Technische Hochschule Ingolstadt
Faculty of Computer Science
Bachelor's course Computer Science

Malware Beaconing Detection with Jupyter Notebooks

Bachelor Thesis

Name and Surname	Jenny Hofbauer
Issued on	October 13, 2022
Submitted on	January 16, 2023
First examiner	Prof. Dr.-Ing. Hans-Joachim Hof
Second examiner	Prof. Dr. Michael Jarschel
Supervisor	Dominik Schaudel, M.Sc.

Abstract

In recent years the spread of ransomware and new malware variants has made the cyber security threat landscape more dangerous. Malware beaconing is a common tactic used by hackers to maintain a connection with a compromised system, send new commands and exfiltrate data. This thesis proposes a method for detecting malware beaconing in security-relevant log information using Jupyter Notebook in a corporate network.

The approach involves analyzing network traffic data for patterns that are indicative of beaconing activity. To do this, signature and periodic-based detections are utilized, as well as visualizing and enriching the detected connections to give an analyst all the information needed to make a quick and informed decision. A working prototype is implemented in Jupyter Notebook with requirements and restrictions based on the needs and infrastructure of a real-life Security Operation Center. The most significant limitation is the restricted network connectivity of the analyst's toolset.

The effectiveness of the approach is evaluated using real-world and simulated data to demonstrate the potential for detecting malware beaconing in a realistic scenario.

Overall, this work provides a practical and effective method for detecting malware beaconing and gives a glimpse into the potential of analyzing, hunting and detecting cyber threats with Jupyter Notebook.

Declaration

I hereby declare that this thesis is my own work, I have not presented it elsewhere for examination purposes and I have not used any sources or aids other than those stated. I have marked verbatim and indirect quotations as such.

Ingolstadt, January 16, 2023

Jenny Hofbauer

Acknowledgements

This thesis concludes my dual study program at the Technische Hochschule Ingolstadt and MBDA Deutschland GmbH.

Special thanks to my coworkers for 3.5 years of exciting projects, technical and emotional support and the possibility to expand my IT security knowledge. A particular mention to my supervisor Dominik Schaudel for giving me the freedom and guidance to work on this interesting and state-of-the-art topic.

I want to thank Prof. Dr.-Ing. Hans-Joachim Hof for accompanying this thesis, answering all of my questions and giving me insights along the way.

Furthermore, I would like to thank Bernhard Büttner, Martin Endres, Martin Kleehaus, Andreas Badenbach, Thomas Tschikste and Taner Kampa for proofreading my thesis and allowing me to improve it further.

Contents

Acronyms ix

1. Introduction 1
1.1. Goals and Scope . . . 1
1.2. Introduction of MBDA Deutschland GmbH . . . 1

2. Fundamentals 3
2.1. Jupyter Notebook . . . 3
2.2. Malware . . . 4
2.2.1. Botnet . . . 4
2.2.2. Malware Beaconing . . . 5
2.3. Botnet Detection . . . 5
2.3.1. Honeypot-based Detection . . . 6
2.3.2. Signature-based Detection . . . 6
2.3.3. Periodic-based Detection . . . 7
2.4. Mathematical Basics . . . 7
2.4.1. Mean . . . 7
2.4.2. Median . . . 8
2.4.3. Mode . . . 8
2.4.4. Median Absolute Deviation . . . 8
2.4.5. Bowley Skewness Measure . . . 8
2.5. Real Intelligence Threat Analytics . . . 9
2.5.1. The difference between User and Beacon Traffic . . . 9
2.5.2. Bowley Skewness Score . . . 10
2.5.3. Median Absolute Deviation Score . . . 10
2.5.4. Connection Count Score . . . 10
2.5.5. Data Smallness Score . . . 10
2.5.6. Overall Beacon Score . . . 10
2.6. Binary Search Tree . . . 11

3. State of the Art 12
3.1. HELK . . . 12
3.2. Real Intelligence Threat Analytics . . . 13
3.3. Suricata . . . 13
3.4. Differentiation from Existing Solutions . . . 14

4. Requirements Engineering 15
4.1. Use Case . . . 15
4.2. Functional Requirements . . . 15
4.3. Non-functional Requirements . . . 16

5. Concept Design **17**
5.1. Selection of the Software Base . . . 17
5.2. Design of a Component Architecture . . . 17
5.2.1. Component Diagram . . . 18
5.3. Resource Gathering Notebook . . . 19
5.4. Malware Beaconing Detection Notebook . . . 19
5.4.1. Security Information and Event Data . . . 19
5.4.2. Allowlist Analysis . . . 20
5.4.3. Traffic Information . . . 20
5.4.4. Time Series Analysis . . . 20
5.4.5. Suspicious Indicator Analysis . . . 23
5.4.6. Investigation and Verfication . . . 24
5.5. Enrichment Notebook . . . 24
5.5.1. Whois . . . 24
5.5.2. GreyNoise . . . 24
5.5.3. VirusTotal . . . 25

6. Implementation **26**
6.1. Data Structure for Malware Beaconing Detection . . . 26
6.2. Importing HTTP Log Data . . . 26
6.3. Domain Generation Algorithm Feed . . . 27
6.4. Detecting Backdoor Activation . . . 27
6.5. Efficent Search . . . 28
6.5.1. Pandas Search . . . 28
6.5.2. Binary Search Tree . . . 29
6.6. Storage of Application Programming Interface Keys . . . 30
6.7. Displaying Enriched Malware Beaconing Activities . . . 30
6.8. Malware Beaconing Detection Report . . . 31

7. Detection Evaluation **32**
7.1. Evaluation of the Implementation of Requirements . . . 32
7.2. Performance . . . 33
7.2.1. RITA . . . 33
7.2.2. Suricata . . . 33
7.2.3. Time . . . 35

8. Conclusion and Outlook **36**

A. Appendix **i**

Appendices **i**

Literatur **iv**

List of Figures

2.1. Jupyter Notebook [6] . . . 3
2.2. Botnet [9] . . . 4
2.3. Taxonomy of Botnet Detection [13] . . . 6
2.4. Snort Blackenergy Signature [17] . . . 7
2.5. User and Beacon traffic [24] . . . 9
2.6. Binary Search Tree . . . 11

3.1. HELK [29] . . . 12

5.1. Component Diagram . . . 18
5.2. 10 Minute Beacon Analysis [45] . . . 21
5.3. 60 Minute Beacon Analysis [45] . . . 21
5.4. Activated Backdoor . . . 22

6.1. Not Activated Backdoor . . . 28

7.1. Suricata Evaluation . . . 34

A.1. Enriched Malware Beaconing Activities . . . ii
A.2. Detection Report . . . iii

List of Tables

3.1. Real Intelligence Threat Analytics Commands [23] 13

4.1. Functional Requirements 16
4.2. Non-functional Requirements 16

6.1. Pandas Search Performance 29
6.2. Binary Search Tree Performance 30

List of Codes

6.1. Variable Mapping 27
6.2. Backdoor Activation 27
6.3. Pandas Search 28
6.4. Build Binary Search Tree Function [57] 29
6.5. Search Binary Search Tree Function [57] 30

Acronyms

AI Artificial Intelligence

API Application Programming Interface

ASCII American Standard Code for Information Interchange

C2 Command and Control

CERT Computer Emergency Response Team

CSV Comma-separated Values

DGA Domain Generation Algorithm

DNS Domain Name System

DoS Denial of Service

ELK Elastic Stack

GUI Graphical User Interface

HELK Hunting Elastic Stack

HTML Hypertext Markup Language

HTTPS Hypertext Transfer Protocol Secure

HTTP Hypertext Transfer Protocol

IP Internet Protocol

IT Information Technology

IoC Indicator of Compromise

IoT Internet of Things

NIDS Network Intrusion Detection System

NIPS Network Intrusion Prevention System

NTP Network Time Protocol

PCAP network packet capture

PDF Portable Document Format

PII personally identifiable information

RITA Real Intelligence Threat Analytics

SIEM Security Information and Event Management

SMTP Simple Mail Transfer Protocol

SOAR Security Orchestration, Automation and Response

SOC Security Operation Center

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TOR The Tor Browser

TSV Tab Separated Values

1. Introduction

The 2022 status report of Information Technology (IT) security in Germany [1] compiled by the Federal Office for Security in Information Technology classified the situation in the reporting period as tense to critical, which is mainly attributed to the spread of ransomware and new malware variants. According to the analysis, an increase of 116,6 million new malware variants was recognized in 2022. This is part of an ongoing battle between cyber criminals and IT security professionals trying to protect businesses and individuals. IT security experts continuously improve protective measures and the detection of hacker attacks while criminals try to circumvent them. The analysis of malware, in particular, requires a high degree of expert knowledge and a considerable investment of time. Since IT security is a relatively young profession and more resources have only been put into the training of IT security experts in recent years, many job openings are not filled [2]. Therefore, the industry increasingly relies on automation to identify malware and analyze security incidents to combat the lack of professionals.

1.1. Goals and Scope

The goal of this thesis is the practical implementation of the analysis of security-relevant network log information to detect malware beaconing with Jupyter Notebook in a realistic corporate network. A proof of concept will be developed to efficiently analyze the data provided, recognize periodic communication, detect beaconing indicators and give a clear and enriched presentation for further manual analysis. This should demonstrate the potential of analyzing, hunting, and detecting cyber threats with Jupyter Notebook for companies and create a starting point for other notebooks.

1.2. Introduction of MBDA Deutschland GmbH

This thesis is conducted in cooperation with MBDA Deutschland GmbH. MBDA S.A.S. is a European defense company represented in Germany, France, Italy, Spain and the United Kingdom. It specializes in developing, testing and manufacturing air missile and air defense systems and components for Army, Navy and Air Force subsystems. Other services include the maintenance and repair of existing systems and those developed in cooperation with other manufacturers. [3]

MBDA Deutschland GmbH has its headquarters in Schrobenhausen where most of the development, administration and explosives testing takes place. In addition, two smaller sites are situated in Germany. Freinhausen is used to test collaborations with the German Airforce. The subsidiary Bayern-Chemie GmbH in Aschau am Inn is a competence center for aerochemical propulsion systems. [4]

Since MBDA S.A.S is active in the defense industry, the company is particularly attractive for advanced persistent threat groups. For this reason, MBDA S.A.S attaches particular importance to the highest possible standard of IT security. In addition to preventive measures, great attention is paid to detecting cyber attacks and analyzing security incidents. However, these tasks are time-consuming and require a

high level of expertise, often unavailable with a limited number of resources. Therefore, the support of modular and easy-to-use automation is needed.

The practical part of this thesis was carried out according to the requirements of the MBDA Deutschland GmbH Security Operation Center (SOC) and served as a proof of concept for the feasibility and usefulness of automating security operation tasks with Jupyter Notebook.

2. Fundamentals

This section defines the knowledge required to understand this paper. It offers a brief overview of malware and common detection methods and a mathematics and data structure knowledge refresher.

2.1. Jupyter Notebook

Jupyter Notebook is an open-source software for interactive scientific data analysis developed in 2014 by Fernando Pérez. The web application enables the simple and modular creation and sharing of interactive live code, visualizations and documentation in numerous programming languages. In recent years, Jupyter Notebook has become increasingly popular in academia as it allows scientists to analyze different datasets in the same way and share studies easily and without significant prior knowledge. This makes it easier to validate and reproduce results. [5]

Jupyter Notebook uses a two-process model based on the kernel-client infrastructure shown in figure 2.1.

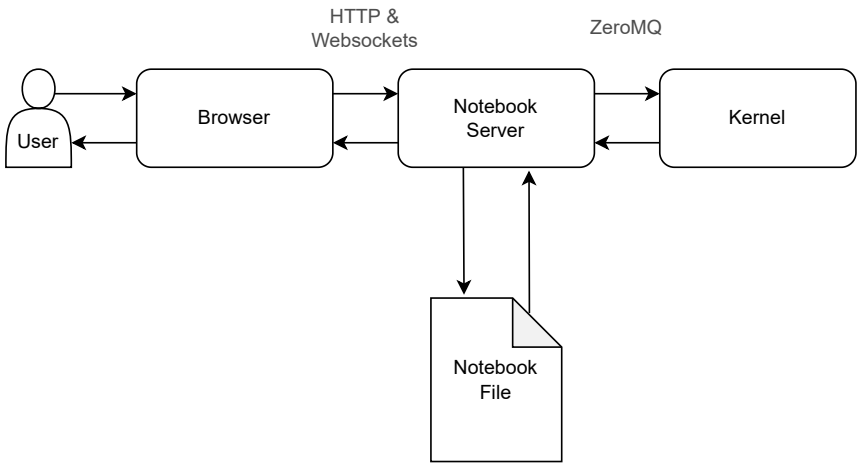

Figure 2.1.: Jupyter Notebook [6]

Notebooks are files with the **.ipynb** file extension, which can contain code, metadata, content and outputs. Notebooks can be written in any browser displaying the notebook server user interface. The notebook server acts as a client which handles the communication between the browser, notebooks, qtconsole and the kernel. It is also possible to send code directly to the kernel using the Qt console a lightweight application which combines the advantages of a terminal and Graphical User Interface (GUI).

