FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Exploring Vulnerabilities of the Internet of Things Devices
Student:	Bc. Zdena Tropková
Supervisor:	Ing. Jiří Dostál, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2023/2024

Instructions

The recent increase of Internet of Things (IoT) devices in all areas makes them an attractive target for attackers. For mitigation of security risks is essential to know common threats and vulnerabilities. In the latest IoT vulnerability ranking created by OWASP in 2018 were the top three vulnerabilities weak passwords, insecure network services, and insecure ecosystem interfaces. The OWASP creates its lists based on public sources and gets data from the contributions of companies as well. After gathering and processing all data, the result is reviewed and published.

The main goal of this thesis is to create an Internet of Things vulnerability ranking list and analyze the explored vulnerabilities.

1. Get familiar with security vulnerabilities in IoT systems and devices.
2. Analyze the current status of vulnerability ranking lists (e.g., OWASP IoT Top 10).
3. Gather data from multiple sources like CVE databases, exploit databases, articles, etc.
4. Create your own IoT vulnerability ranking list based on the created dataset.
5. Choose at least three vulnerabilities from the created list and analyze them in the context of IoT. Describe possible defense and attack vectors.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 15 January 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

Exploring Vulnerabilities of the Internet of Things Devices

Bc. Zdena Tropková

Department of Information Security
Supervisor: Ing. Jiří Dostál, Ph.D.

May 3, 2023

Acknowledgements

I would like to thank my supervisor Ing. Jiří Dostál, Ph.D. for the academic help and guidance. Furthermore, I am also grateful to my boyfriend Petr, family, and friends for their support during my studies and work on this thesis.

Declaration

In Prague on May 3, 2023 ....

Citation of this thesis

Tropková, Zdena. *Exploring Vulnerabilities of the Internet of Things Devices.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstrakt

V této diplomová práce představujeme žebříček deseti nejčastějších zranitelností objevujících se v zařízeních Internetu Věcí. Hlavním cílem práce bylo vytvořit žebříčky z veřejných dat s transparentním postupem protože nyní neexistují žebříčky, které by tyto požadavky splňovaly. Například projekt OWASP publikoval nejnovější žebříček v roce 2018 a ostatní aktuální žebříčky nemají popsaný postup a doložené zdroje dat. V této práci představujeme žebříček deseti nejčastějších zranitelností v zařízeních Internetu Věcí. Dále prezentujeme podobný žebříček pouze pro kamerová zařízení a nejobvyklejší zranitelnost pro různé kategorie chytrých zařízení. Pro vytvoření žebříčků byl implementován nástroj pro sběr zranitelností ve frameworku Scrapy. Navíc byla provedena analýza třech kategorií zranitelností v kontextu Internetu Věcí. Jedná se o zranitelnosti z kategorií Access Control, Overflow a Password Management.

Klíčová slova Internet věcí, chytrá zařízení, žebříček zranitelností, zranitelnost, vytěžování dat, framework Scrapy

Abstract

We introduce in this thesis a ranking list of the ten most common vulnerabilities in Internet of Things devices. The main aim was to provide ranking lists created from public data with a transparent creation methodology because ranking lists with these requirements currently do not exist. For example, the popular project OWASP published the most recent ranking list in 2018, and other existing up-to-date ranking lists do not provide a transparent creation methodology and used data sources. We introduce in this thesis a ranking list of the ten most common vulnerabilities in Internet of Things devices. Furthermore, we propose a similar ranking list only for camera devices. Also, we present the most common vulnerability for different smart device categories. In addition, the scraping tool for vulnerability collection was implemented in the framework Scrapy, and an analysis of three vulnerabilities in the context of the Internet of Things devices was performed. The selected vulnerability categories are Access Control, Overflow, and Password Management.

Keywords Internet of Things, smart devices, vulnerability ranking list, vulnerability, data mining, framework Scrapy

Contents

List of Figures

List of Tables

Introduction

The emphasis on the security of the Internet of Things devices is still insufficient despite the recent rapid increase in their use. Nowadays, smart devices are available to everyone and used daily in numerous diverse areas. We can easily purchase them to equip our households with smart bulbs, plugs, and other smart appliances. In addition, smart buildings use thermostats and controllers. Many wearables, such as smart watches and rings, are widely available. Furthermore, smart devices like sensors, monitors, and controllers are frequently used in the automotive, transportation, healthcare, and other industries.

