechanisms. The selection of which data to collect together with which authentication mechanisms to support was discussed with Turris Project and The National CSIRT of the Czech Republic [2] teams' members. In general, authentication mechanisms using plaintext transfer of authentication data were chosen because they make the data collection simple and easy.

In principle, minimal honeypots interact with clients - attackers according to given protocol and send to Turris servers collected data in the form of standard Sentinel messages 6.2. The standard messages are sent to the servers only when data are fully available. It means in general e.g. when a valid login attempt was made.

8.2.3 Software Design

The new minimal honeypots combine Minipot's event-driven programming paradigm 7.3.2 and all the approaches used in Telnet minimal honeypot design 7.3.6.

8.3 Hypertext Transfer Protocol 1.1

Apache HTTP server [3] was selected as a template for HTTP minimal honeypot. According to [38] it is widely deployed.

HTTP is by design stateless protocol. Each request is independent of others. Also, authentication is done in one request-response sequence, where server challenge and c client responses are both sent in dedicated headers of HTTP request/response messages [14]. Currently, the only supported HTTP authentication scheme [36] by HTTP minimal honeypot is Basic [15], because as said in 8.2.2, the data extraction is easy.

HTTP minimal honeypot besides authentication data also collects method and URL fields contained in the HTTP message start line and content of User-Agent header. Method and URL fields always contained in the HTTP minimal honeypot generated message. All the headers are an optional part of the HTTP message, which implies that also the corresponding *authorization* and *user_agent* fields of the generated message are optional. Based on this, two kinds of Sentinel messages 6.2 are generated by HTTP minimal honeypot. Both messages have *type* field *http*. The *actions* of the generated messages together with the messages' data are depicted in tables 8.1 and 8.2.

Action	Meaning	Data
connect	connection of a new client	no
message	HTTP message received from client	yes

Table 8.1: Actions of messages generated by HTTP minipot

[2]https://csirt.cz/
[3]https://httpd.apache.org/

Key	Meaning	Presence
method	method field	always
url	URL field	always
authorization	content of Authorization header value	optional
user_agent	content of User-Agent header value	optional

Table 8.2: Data structure of *http* Sentinel message with action *message*

8.4 File Transfer Protocol

The vsftpd [4]FTP server implemenation was chosen to be emulated. As stated at [4] some large sites use it.

FTP is a stateful protocol. Usually, some operations require some particular sequence of commands. Only data collected by FTP minimal honeypot are authentication data. FTP has a built-in authentication mechanism consisting of USER and PASS commands. The authentication attempt is then ordered sequence of these commands. The authentication data are the commands' parameters. Ans their retrieval is, in the case of FTP, very trivial. Two kinds of Sentinel messages 6.2 are generated by FTP minimal honeypot. Both messages have *type* field *ftp*. The *actions* of the generated messages together with the messages' data are in tables 8.3 and 8.4.

Action	Meaning	Data
connect	connection of a new client	no
login	authentication attempt	yes

Table 8.3: Actions of messages generated by FTP minipot

Key	Meaning
user	authentication data
password	authentication data

Table 8.4: Data structure of *ftp* Sentinel message with action *login*

8.5 Simple Mail Transfer Protocol

Postfix [5] was selected as base for SMTP minimal honeypot design. [39] shows that it is widely used across the Internet.

SMTP is a stateful protocol. Usually, requested data operations require some particular sequence of commands. Only data collected by SMTP minimal honeypot are authentication data. SMTP has no built-in authentication

[4]https://security.appspot.com/vsftpd.html#about
[5]http://www.postfix.org/

mechanisms. The authentication mechanisms are provided by Simple Authentication and Security Layer [20]. Only supported SASL mechanisms [35] by SMTP minimal honeypot are Plain [32] and Login [34].

In both mechanisms, the first AUTH command with the mechanism as its parameter is issued by the client. When the Plain mechanism is selected, a server sends the Base64 [22] encoded challenge. The client then sends the Base64 encoded authentication response. When the login mechanism is selected, authentication is done in two rounds of challenge-response exchanges. This corresponds to exchanging usernames and passwords. The server challenges and client responses are also Base64 encoded as in the case of Plain mechanism.

SMTP minimal honeypot generates three kinds of Sentinel messages 6.2 with *type* field *smtp*. The *actions* of the generated messages together with the messages' data are in tables 8.5, 8.6 and 8.7.

Action	Meaning	Data
connect	connection of a new client	no
plain	authentication attempt using Plain scheme	yes
login	authentication attempt using Login scheme	yes

Table 8.5: Actions of messages generated by SMTP minipot

Key	Meaning
data	authentication data

Table 8.6: Data structure of *smtp* Sentinel message with action *plain*

Key	Meaning
username	authentication data
password	authentication data

Table 8.7: Data structure of *smtp* Sentinel message with action *login*

Chapter 9
Minipot Redesign

The original code base 7, i.e., master process and Telnet minimal honeypot, were heavily refactored to accommodate new functionalities and requirements for robustness, security, and scalability. The three new minimal honeypots and further improvements extend Minipot component functionality.

9.1 Original Codebase Refactoring

9.1.1 Command-line Options Parsing

Mew options were introduced to support new minimal honeypot types. Original options 7.3.1 are kept.

- *-H*/*--http* Port to run HTTP minimal honeypot on.
- *-F*/*--ftp* Port to run FTP minimal honeypot on.
- *-S*/*--smtp* Port to run SMTP minimal honeypot on.

The command-line options and their arguments repesents meachanism for mapping which honeypot to run on which port. Now there are supposed to run more than one honeypot at the same time. For simple control of honeypots to ports mapping, there are no default ports values for any honeypot. The mapping is done strictly by the command-line options and their arguments. At least one minimal honeypot must be defined (mapped to some port) for the run of the Minipot component. It has no sense to run the master process without any child data-collecting processes. The meaning of *--user*, *--topic* and *--socket* options remains the same. In the original implementation, CLI options' arguments values were not checked e.g., for allowed port value. Thus the options argument values check was added.

9.1.2 Creating Child Processes

Original implementation 7.3.3 was hardcoded to create just one child process and run Telnet honeypot into it. To support the execution of multiple child processes, the function was refactored. To store all child process-specific data, e.g., for sending signals, pipe communication, and easy memory management, the data pool principle is used. The pool entry data structure groups all child process-related variables. The size of the pool represents the maximal number of honeypots that can be run. This again prevents from resource exhaustion by running extensive amounts of honeypots - child processes. If command-line options define more minimal honeypots, only the maximal number of processes defined by the data pool size is created. The rest of the minimal honeypots defined by command-line options is ignored, and a user is informed about it by a message sent to standard output. If more than one honeypot is mapped to the same port, the Minipot component execution is terminated during the process of creating new methods and running honeypots.

9.1.3 Reporting Mechanism

Original reporting mechanism; master-child process communication 7.3.4 and master process - Proxy communication 7.3.5 design and the implementation were sufficient, correctly working and scaling in the case only one minimal honeypot child process running.

During integration testing 10.2 this original codebase part was discovered not working correctly and causing on first sight random tests fails. The detailed examination showed the problem originated in the implementation of the master - child process communication protocol. Besides of that both parts: master-child protocol and master process - Proxy communication lacked proper allocated memory tracking and freeing and proper error handling and Minipot component termination in such a case. These parts' implementations used unnecessary memory allocations and deallocations for each read from any pipe, causing performance shortage and higher system load. These parts were redesigned, and their code refactored for better performance, scalability, and proper memory and error handling.

The complete Sentinel standard message 6.2 is now composed in a minimal honeypot child process, serialized by MessagePack 6.1.3 and sent to the master process. The master process receives the data from a pipe and buffers them first because pipe reads are scheduled by libevent and handled by defined read event callback. There are supposed to be more child processes sending the data to the master process simultaneously. The serialized Sentinel message than can arrive in several read events, not immediately following each other. To overcome this, each pipe now has a dedicated data buffer. The buffer memory is allocated during a child process creation and deallocated at Minipot teardown. There are no intermediate memory allocations and deallocations. Child process sends data length first, so the master process has the information on how many data belongs to the current message. This approach is more robust and modular than the previous design.