The IPython kernel processes the code forwarded by the notebook server and sends back the result. The kernel can simultaneously communicate with different clients over the low-level transport protocol ZeroMQ and web sockets. ZeroMQ [7] is an asynchronous messaging library for high-throughput computing specifically designed for simultaneous execution in distributed systems.

Furthermore, the Jupyter project provides the web-based development environment Jupyter Labs, which can be personalized with extensions and tailored to the respective task. [6]

2.2. Malware

Malware is software created to fulfill an attacker's harmful intent. The word is used as an umbrella term for a variety of more specialized harmful software. Malware is developed to penetrate devices and networks by using vulnerabilities and social engineering techniques to carry out predefined tasks depending on the type of malware. For example, Spyware collects data from a device; Ransomware, on the other hand, encrypts user data and demands a ransom to restore it. [8]

2.2.1. Botnet

A bot is a type of malware that works remotely on smartphones, computers, Internet of Things (IoT) devices and many other systems. If a large number of systems are infected by one malicious actor and misused for specific actions, it is called a botnet, as shown in figure 2.2. Once a device is infected by botnet malware, it tries to establish communication with its botmaster over a Command and Control (C2) server to receive commands. A botmaster is a person who operates and controls the botnet. This regular communication serves as a keep-alive message and to manage further activities of the systems controlled by the attacker. The infected devices can be misused for various purposes, including reloading other malware, such as encryption trojans. Botnets are often utilized because of their strong resources. For example, by sending spam mail, renting it out to third parties or Denial of Service (DoS) attacks that overwhelm websites and services with a flood of requests. [9]

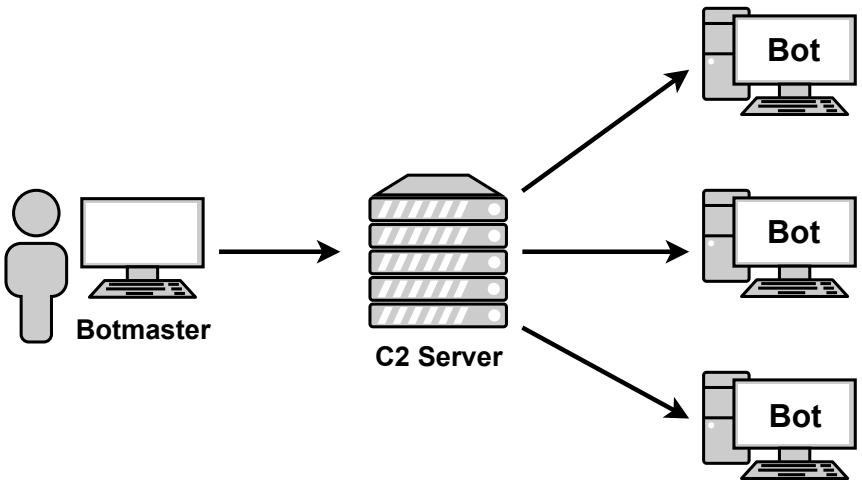

Figure 2.2.: Botnet [9]

2.2.2. Malware Beaconing

After the early stages of the Cyber Kill Chain [10], consisting of reconnaissance, weaponization, delivery, exploitation and installation, attackers try to establish remote control and persistence through C2 techniques. Malware beaconing is a C2 technique to establish continuous communication between malware and a server controlled by the attacker. The attacker uses the server to share commands to control the malware individually at any time after the initial infection and to exfiltrate data from the host. Some of the most critical characteristics of malware beacons are described below. [11]

- Communication occurs through **common protocols** such as Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS), Domain Name System (DNS) and Simple Mail Transfer Protocol (SMTP) to look like ordinary traffic and to tunnel firewalls.
- **Popular cloud services** like Google Documents or Dropbox are used to evade restrictions.
- C2 communication can be **encrypted with HTTPS**, only leaving the traffic destination visible.
- Beacons **communicate in regular intervals** ranging from every few seconds to days.
- Most requests from the compromised system will include the same commands being transmitted, resulting in a **similar packet size**.
- To evade detection, **malware beacon payloads can be encoded or encrypted.**
- A **jitter** is often used to add randomness to the beaconing communication. The jitter is set to a time interval in which bounds the beacon generates a random time to request the next instruction. [12]

2.3. Botnet Detection

The paper "A Taxonomy of Botnet Behavior, Detection, and Defense" [13] divides botnet detection into three main approaches: recognizing individual bots, the botmaster and command and control communication. This thesis focuses on the detection of communication, as shown in figure 2.3.

Active detection is involved in bot operations by manipulating them, which can be done through injection or suppression. **Injection detection** injects packets into suspicious communication to infer C2 communication based on the response. **Suppression detection** works similarly, but instead of including new packets, the sending and receiving of certain incoming and outgoing packets is suppressed.

On the other hand, passive detection observes communication without actively intervening. **Syntactic detection** works with signatures and predefined patterns, such as strings or sequences, that have emerged from the analysis of known malware communication. In **semantic analysis**, heuristics are used to identify properties of defective communication. One of the sub-detections of semantics is the **correlation**, which detects bots with similar communication behavior. In **behavioral analysis**, deviations from regular machine traffic are detected. The last sub-item is the **statistical approach**, which is largely implemented by machine learning, whereby recognition is trained from prominent features of C2 communication, such as the range of packet length or flow duration.

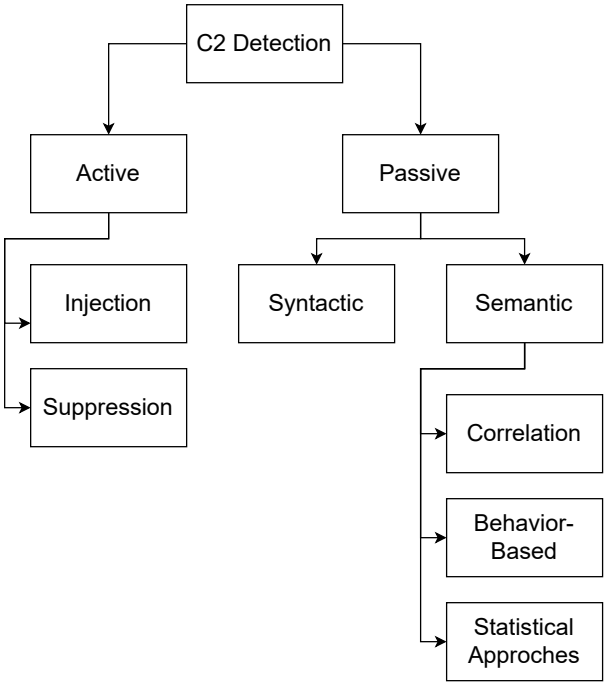

Figure 2.3.: Taxonomy of Botnet Detection [13]

2.3.1. Honeypot-based Detection

One of the first attempts to detect botnets and C2 communication was developed by the Honeynet Project [14]. So-called honeypots were placed on the internet and C2 communication was captured. The monitored traffic was analyzed to block the botnet domains and Internet Protocol (IP) addresses found and to create patterns to search for the detected botnets. A honeypot is a computer system deliberately equipped with security gaps to divert attackers from the actual targets or to analyze hacker attacks more precisely. The honeypot is designed to mimic an actual computer system but does not contain legitimate data or provide access to the internal network. In addition to distracting from real live networks, honeypots are equipped with extensive monitoring for information gathering. This allows to gain an overview of the current threat situation or, in the case of specific built-in vulnerabilities, insight into the attacker's tactics, techniques and procedures. Honeypots are still actively used today to collect malware samples, threat information data for detection engineering and threat intelligence.

2.3.2. Signature-based Detection

A traditional method of detecting malware C2 communication is through pattern-based signatures. Signatures are Indicator of Compromise (IoC) that indicate a known attack or abusive system behavior. IoCs can be specific objects or activities like IPs, domains, hashes, byte sequences, privileged logins from foreign countries and more. With signature-based C2 detection, data traffic is observed and the behavior and content is compared to the IoCs. Signatures are frequently used in Network Intrusion Detection System (NIDS), which monitor traffic and activities and raise alarms if an anomaly is detected. Figure 2.4 shows a snort [15] signature to detect BLACKENERGY malware [16]. The signature enables to check whether a connection from the internal network to the internet via Transmission Control

Protocol (TCP) has specific properties. The packet data has to be less than 400 bytes and data traffic must be a HTTP POST request with the header "Cache-Control—3a— no-cache". Furthermore, the body must contain specific strings that indicate the type of malware. If all these properties are given, an alarm is sent with the message "ET TROJAN Blackenergy Bot Checkin to C&C".