Despite the broad area of use of Internet of Things devices, many vendors still do not put enough afford into vulnerability management and product security. Regardless, people equip their households with vulnerable smart devices with wide-known security threats, leading to severe security issues for home networks. The same risks apply to companies and organizations when the attacker can misuse vulnerable Internet of Things devices to gain control over the entire system and network.

However, due to many concerns from organizations and security experts, the situation is slowly improving. For example, by the IoT Security Foundation reports [1], the trend of disclosure policy is gradually increasing. Below 10% of vendors had a disclosure policy in 2018. Later in 2022, over 27% companies offered it. Transparency and vulnerability publishing from vendors is essential for the security of smart devices. Security experts must know common problems and threats to mitigate security risks in systems. Therefore, it needed to have relevant data and reference lists.

For web applications is used the awareness document OWASP TOP 10 [2]. It repeatedly introduces and describes the most common threats and offers a guide on how to prevent them. Companies adopt it and use it in development to secure their software. OWASP also published ranking lists for the Internet of Things devices in 2014 and 2018 [3]. However, with the recent increase in the use of smart devices, it may be outdated. The more recently created ranking lists of smart devices are available online from many different sources, but their creation and data source methodology is not transparent.

The main goal of this thesis is to propose a vulnerability ranking list of Internet of Things devices with clarified methodology and known public data sources. In addition, create the dataset containing vulnerability records about the Internet of Things devices and implement a tool to collect the required data.

The thesis is divided into the following six chapters. The first chapter introduces terms related to this work—Internet of Things, vulnerability, and ranking lists. In the second chapter is described the entire scraping process. The data sources are introduced, and the implementation of the scraping tool is described, including the challenges and structure of the tool. The third chapter is about the created dataset and its methodology. All dataset columns are explained, and device and vulnerability categories are presented. In the fourth chapter, the formed ranking lists are presented. In the last chapter, three vulnerabilities are introduced and analyzed in the Internet of Things security context. Finally, the conclusion of the thesis is proposed with future work to improve the ranking list creation process.

Chapter 1

State-of-the-art

In the first part of this chapter, the term Internet of Things is introduced. Then, vulnerability and associated phrases like CWE a CVSS are explained. Finally, the ranking lists are presented with their methodology—*OWASP TOP 10* and *CWE Top 25 Most Dangerous Software Weaknesses*.

1.1 Internet of Things

The Internet of Things comprises diverse smart devices with various features and purposes. Hence, there is no exact definition for it. For example, this work [4] defines the Internet of Things as devices that are not computers but have computing power and can exchange and process data. Next, [5] describes it as a network of intelligent objects used on daily bases. In addition, [6] presents the Internet of Things as a network of autonomous devices that collect data, exchange information, and perform computations cooperatively.

Furthermore, opinions differ on what exact devices belong to the Internet of Things. For illustration, somebody considers mobile phones or network appliances as smart devices. However, on the opposite, many people think these devices are closer to computers based on their conduct and computation power. Therefore, they do not belong in the Internet of Things. The inclusion of specific device types mostly depends on the used definition in the specific case.

In addition, many devices can be clearly considered the Internet of Things. First, it is intelligent home gadgets like bulbs, sockets, thermostats, doorbells, or locks. Similarly, bigger appliances like fridges, vacuums, or washing machines. Second, wearable devices nowadays broadly used by people—for example, smart watches, rings, or glasses. Next, the gadgets operating in various industries, like sensors, monitors, or controllers. All devices we view as the Internet of Things are presented in section Device Categories.

The popularity of the Internet of Things is still rapidly growing. The statistic from [7] suggests that in 2019 almost 8 billion smart devices were used. Furthermore, it excepts that in 2030 around 25.5 billion Internet of Things devices will be connected to the network. That is an expansion of approximately 200%. Hence, increased use of the Internet of Things in numerous areas and finding new applications in additional industries is to be expected.

1.2 Vulnerabilities

The vulnerability is a system weakness or flaw the malicious actor can misuse for the attack. Vulnerabilities usually arise from programming bugs or incorrect system architecture. [8] The most common vulnerabilities are, for example, cross-site scripting, SQL injection, path traversal, or broken authentication.