When the message from a child process is wholly received, it is copied to the messages buffer. The design of this part stays the same as in 7.3.5. The content of the messages buffer is again serialized by MessagePack and sent to Proxy in regular time intervals or when it is full.

9.1.4 Error Handling and Proper Teardown

When a handled error occurred, any process only exited with an error return code in the original implementation. There were also unhandled errors in master process and Telnet minimal honeypot code such as pipe read and write, ZMQ socket write, TCP [33] socket write errors. As mentioned above, the proper error handling was added to all original parts of the code. It is tightly coupled with a proper teardown process. The proper teardown consists of allocated memory free and closing all possible file descriptors (pipe ends, TCP connections, ZMQ sockets), terminating event loop and exiting with appropriate error code.

9.1.5 Telnet Minimal Honeypot

The functionality on the Telnet protocol level was not changed. In addition to 9.1.4, code refactoring for proper and optimal dynamically allocated memory management was done. All buffers are allocated dynamically, saving some memory when no Telnet minimal honeypot child process is running. Connection inactivity timer with its callback were added to close hanging connections.

9.2 New Minimal Honeypots

The implementation of a new minimal honeypots uses principles from Telnet minimal implementation 7.3.6 together with enhancements introduced in 9.1.5. HTTP 3, FTP 4 and SMTP 5 protocols are implemented by state automatas. Perfect hash function generator [1] is used for generation of hash functions and lookup tables for fast and easy to use recognition of protocol specific keywords e.g. HTTP header names 3.3.2, FTP commands 4.2.1, SMTP commands 5.3.1, SASL mechanisms 5.2.1.

Event loop 7.3.2 and events' callbacks are used to handle incomming TCP [33] connections, signals, timeouts and protocols' state atomatas. New standard callback was for connection incativity timer was introduced with new minimal honeypots. As stated in 9.1.5 it was alaso added in Telnet minimal honeypot. The table 9.1 shows the common callbacks used in all the minimal honeypots. The concept of connection data pool 7.3.6 is encorporated to store

Callback name	Occurance
on_accept	new connection request on listening port
on_read	data arrived from a TCP socket
on_timeout	connection timeout elapsed
on_sigint	SIGINT [2] signal caught

Table 9.1: Standard minimal honeypots' callbacks

[1]https://www.gnu.org/software/gperf/

[2]https://www.man7.org/linux/man-pages/man7/signal.7.html

communication state between read events and to limit number of simultaneous TCP connections. All buffers are allocated dynamically to save memory in case some minimal honeypot is not run. The new minimal honyepots also come with proper error handling and teardown.

Chapter 10
Minipot Testing

Minipot component 7 supposed to be directly exposed to Internet and interacting with any entity who connects to it. Its functionality is complex and it must be preserved in every conditions same well as nonexploitability. To ensure this properties Minipot component was thoroughly tested.

Valgrind Memcheck [1]: a memory detector tool was used to detect memory errors in Minipot component. It ran in Valgrind environment during all the tests execution to reveal and fix all detected memory errors.

10.1 Manual Testing

Obviously, user interaction with Minipot component was tested manually. The interaction means running Minipot executable with various command-line options and arguments. All the minimal honeypot connections timeouts were also tested mmanualy - by simple time of closure observation.

10.1.1 Protocols' Implementations Validation

To validate wether the implementation of HTTP 3, FTP 4 and SMTP 5 protocols authentications meachanisms are correct the Minipot component was tested with various existing protocols client implementations. The various clients were used to connect and interact with implemented servcices emulations - minimal honeypots. The authentication data manualy inserted into the clients' user interfaces was visualy compared with data received from ZMQ 6.1.2 socket.

Telnet - telnet [2],

HTTP - Firefox [3], Vivaldi [4], Chromium [5], curl [6], httpie [7],

[1]https://www.valgrind.org/docs/manual/mc-manual.html
[2]https://linux.die.net/man/1/telnet
[3]https://www.mozilla.org/en-US/firefox/new/
[4]https://vivaldi.com/
[5]https://www.chromium.org/Home
[6]https://curl.haxx.se/
[7]https://httpie.org/

FTP - FileZilla [8], gFTP [9], ftp [10], lftp [11], ncftp [12],

SMTP - curl [6], Swaks [13], smtp-cli [14],

10.2 Integration Testing

As mentioned above, Minipot's functionality is complex. The complexity arises from the fact that there are several simultaneous communications between several components by multiple protocols happening at the same time, as depicted in figure 10.1. The manual testing is not feasible to test all

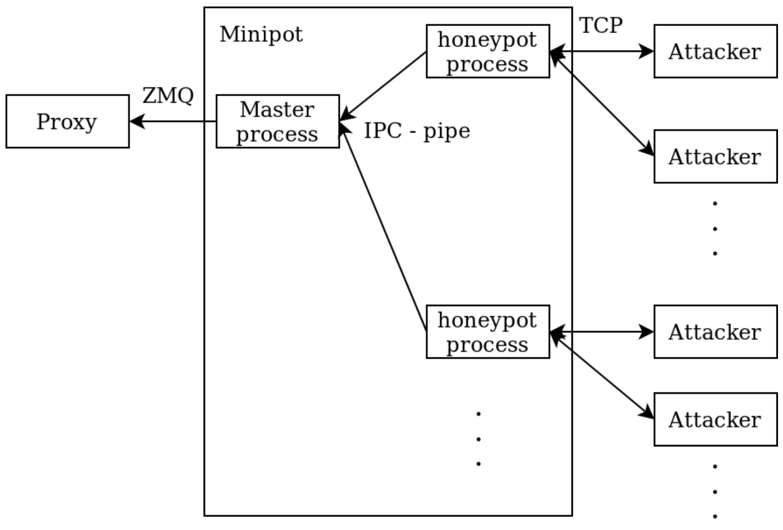


Figure 10.1: Interactions of minipot component

the scenarios and cases which could happen during all the communication. Integration testing serves for this purpose. During the integration testing, the Minipot component is seen as a white box. Its design and implementation details are known. Testing itself is done by stimulation of the box with inputs and capturing its outputs.

Inputs - streams of bytes received from attackers on particular TCP [33] connections

Outputs - streams of bytes sent to attackers on particular TCP connections, data reported to Proxy component

[8] https://filezilla-project.org/index.php
[9] https://www.gftp.org/
[10] https://linux.die.net/man/1/ftp
[11] https://lftp.yar.ru/
[12] https://www.ncftp.com/
[13] http://jetmore.org/john/code/swaks/
[14] http://www.logix.cz/michal/devel/smtp-cli/

From the design specification of Minipot, the exact output for the given input is known. So the real captured outputs can be easily compared with designed, intended outputs.

10.3 Integration Testing Framework

Simple test scripts are not sufficient to test that complex interactions as mentioned in 10.2. The integration test framework was designed for that purpose. The integration tests framework provides a flexible environment for the following tasks:

1. Minipot input generation
2. Minipot intended output generation based on given input
3. Interactions with Minipot
4. Minipot output capture
5. Comparison of real captured output with the generated intended output

10.3.1 Architecture

Based on the interactions overview on figure 10.1 and mentioned functionalities the testing framework architecture depicted on figure 10.2 is designed.

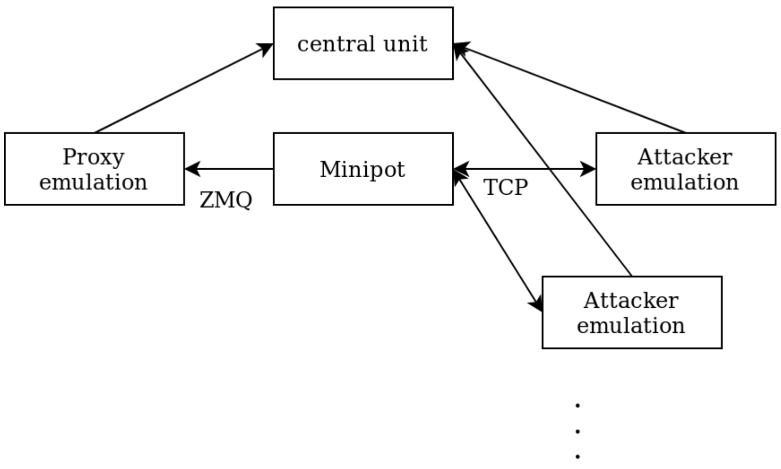


Figure 10.2: Integration tests framework components

Attacker emulation generates *Minipot* input and *Minipot* intended output based on the generated input. It sends generated input to *Minipot* by TCP [33]. It receives and checks the correctness of the received data on a TCP connection. The generated intended *Minipot* output is also sent to *Central unit* for later comparison.