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
(msg: "ET TROJAN Blackenergy Bot Checkin to C&C";
flow:      established,to_server;          dsize:<400;
content: "POST"; nocase;                   http_method;
content: "Cache-Control|3a| no-cache";     http_header;
content: "id=";                            http_client_body;
content: "&build id=";                     http_client_body;
pcre:     "id=x.+ [0-9A-F] {8}&build id=.+/P";
classtype:trojan-activity; sid:2007668;)
```

Figure 2.4.: Snort Blackenergy Signature [17]

Signatures have the disadvantage that only known and analyzed threats can be detected. Furthermore, signatures work best on unencrypted data and must be updated regularly.

2.3.3. Periodic-based Detection

Periodic-based detection focuses on detecting malware beacons. Since these queries contact the attacker via the C2 infrastructure at regular intervals, these signals can be recognized in the network logs. The difficulty lies in discerning C2 communications among the multitude of non-malicious network activities. There are different approaches, such as the recognition by Discrete Fourier Transforms [18], which can recognize multiple distinct period lengths in a given time series. Alternatively, calculating the signal-to-noise ratio [19] that compares the level of a harmonic signal to the background noise and therefore recognizes the periodic malware beaconing as a signal. The advantage of this approach is that even unknown malware can be detected this way. However, attackers have also evolved and developed methods such as configuring a jitter to distort the regular queries. A jitter randomizes the time delta a beacon sleeps in between contacting the C2 infrastructure [20]. For example, a beacon is set to ask for new commands every 60 minutes. By adding a jitter of 50%, beaconing calls can now vary by up to 30 minutes. Interactions would occur within a time delta interval of 30 to 90 minutes. This variance in timing makes detecting beacons more difficult since they are less predictable.

2.4. Mathematical Basics

2.4.1. Mean

The statistical mean is a number within a dataset that represents the center point or a typical value. To calculate the mean, all numbers within a data set are added and divided by the number of observations.

2.4.2. Median

The median is the middle of a dataset, which separates the higher and lower half. To calculate the median, the dataset is sorted from the smallest to the highest number. Then the number in the middle is selected as the median; if the dataset has an even number of values, the median is defined as the mean of the two middle values.

2.4.3. Mode

Mode is the most common number presented in a dataset. A set of numbers can have one, multiple or no mode. It is calculated by placing all numbers in ascending or descending order and then counting how many times each number appears.

2.4.4. Median Absolute Deviation

Median absolute deviation is a robust measure of how spread out a dataset is. The median absolute deviation is best suited if the data is normally distributed and does not have extreme highs and lows. It is calculated by sorting the dataset to find the median, which is then subtracted from every data point. The resulting absolute value of each number is sorted again, whereas the newly calculated median is the median absolute deviation. [21]

$$mad = median(x_i - m)$$

x_i = Each of the values in the dataset
m = Median of a dataset

2.4.5. Bowley Skewness Measure

The Bowley skewness calculates whether a distribution is negatively or positively skewed. Skewness describes the degree of deviation from a symmetrical distribution. In the case of a symmetrical distribution, the quartiles are equidistant. [22]

$$(Q_2 - Q_1 = Q_3 - Q_2)$$

Thus, the Bowley measures asymmetry by calculating if the quartiles are not equidistant from the median.

$$bowley = \frac{Q(2 - Q_2) - (Q_2 - Q_1)}{Q_3 - Q_1}$$

- $0 =$ Curve is symmetrical
- $0 >$ Curve is positively skewed
- $0 <$ Curve is negatively skewed

2.5. Real Intelligence Threat Analytics

Real Intelligence Threat Analytics (RITA) [23], as described in more detail in section 3.2, is an existing framework written in Go for network traffic analysis which, among other things, can detect malware beaconing through statistical analysis. The following section describes how the periodic-based malware beacon analyzer works.

2.5.1. The difference between User and Beacon Traffic

Real Intelligence Threat Analytics (RITA)'s analysis process [24] is based on the distribution of user and beacon traffic. To visualize the differences, a sufficient amount of beacon communication with the following properties was generated.

- **Time delta:** 300 seconds
- **Jitter:** 20%
- **Time delta interval:** 240-360 seconds

To create a realistic environment, the beacon traffic is examined with regular user traffic generated by visiting and refreshing a website.

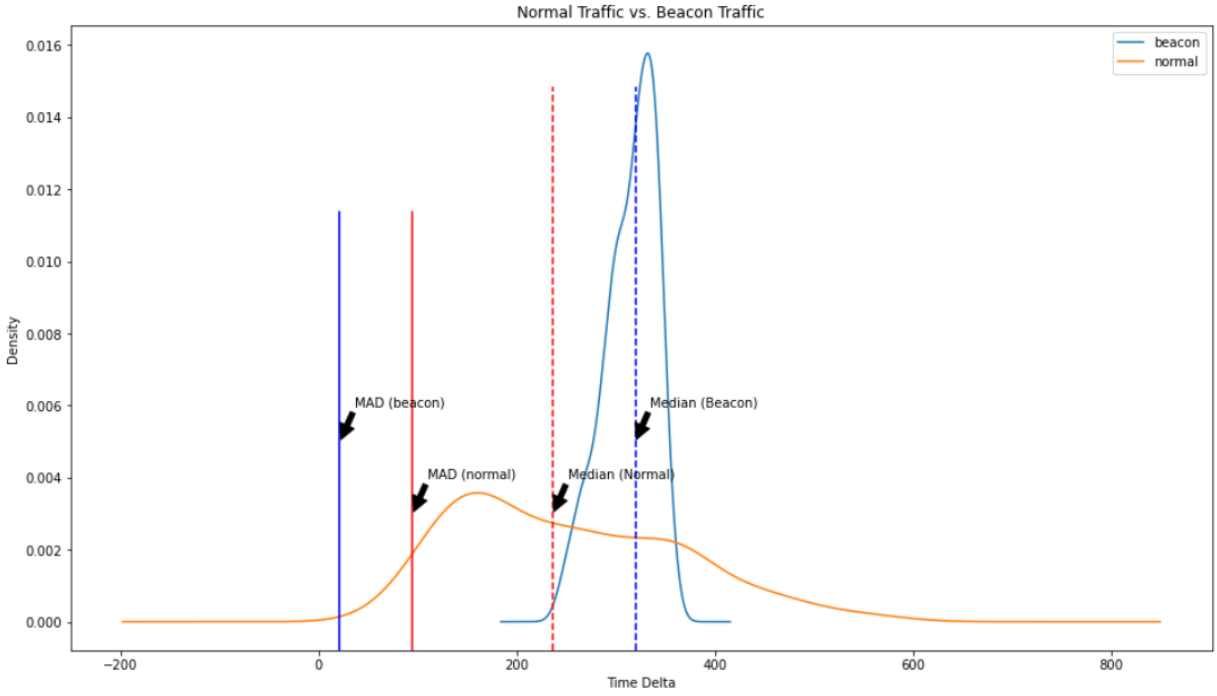

Figure 2.5.: User and Beacon traffic [24]

The figure 2.5 shows that beacon traffic has an even distribution and a small median absolute deviation. In contrast, more random user traffic is skewed in one direction and has a larger distribution. Based on this knowledge, RITA analyzes both the chronological sequence of requests and the transmitted data to calculate a beacon score. Furthermore, indicators like packet size and the number of queries are considered. [25]

2.5.2. Bowley Skewness Score

Perfect beacons have a symmetric time delta and data size distribution. The Bowley skewness measure provides information about symmetry and how it deviates. The symmetry for all connections between two systems can be calculated using the formula described in section 2.4.5. The closer the value is to zero, the higher the symmetry and the higher the probability of a malware beacon.

2.5.3. Median Absolute Deviation Score

The median absolute deviation is also calculated for the time delta and data size. Beacons have very low dispersion around the median, which can be calculated using the formula in section 2.4.4. Therefore, the smaller the median absolute deviation, the higher the probability of a malware beacon.

2.5.4. Connection Count Score

A good indicator for beacons is a high connection count, which measures the communication between two systems. First, the time between the first and last recorded connection is determined to calculate the time delta. The number of all established connections is then divided by the previously determined time delta of the first and last request. The greater the number of connections in a set timeframe, the greater the probability it is a beacon.

2.5.5. Data Smallness Score

The data smallness score assumes that numerous small packets are sent if the C2 infrastructure does not have new commands for the malware. A smaller data average additionally indicates malware beaconing since user traffic usually involves data-intensive pictures and media. To calculate the data smallness score, an average data packet is divided by 16383. RITA [26] uses the value 65535, whereas RITA-J [27] divides by 8192 because perfect beacons only send very little data and are mostly idle. The smaller the value, the more sensitive the score. The number used in this implementation is calculated as the average size of beacon data packets in the test data set and serves as a guide value.

2.5.6. Overall Beacon Score

The final result is calculated from the sum of the datascore and timescore divided by two.

$$Timescore = \\ (BowleySkewnessScore + MedianAbsoluteDeviationScore + ConnectionsCountScore)/3$$

$$Datascore = \\ (BowleySkewnessScore + MedianAbsoluteDeviationScore + DataSmallnessScore)/3$$

$$Finalscore = (Timescore + Datascore)/2$$

2.6. Binary Search Tree

A binary search tree is a node-based data structure that allows one to search, insert and delete items within it quickly. The "binary" part of the name comes from the property that each node has a maximum of 2 children. A binary search tree can be generated by locating all nodes with a value smaller than the root note on the left subtree and all larger nodes on the right. An example of a correctly constructed binary search tree is shown in figure 2.6.

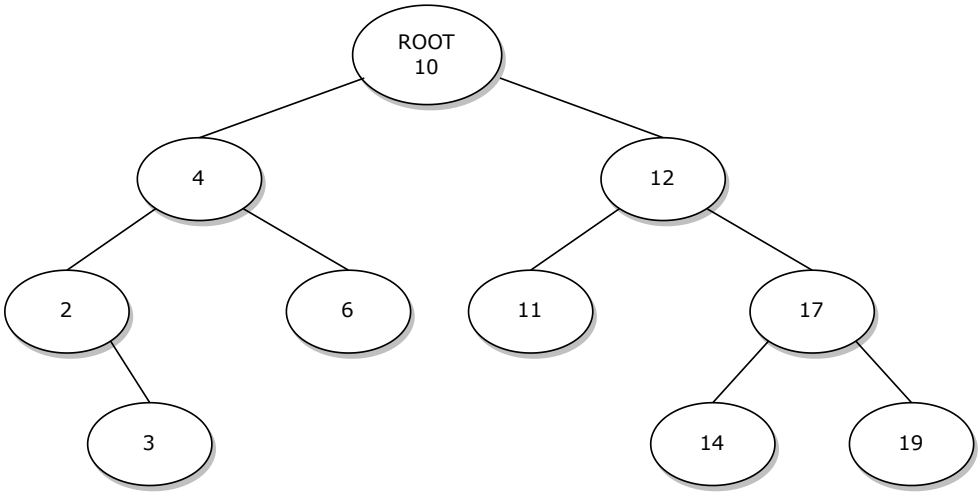

Figure 2.6.: Binary Search Tree

A search in the tree-like structure has a time complexity of $O(logn)$ in the best and $O(n)$ in the worst case. If the searched number is smaller than the current node, the search continues on the left subtree otherwise, the number is larger and located on the right subtree. A NULL is returned if the search has reached the end of a search tree and has not found any node with the specified value. [28]

3. State of the Art

This section presents software with similar functionalities to gain an insight into the state of the art and compare it with the solution presented in this paper. The statements are based exclusively on the information provided by the manufacturer and have not been verified by testing the software.

3.1. HELK

Hunting Elastic Stack (HELK) [29] is an open-source threat hunting platform based on Jupyter Notebook and Apache Spark over an Elastic Stack. Elastic Stack (ELK) [30] is a real-time tool to store, search, analyze and visualize data. The HELK project was launched in 2017 and is currently in the alpha phase. There have been no contributions for over a year and currently, no information about further updates is available. It was primarily developed for research and to make threat hunting and data analysis more accessible and faster. The most important features include cluster capability with Spark, visualization with Kibana, data collection with real-time pipelining capabilities using Logstash and distributed publish-subscribe messaging with Kafka. A complete overview of the architecture is shown in figure 3.1.

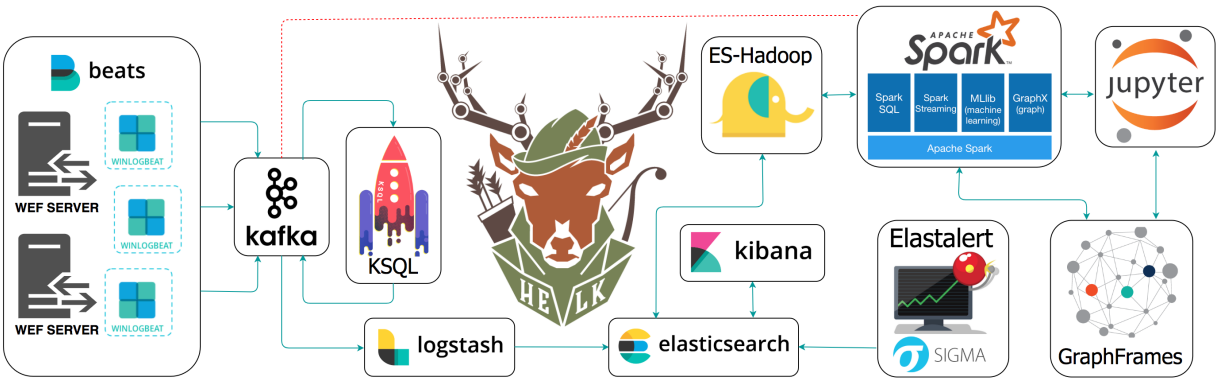

Figure 3.1.: HELK [29]

The most significant functionality HELK provides is converting Sigma rules into notebooks [31]. Sigma is a generic and open format to describe log events. Most malware detections are written in Sigma since the language is platform-independent. A few hundred Sigma rules have already been implemented as a notebook, including a few detections for common C2 tools.

3.2. Real Intelligence Threat Analytics

RITA [23] is an open-source framework for network traffic analysis capable of detecting malware beaconing behavior, DNS tunneling and blocked domains and hosts. The framework scans both in a Tab Separated Values (TSV) formatted database or in real-time using the open-source network security monitoring tool Zeek [32]. Mainly written in Go, RITA is controlled over the terminal and does not have a GUI. RITA can only be installed natively in Linux, but there is also a prebuilt Docker container. The configuration and the evaluation of the results require appropriate specialist knowledge. Currently, the range of functions includes the following commands.

KeyWord	Description
show-databases	Print the datasets currently stored.
show-beacons	Print hosts which show signs of C2 software.
show-bl-hostnames	Print blacklisted hostnames which received connections.
show-bl-source-ips	Print blacklisted IPs which initiated connections.
show-bl-dest-ips	Print blacklisted IPs which received connections.
show-exploded-dns	Print DNS analysis. Exposes covert DNS channels.
show-long-connections	Print long connections and relevant information.
show-strobes	Print connections which occurred with excessive frequency.
show-useragents	Print user agent information.

Table 3.1.: Real Intelligence Threat Analytics Commands [23]

The project is updated irregularly at the time of writing. However, the documentation indicates that the commercial product AC-Hunter [33] from the same developers is an up-to-date, GUI-based and feature-rich network threat hunting alternative. It is also worth mentioning that the GitHub user Cyb3r-Monk [34] has published RITA-J [24], a Jupyter Notebook implementation of RITA. Currently, the notebook only offers the ability to detect beacons through the RITA algorithm in Comma-separated Values (CSV) files and no additional features.

3.3. Suricata

Suricata [35] is an open-source NIDS with Network Intrusion Prevention System (NIPS) capabilities developed by the Open Information Security Foundation. NIDS is a device or application that detects malicious traffic on a network. It usually needs promiscuous network access and does not interfere with the monitored traffic. The NIDS alerts a management server or NIPS if a threat is detected. NIPS then takes active steps to stop the malicious activity. Among functionalities like network packet capture (PCAP) file analysis, automatic logging, protocol detection and parsing, IP reputation and file extraction, Suricata can recognize dangerous network traffic through signatures. To detect malware beacons, Suricata offers various rulesets with signatures to identify known beacon behavior. In addition, some unofficial advancements use Suricata's logging functionality to detect periodic communication behavior.

3.4. Differentiation from Existing Solutions

The system developed in this work differs from already established solutions in several aspects.