1.2.1 Common Weakness Enumeration (CWE)

Common Weakness Enumeration is a list presenting categories of the system's weaknesses. [9] The categories are structured into a tree defining relations between them. For example, CWE-862:*Missing Authorization* is the parent of the weakness category CWE-425:*Direct Request*.

The first CWE list was published in 2006, focusing mainly on software. However, due to the rapid spread of threats to the Internet of Things and other hardware devices, in 2020, the categories related to them were added. [10]

Besides defining the structure of weaknesses, CWE includes extensive information about each category. For example, short and long descriptions. Further, examples of attacks, impacts of weakness, and affected platforms are provided.

The primary purpose of the database is to help developers and security experts understand weaknesses in the systems. Based on that knowledge, they can eliminate weaknesses in design or development, react efficiently to threats, and secure products. Second, CWE offers a united structure to communicate and exchange information about weaknesses.

1.2.2 Common Vulnerabilities and Exposures (CVE)

Common Vulnerabilities and Exposures is a database of published and reviewed vulnerabilities. The main goal of it is to categorize the vulnerabilities from numerous sources affiliated with the CVE program and provide the public with unified vulnerability data records. [11]

Each database record describes one specific vulnerability and has assigned its unique CVE identifier. The ID has a specified form. First is **CVE-** followed by prefix and year of vulnerability publishing. The last part contains at least four random digits. The identifiers are mainly used to efficiently specify vulnerabilities in academic papers, articles, vendor security publications, or software documentation.

Before the vulnerability is officially accepted in the database, it must pass through several acceptance and confirmation stages. Also, the source must provide all required data about vulnerabilities like type, consequences, sources, and affected devices.

1.2.3 Common Platform Enumeration (CPE)

Common Platform Enumeration is a string providing structured information to identify the system or the device part. The format of the CPE string is strictly given because it aims to be automatically processed by computers. Colons separate all 12 parameters like vendor, product, and version. If the parameter is missing, it is substituted with an asterisk. [12]

For example, the CPE string *cpe:2.3:o:planex:cs-wmv02g_firmware:*:*:*:* :*:*:*:**. The first part *cpe:23* defines the CPE format and its version. The following letter *o* specifies a part of the described system. In this case, it is the operating system. The other options are *h* for the hardware and *a* for the application. The next two parameters are vendor *Planex* and product *cs-wmv02g_firmware*. All remaining parameters are missing, and void places are filled with asterisks.

The database containing all CPE records is freely available and updated daily. The CPE strings are primarily used to associate vulnerability records with specific affected products.

1.2.4 Common Vulnerability Scoring System (CVSS)

The Common Vulnerability Scoring System is an open standard providing a formula for calculating the severity of the vulnerabilities. The input metrics for the CVSS formula slightly differ by version and are usually chosen by security experts. However, all versions operate with the following three main metrics groups [13]:

1. **Base:** immutable information about vulnerability
2. **Temporal:** information about a vulnerability that could be modified in future
3. **Enviromental:** vulnerability information related to the concrete environment—for example, specific company or organization

The output contains two parts. First is the computed score ranging from 0 to 10. Based on that value, the vulnerability is assigned a severity category—low, medium, high, and critical. The second part is the vector. It is a string with capital letters representing vulnerability metrics divided with slashes. For example, this *CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H /A:H* is CVSS version 3.1 vector for vulnerability *CVE-2022-3240*.

Companies and security experts use CVSS widely in vulnerability and risk management. It helps them classify vulnerabilities by priority to primarily focus on more severe vulnerabilities and threats. Furthermore, the CVSS score and vector are usually included with records in vulnerability databases.

1.2.4.1 CVSS 2

The CVSS2 was used from 2007 to 2015. It was mainly created to fix issues in the previous version of CVSS. [14] Due to complaints from the public about inconsistencies and difficulties in score calculation, it was replaced in 2015 by version 3.

The CVSS 2 group parameters are the following [14]:

1. **Base:** Access vector, access complexity, authentication and impact metrics (confidentiality, integrity, availability)

2. **Temporal:** Exploitability, remediation level, report confidence

3. **Enviromental:** Collateral damage potential, target distribution and requirements (confidentiality, integrity, availability)

This is an example of CVSS 2 vector *AV:L/AC:H/Au:N/C:N/I:C/A:C* for vulnerability CVE-2010-4027. The score for it is 5.6. Thus, the vulnerability has medium severity. From the vector, we can extract the following information, access vector *(AV:L)* is local, access complexity *(AC:H)* has a value high, and authentication *(Au:N)* is not required. Furthermore, for impact metrics, confidential *(C:N)* impact is none, but integrity *(I:C)* and availability *(A:C)* impacts are complete.