Proxy emulation captures output from Minipot and forwards it to *Central unit.*

Central unit receives captured Minipot output from Proxy emulation and intended generated output from *Attacker* emulation. The both outputs are then compared.

10.3.2 Evaluation Process

The communication of Minipot and Attacker emulation is validated in realtime by the attacker emulation component. If the response received from Minipot is not correct, the testing is immediately terminated with the negative result. The Standard sentinel messages generated by the minimal honeypot are buffered and send in batches in regular time intervals. 7.3.5 So it would be nearly impossible to check it in realtime feasibly. The Minipot proxy communication is checked after all the interactions between Minipot and Attacker emulation components if the Minipot passes it. If there is any message mismatch, the Minipot failed the test.

10.3.3 Implementation

The framework is implemented in Python language, because of its easy-to-use nature.

Attacker Emulation

The Attacker emulation component implementation is not universal, like, e.g., minimal honeypot one. Implement fully compatible and comprehensive attacker emulation for testing purposes would be too complicated, time-consuming, and error-prone as the implementation of the minimal honeypots for which testing this component is intended. Attacker emulation is instead implemented in a manner of simple script consisting of code primitives like send, receive a message, compare two messages, generate Sentinel message returning generated messages, and taking Internet endpoint as input. The code primitives have a standard interface and are used in the whole framework. This approach allows for creating various communications testings scenarios simply and straightforwardly. The attacker emulation scripts are implementations of the test cases.

Proxy Emulation

It collects data incoming from the ZMQ socket. The Minipot master process sends data at regular intervals; thus, polling with a slightly longer interval is used to capture data from the socket.

Central Unit

The central unit is implemented as a class with a single method run. Thread pool is used to run all testing scripts, emulating an attacker on given Internet endpoints and Proxy emulation concurrently. After all the threads finish their jobs, generated and captured Sentinel messages are retrieved from them for the comparison.

Chapter 11

Server Components Enhancements

All the server components are implemented in Python programming language using the Sentinel Network library. It is internal, a non-public library providing API for smooth implementation of Sentinel server data processing components It solves ZMQ networking, common configuration framework, logging in a standardized way, message queue for handling messages in a safe and good-performing way. Thus a software developer can focus on data processing functionality itself of a particular component. Practically, each of the data processing components is implemented by inheriting from one of the classes provided by the Sentinel Network library. Each class has a predefined interface for its intended usage. Sanity check, Dumper, and database were extended to process and store new types of Sentinel messages generated by new minimal honeypots.

11.1 Sanity Check

It checks if received and unpacked Sentinel messages have the same structure, which is described by YAML [1] validation schema. Sanity check itself didn't need to be extended. Definitions of new Sentinel messages structures were added to the YAML validation schema.

11.2 Dumper

Dumper is the very last component of the server data processing pipeline. It extracts payload data from received Sentinel messages and stores them into PostgreSQL [2] database. Psycopg [3] - the most popular PostgreSQL adapter for Python programming language is utilized as an interface between Python and PostgreSQL. Each *type* of Sentinel message is handled by a dedicated handler, which is implemented as class inheriting from the general handler class providing access to the database. New handlers for new Sentinel

[1]https://yaml.org/
[2]https://www.postgresql.org/
[3]https://www.psycopg.org/

messages types were implemented by inheriting from general handler class and defining SQL queries and their data.

11.3 Database

Data from Sentinel messages generated by minimal honeypots are stored in PostgreSQL object-relational database system. Data in relational database management systems are stored in tables. New tables for the new type of data from new minimal honeypots were added. The design of the new tables originates from the design of the table for Telnet minimal honeypot data. There is one-to-one mapping from Sentinel messages fields 6.2 to database tables columns. Additional *identity_id* attribute serves for purposes

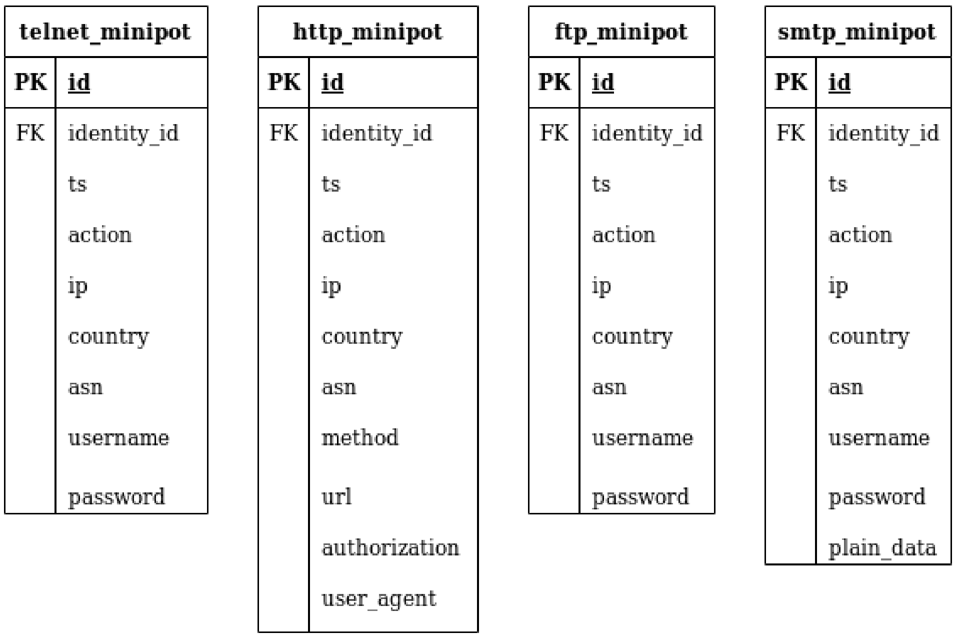


Figure 11.1: ERD for minimal honeypots data

of particular Turris router device identification and authorization within Sentinel system. Its design is not relevant for this thesis, and thus it is omitted. There are no *country* and *asn* fields defined in 6.2 nor gerated by minimal honepots 8.3 8.4 8.5. These fields are added to passing Sentinel messages by Enrichments component, as described in 6.3.

Part IV

Data Analyses

Chapter 12
Base statistics

In this chapter, the basic statistics and observations of collected data are presented. They serve as a starting point for later attackers and groups of attackers identification in chapter 13. The key idea for a finding of attackers groups is to perform various unsupervised machine clustering algorithms on authentication data used by each attacker. An IP address is considered a unique identifier of an attacker. Although the authentication data were captured by emulating different authentication mechanisms in various protocols, in principle, they are composed of username and password. Particular types of data collected by each minimal honepot can be found in 8.3, 8.4, 8.5 and 7.3.6.

The analyzed data are captured single events expressing some action made by an attacker to a minimal honeypot. How the data are stored is stated in 11.3 All the statistics and following data analyses are performed on the data set of 5 million events collected by HTTP , FTP, SMTP, and Telnet minimal honeypots deployed on 48 Turris routers in range of 9 days from 31st of July 2020 to 8th of August 2020. Section 12.1 gives general events and their counts and types of overview. In section 12.2 various IP addresses - attackers related statistics are listed. Section 12.3 presents statistics of usernames and passwords used by attackers.