Through the implementation with Jupyter Notebook, the detection works without any dependency on external applications and can be run offline with local data. This can prevent sensitive information from becoming public and protect personally identifiable information (PII) recorded in the network traffic logs. The notebooks offer a graphical, interactive user interface and plots to visualize beacon activities. The user can also enrich potential C2 Infrastructure with more information, making the application usable without a deep technical understanding of malware beacons. The application also uses a mix of signature and periodic-based detec, resulting in a higher detection rate.

4. Requirements Engineering

In this thesis, the use of Jupyter Notebook to detect malware beaconing in a realistic corporate network is to be tested. The specific requirements and restrictions are based on the needs and infrastructure of the MBDA Deutschland GmbH SOC.

4.1. Use Case

The application is used for security audits and forensic analyses and should be able to operate on HTTP network traffic logs from different environments. The log data is primarily provided by a Security Information and Event Management (SIEM) system but should be as adaptable as possible to support log data from additional sources. Malware beaconing detection should be both signature and periodic-based. Since the analysis is usually carried out on-site on a standalone system, data transmission and detection must occur offline. Therefore, the installation and time required for on-site use of the application must be kept to a minimum. A clear overview of the suspicious systems should be given immediately after the analysis. Furthermore, the functionality to enrich the detected connections in an online system with further information is required. This enrichment should be visualized as clearly and comprehensibly as possible and is not subject to any restrictions concerning online resources. The number of connections to be analyzed online should be as small as possible to protect Application Programming Interface (API) license limits and enable the SOC analyst to analyze only IPs and domains authorized for the public eye. To make investigations comprehensible and revisable at a later point in time, the suspicious connections and the enriched data has to be saved. The entire application will be implemented using Jupyter Notebook to share the notebooks with other analysts and generate reproducible results. Therefore, the application should be programmed as simply and understandably as possible.

4.2. Functional Requirements

The functional requirements are developed from the given scenario and system context. Functional requirements are specifications that describe the functions and capabilities of a system. They define the system's fundamental behavior and must be fulfilled for the system to be considered successful. [36, pages 51 - 52]

ID	Description
FR1	The application shall be built with Jupyter Notebook.
FR2	The application shall be able to import and use security-relevant log information.
FR3	The application shall analyze malware beaconing with periodic-based detection.
FR4	The application shall analyze malware beaconing with signature-based detection.
FR5	The application shall be able to look up reputations on IPs and domains.
FR6	The application shall be able to enrich IPs and domains with information.
FR7	The application should be able to archive investigation results.

Table 4.1.: Functional Requirements

4.3. Non-functional Requirements

Non-functional requirements describe the general characteristics of a system. If these are not met, the application will continue to function but will not meet the expectations of the stakeholders or the company. They are called quality attributes and describe requirements such as accessibility and compatibility. [36, page 52]

ID	Description
NFR1	The application shall be easy to understand and customize.
NFR2	The application shall be able to detect malware beaconing offline.
NFR3	The offline application shall be easy and quick to install.
NFR4	The offline application shall be able to use up-to-date signatures.
NFR5	The online application shall provide an understandable and visual evaluation.
NFR6	The online application shall only enrich connections on demand during investigation.

Table 4.2.: Non-functional Requirements

5. Concept Design

This chapter describes the specific technical implementation of the given system requirements.

5.1. Selection of the Software Base

The requirement FR1 already establishes that the system has to be implemented with **Jupyter Notebook**. Jupyter Notebook used to be represented primarily in academics [5] but has become an inherent part of many big companies like Microsoft and Google [6]. Especially the automation of IT security tasks has become more prevalent in recent years, with open-source projects like HELK and companies like IBM [37] adding support to their products. Besides the variety of free and ready-to-use notebooks, Jupyter Notebook has many advantages for analyzing, hunting and detecting cyber threats. The open-source software can be used as a free, efficient and programming language-independent way to analyze security events in nearly every environment. Deployment is possible both locally and in the cloud. Once installed, Jupyter Notebook can be used offline in every up-to-date web browser, which makes it a great tool to bring on-site when investigating an incident. Since Jupyter Notebook is language-agnostic, interactive and generates reproducible outputs, it is an excellent standardized way to document and share knowledge.

To make the application easy to understand and customize (NFR1), a widely used programming language, is chosen. According to Statista [38], 48.07% of global developers used the **Python** programming language in 2022, making it the most used programming language after script and structural languages such as JavaScript and HTML. Furthermore, the Jupyter Notebook installation already comes with a native Python execution backend, which aids the simple and fast installation required in NFR3. Another advantage is the extensive selection of additional libraries. Since security-relevant log information is to be imported and evaluated (FR2), libraries for Big Data, such as NumPy [39] and Pandas [40], are required. **NumPy** is imported for easy numerical data processing. It simplifies working with multidimensional arrays, which is required as a basis for programming with matrices and vectors. In addition to data structures, functions for numerical calculations are provided. **Pandas** is used to analyze large amounts of data efficiently. The open-source tool offers features for easy analysis and manipulation of Big Data. It is primarily designed for operation and access to numeric tables and series of numbers.

5.2. Design of a Component Architecture

In order to meet all requirements, the entire application is divided into three notebooks on an online and offline device. The signatures required for the detection are collected in the **Resource Gathering Notebook**, described in more detail under section 5.3. The notebook runs on a device connected to the internet and downloads IoCs and signatures, which are converted into a standardized format and

stored collectively in a folder. As required in NFR4, the up-to-date signatures can then be transferred to the standalone system using removable media.

When transferring data, it is important that the removable media is checked by a security gateway for removable media before it is connected to another system. A security gateway for removable media is a device to which removable media is connected to check the data carrier with various anti-virus software. This prevents malware software from spreading through infected files and mobile media.

In addition to the signatures, the HTTP log data to be analyzed is also transferred to the standalone system. Therefore, the primary **Malware Beaconing Detection Notebook** can be operated offline (NFR2) with the shared data to detect malware beacon activities, further explained in 5.4. Furthermore, by outsourcing all other functions, the notebook is operated without external resources and with few libraries, thus complies with NFR3. After the signature and periodic-based detection, all suspicious connections are exported to a file.

The suspicious connections can now be transferred to an online system with the help of a removable medium. The last **Enrichment Notebook**, described in 5.5, is responsible for interactively enriching the suspicious connections with information from the internet and displaying all collected knowledge.

5.2.1. Component Diagram

To better represent the components in the system and their tasks, a component diagram is created that shows the static implementation view of a system.

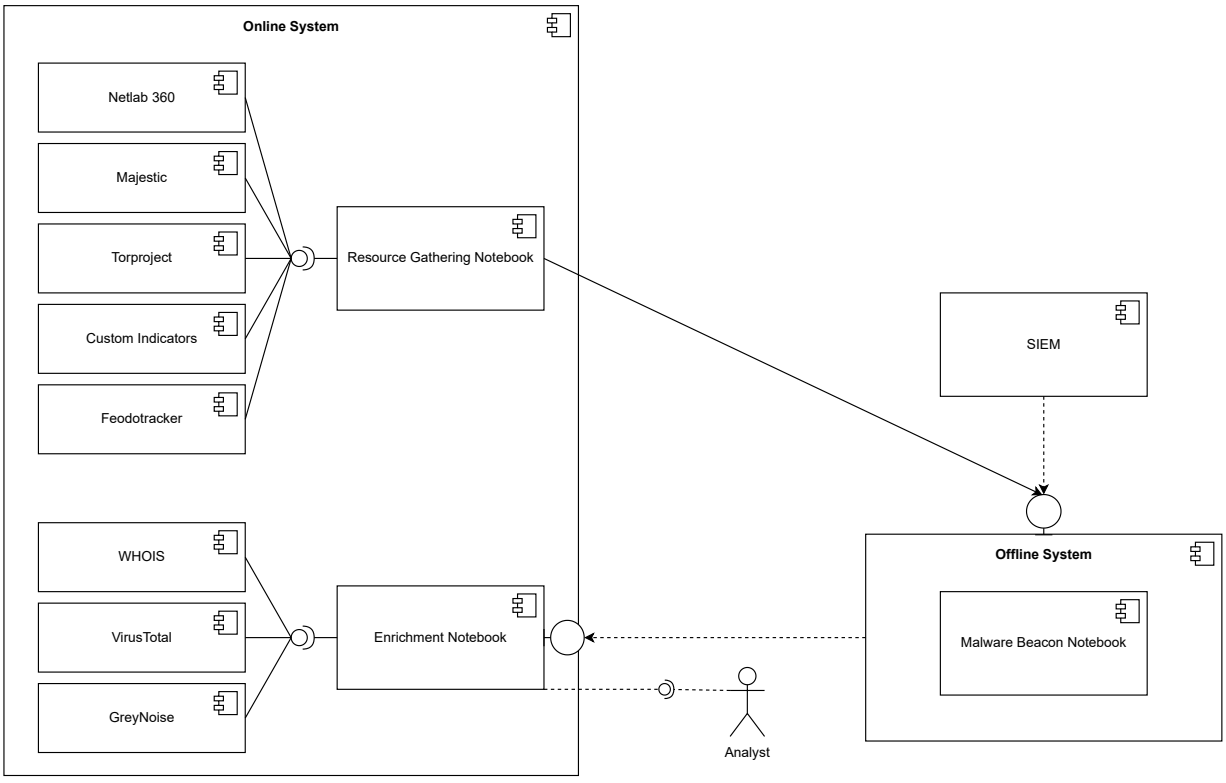

Figure 5.1.: Component Diagram

5.3. Resource Gathering Notebook

When executed, the Resource Gathering Notebook creates a folder containing IoCs and signatures for offline usage. The notebook first allows the user to enter custom domain and IP indicators in an input field. This enables the user to customize the application to their needs by incorporating IoCs from e. g. Computer Emergency Response Team (CERT) reports or threat intelligence sources. In addition to the individual indicators, the latest feed of The Tor Browser (TOR) exit nodes from the onion project [41], known C2 servers and known malware Domain Generation Algorithm (DGA) domains are collected. Lastly, a list of popular websites is downloaded for the allowlist. After the download, all unrelated data is deleted and the remaining indicators are converted into a uniform format.

5.4. Malware Beaconing Detection Notebook

The concept of this notebook is based on BAYWATCH [42], a robust beaconing detection methodology divided into four phases. Before the first phase begins, requirement FR2 is fulfilled by importing and standardizing security-relevant log information in preparation for further processing. The first phase, called **allowlist analysis**, filters out trusted traffic from the preprocessed data to reduce processing time. In the next step, the **time series analysis**, the filtered data is divided into communication pairs and checked for consistent behavior. To further enrich the periodic-based analysis (FR3), the third step is the signature-based (FR4) **suspicious indicator analysis** which searches for additional beacon indicators. Finally, the indicators are evaluated in the **investigation and verification phase** and suspicious connections are displayed. The human analyst is presented with an overview of all suspicious connections and why they were detected as a beacon. To further enrich the detections, all suspicious connections are exported into a file and the analyst can use the additional online Enrichment Notebook to gain more information.

5.4.1. Security Information and Event Data

The detection is based on security information and event data, generated from events that affect confidentiality, integrity or availability. The data is collected from various sources, the most common being sensors, the network, applications, databases, servers and user devices which generate data such as log files, configurations, messages, alarms, metrics, changes, or tickets.

The Malware Beaconing Detection Notebook is primarily designed to work with exported HTTP data from a SIEM system within a defined timeframe. SIEM systems are a tool to analyze security information and event logs to monitor network and user activities. Specifically, proxy, DNS and firewall logs can be exported from the system in various formats and integrated into Jupyter Notebook for a detailed analysis of beaconing communication. The notebooks support the filetype CSV, which most SIEM systems like Splunk, Big Blue, LogRhythm or Azure Sentinel use.

One of the main problems in structuring the available SIEM data is the labeling which is different for most sources and networks. The data required for the analysis must be named consistently. To make this task more accessible, the user has to manually define variables with the respective designation once at the beginning of the notebook. The rest of the notebook uses the mapped variables instead of the individual labels of the source. [43]

5.4.2. Allowlist Analysis

To minimize the data to be analyzed and enhance the overall performance, harmless and trusted network traffic is filtered before further processing. This includes benign websites like search engines or environment-specific intranet sites or services. The allowlist is divided into a universal and local list to make the customization optional and easy to update. The universal list can be used with every dataset, whereas the local allowlist consists of custom data provided by the user.

Universal Allowlist

The universal allowlist is comprised of popular public websites such as search engines, news, shopping, mail or social media. The allowlist can be easily compiled by importing a list of popular websites from resources like majestic's top one million [44]. It should be noted that in some cases, malicious usage can occur from those web services. However, such web providers have security measures and detections in place and publicly announce malicious use. The allowlist is a file that does not affect the rest of the notebook. This allows for easy updates and the removal if an extended beacon search through all data is desired.