1.2.4.2 CVSS 3

The CVSS3 was released in 2015 to replace CVSS 2. The main goal was to improve scoring based on complaints and requirements from the public. The three main groups remained the same, but some parameters were adjusted. New parameters were added to the group *Base* to describe vulnerability more clearly. Furthermore, in *Enviromental* group, some parameters were removed and substituted by modified base metrics. The last group *Temporal* was not changed. Finally, there is the list of categories and parameters for CVSS 3 [15]:

1. **Base:** Attack vector, attack complexity, privileges required, user interaction, scope and impact metrics (confidentiality, integrity, availability)

2. **Temporal:** Exploit code maturity, remediation level, report confidence

3. **Enviromental:** Modified base metrics and requirements (confidentiality, integrity, availability)

1.2.4.3 CVSS 3.1

The main aim of CVSS 3.1, launched in 2019, was to implement improvements into version 3.0 without any significant changes. Thus, no matrics group parameter was added or modified. However, their descriptions were modified and clarified. By creators, the changes should remind users that CVSS is mainly for severity measures and not only risk management. Also, some changes were applied to attack vector metrics and scoring guidance. [16]

1.2.4.4 CVSS 4

The CVSS 4 is currently in draft. The primary goal of creators is to clarify the score and implement new parameters according to the current security and system state. Some of the approved proposals have already been published. For example, the new main group should be added, and some of the current parameters like *Remediation level* removed. Furthermore, the attack's automation should also be considered in score calculation. For the Internet of Things devices is relevant a proposal to include a kinetic/safety impact metric. [17]

1.3 Vulnerability Ranking Lists

The vulnerability rankings lists usually introduce and describe the most common threats to the systems in the specified area. For example, OWASP TOP 10 is a popular list specializing in web application vulnerabilities. Ranking lists are a crucial tool for developers, vendors, and security experts. It can be used to design proper and secure architecture. Also, the developers may use it to mitigate security flaws in code during development. Furthermore, it helps security experts in risk management to focus on widespread and more potential threats primarily.

1.3.1 OWASP

The Open Web Application Security Project (OWASP) is a non-profit foundation dedicated primarily to web security. [18] It includes numerous projects related to cybersecurity. One of the most recognized is the project **OWASP TOP 10**, a standard used widely by companies, developers, and security experts to secure web applications.

Table 1.1: OWASP vulnerability ranking list for Internet of Things 2018

Position	Vulnerability
1	Weak Guessable, or Hardcoded Passwords
2	Insecure Network Services
3	Insecure Ecosystem Interfaces
4	Lack of Secure Update Mechanism
5	Use of Insecure or Outdated Components
6	Insufficient Privacy Protection
7	Insecure Data Transfer and Storage
8	Lack of Device Management
9	Insecure Default Settings
10	Lack of Physical Hardening

1.3.1.1 OWASP TOP 10

OWASP TOP 10 is a periodically published ranking list of the top ten most common web application vulnerabilities. [2] The last one was posted in 2021. The ranking lists aim to improve web application security by pointing out widespread vulnerabilities. Thus, all listed vulnerabilities are associated with descriptions, examples, and prevention advice.

The OWASP uses numerous data sources to create a list—companies, vendors, or bug bounty programs. The acquired data are verified and reviewed before acceptance to the dataset. In further processing, the data are normalized, and vulnerability CWEs are revised. The final placement is formed based on the number of vulnerable applications with specific CWE instead of the count of the CWE itself. This approach is adopted because a single application may have multiple identical CWEs. Furthermore, the result computation also considers the CWSS score. [2]

1.3.1.2 OWASP Internet of Things

In 2014, OWASP launched a project dedicated solely to the Internet of Things devices. [3] Similarly to OWASP TOP 10, it aims to help with the security of smart devices by providing a ranking list to vendors and companies. Since the project started, two lists were released in 2014 and 2018.

The top 10 vulnerabilities for the Internet of Things for 2018 are presented in the table 1.1.