12.1 Events

Action	**HTTP**	**FTP**	**SMTP**	**Telnet**	**Action sum**
connect	298 982	307 504	2 028 785	481 154	3 116 425
message	635 224	-	-	-	635 224
login	-	279 070	1 361 924	362 551	2 003 545
plain	-	-	0	-	0
Protocol sum	934 206	586 574	3 390 709	843 705	5 755 194

Table 12.1: Events types, counts and protocols summary

From table 12.1 is visible that more than half of the total number of captured events was made through SMTP protocol. Counts of events captured by

HTTP, FTP, and Telnet minimal honeypots are in order lower and relatively close to each other. HTTP is the second, and Telnet is the third, and FTP is the last in the comparison of the number of the events. More than half of the total events count are *connect* events. This also applies to FTP, SMTP, and Telnet. Only HTTP has twice more *message* events than *connect*. It indicates that a significant number of IP addresses establish a minimal honeypot connection, but don't do any further actions. These are mostly open ports scans and, in a small amount, misleading user connections. The interesting fact is no occurrence of SMTP *plain* events which corresponds to login attempt made by SASL Plain authentication mechanism [32]. All login attempts made to SMTP minimal honeypots were made through already obsolete (according to [35]) SASL Login authentication mechanism [34].

12.2 IP addresses - attackers

	HTTP	**FTP**	**SMTP**	**Telnet**	**All**
with connect actions	30 412	1 635	922	51 716	78 471
without connect actions	29 271	985	75	11 350	39 627
Percentage count drop	4%	40%	92%	78%	50%

Table 12.2: Unique IP addresses counts summary

Number of protocols	1	2	3
IP adresses count	37574	2052	1

Table 12.3: IP addresses envolvements in protocols

From table 12.2 we can see significant differences of unique IP addresses counts with and without counting connections events. It confirms the theory from the previous section 12.1 that a lot of IP addresses only connect to minimal honeypots without any further action. This applies especially to FTP, SMTP, and Telnet protocols where the drops are significant. The most significant percentage drop in the unique IPs is SMTP. It means that the largest amount of login attempts seen in table 12.1 (more than 1 million) is made by a minimal amount of IP addresses. Most probably, those are email spambots. The smallest drop of IP addresses is in HTTP.

From figure 12.1 is visible that even after filtering out connection events most of the captured IP addresses made only a small amount of login attempts. And only a very few of them did hundreds of thousands of the events. Table 12.3 shows that most of the IP addresses interacted with only one type of minimal honeypot.

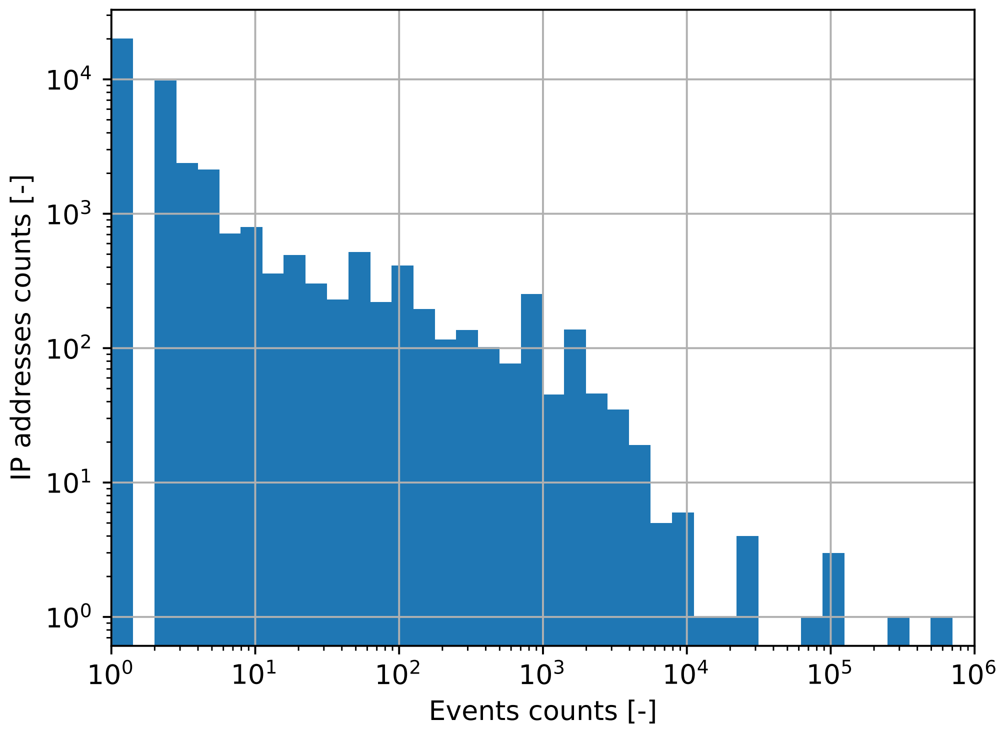


Figure 12.1: Histogram of IP addresses records counts without connect actions

12.3 Usernames and Passwords

Protocols	HTTP	FTP	SMTP	Telnet	All
Usernames counts	51	30	7946	176	8122
Passwords counts	290	480	11 579	868	12872

Table 12.4: Unique usernames and passwords counts in protocols

Number of protocols	1	2	3	4
Usernames count	8040	44	10	6
Passwords count	12550	207	32	25

Table 12.5: Usage of usernames and passwords in protocols

Based on login events counts in table 12.1 and counts of unique usernames and passwords in table 12.4 we can deduce that usernames and passwords used in login attempts are heavily reused within one protocol. But counts of unique passwords in each particular protocol and in general are higher than counts of unique usernames. Most of the usernames and passwords are used only in one particular protocol, according to table 12.5. Figures 12.2 and 12.3 shows that vast majority of usernames and passwords are used in