Local Allowlist

The second allowlist is meant to be provided by the user and is different for every environment. Internal services like a Network Time Protocol (NTP) server are common false positives. NTP ensures the time is accurate on every local system by beaconing in intervals to synchronize the system clock. Other services potentially generating false positives are configuration management systems, patch management systems or licensing systems. A list of the most popular internal traffic destinations can be compiled with the help of a SIEM system or other network tools.

5.4.3. Traffic Information

Some malware beacon identifiers are easier to calculate across the entire HTTP traffic directly after the allowlist filtering. This includes a high amount of HTTP POST requests. Malware beacons regularly check the C2 infrastructure and ask for new commands. Therefore, the devices are sorted and displayed according to the number of HTTP POST requests. The same applies to a high number of messages with an HTTP 404 error code, meaning that a page or file was not found. For example, if a DGA described in section 5.4.5 is used to generate the next C2 domain, it may already be owned or not set up yet and the request returns an HTTP 404 response.

5.4.4. Time Series Analysis

For malware to be used efficiently, it is programmed to receive new commands regularly. This has the advantage that one universal malware can load additional software as required after infection and for example, change the target of a DoS attack on command. Therefore, a beacon is often built into malware. However, the beacon needs a way to communicate with its botmaster, which can be detected through the time series analysis. The beacons require the internet to establish a connection, which offers a central monitoring point for analysis. The captured data can be searched for persistent

beacons by filtering for long-standing connections; if these are not known, a more detailed analysis can be initiated. Furthermore, persistent connections do not scale well; if the C2 infrastructure has to maintain several thousand connections, a lot of hardware resources are required. Additionally, more connections mean more opportunities for security appliances like NIDS to detect the malicious activities. Therefore, beacons are used to reach out to the attacker in intervals. Since commands must be executed promptly, communication has to be frequent. This limitation can be discovered in the network traffic through appropriate analysis. [25]

Time Bucket Length

An essential factor to consider is the so-called time bucket in which the analysis is conducted. Anomalies or extreme values regulate themselves when looking at a more extended time period, which is why analyzes should ideally be made over more than 24 hours. The evaluation of the time deltas of a 10-minute beacon analysis can be viewed in figure 5.2, the red line shows the mean average of all connections. The mean average never matches the connections and many extremes can be seen.

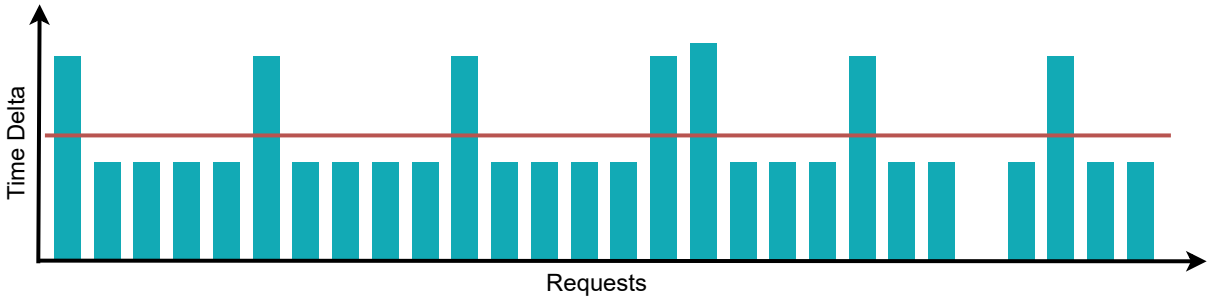

Figure 5.2.: 10 Minute Beacon Analysis [45]

A 60-minute analysis of the same beacon shows a different picture when values within a smaller period have been grouped and only the mean is displayed in the graph. For example, in figure 5.3, the extremes have balanced out and the mean average is close to the correct time delta for almost all connections.

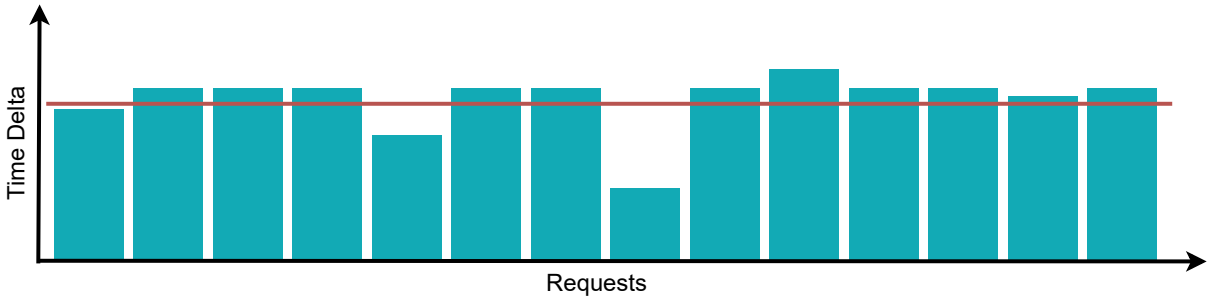

Figure 5.3.: 60 Minute Beacon Analysis [45]

RITA

The periodic-based detection is implemented by the open-source framework RITA. The precise technical implementation is described in 3.2. RITA uses the time deltas between requests and the transmitted data to detect periodic connections between systems. The analysis is also designed to find malware beacons with jitters and uses several detection methods. The framework is written in Go and has to be reimplemented in Python since it was selected as the programming language for this prototype. After the execution of RITA, a beacon score is given; the closer the score is to 1, the higher the probability it is a beacon. If the value is 1, it is a so-called perfect beacon sending the same query at regular intervals without a jitter.

Detecting Backdoor Activation

A backdoor is the part of the malware that allows the attacker to gain access or control a system after infecting a device. In the case of malware with a beacon function, the backdoor is said to be activated when the C2 infrastructure sends the first command to the beacon. If the backdoor still needs to be activated, the beacon traffic looks uniform. The malware asks the C2 infrastructure for new commands at regular intervals, which then responds with a sleep command. The answer is always the same and, therefore, always has the same size when the data is transmitted. However, the backdoor has probably been activated if there are other data transfers of different sizes, as shown in figure 5.4. This activation can be obfuscated by padding every data package.

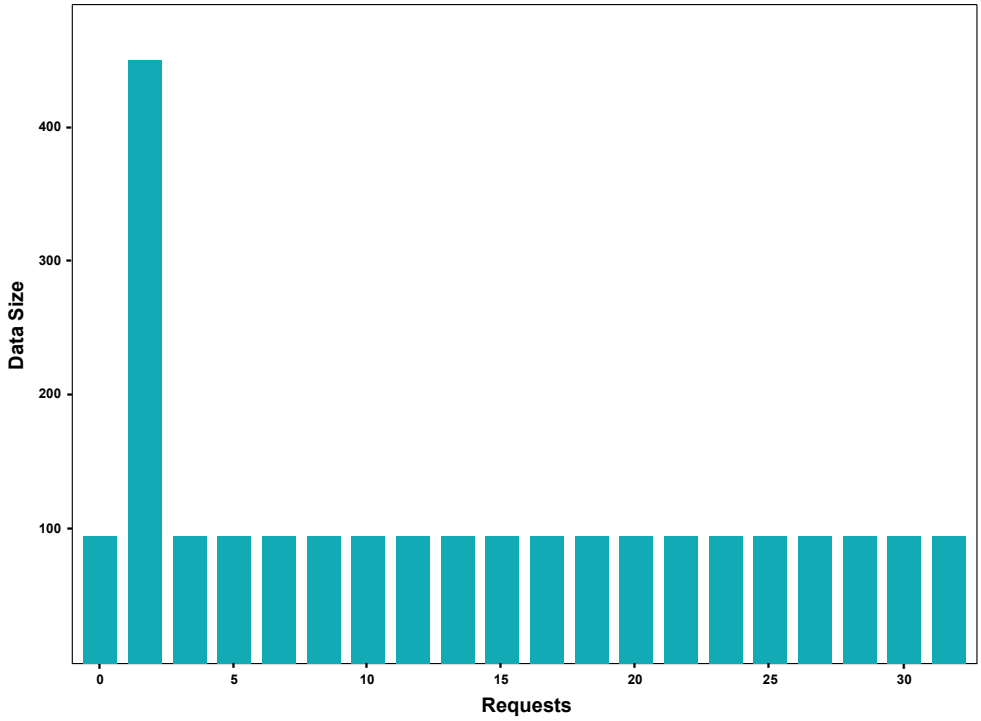

Figure 5.4.: Activated Backdoor

5.4.5. Suspicious Indicator Analysis

After a hacker attack, the methods of compromise are analyzed and the IoC shared. IoC are objects or activities used in security operations that indicate a compromise. IoCs can be unusual DNS requests, multiple login attempts, hashes, IPs or file and web server names. If malware beaconing is used in an attack, the server used by the hacker may be shared as an IoC so that it can be blocked or detected in other networks. [46]

Known Malicious Indicators

To make known C2 server IPs accessible offline, a comprehensive list and a real-time tracker are necessary. Feodo tracker by abuse.ch [47] is a free and regularly updated tracker for various well-known malware families. Feodo offers its collection of currently active botnets and all C2 servers ever tracked as a CSV file or Suricata botnet C2 IP ruleset. Initially, the notebook will be preconfigured with a comprehensive list of all botnets. However, this method will have a much higher chance of false positives since IPs are often re-used. To avoid this problem, the files from the Resources Gathering Notebook offers a quick and easy way to use only the most recent Feodo tracker feed.

Domain Generation Algorithm Detection

A DGA is a function that generates new domains on demand. Since botnet malware needs a server to get commands from to work correctly, using only one fixed domain or IP is often too risky for the attacker. The domain or IP can be blocked or taken down when detected, bringing an entire botnet to a halt. That is why modern malware families use the DGAs to switch to a new domain regularly. This constant change is also called domain fluxing. The algorithm generates the same predictable domains for the infected device and the attacker, but the result can not be analyzable by malware analysts without the seed. Furthermore, domains are generated where the probability is low that they are already taken and registration should be as cheap and anonymous as possible. The simplest way to meet these requirements is through an algorithm that generates a modifiable domain name from seed and uses a cheap persistent top-level domain. Since countless DGA algorithms can deliver diverse results through seeds, most detection approaches are based on signatures or machine learning. Machine learning would go beyond this paper's scope and is incompatible with the established requirements. That is why the notebook uses a signature-based approach. Netlab 360 [48] maintains a free and regularly updated list of known DGA families, which can be used to compare network traffic with known DGA-generated domains. [49]

TOR Nodes

TOR [41] is an overlay network for anonymizing connection data. To use it, the user installs a so-called onion proxy on his computer, which connects to the TOR network. TOR provides a list of TOR servers from which a random route is selected at regular intervals. The random connection chain is at least three servers long, with none of the systems knowing its predecessor or successor. It is, therefore, very time-consuming to track TOR communication. To protect their identity, some malware actors use the TOR network to obfuscate communications between the infected system and the C2 infrastructure. An easy way to find out if communication has occurred over a TOR network is to check the list of of known TOR servers. [50]

Custom Indicators

In the Resource Gathering Notebook, the user can add custom IP and domain indicators. The custom IoCs are compared with the HTTP log data and matches are marked as suspicious.

5.4.6. Investigation and Verfication

In the last part of the notebook, the analyst can sort the calculated results based on his priorities and examine individual connections more closely. For example, it is possible to choose between all time series analysis scores above a specific value, known malicious indicators, TOR nodes and the DGA detection. Furthermore, there is the option to select a particular connection with the row index which is then clearly displayed in a table and plots.

5.5. Enrichment Notebook

The last notebook is designed to give the analyst a quick and straightforward way to analyze previously detected suspicious communication. The findings are enriched with additional information (FR6), such as WHOIS or X.509 certificate data and reputation services like VirusTotal or GreyNoise (FR5), described in more detail in the sections below. Since companies use different services and APIs change, making the queries and the graphical representation as simple and modular as possible is crucial. To evaluate the connections required by the analyst (NFR6) only, a widget is created that takes the index of the connection to be analyzed and carries out the investigation on command. This saves time since the notebook does not automatically enrich all suspicious connections. Furthermore, the gathered information is simultaneously archived in a text file when an investigation is run (FR7).

5.5.1. Whois

WHOIS provides information about internet domains, IP addresses and their owners. An analyst must know when and where the domain was registered. Especially the geolocation of the server is a beneficial threat intelligence information. For example, connections to countries with no company locations or suppliers can be classified as suspicious. Another critical piece of data is the status of the domain and when it was registered. Newly registered domains should be analyzed more closely. C2 infrastructure is often blocked as soon as its use is known, so the domains change frequently. Furthermore, if a DGA is used, domains are regularly regenerated and have only been in use for a short time. [51] [52]

5.5.2. GreyNoise

GreyNoise analyzes internet scanning traffic to filter legitimate threats from background noise. The platform collects, analyzes and labels data on IPs that scan the Internet. This helps an analyst to classify the IP intent quickly and gives a detailed context. [53]

5.5.3. VirusTotal

VirusTotal is a free online service owned by Chronicle LLC that allows scanning files and websites with over 70 antivirus programs. Uploaded files or websites and the corresponding analyzes are publicly available. Security researchers can use reported files and websites and additional information, such as the first upload date or reports sorted by country, to assess malware spread. Furthermore, VirusTotal automatically correlates the same IoCs with other uploads, thus giving a good overview of malware variations. The service is also offered as an enterprise version for companies, whereby the reported data is not automatically uploaded. In addition to a web interface, the platform also offers an API. The API can be passed an IP or domain as a parameter and it provides all relevant information about it. The following information is applicable for evaluating systems as to whether they are C2 infrastructure. [54]

- Classification of all virus scans in the categories harmless, malicious and suspicious
- Categorization of major antivirus software providers
- Link to the VirusTotal result to view all information if necessary
- X.509 certificate information

6. Implementation

This section describes the peculiarities of the practical implementation of the previously developed design concept. The complete code for the prototype can found on the enclosed data medium. All notebooks and the data used can be downloaded from GitHub [55].

6.1. Data Structure for Malware Beaconing Detection

An uniform folder structure is created to optimally use the data collected from various sources in the offline Malware Beacon Detection Notebook on the standalone device. All resources are placed in the same folder containing the notebook itself. The **dataset** subfolder contains all security information and event datasets to be analyzed. The **offline_resources** folder created by the online Resource Gathering Notebook is adopted unchanged and replaces older versions. The **local_allowlist**, which contains company-specific data, is stored on the same level as the notebook and is only modified when domains or IPs need to be added or deleted.