1.3.2 CWE Top 25 Most Dangerous Software Weaknesses

The ranked lists made by CWE present 25 of the most common weaknesses in software. [19] The latest list was published in 2022 and was created from over

37 000 CVE records from the preceding two years. As ranking lists described above, it should help spread awareness about common weaknesses and help with risk management.

CWE uses as a data source for the ranking list a public database of CVE records—National Vulnerability Database maintained by NIST. First, the data are collected for the desired period. Second, the acquired records are divided into groups by weakness type and then automatically and manually examined for inaccurate CWE assignation. Finally, the placement is calculated by the formula considering vulnerability prevalence and CVSS score. [19]

Chapter 2

Data Scraping

First, we introduce in this chapter the process of dataset creation. Second, we propose sources of vulnerability data and lists of keywords. Next, the implementation of the scraping tool is described. Thus, we present tools and libraries used for scraping. Also, implementing requirements and challenges, like the Internet of Things vulnerability detection, are introduced. Finally, scrapers are individually presented with their functions.

2.1 Proposal

To be able to create ranking lists, we needed to collect data and process them into the dataset. First, we needed to choose public sources of Internet of Things vulnerability. We aimed to find sources that are possible to scrape and offer a satisfactory amount of vulnerability records. Furthermore, the vulnerability records needed to be in a form that can be processed at least partly automatically. Based on these requirements, we selected three sources that are presented in detail in the next section Data Sources. One of the sources is a database containing all CVE records. Thus, our next goal was to find additional sources that collect vulnerabilities also without assigned CVE. That fulfills, for example, exploit databases. We did not use articles about vulnerabilities because they almost every time covered the vulnerability with assigned CVE, which is already covered from the source of the CVE database.

Because the chosen sources also include vulnerabilities not associated with the Internet of Things, we needed to be able to search for required vulnerabilities. We proposed keywords that we used for that. They cover types of smart devices or vendors of the Internet of Things. We used numerous sources like e-shops with smart devices or vulnerability records to construct lists of them. They are closely introduced in section Keywords.

Based on chosen sources and created keywords, we defined the dataset's structure. We selected column names and types of values. Also, we formed

17 categories of vulnerabilities and 13 categories of smart devices for further identification of vulnerability record types.

Then we had to pick a tool enabling the scraping of requested sources and data processing. After researching and evaluating the pros and cons, we decided to use the framework Scrapy. The process of selecting the suitable tool for scraping is described Data Scraping Tools.

While implementing the scraping tool, we aimed to process as much data as possible automatically. Also, we wanted to have data in the dataset consistent and reduce the number of empty values. However, it was not possible in some cases, and various parts of the dataset had to be further edited and processed manually. During the scraping tool implementation, we dealt with numerous challenges associated with the data sources nature. They are described in section Challenges.

After processing from Scrapy and manual work, we were able to form a dataset suitable for ranking lists creation. Based on the dataset, we further formed three ranking lists fulfilling our requirements on public data sources and transparent methodology.

2.2 Data Sources

Our primary goal was to choose public sources for the dataset creation. Therefore, we select the following three web pages to collect vulnerability records about Internet of Things devices.

2.2.1 National Vulnerability Database

The National Vulnerability Database, run by the U.S. government, is a repository of standards for vulnerability management. [20] The primary purpose is aggregating CVE records and collecting vulnerability data from multiple sources. Also, the vulnerability records have CVSS scores calculated and are assigned CWE, CPE, and other metrics based on findings. With their used methodology, they offer consumers an extensive vulnerability database.

2.2.2 Exploit Database

The Exploit Database is a non-profit project operated by Offensive Security. [21] Project offers a freely available database of public exploits and proofs-of-concept for security experts and developers. The database consists of records from multiple sources. They collect data from various public origins, mailing lists, or direct submissions.

2.2.3 Packet Storm

Packet Storm is a webpage created in 1998 about computer security. [22] The page aims to point to security issues and provide information to developers, security experts, or companies. The webpage offers a database of files like tools, advisories, and exploits. Furthermore, articles with information about vulnerabilities and security incidents are published daily.

2.3 Keywords

We created numerous lists and dictionaries consisting of keywords for easier vulnerability searching and classification. Due to the enormous amount of data about the vulnerabilities, we needed to search and pick Internet of Things-related vulnerabilities effectively. Therefore, we propose lists for vendors, device types, and vulnerabilities to be used as keywords for searching on the source webpages and further automatic labeling in the scrapers.