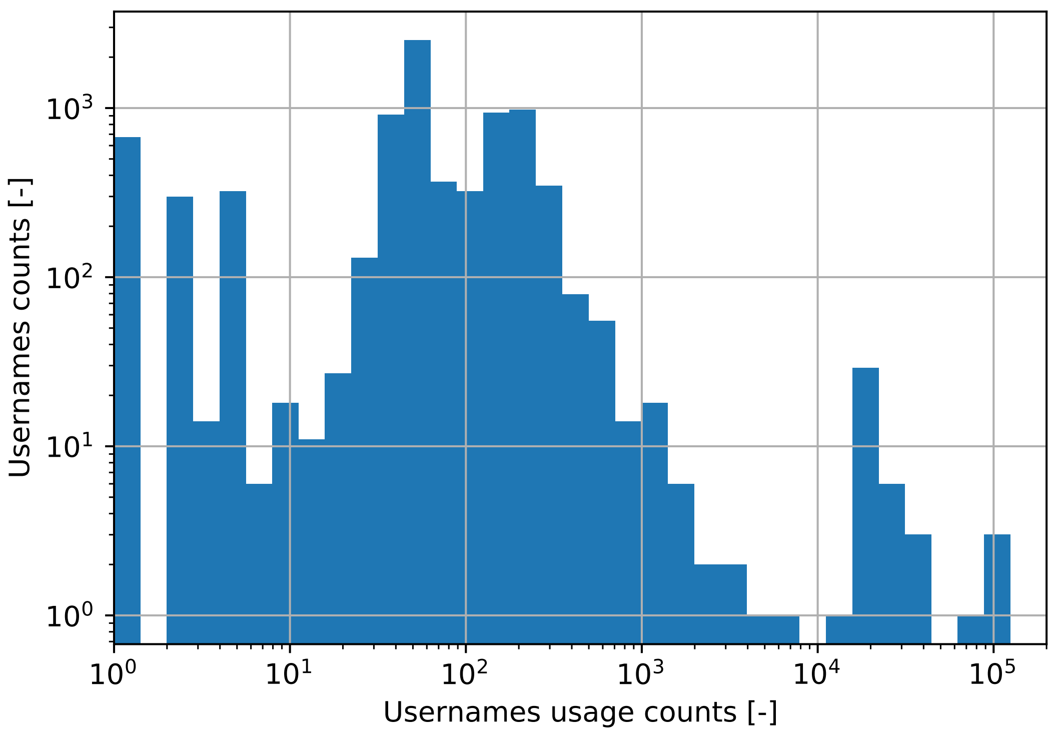


Figure 12.2: Histogram of usernames usages

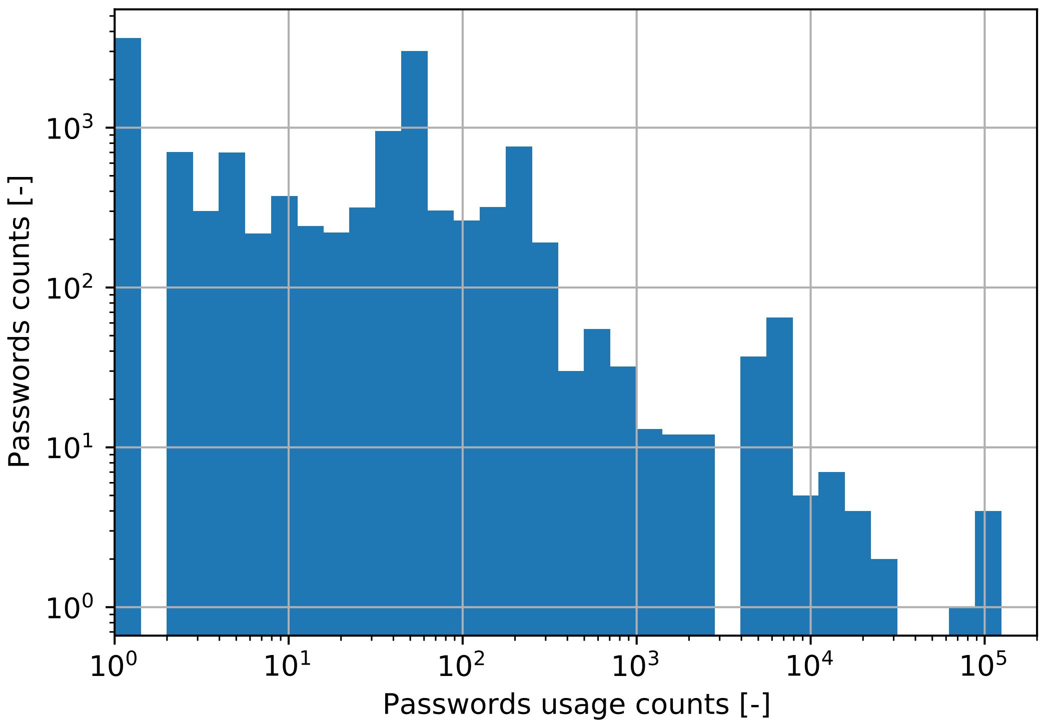


Figure 12.3: Histogram of passwords usages

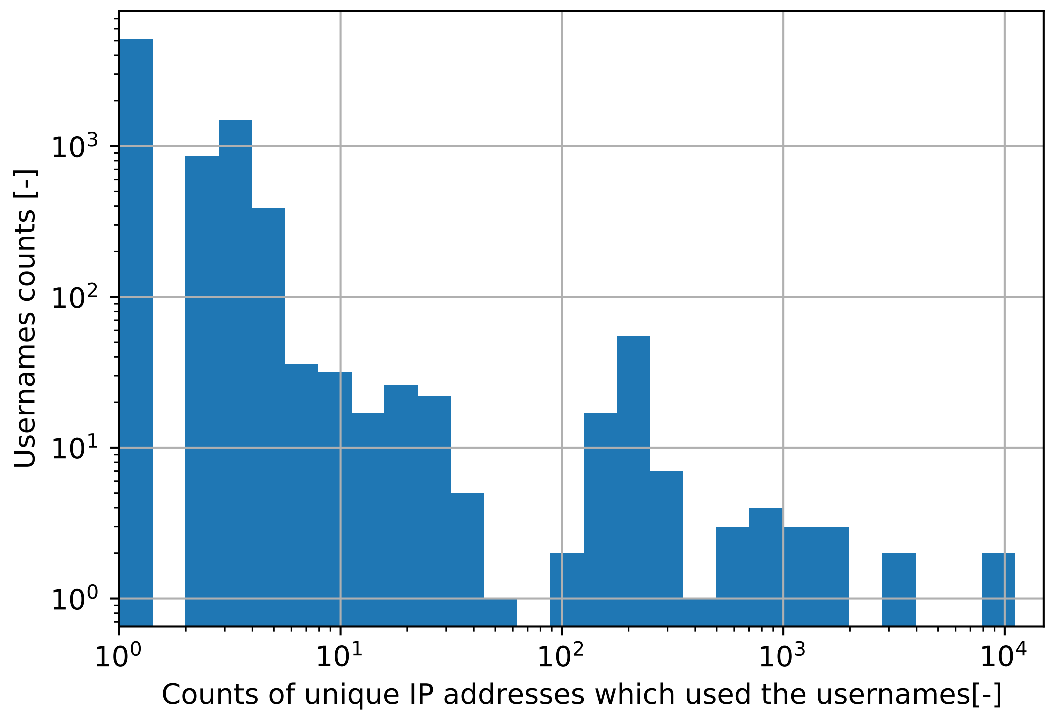


Figure 12.4: Histogram of usernames usages by unique IP addresses

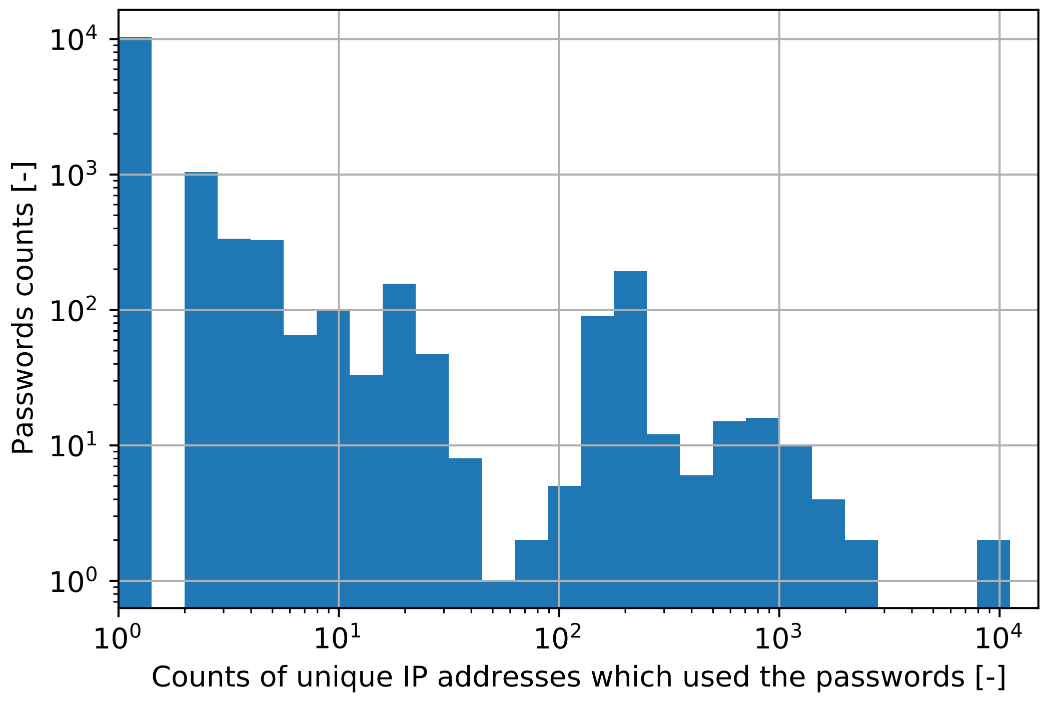


Figure 12.5: Histogram of passwords usages across unique IP addresses

orders odf units, tens and hundreds. Higher usage is rare. In comparison of figures 12.2 and 12.3 we can observe that passwords are in general reused less than usernames. However, the difference is not very big. It is because there are more unique passwords than usernames. Figures 12.4 and 12.5 depict very similar trends in usage of the usernames and passwords by unique IP addresses. IP addresses use most of them in counts in orders of units or tens.

Chapter 13

Attacker and Attackers Groups Identification

The main goal why the methods for attacker and especially attackers groups identification are elaborated is their possible application in Distributed adaptive firewall introduced in chapter 1 for recognition of new IP addresses as attackers.