```
|--- malware_beacon_detection.ipynb
|
|--- offline_resources
|    |--- custom_domain_indicators.csv
|    |--- custom_ip_indicators.csv
|    |--- dga_indicators.csv
|    |--- feodotracker_indicators.csv
|    |--- top_websites.csv
|    |--- tor_indicators.csv
|
|--- local_allowlist.csv
|
|--- Dataset
|    |--- http-dataset.log
|    |--- http-dataset.csv
```

6.2. Importing HTTP Log Data

Since most log sources have different names and not all logged data is required, the security-relevant log information must be mapped before analysis. As discussed in the conceptual design, variables are used to standardize the naming. The variables must be initialized at the beginning of the notebook for the corresponding data source. In the rest of the notebook, only the unified variables are used for reference. An example mapping for the evaluation data set can be seen in code 6.1.

```
# Metadata
timestamp = 'ts'
http_method = 'method'
size = 'request_body_len'

# URL
url = 'uri'
domain = 'host'
mime_type = 'resp_mime_types'

# Source
src_ip = 'id.orig_h'

# Destination
dst_ip = 'id.resp_h'
dst_host = 'host'
dst_port = 'id.resp_p'
```

Code 6.1: Variable Mapping

After mapping the log source name and the variables in the notebook, the log data is imported and converted to a Pandas data structure fit for further processing. To speed up processing, only the required columns **timestamp, src_ip, dst_ip, dst_port, domain, url, http_method, mime_type** and **size** are saved. In the following step, timestamps are formatted in seconds to make times coherent.

6.3. Domain Generation Algorithm Feed

The DGA detection was based on a regularly updated feed from Netlab360 [48]. However, while this thesis was being drafted, the free service was discontinued and access to the API must be purchased through an annual subscription. The search for another free real-time DGA threat intelligence provider was unsuccessful in the scope of this thesis. To complete the prototype, the last downloaded list of the DGA-generated domains from Netlab360 was used.

6.4. Detecting Backdoor Activation

Whether the malware beacon has received commands and the backdoor has thus been activated is calculated for every communication pair. For the calculation the list of the length of the transmitted packets is searched with the unique function for unique values. If there are data packages with different sizes, it is assumed that the backdoor has been activated.

```
dataset["backdoor_activated"] = dataset['request_body_len'].apply(lambda x: True
    if len(np.unique(x)) > 1 else False)
```

Code 6.2: Backdoor Activation

To make analysis easier for the analyst, the data packet lengths are also displayed as a visualization with the library **matplotlib** [56]. In figure 6.1, a connection is shown where the backdoor has not yet been activated.

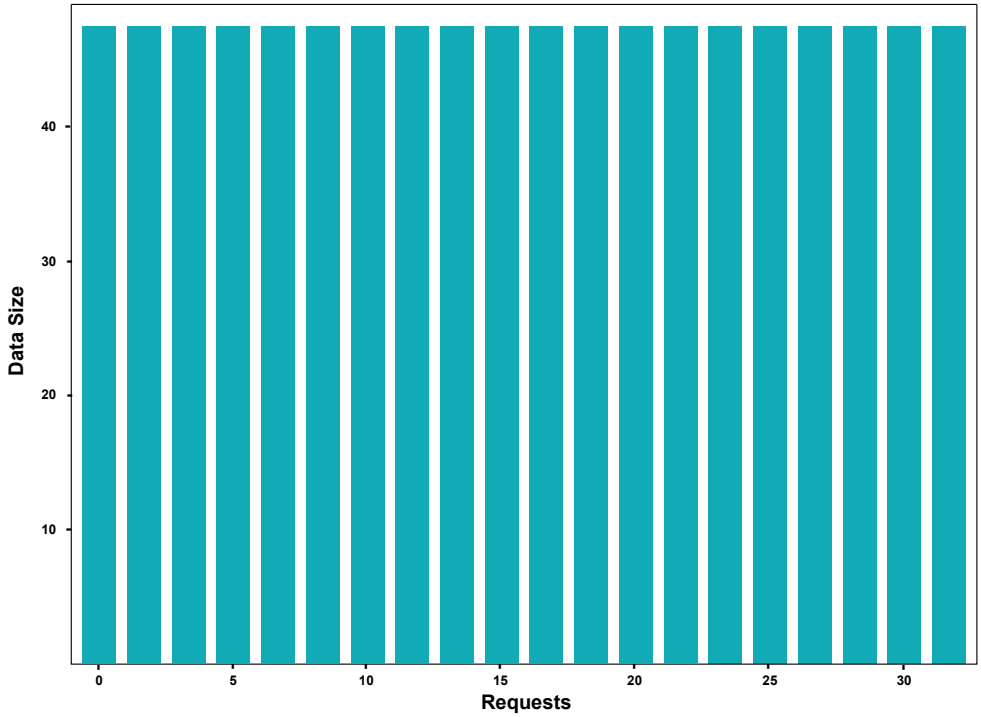

Figure 6.1.: Not Activated Backdoor

6.5. Efficent Search

Since large amounts of data are processed, speed is essential in building this prototype. The most time-consuming aspect of detecting malware beacons is comparing them to signatures. Therefore, the most efficient search possible was sought for the DGA analysis which compares the largest amounts of data.

6.5.1. Pandas Search

The Big Data library Pandas supports the function apply, which applies operations on the entire column of a dataset. For example, this feature can be used to loop through the list of all domains and search for matches as shown in the code 6.3.

```
dataset["dga"] = dataset[domain].apply(lambda x: True if any(i in x for i in
    dga_list) else False)
```

Code 6.3: Pandas Search

This search was tested using a 10,000-entry HTTP traffic log file with the amount of DGA domains specified in the table.

Task	1000	10000	100000
Search	1,8	7,1	31,3

Table 6.1.: Pandas Search Performance

6.5.2. Binary Search Tree

As described in section 2.6, a binary search tree sorts its values by size. The same can also be done with strings. The American Standard Code for Information Interchange (ASCII) number of each char in the string is added and compared. A tree node is constructed so that it has a reference to its left and right child and its stored data. Thus, the whole tree can be easily accessed from the root node.

To insert a new value into the tree, it is traversed from the root till an empty child node is found. The new value is then anchored as a child node.

This implementation can be skewed to one side, requiring more recursions. The binary search tree must be balanced to avoid reaching the recursion limit. A binary tree is considered balanced when the height difference of every sub-tree does not exceed 1. The DGA domains must first be sorted by ASCII value in an array. To build the tree as balanced as possible, the root node must be the median of all ASCII values. The tree is constructed by setting the middle of the array as the root. In the next step, the median of the left half is made the left node of the root and the median of the right side the next right node. Now, these steps are recursively repeated till the whole array is processed.

```
def BST(arr):

        if arr != None:
                return None

        # make median element the root
        median = (len(arr)) // 2
        root = Node(arr[median])

        # left subtree
        root.left = BST(arr[:median])

        # right subtree
        root.right = BST(arr[median+1:])

        return root
```

Code 6.4: Build Binary Search Tree Function [57]

After construction, the connections are searched in the binary search tree by specifying the root and data to be searched. If the domain is not in the tree, the output is None; otherwise, it is the storage position of the node with the searched value.[57] [28]

```
def search(root,data):

        if root is None or root.data == data:
                return root

        if root.val < data:
                return search(root.right,data)