1. **vendors_keywords:** The list comprises vendors associated with the Internet of Things devices.

2. **devices_keywords:** This list consists of devices in the Internet of Things.

3. **specific_devices_keywords:** The list of keywords with specific devices from companies not producing only the Internet of Things products. For example, *Amazon Echo* or *Google Home*.

4. **ambiguous_keywords:** The words in this list do not have meaning only in the Internet of Things like *monitor*. The monitor can also be associated with network or monitoring vulnerabilities. The keywords in this list have added *iot* shortcut to clear and more effective search.

5. **other_keywords:** The list of general terms about the Internet of Things or keywords not belonging to any previous list.

6. **devices_dict:** This dictionary has keys defined by categories of devices like in section Device Categories. Every category has a list of devices belonging to that group.

7. **vulnerability_dict:** The dictionary with keys representing categories described in section Vulnerability Categories. Each category has a list of keywords associated with vulnerability used mainly for labeling further in the scraping process.

2.4 Data Scraping Tools

This section describes tools that we considered using for data scraping. All of them are compatible with the programming language Python and enable data collection from web pages. However, they differ in many functionalities and abilities.

2.4.1 Beautiful Soup

The Python library Beautiful Soup extracts data from XML and HTML files. It converts given documents into Python objects for further processing. [23] Therefore, it is not sufficient for scraping data from web pages. Another tool, for example, Python library *requests* or *urllib3*, is needed to perform requests and get data from the source pages. It is mainly used for small projects and recommended for beginners in data scraping.

2.4.2 Selenium

Selenium is an open-source project containing various tools for web browser automation. It is mainly used for web application automation testing because it offers many components to write tests in multiple programming languages. [24] However, due to its capabilities, this framework is frequently used for data scraping.

2.4.3 Playwright

Playwright is a library for automated web page testing using browsers similar to Selenium. It offers multiple modern browsers like Firefox or Chromium running in isolation and different configurations. Their main goal is fast and reliable use. In addition, it provides API to browsers which can also be used for web page scraping and following data extraction and manipulation. [25]

2.4.4 Scrapy

Scrapy is an application framework for data scraping and web crawling. It supports more robust projects with the ability to build architecture containing multiple spiders. For programmers, it offers wide flexibility and many features helpful in scraping, like working with proxies, data pipelines, or setting request parameters. [26]

2.5 Implementation of Scraping Tool

We decided to use the framework Scrapy to implement the data scraping tool. First, it allows us to create a robust system to scrape multiple web pages and easily add new scraper for additional sources of vulnerabilities in the future.

Second, it offers mechanisms to process gathered data in a united form and create pipelines. Furthermore, the scraping parameters, like request headers or parallelization, can be easily set in the setting file shared by all scrapers. Also, it offers saving the scraped data into CVS files which we preferred for further data processing. Last, the Scrapy is fast due to its nature—it is a framework and not only a library, also it does not run a browser internally to send requests.

Other tools could not fulfill some of our following requirements:

1. Scalability
2. Easy code modification
3. Repeated launching and crawling
4. Data unification and processing

2.5.1 Challanges

We rose to numerous challenges during the design and implementation of the scrapers. They were mainly associated with the accurate detection of the Internet of Things vulnerabilities and the implementation of specific functionalities of the scrapers. The most significant challenges are described with their solution below.

2.5.1.1 Identification of Internet of Things Vulnerabilities

We use as data sources the web pages maintaining vulnerabilities information. However, they not only offer data related to the Internet of Things. Therefore, we had to use mechanisms to detect vulnerabilities related only to the Internet of Things. Due to the amount of scraped data, it had to be done mainly automatically to be able to process it.

In the NIST CVE database are no rules on how the description of the CVE record should look. Therefore, it is on the reporter of CVE how he defines and explains the vulnerability. Some of the descriptions are very short or unclear. Thus, sometimes it is even for humans complicated to decide based on the description if the vulnerability is relevant to the Internet of Things.

For example, the description of vulnerability: *Meross MSS110 devices through 1.1.24 contain an unauthenticated admin.htm administrative interface.*

If somebody does not know that *Meross* is the company selling smart home devices and devices with identification *MSS110* are smart wifi plugs, the person can not simply decide that it is the Internet of Things vulnerability. Therefore, it is complicated to label the vulnerability automatically.