13.1 Attacker Identification

Interactions with a honeypot should be a reliable indication that given IP is having malicious behavior as stated in 2.2.1. But as shown in table 12.2, approximately half of the IP addresses did only connectin attempts without further action, which is not malicious, but somewhat suspicious. More a given IP address has connection events captured, the more it is suspicious. An interesting fact about the IPs which has only connection events in the database could be whether they have one or a small number of connections captured from a higher number of devices where minimal honeypots are deployed or whether they have a higher number of connections done on one or a small amount of minimal honeypots. But the connection attempts are not inspected further than in 12.1 and 12.2, because it is not in the focus of this thesis. The small number of connections on the higher number of devices would most probably mean network scans. The higher amount of connection on the small number of devices would be more suspicious. In reality, some of the connections established with minimal honeypots are just mistakes when someone could, for example, mistype IP address of some server.

With login events, it is more probable that a given IP address is malicious. But again is not that simple state that IP address is bad because of single or a few logins attempts made to minimal honeypots. In this case, it would be hard to say if it is an attack attempt or whether some user accidentally connected to a minimal honeypot and tried to log in thinking he is connected to some real service. Unfortunately, from figure 12.1 is visible that significant part of IP addresses has a single or units of login attempts, whose classification as an attacker or mislead user is hard. An idea for recognition of misleading users from attackers is to use methods for attackers group identification, which are together with the idea discussed in the next chapter 13.2. If an IP address has tens or even more login attempts, we can certainly say that the IP address is malicious.

13.2 Attackers Groups Identification

For finding groups of attackers some property which is common to all attackers but which can also uniquely represents a single attacker is needed. The authentication data consisting of username and password were selected for this purpose. In 12.3 is observed that there is significant inequality of number of unique usernames and the number of unique IP addresses and that the usernames are thus heavily reused. Used usernames are thus not very good measure that more IP addresses have something in common. With passwords, it is the same as with the usernames. But there are more unique passwords than usernames. IPs thus less reuse passwords than the usernames. The first strategy for finding attackers groups is to group them by a set of unique passwords that a given IP address used. The second strategy is to use both sets of unique usernames and passwords for finding groups of attackers.

Since the counts and the sizes of attackers groups are not known before, and there is no training labeled dataset, unsupervised machine learning clustering algorithms are utilized for the grouping. Attackers clustering by authentication data can also be used to determine whether an IP address with a few login attempts is malicious or whether it is a misleading user. There is a significant amount of IP addresses with only a few login attempts, according to figure 12.1. If an IP address with a small amount of logins is grouped to a group of other attackers by a clustering algorithm, it is more probably an attacker than a misleading user. But of course, if a misleading user uses the same login credentials as those used by attackers, there is no chance to distinguish the user from an attacker neither by this approach. The next chapter 14 contains all related to the clustering, such as particular algorithms selection, overview, and testing.

Chapter 14
Clustering Algorithms

14.1 Introduction

Clustering is an unsupervised machine learning task involving automatical discovery of natural groupings in the feature space of input data. A cluster is considered as an area in a feature space where points from a given dataset are closer to each other than to other points in the dataset. It is hard to evaluate the quality of an output of any clustering method, because, in comparison with supervised learning, it only interprets the input data. So it is hard to determine what is good and what is a bad result. Evaluation of identified clusters depends on a concrete dataset and may require an expert from a domain from which the data came.

There are many clustering algorithms with different approaches and no single one is the best method for all datasets. Thus there is no easy way to select the best algorithm correctly for a given dataset without controlled experiments.

Many algorithms use similarity or distance measures between data samples in a feature, space to discover dense regions - clusters. Some of the algorithms require to specify an estimated number of clusters for the discovery. Other ones may need a specification on some minimum distance or similarity in which data points are considered close enough to be in the same cluster. In general, cluster analysis is an iterative process where subjective clusters evaluation is feedback for changes of particular clustering algorithm parameters until an appropriate result is achieved. [3] [2]

14.2 Attackers Clustering Algorithms Selection

Based on strategies from 13.2 for attackers groups identification and clustering algorithms introduction 14.1 we can select particular algorithms for basic testing and evaluation and for future fine-tunning. Since our data points are the single IP address and their features are sets of unique passwords and usernames (in general sets of text strings), which was used during login attempts made to minimal honeypots, it is not feasible to project the data points into the feature space.

But finding distances or similarities between data points represented by sets of strings is doable. How to find a distance or similarity between two sets of strings is discussed in 14.3. The number of clusters, in this case, unknown and due to the size of the dataset, it is also nearly impossible to estimate it in a reasonable scope. In summary, the algorithms not requiring cluster count estimation and taking as input distances and similarities between data points are needed.

DBSCAN [10], Hierarchical Agglomerative Clustering, Affinity Propagation [11] and Spectral Clustering [9] algorithms were selected for the basic testing and evaluation whether they are suitable for the attackers' group identification and possibly for later fine-tuning and optimization, because they satisfy the constraints mentioned above. They represent, in principle, totally different approaches for the task of the clustering. The brief theoretical overviews, introducing the basic concepts of the algorithms are in 14.2.1, 14.2.2, 14.2.3, 14.2.4.

Scikit-learn [1] - full-featured open-source machine learning library for the Python programming language was chosen as an implementation of the selected clustering algorithms. It provides easy-to-use, consistent, well-documented API. It is in active development since 2007, currently maintained by a group of volunteers. [8]

14.2.1 DBSCAN

DBSCAN - Density-Based Spatial Clustering of Applications with Noise algorithm is based on the density notion of clusters. Clusters are areas of high density separated by regions of low density. It is suitable for a dataset, which contains clusters of similar density. For each data point of a cluster the neighborhood of a given radius has to contain at least a minimum number of points - the density in the neighborhood has to exceed some threshold. The choice of a distance function for two data points defines the shape of a neighborhood.

The algorithm starts by finding core samples of high density and then expands clusters from them. A cluster is a set of core samples built recursively by taking a core sample and finding all of its neighbors that are core samples. A cluster also has non-core samples, which are samples in a neighborhood of core samples in a cluster but are not themselves core samples. Any data point that is not a core sample, and is at least in the minimum required distance from any core sample, is considered an outlier by the algorithm. Clusters found by DBSCAN can have an arbitrary shape.

There are two main parameters of the algorithm, a minimum number of points in a neighborhood and a minimum distance of two points to be considered neighbors. Higher the minimum number of points or lower the distance, the indicate higher the density necessary to form a cluster. [10] [46]

[1]https://scikit-learn.org

14.2.2 Hiearchical Agglomerative Clustering

Hierarchical Agglomerative Clustering is a process of recursive merging of a pair of clusters that has a minimal linkage distance. It builds a binary merge tree, starting from each data point stored at leaves of the tree and successively merges two by two, the closest clusters until a root of the tree is reached. The root of the tree is one cluster that gathered all the samples, the leaves are clusters containing only a single data point. The representation of the binary merge tree is called a dendrogram. The linkage distance is a distance measure of two clusters. Most common methods to compute the linkage distance are:

Single Linkage takes the distance of two of the closest data points in the clusters.

Complete Linakge takes the distance of two of the furthest data points in the clusters.

Average Linkage computes the average distance of the sets from distances of all pairs of data points from the sets.

[4] [46]

14.2.3 Affinity Propagation

Affinity Propagation algorithm creates clusters based on the iterative exchange of messages between pairs of data points. The given input data set is represented as a network, where each sample is a node of the network. The messages sent between the data samples - the network nodes, are suitabilities for one data point - node to be the exemplar of the other node. The appropriate examplar samples are thus chosen on information gained about other prospective, representative data samples. The exchanges are done until a convergence, at which point the final exemplars represent the data clusters. The algorithm's important parameter is the damping factor, which determines how much the messages are communications dumped to avoid numerical oscillations.