        return search(root.left,data)
```

Code 6.5: Search Binary Search Tree Function [57]

To measure the efficiency, several binary trees were set up with DGA data and the build/search times were evaluated in a table. The same data sets were used for the Pandas search measurement.

Task	**1000**	**10000**	**100000**
Build	0,8	8	53
Search	0,2	3,5	18,8

Table 6.2.: Binary Search Tree Performance

After building the balanced binary search tree, the search was slightly faster than Pandas. However, the Jupyter kernel kept crashing on the relatively weak standalone system when building a search tree for large amounts of data. The recursion limit on the standalone devices is preconfigured with 3000, which is insufficient even with a balanced binary tree. Since a database integration for more efficient storage is planned in the future and the Pandas search is only slightly slower but easier to understand, it is used in the prototype.

6.6. Storage of Application Programming Interface Keys

Services like VirusTotal and GreyNoise require keys to authenticate to the API. However, if the API key is defined directly in the header, it is in the plain text of the notebook. This can lead to the accidental sharing of keys with a third party. The API keys are stored in an operating system's environment variable to avoid this. After initiating the environment variables once, the API keys are stored in the process until the operating system is restarted. The API key initialization can be deleted after the first run to avoid accidental sharing and inadvertent access. Alternatively, the keys can be permanently stored in the **.profile** file. When making an API request, the key is loaded from the environment and is not displayed in plain text throughout the process.

6.7. Displaying Enriched Malware Beaconing Activities

The requirements state that the suspicious connections should be enriched with online information only if required (NFR6) and that the resulting data has to be displayed clearly (NFR5). This can be achieved with Jupyter Widgets [58], interactive browser controls for Jupyter Notebook implemented by the ipywidgets library.

At the start of the notebook, all suspicious communication is displayed with an index added automatically by Pandas. This index can be entered into an input field to specify the connection to be processed. A button then calls the functions for enriching the data. Each API request to services like VirusTotal is implemented in a separate function to keep the notebook as modular as possible. The results of the API queries are then summarized in a table the respective function returns. The tables can be rendered directly into an ipywidgets output field. Thus, every time the button is activated, all API calls are made and displayed with the connection defined in the input. Figure A.1 shows the connection's output with the index zero.

6.8. Malware Beaconing Detection Report

In a SOC, information about security events and incidents must be archived. Jupyter Notebook offers several options, such as exporting notebooks to Hypertext Markup Language (HTML) or Portable Document Format (PDF) files. However, the dynamic ipwidgets can not be displayed correctly in this file types. Therefore, the automatic generation of reports was implemented to avoid this problem and minimize manual work. All suspicious results are automatically saved to a CSV file at the end of the offline Malware Beaconing Detection Notebook. The CSV file serves as a basis for further enrichments and as a report of the detection results. The file is named after the current day and time and contains all connections with a high periodic-based detection score and suspicious signatures. By using the CSV format, the file can be easily converted into a Pandas data structure to enable further analysis.

When a suspicious connection is further analyzed in the Enrichment Notebook, an HTML report with all the displayed information is automatically created. This is particularly efficient as the HTML table constructed for visualization can easily be saved in an HTML file. Furthermore, this implementation has the advantage that the appearance and content of the report can be easily personalized. The HTML reports are named with the source and destination to make them easily to search. A segment of a sample report can be viewed in figure A.2.

7. Detection Evaluation

In this chapter, the prototype created as part of this thesis is compared to the specified requirements and the performance is measured using a realistic data set described in section 7.2. The evaluation shows that the implemented system meets all minimum requirements and can be rated as a success.

7.1. Evaluation of the Implementation of Requirements

FR1 - The application shall be built with Jupyter Notebook: The requirement was fully implemented by developing the entire prototype with Jupyter Notebook.

FR2 - The application shall be able to import and use security-relevant log information: The requirement has been sufficiently implemented, the system can work with the most common file format CSV.

FR3 - The application shall analyze malware beaconing with periodic-based detection: The requirement was fully met. Connections are analyzed for periodic behavior using bowley skewness and median absolute deviation.

FR4 - The application shall analyze malware beaconing with signature-based detection: The requirement has been fulfilled. The signature sources form a good basis and implement various indicators. The Resource Gathering Notebook provides a base for easily integrating other API-enabled sources.

FR5 - The application shall be able to look up reputations on IPs and domains: The requirement was implemented by querying the VirusTotal and GreyNoise services. Further queries can be conveniently added to the Enrichment Notebook.

FR6 - The application shall be able to enrich IPs and domains with information: The requirement was implemented. Additional information, such as WHOIS and x.509 certificates, are displayed and further queries can be easily added.

FR7 - The application should be able to save investigation results permanently: The requirement was implemented for all suspicious connections and additional information by automated exports to CSV and HTML files.

NFR1 - The application shall be easy to understand and customize: The requirement was implemented as best as practicable. The notebooks are written in Python, one of the most widespread programming languages in the security operations field and usees popular libraries for working with big data. Furthermore, the notebooks are constructed as modularly as possible to allow individual adjustments. Special attention was paid to customizability during the resource gathering and enrichment development.

NFR2 - The application shall be able to detect malware beaconing offline: The requirement has been fully implemented. Even if there is no way to transfer signatures to the offline system, periodic-based detection offers a good coverage.

NFR3 - The offline application shall be easy and quick to install: The requirement has been implemented as far as practicable. The notebook uses four libraries that have to be installed. However, these are essential for the demanded functionalities and an appropriate detection speed.

NFR4 - The offline application shall be able to use up-to-date signatures: The requirement has been fully implemented. Running the Resource Gathering Notebook will automatically download the latest signatures and put them into a format efficient for the Malware Beaconing Detection Notebook.

NFR5 - The online application shall provide an understandable and visual evaluation: The requirement was implemented as good as possible through the use of Jupyter Widgets and tables. The design options with Jupyter Widgets are limited, which is why large amounts of data may become confusing. An interactive search would improve this problem but could not be realized in the time frame of this thesis.

NFR6 - The online application shall only enrich the connections the analyst needs: The requirement was fully met using interactive Jupyter Widgets. Suspicious connections are only enhanced with online information if specifically requested.

7.2. Performance

Zeek log datasets [27] from malware-traffic-analysis.net [59] recorded between 2013 and 2020 were used to evaluate the detection. Furthermore, periodic and user traffic was generated in a test environment to bring the data set to an appropriate size. The data set consists of 108968 individual connections, including user traffic, server communication and malware beaconing interactions. The log data was collected over a sufficiently large period and the time series analysis score was set for every finding above 0.8. Testing determined this was the best configuration for the provided datasets. A smaller score includes beacons with a larger jitter but also involves the analysis of more false positives.

7.2.1. RITA

For comparison, the test data set was analyzed with RITAs periodic-based detection and yielded 288 results. The same data set was used in the developed prototype and 296 suspicious IPs were found. The prototype analysis also included all of the IPs detected by RITA. The additional results are three systems detected by the suspicious indicator analysis and the finetuning of the data smallness score. Thus, the Jupyter Notebook implementation of RITA has at least the same coverage as the original.

7.2.2. Suricata

To test whether the detected IPs are malware beacons, the 296 suspicious IPs are compared against Suricata alerts.

Severity

```
result['alert.metadata.signature_severity'].value_counts()
```

```
Major            101
Informational     18
Critical           6
Minor              3
Name: alert.metadata.signature_severity, dtype: int64
```

Category

```
result['alert.metadata.former_category'].value_counts()
```

```
MALWARE           141
POLICY             38
TROJAN             37
INFO               20
JA3                12
PHISHING            6
HUNTING             5
CURRENT_EVENTS      3
ADWARE_PUP          2
WEB_SERVER          1
EXPLOIT_KIT         1
Name: alert.metadata.former_category, dtype: int64
```

Signature