On the other side, it is simple to clearly classify the vulnerabilities with descriptions when the type of device, like a smart socket or camera, is specified.

For example, it fulfills the following description: *Syska Smart Bulb devices through 2017-08-06 receive RGB parameters over cleartext Bluetooth Low Energy (BLE), leading to sniffing, reverse engineering, and replay attacks..*

In Exploit Database, the exploits have parameters like *Type* or *Platform*. They can help to specify if the exploit is for the Internet of Things Devices. Furthermore, as the source of the vulnerability data can be used the title of the exploit or published code with comments. However, there are no strict rules or specifications similar to the NIST database. Thus, relying on it entirely during the automatic classification is not possible.

The vulnerability can be identified in the Packet Strom based on the title and description. The problem is equivalent to previous sources. No specific rules are applied. Thus, they can not be used to classify Internet of Things vulnerabilities fully automatically.

To identify Internet of Things vulnerabilities based on the available data, we proposed the Internet of Things Score system. It should help determine how much is a specific vulnerability related to the Internet of Things. The score is in the range from 0 to 10. A lower value indicates that the vulnerability is less likely to be associated with the Internet of Things. To calculate the score, we used the presented keywords and created a dictionary assigning values to specific keywords based on their association with the Internet of Things. The whole process is further explained in the section Internet of Things Score.

The IoT score enabled easier automatic identification of Internet of Things vulnerabilities. However, due to the problems described above, it is impossible to classify all vulnerabilities precisely. Nevertheless, it allows the rejection of vulnerabilities with a low score and vice versa, labeling the vulnerability with a high score as the Internet of Things related. In addition, it helps with more effective manual processing and labeling of the scraped data.

2.5.1.2 Unification of CVSS Score Versions

The used CVE database from NIST offers a CVSS score for all vulnerabilities. However, the older vulnerabilities have the only score available in the CVSS 2. Because we aimed to have a unified form of data in the created dataset, we decided to work with only one version of the score. In addition, it enables more precise results when working with CVSS scores of vulnerabilities. We chose to use CVSS in version 3.1 because it is newer and more commonly used in enterprises and by security experts.

For CVSS score transformation in scrapers was implemented function *convert_cvss_2_to_3*. It takes the CVSS vector in version 2 as input. After the conversion process, it returns a vector in version 3.1 and a score calculated based on it. The entire methodology and function are described in more detail in the section CVSS Converter. Furthermore, in addition to the newly calculated score in version 3.1, we decided to save also the old version to maintain data consistency.

2.5.1.3 Web Pages Using Java Script

The Exploit Database uses JavaScript to display a list of exploits. First, we considered that we would access the exploits records directly based on the index in the URL without the need to process JavaScript content. However, the quantity of scraped data would be enormous, with only a small amount of Internet of Things-related vulnerabilities. Thus, the further manual labeling and processing of the data would be unmanageable.

We needed to access primarily the Internet of Things exploits. Therefore, we used keywords to search the website to acquire the required exploits. However, the search function has, as a result, a list of the exploits run by Java Script. Unfortunately, Scrapy does not support by default scraping webpage content run by JavaScript. For that reason, we used, in addition, Playwright integration for Scrapy [27]. It enables a hander in Scrapy that sends Playwright requests and gathers required JavaScript content.

If Scrapy sends a request to the exploit database without using the Playwright handler, the required exploit list will not be present in response because the Java Script was not yet loaded. Playwright provides functionalities that enable waiting for the entire webpage content. Therefore, the exploit list can be extracted for further use. This way, we could scrape required vulnerabilities mainly associated with the Internet of Things devices. All implementation details are further described in the section EXPLOIT_DB_WEB_PLAYWRIGHT, dedicated to the Exploit database.

2.5.2 Used Libraries

We used numerous commonly used libraries in Python like *re* and *datetime* to implement the data scrapers. Furthermore, the library *nltk* was used for natural language processing in descriptions and titles. Due to the usage of Playwright, the *asyncio* had to be used to implement JavaScript content scraping and processing.