Affinity propagation has the potential to achieve good results, as it chooses the clusters based on the input data in comparison with other methods, where initial representatives are selected randomly and then refined. A drawback is that the algorithm's time complexity is given by a number of data points and by a number of iterations. [46] [5]

14.2.4 Spectral Clustering

Spectral clustering is in a general class of clustering methods based on linear algebra theory. First, low-dimension embedding of a similarity matrix between data points is performed. Secondly, clustering, e.g., KMeans, of the eigenvectors in the low dimensional space follows. There are many ways how to derive the clusters based on which exactly eigenvectors are used. Spectral clustering groups data that are connected but not necessarily in convex space or a compact cluster. It achieves good results when the clusters have a highly non-convex structure and when a center and spread of cluster is not a suitable description. [46] [9]

14.3 Strings Sets Similarity and Distance Metrics

The presented similarity metrics are needed for computing an input data - similarity or distance matrix of clustering algorithms discussed in 14.2. Only the similarity metrics are discussed. Similarity expresses in this context how much are two sets of strings similar according to the contained strings.

Similarity of two sets of strings S_1 and S_2 denoted as $s_{12}[-]$ is for this usecase defined in interval $< 0, 1 >$. Is $s_{12} = 1$, S_1 and S_2 are considered to be identical or very similar. If $s_{12} = 0$, S_1 and S_2 are considered as totally different or have nothing in common. The distance $d[-]$ is obatined from similarity $s[-]$ as $d = 1 - s$.

14.3.1 Overlap Coefficient

Overlap coefficient also called as Szymkiewicz-Simpson coefficient is defined by following formula: $overlap(X, Y)[-] = \frac{|X \cap Y|}{min(|X|, |Y|)}$, where X and Y represents in general sets of some objects. In this case the objects are text strings representing usernames and passwords used when interacting with minimal honeypots. The coefficient measures overlap between the two sets. It considers the two sets as a full match if one is a subset of another. [7]

14.3.2 Levenhstein Library *setratio* Function

Function *setratio* from Levenshtein library [2] was found to be a suitable implementation of similarity measure of two strings sets. The similarity is determined based on the best match string from the given sets [47].

[2]https://github.com/ztane/python-Levenshtein/

Chapter 15
Conclusion

15.1 Sentinel Enhancements

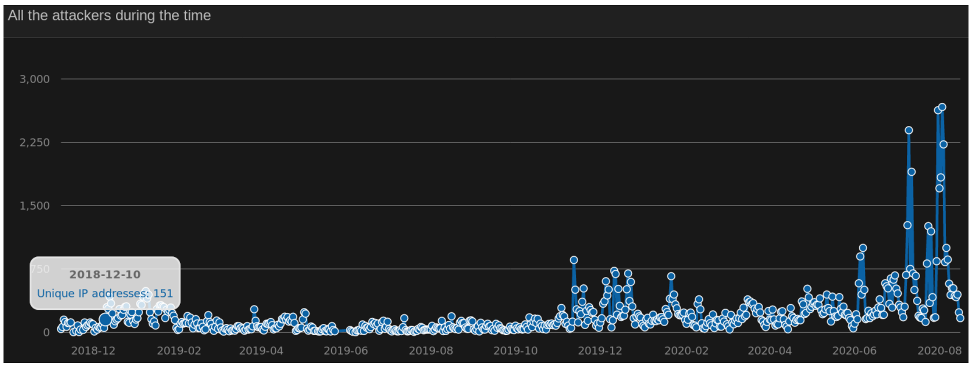
HTTP, FTP, and SMTP minimal honeypots were added to the Sentinel data collecting system running on Turris routers. The minimal honeypots design 8 was based on analyses of real servers' implementations behavior rather than on particular application protocol to mitigate possibilities of the minimal honeypots fingerprinting 2.4.

Minipot - Sentinel component was redesigned and wholly refactored 9 to accommodate the implementation of the new minimal honeypots. Minipot component was also thoroughly tested 10 to validate the correct functionalities and implementation of the new minimal honeypots. Integration testing framework 10.2 was designed and implemented for the purpose of Minipot component validation and testing. The new minimal honeypots were also tested manually with various client implementations to validate particular protocol compatibility 10.1.1. During the tests, Minipot component run in Valgrind Memcheck [1] environment to detect any possible memory handling flaws.

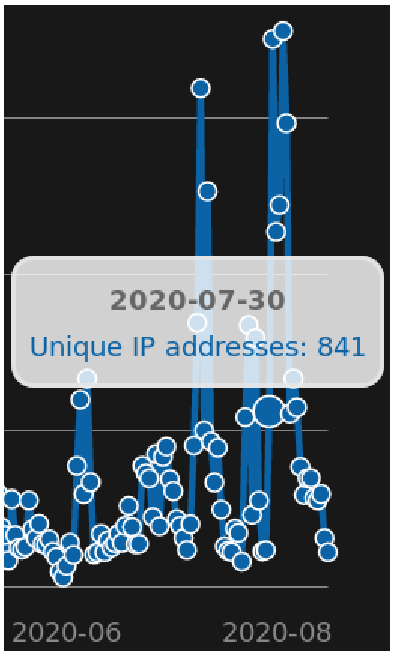
Sanity Check, Dumper and database - Sentinel server components were enhanced 11 to be able to process and store data from new minimal honeypots.

Five million events were captured by new minimal honeypots deployed on 48 Turris routers in dates from 31st of July to the 8th of August. Which is a reliable indication of correct minimal honeypot design and implementation as well as proper Sentinel server components enhancements? Immediately after the new minimal honeypots deployment on 31st of July 2020 the number of IP addresses in the Distributed adaptive firewall [2] tagged as attackers and automatically blocked on all Turris devices was more than tripled. Screenshots of the list of blocked IPs - Greylist visulalisation [3] are in figure 15.1. It clearly shows the positive direct impact of new minimal honeypots on the security of thousands of Turris routers users.

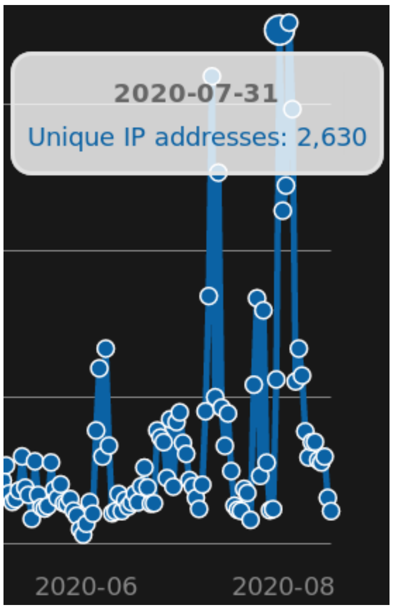
[1]https://www.valgrind.org/docs/manual/mc-manual.html
[2]https://project.turris.cz/en/security
[3]https://view.sentinel.turris.cz/



(a) : Overall



(b) : 30th of July - before the new minimal honepots deployment



(c) : 31st of July - after the new minimal honeypot deplyement

Figure 15.1: Screenshots of Greylist visualization

15.2 Data Analyses

The high volume of the data collected by the newly implemented and deployed minimal honeypots 12.1 in that short period of nine days is surprising. Many interesting facts come out from the initial statistics, which are discussed in 12. Identification of an IP address, which has tens or more occurrences in the database as an attacker, as an attacker, is quite reliable. With an IP address having only units of occurrences in the database is hard to classify as an attacker or a misleading user. The promising way how to estimate whether an IP address with the low number of captured events is an attacker or mislead user is to try whether it will fit a group of IP addresses known as attackers 13.2. If an IP address with uncertain classification is grouped to a group of attackers, it is more probably an attacker than a misleading user. Methods for grouping IP addresses based on used authentication data was established in 13.

According to these methods, constraints for algorithms used to reveal clusters of attackers in the data set were defined 14.2. The clustering algorithms may not require a number of clusters as a parameter and they also have to accept precomputed similarities or distances, because in this case is not feasible to project the data in feature space. Various metrics for computing the distance or the similarity of data points (IP address) based on sets of strings (used authentication data) were researched. Two such metrics were selected for this use case 14.3. Based on the constraints, four unsupervised machine learning clustering algorithms with different approaches - DBSCAN 14.2.1, Afinite Propagation 14.2.3, Spectral14.2.4 and Hierarchical Agglomerative 14.2.2 clustering were studied and selected as the starting point for the groups of attackers analyses.