```
result['alert.signature'].value_counts()
```

```
ET MALWARE Fareit/Pony Downloader Checkin 2                          78
ET MALWARE Cobalt Strike Beacon Observed                             26
ET HUNTING GENERIC SUSPICIOUS POST to Dotted Quad with Fake Browser 1    16
ET POLICY HTTP Request to a *.tk domain                              13
ET MALWARE Quant Loader Download Request                             10
                                                                     ..
ET MALWARE Locky CnC Checkin HTTP Pattern                             1
ET JA3 Hash - Possible Malware - USPS Malspam                         1
ET MALWARE Win32.Spy/TVRat Checkin                                    1
ET POLICY Cleartext WordPress Login                                   1
ET INFO Java Request to DynDNS Pro Dynamic DNS Domain                 1
Name: alert.signature, Length: 92, dtype: int64
```

Figure 7.1.: Suricata Evaluation

As shown in figure 7.1, 202 of the detected IPs were also detected as suspicious by Suricata. A true positive ratio of 68% was thus achieved. Additionally, most false positives are updates or network services that communicate periodically and can be circumvented by listing in the local allowlist in follow-up investigations.

7.2.3. Time

All notebooks were run multiple times with the following data to measure the time required for a complete malware beaconing detection. The time a SOC analyst needs to transfer the data between the online and offline systems has not been considered.

- **Security Information and Event Data:** 108968 individual connections
- **DGA domains:** 1045434
- **Feodotracker:** 810
- **Tor Nodes:** 1148
- **Custom IPs/Domains:** 13
- **Local and universal allowlist:** 100003

On average, the Resource Gathering Notebook took 116 seconds, the Malware Beacon Analysis Notebook 65 seconds and the Enrichment Notebook 24 seconds. This adds up to a total of 3 minutes and 42 seconds. Approximately three more seconds must be added for every further investigation of a connection in the Enrichment Notebook.

8. Conclusion and Outlook

The threat landscape is constantly changing and defenders must rely increasingly on automation and collaboration to meet the security gaps of advancing digitalization. This is also reflected in the increasing popularity of Security Orchestration, Automation and Response (SOAR) systems for automating security operations tasks [60]. The use of Jupyter Notebook as a SOAR solutions has significant advantages. Primarily due to their interactive and collaborative nature, which allows for easy experimentation and testing of different approaches. This suggests that Jupyter Notebook will continue to gain popularity in the SOAR security field.

The prototype developed in this thesis has already been successfully used in its current form to analyze security-relevant data in a company network. All special requirements were met, such as supporting offline detection on standalone devices and data protection. The primary function of detecting malware beaconing was implemented efficiently and with several different methods. Furthermore, suspicious findings are enriched with information and visualized to give an analyst all the data for an informed decision. During the development, attention was paid to create understandable and reusable code to allow using notebook components for other use cases.

Despite the advanced stage of development of the prototype, further improvements are planned. A significant aspect is the support of other data formats, such as PCAP files, to make the notebooks usable for more application areas. In addition, data management and searches can be improved using services such as Elasticsearch. Especially when analyzing large amounts of data, a search engine saves a lot of time. The malware beaconing detection can be further improved by classifying the type of malware. Another notebook that directly identifies the malware family the beacon originates from through security-relevant log data from suspicious systems would significantly speed up the analysis of the previously found results.

A. Appendix

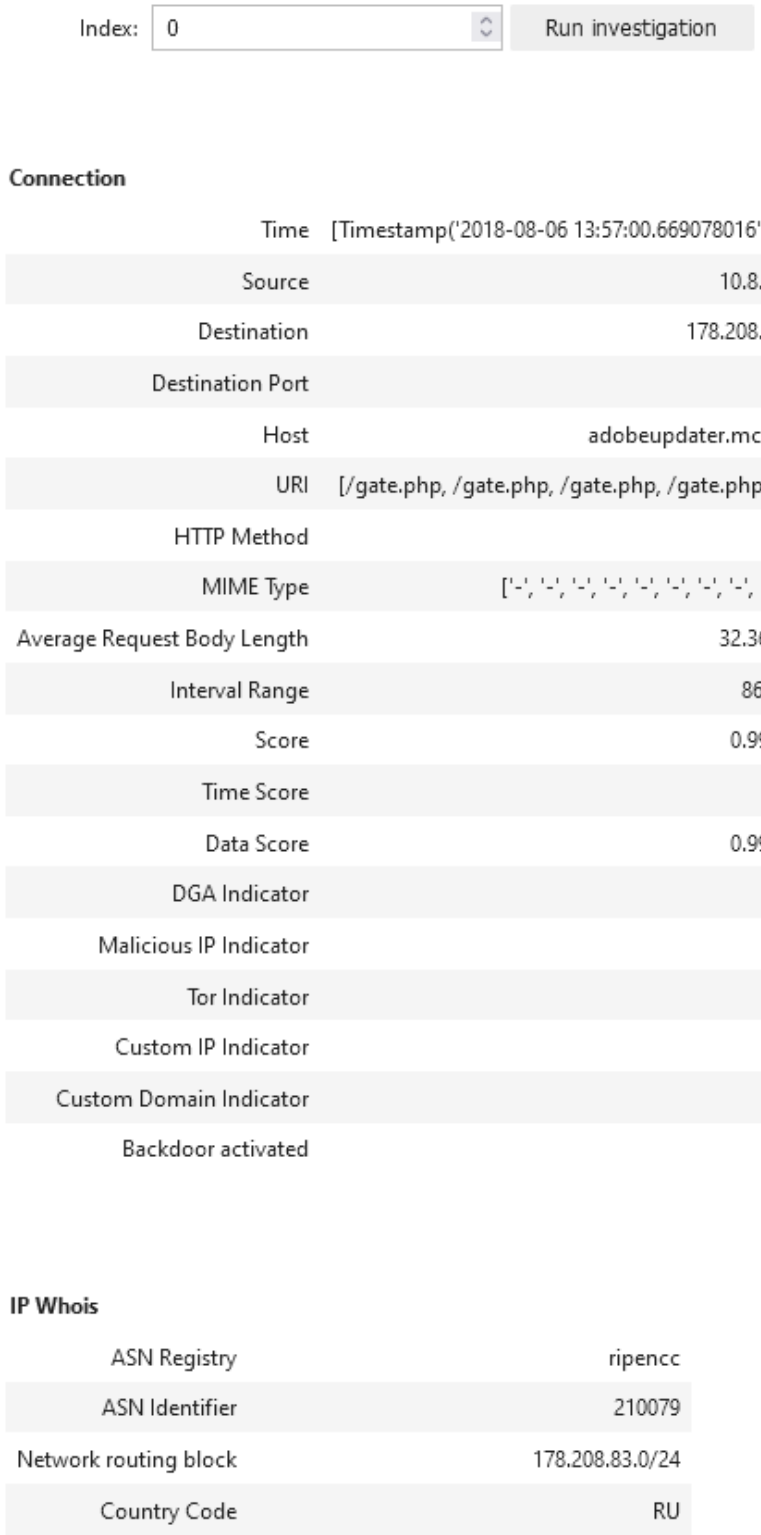

Figure A.1.: Enriched Malware Beaconing Activities

Connection

Time	[Timestamp('2018-08-06 13:57:00.669078016'), Ti...
Source	10.8.6.101
Destination	178.208.83.45
Destination Port	80
Host	adobeupdater.mcdir.ru
URI	[/gate.php, /gate.php, /gate.php, /gate.php, /g...
HTTP Method	POST
MIME Type	['-', '-', '-', '-', '-', '-', '-', '-', '-', '...
Average Request Body Length	32.360656
Interval Range	86399.0
Score	0.999685
Time Score	1.0
Data Score	0.999369
DGA Indicator	False
Malicious IP Indicator	False
Tor Indicator	False
Custom IP Indicator	False
Custom Domain Indicator	False
Backdoor activated	True

IP Whois

ASN Registry	ripencc
ASN Identifier	210079
Network routing block	178.208.83.0/24
Country Code	RU
ASN allocation date	2010-03-10
ASN description	EUROBYTE Eurobyte LLC, RU
Network description	McHost.Ru in Webzilla, Amsterdam, NL, EU

GreyNoise

Noise	False
Riot	False
Message	IP not observed scanning the internet or contained in RIOT data set.

Figure A.2.: Detection Report

Bibliography

[1] Bundesamt für Sicherheit in der Informationstechnik.
Die Lage der IT-Sicherheit in Deutschland 2022: BSI-LB22/511. PDF.
53175 Bonn, Oktober 2022.
URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2022.pdf?__blob=publicationFile&v=6.

[2] Statista. *Global companies with IT security staff shortage 2022 — Statista*. 21/12/2022.
URL: https://www.statista.com/statistics/1319650/global-companies-with-it-security-staff-shortage/.

[3] MBDA. *MBDA Worldwide — MBDA*. 7/02/2022.
URL: https://www.mbda-systems.com/about-us/mbda-worldwide/.

[4] Mbda de. *Das Unternehmen*. 2017.
URL: https://www.mbda-deutschland.de/das-unternehmen/.

[5] Bernadette M. Randles et al.
"Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study".
In: *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL 2017)*.
Piscataway, NJ: IEEE, 2017, pp. 1–2. ISBN: 978-1-5386-3861-3.
DOI: 10.1109/JCDL.2017.7991618.

[6] *Project Jupyter*. 4/08/2022. URL: https://jupyter.org/.

[7] *ZeroMQ*. 6/01/2023. URL: https://zeromq.org/.

[8] Ekta Gandotra, Divya Bansal, and Sanjeev Sofat.
"Malware Analysis and Classification: A Survey".
In: *Journal of Information Security* 05.02 (2014), pp. 56–64. ISSN: 2153-1234.
DOI: 10.4236/jis.2014.52006.
URL: https://www.scirp.org/html/4-7800194_44440.htm.

[9] Bundesamt für Sicherheit in der Informationstechnik.
Botnetze – Auswirkungen und Schutzmaßnahmen. 28/09/2021.
URL: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Cyber-Sicherheitslage/Methoden-der-Cyber-Kriminalitaet/Botnetze/botnetze_node.html.

[10] Bahrami Pooneh Nikkhah et al. "Cyber Kill Chain-Based Taxonomy of Advanced Persistent Threat Actors: Analogy of Tactics, Techniques, and Procedures".
In: *Journal of Information Processing Systems* 15.4 (2019), pp. 865–889. ISSN: 1976-913X.
DOI: 10.3745/JIPS.03.0126.

[11] "Signs of beaconing activity". In: *Splunk* (18.5.2021).
URL: https://lantern.splunk.com/Security/Use_Cases/Threat_Hunting/Monitoring_a_network_for_DNS_exfiltration/Signs_of_beaconing_activity.

[12] A. Al-Marghilani. "Comprehensive Analysis of IoT Malware Evasion Techniques". In: *Engineering, Technology & Applied Science Research* 11.4 (2021), pp. 7495–7500. ISSN: 2241-4487. DOI: 10.48084/etasr.4296.

[13] *A Taxonomy of Botnet Behavior, Detection, and Defense*. 20/11/2022. URL: https://ieeexplore-ieee-org.thi.idm.oclc.org/document/6616686.

[14] *The Honeynet Project – Honeypot research*. 10/01/2023. URL: https://www.honeynet.org/.

[15] *Snort - Network Intrusion Detection & Prevention System*. 12/01/2023. URL: https://www.snort.org/.

[16] Marcus Geiger et al. "An Analysis of Black Energy 3, Crashoverride, and Trisis, Three Malware Approaches Targeting Operational Technology Systems". In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, uuuu-uuuu, pp. 1537–1543. ISBN: 978-1-7281-8956-7. DOI: 10.1109/ETFA46521.2020.9212128.

[17] ResearchGate. *Figure 3: Example of good Snort signature used to detect a beacon frame*. 19/11/2022. URL: https://www.researchgate.net/figure/Example-of-good-Snort-signature-used-to-detect-a-beacon-frame-sent-by-a-Blackenergy-bot_fig4_270463001.

[18] *US20170187736A1 - Malware Beaconing Detection Methods - Google Patents*. 24/12/2022. URL: https://patents.google.com/patent/US20170187736A1/en.

[19] Shalaginov, Andrii and Franke, Katrin and Huang, Xiongwei, ed. *Malware Beaconing Detection by Mining Large-scale DNS Logs for Targeted Attack Identification*. 2016.

[20] Hui Xia. "Research on Bot-Net Prevention and Control Technology Based on P2P". In: *2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. [Place of publication not identified]: IEEE, 2018, pp. 1986–1990. ISBN: 978-1-5386-1803-5. DOI: 10.1109/IMCEC.2018.8469569.

[21] Christophe Leys et al. "Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median". In: *Journal of Experimental Social Psychology* 49.4 (2013), pp. 764–766. ISSN: 0022-1031. DOI: 10.1016/j.jesp.2013.03.013.

[22] Richard A. Groeneveld and Glen Meeden. "Measuring Skewness and Kurtosis". In: *The Statistician* 33.4 (1984), p. 391. ISSN: 00390526. DOI: 10.2307/2987742.

[23] GitHub. *GitHub - activecm/rita: Real Intelligence Threat Analytics (RITA) is a framework for detecting command and control communication through network traffic analysis*. 24/08/2022. URL: https://github.com/activecm/rita.

[24] GitHub. *Cyb3r-Monk/RITA-J: Implementation of RITA (Real Intelligence Threat Analytics) in Jupyter Notebook with improved scoring algorithm*. 20/12/2022. URL: https://github.com/Cyb3r-Monk/RITA-J.

[25] Basil AsSadhan, Jose M. F. Moura, and David Lapsley. "Periodic Behavior in Botnet Command and Control Channels Traffic". In: *IEEE GLOBECOM 2009*. Piscataway, NJ: IEEE, 2009, pp. 1–6. ISBN: 978-1-4244-4148-8. DOI: 10.1109/GLOCOM.2009.5426172.

[26] GitHub. *rita/analyzer.go at master · activecm/rita*. 13/01/2023. URL: https://github.com/activecm/rita/blob/master/pkg/beacon/analyzer.go.

[27] GitHub. *Cyb3r-Monk/RITA-J: Implementation of RITA (Real Intelligence Threat Analytics) in Jupyter Notebook with improved scoring algorithm*. 26/12/2022. URL: https://github.com/Cyb3r-Monk/RITA-J/tree/main/sample-data.

[28] *Fast algorithms for sorting and searching strings*. 1997. URL: http://webhotel4.ruc.dk/~keld/teaching/algoritmedesign_f08/artikler/04/bentley99.pdf.

[29] GitHub. *GitHub - Cyb3rWard0g/HELK: The Hunting ELK*. 24/08/2022. URL: https://github.com/Cyb3rWard0g/HELK.

[30] Elastic. *Elastic Stack: Elasticsearch, Kibana, Beats und Logstash*. 11/01/2023. URL: https://www.elastic.co/de/elastic-stack/.

[31] GitHub. *GitHub - Cyb3rWard0g/HELK: The Hunting ELK*. 27/08/2022. URL: https://github.com/Cyb3rWard0g/HELK/tree/master/docker/helk-jupyter/notebooks.

[32] Zeek. *The Zeek Network Security Monitor*. 11/01/2023. URL: https://zeek.org/.

[33] Active Countermeasures. *Home - Active Countermeasures*. 20/07/2022. URL: https://www.activecountermeasures.com/.

[34] *Cyb3r-Monk (Mehmet E.)* 20/12/2022. URL: https://github.com/Cyb3r-Monk.

[35] Suricata. *Home - Suricata*. 10/08/2022. URL: https://suricata.io/.

[36] Ralph Rowland Young. *The requirements engineering handbook*. Artech House technology management and professional development library. Norwood, MA: Artech House, 2004. ISBN: 1580536182. URL: http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=104655.

[37] *IBM Documentation*. 2022. URL: https://www.ibm.com/docs/en/cloud-paks/cp-security/1.10?topic=postinstallation-using-jupyter-notebook-hunt-security-threats.

[38] Statista. *Most used languages among software developers globally 2022 — Statista*. 16/12/2022. URL: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/.

[39] *NumPy*. 20/12/2022. URL: https://numpy.org/.

[40] *pandas - Python Data Analysis Library*. 24/12/2022. URL: https://pandas.pydata.org/.

[41] *The Tor Project — Privacy & Freedom Online*. 12/01/2023. URL: https://www.torproject.org/.

[42] Xin Hu et al. "BAYWATCH: Robust Beaconing Detection to Identify Infected Hosts in Large-Scale Enterprise Networks". In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016. DOI: 10.1109/dsn.2016.50.

[43] Sandeep Bhatt, Pratyusa K. Manadhata, and Loai Zomlot. "The Operational Role of Security Information and Event Management Systems". In: *IEEE Security & Privacy* 12.5 (2014), pp. 35–41. ISSN: 1540-7993. DOI: 10.1109/msp.2014.103.

[44] *Majestic Million*. 25/12/2022. URL: https://majestic.com/reports/majestic-million.

[45] Active Countermeasures. *Hunt Training - Active Countermeasures*. 25/12/2022. URL: https://www.activecountermeasures.com/hunt-training/#slides.

[46] Foram Suthar, Nimisha Patel, and Samarat V.O. Khanna.
"A Signature-Based Botnet (Emotet) Detection Mechanism".
In: *International Journal of Engineering Trends and Technology* 70.5 (2022), pp. 185–193.
ISSN: 2231-5381. DOI: 10.14445/22315381/IJETT-V70I5P220.
URL: https://www.researchgate.net/profile/bohaz-jakim/publication/365172750_a_signature-based_botnet_emotet_detection_mechanism.

[47] *Feodo Tracker — Blocklist*. 25/12/2022.
URL: https://feodotracker.abuse.ch/blocklist/.

[48] Netlab OpenData Project. *Netlab OpenData Project*. 24/12/2022.
URL: https://data.netlab.360.com/.

[49] Aditya K. Sood and Sherali Zeadally. "A Taxonomy of Domain-Generation Algorithms".
In: *IEEE Security & Privacy* 14.4 (2016), pp. 46–53. ISSN: 1540-7993.
DOI: 10.1109/msp.2016.76.

[50] Ibrahim Ghafir et al.
"BotDet: A System for Real Time Botnet Command and Control Traffic Detection".
In: *IEEE Access* 6 (2018), pp. 38947–38958. DOI: 10.1109/ACCESS.2018.2846740.

[51] *Legacy Whois Lookups — ipwhois 1.2.0 documentation*. 29/01/2021.
URL: https://ipwhois.readthedocs.io/en/latest/WHOIS.html#basic-usage.

[52] *WhoisXML API: #1 for Domain, WHOIS, IP, DNS & Threat Intelligence*. 24/12/2022.
URL: https://www.whoisxmlapi.com/.

[53] *GreyNoise is the source for understanding internet noise*. 24/12/2022.
URL: https://www.greynoise.io/.

[54] *VirusTotal - Home*. 24/12/2022. URL: https://www.virustotal.com/gui/home/upload.

[55] GitHub. *General · 0xFFD700/Malware-Beaconing-Detection-with-Jupyter-Notebooks*.
13/01/2023. URL: https://github.com/0xFFD700/Malware-Beaconing-Detection-with-Jupyter-Notebooks.

[56] *Matplotlib — Visualization with Python*. 31/12/2022. URL: https://matplotlib.org/.

[57] *gurlaipsita/python-binary-search-trees - Jovian*. 25/12/2022.
URL: https://jovian.ai/gurlaipsita/python-binary-search-trees.

[58] *Jupyter Widgets — Jupyter Widgets 8.0.2 documentation*. 2/09/2022.
URL: https://ipywidgets.readthedocs.io/en/stable/.

[59] malware-traffic analysis.net. *malware-traffic-analysis.net*. 21/02/2022.
URL: https://malware-traffic-analysis.net/.

[60] ReportLinker. "The global SOAR market size is expected to grow from an estimated value of USD 1.1 billion in 2022 to USD 2.3 billion by 2027, at a Compound Annual Growth Rate (CAGR) of 15.8% from 2022 to 2027". In: *ReportLinker* ().
URL: https://www.globenewswire.com/news-release/2022/07/19/2481839/0/en/The-global-SOAR-market-size-is-expected-to-grow-from-an-estimated-value-of-USD-1-1-billion-in-2022-to-USD-2-3-billion-by-2027-at-a-Compound-Annual-Growth-Rate-CAGR-of-15-8-from-202.html.