Because the release of Turris OS [4] version 5.1 which involved a lot of changes and new features including new minimal honeypots needed more time than originally planned, the data were available only recently, and there was only minimal time for testing and following the finetuning of clustering algorithms on real data. The algorithms were run on the dataset with their initial parameters, the results seem to be promising on the first look.

15.3 Future Work

The high volume of the data collected during the units of days 12, is going certainly require their aggregation. In such a case, the appropriate aggregation methods will need to be designed and implemented. Since there was no time for clustering algorithms 14 testing and finetuning, it is going to be done in the near future. If after the clustering algorithms optimizations for attackers clustering and outliers (mislead users) detection, is going to give decent results, it will be used for enhancing Distributed adaptive firewall [2] functionality.

[4]https://project.turris.cz/en/software

Bibliography

[1] Pathan, A. K. (2016). State of the art in intrusion prevention and detection. CRC Press. ISBN 9781138033986

[2] Witten, I. H., Frank, E., Hall, M. A., Pal, C. J. (2017). Data mining: Practical machine learning tools and techniques. Morgan Kaufmann. Fourth Edition. Elsevier. ISBN: 978-0-12-804291-5

[3] Murphy, K. P. (2013). Machine learning: A probabilistic perspective. Cambridge, MA: MIT Press. ISBN 978-0-262-01802-9

[4] Nielsen, F. (2016). Introduction to HPC with MPI for data science. Cham; Heidelberg: Springer. ISBN: 978-3-319-21902-8

[5] Frey, Brendan and Dueck, Delbert. (2007). Clustering by Passing Messages Between Data Points. Science (New York, N.Y.). 315. 972-6. DOI: 10.1126/science.1136800.

[6] Ng, Andrew and Jordan, Michael and Weiss, Yair. (2002). On Spectral Clustering: Analysis and an algorithm. Adv. Neural Inf. Process. Syst. 14.

[7] M K, Vijaymeena and K, Kavitha. (2016). A Survey on Similarity Measures in Text Mining. Machine Learning and Applications: An International Journal. 3. 19-28. DOI 10.5121/mlaij.2016.3103.

[8] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res. 12, (2/1/2011), 2825–2830. DOI 10.5555/1953048.2078195

[9] Ng, Andrew and Jordan, Michael and Weiss, Yair. (2002). On Spectral Clustering: Analysis and an algorithm. Adv. Neural Inf. Process. Syst. 14.

[10] Ester, M, Kriegel, H P, Sander, J, and Xiaowei, Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. United States: N. p., 1996.

[11] Brendan J. Frey, Delbert Dueck, Clustering by Passing Messages Between Data Points Science 16 Feb 2007: Vol. 315, Issue 5814, pp. 972-976 DOI: 10.1126/science.1136800

[12] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, https://www.rfc-editor.org/info/rfc7230

[13] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, https://www.rfc-editor.org/info/rfc7231

[14] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, https://www.rfc-editor.org/info/rfc7235

[15] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, https://www.rfc-editor.org/info/rfc7617

[16] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, October 1985, https://www.rfc-editor.org/info/std9

[17] Klensin, J. and A. Hoenes, "FTP Command and Extension Registry", RFC 5797, DOI 10.17487/RFC5797, March 2010, https://www.rfc-editor.org/info/rfc5797

[18] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, DOI 10.17487/RFC0854, May 1983, https://www.rfc-editor.org/info/rfc854

[19] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, https://www.rfc-editor.org/info/rfc5321

[20] Melnikov, A., Ed., and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, https://www.rfc-editor.org/info/rfc4422

[21] Siemborski, R., Ed., and A. Melnikov, Ed., "SMTP Service Extension for Authentication", RFC 4954, DOI 10.17487/RFC4954, July 2007, https://www.rfc-editor.org/info/rfc4954

[22] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, https://www.rfc-editor.org/info/rfc4648

[23] Zeilenga, K., Ed., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism", RFC 4616, DOI 10.17487/RFC4616, August 2006, https://www.rfc-editor.org/info/rfc4616

[24] Hoffman, P., "SMTP Service Extension for Secure SMTP over Transport Layer Security", RFC 3207, DOI 10.17487/RFC3207, February 2002, https://www.rfc-editor.org/info/rfc3207

[25] Gellens, R. and J. Klensin, "Message Submission for Mail", STD 72, RFC 6409, November 2011, https://www.rfc-editor.org/info/std72

[26] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, https://www.rfc-editor.org/info/rfc1035

[27] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, DOI 10.17487/RFC1939, May 1996, https://www.rfc-editor.org/info/rfc1939

[28] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, https://www.rfc-editor.org/info/rfc3501

[29] Hutzler, C., Crocker, D., Resnick, P., Allman, E., and T. Finch, "Email Submission Operations: Access and Accountability Requirements", BCP 134, RFC 5068, DOI 10.17487/RFC5068, November 2007, https://www.rfc-editor.org/info/rfc5068

[30] Moore, K. and C. Newman, "Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access", RFC 8314, DOI 10.17487/RFC8314, January 2018, https://www.rfc-editor.org/info/rfc8314

[31] De Winter, J., "SMTP Service Extension for Remote Message Queue Starting", RFC 1985, DOI 10.17487/RFC1985, August 1996, https://www.rfc-editor.org/info/rfc1985

[32] Zeilenga, K., Ed., "The PLAIN Simple Authentication and Security Layer (SASL) Mechanism", RFC 4616, DOI 10.17487/RFC4616, August 2006, https://www.rfc-editor.org/info/rfc4616.

[33] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, https://www.rfc-editor.org/info/std7.

[34] Murchison, K. Crispin, M., "The LOGIN SASL Mechanism", Internet-Draft, draft-murchison-sasl-login-00, August 2003, https://datatracker.ietf.org/doc/html/draft-murchison-sasl-login-00

[35] Simple Authentication and Security Layer (SASL) Mechanisms. (2015, November 02). Retrieved August 14, 2020, from https://www.iana.org/assignments/sasl-mechanisms/sasl-mechanisms.xhtml

[36] Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry. (2017, November 29). Retrieved August 14, 2020, from https://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml

[37] Message Headers. (2020, July 13). Retrieved August 14, 2020, from `https://www.iana.org/assignments/message-headers/message-headers.xhtml`

[38] World Wide Web Technology Surveys. (n.d.). Retrieved August 14, 2020, from `https://w3techs.com/`

[39] Mail (MX) Server Survey. (2020, May 1). Retrieved May 6, 2020, from `http://www.securityspace.com/s_survey/data/man.202004/mxsurvey.html`

[40] What is Entity Relationship Diagram (ERD)? (n.d.). Retrieved August 14, 2020, from `https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/`

[41] Tutorialspoint. (n.d.). SQL - RDBMS Concepts. Retrieved August 14, 2020, from `https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm`

[42] Hrušecký, M. (2020, January 22). Protecting your servers with Turris Sentinel. Retrieved August 14, 2020, from `https://en.blog.nic.cz/2020/01/22/protecting-your-servers-with-turris-sentinel/`

[43] CZ.NIC z.s.p.o. (2019, November 11). Sentinel. Retrieved August 14, 2020, from `https://docs.turris.cz/basics/apps/sentinel/`

[44] MQTT FAQ, What is MQTT? (n.d.). Retrieved August 14, 2020, from `https://mqtt.org/faq`

[45] MQTT man page. (2018, September 20). Retrieved August 14, 2020, from `http://mosquitto.org/man/mqtt-7.html`

[46] Clustering. (n.d.). Retrieved August 14, 2020, from `https://scikit-learn.org/stable/modules/clustering.html`

[47] Python-Levenshtein doc. (n.d.). Retrieved August 14, 2020, from `https://rawgit.com/ztane/python-Levenshtein/master/docs/Levenshtein.html`

[48] CZ.NIC z.s.p.o. (2020, July). Minipot doc. Retrieved August 14, 2020, from `https://gitlab.nic.cz/turris/sentinel/minipot/-/tree/master/doc`

[49] Obůrka, R. Turris:Sentinel Nový systém pro sběr dat, Přednáška, Konference Internet a Technologie, November 16, 2018. Available at `https://www.nic.cz/files/nic/IT_18/prezentace/22_it18_oburka.pdf`