2.5.2.1 Natural Language Toolkit

The Natural Language Toolkit (nltk) is a tool for processing human language. It supports over 50 languages, like English, Spanish, and Greek. NLTK provides many procedures and functions associated with the text, like parsing, tokenization, and stemming. In addition, it offers various wordlists. For example, the wordlist *stopwords* contains the most common words like articles and prepositions. The library is commonly used for text processing before analyzing or applying machine learning algorithms. [28]

Figure 2.1: Structure of the Scrapy project files. It consists of the default Scrapy files and files including helper functionalities. In the folder *spiders* are files with implemented Spiders used for scraping.

```
vuln_scraping
├── spiders
│   ├── __init__.py
│   ├── cve_nist_api.py
│   ├── exploit_db_web_playwright_keywords.py
│   └── exploit_packetstorm_web.py
├── __init__.py
├── cvss_converter.py
├── items.py
├── keywords.py
├── middlewares.py
├── pipelines.py
├── score.py
└── settings.py
```

2.5.2.2 Asyncio

Asyncio is a standard Python library for writing concurrent applications. It introduces async/await syntax in Python, making it easier to write concurrent code. Furthermore, Asyncio enables running multiple coroutines simultaneously and provides asynchronous queues and async generators. Also, the library can be used when dealing with I/O intensive operations such as communication with a remote server and downloading data from it. [29]

2.6 Scrapers

The implemented scrapers for source web pages are united in a single Scrapy project. We used the structure of folders and files that Scrapy automatically generates in the creation of the new project. In addition, we created new spiders and files with helper objects and functions. The structure of the files is illustrated in figure 2.1.

We used for the implementation of the scraping process Scrapy objects *Item* and *Spider*. The *Item* was used to store scraped vulnerabilities and *Spider* for implementing the scrapers. They are described further in the section. Additionally, we present the scraping process flow and each scraper's specifications. Finally, at the end of the section are described in detail functions for converting CVSS, processing descriptions, and calculating Internet of Things scores.

Figure 2.2: Example of Item Vulnerability. It presents the structure of scraped vulnerability saved into Item Vulnerability. The example vulnerability is CVE-2020-11950 from NIST API.

```
{'cve': 'CVE-2020-11950',
 'cvss31': 8.8,
 'cvss_original': 8.8,
 'cvss_vector': 'CVSS:3.1/AV:N/.../S:U/C:H/I:H/A:H',
 'date_scraped': '2023-04-01 18:04:54',
 'date_vuln_published': '2020-05-28 13:15:11',
 'description': 'VIVOTEK...(and '
                'before...an authenticated user'
                'to upload...resultant execution of'
                'OS commands...devices.',
 'device_group': 'Cameras'
 'device_type': 'camera',
 'iot_keywords': ['camera', 'device'],
 'iot_score': 8.0,
 'searched_keyword': 'camera',
 'source_type': 'CVE_API_NIST',
 'source_url': 'https://se...?keywordSearch=camera',
 'vendor': ['vivotek'],
 'vuln_description': 'improper...used in an '
                     "os...('os command injection')",
 'vulnerability': 'Execution of malicious code'}
```

2.6.1 Items

The *Item* in the framework Scrapy is the tool for manipulating and storing structured data returned by the spiders. It enables to structure the scraped data in the following forms: dictionaries, Item objects, dataclass objects, and attrs objects. The objects are primarily declared in a file generated by Scrapy *item.py*. Using *Item* offers many benefits, like structuring large amounts of data, creating pipelines processing data, and unifying data from multiple spiders/different sources. [30]

2.6.1.1 Item Vulnerability

We proposed the *Item* named *Vulnerability* for structuring scraped vulnerabilities from different sources. We decided to use the *Item object* type because it offers various features. The values and their names in *Item Vulnerability* corresponds with columns described in section Dataset Columns. Additionally, the values in *Item* are defined by object *Field*.

An example of the vulnerability record saved into *Item* is displayed in figure 2.2.

2.6.2 Spiders

The *Spider* is a class defined for implementing the crawling and scraping process of the selected source. In addition, it introduces numerous methods for a more straightforward execution of the scraping procedure. Furthermore, it allows setting class attributes specifying multiple parameters for scraping behavior like *allowed_domains*. [31]

First, the URL of the chosen web page is inserted. Second, when the *Spider* is launched, the start method *start_request* is called. Then, it performs the request to a given webpage URL and forwards the content to the method *parse*. Furthermore, the method processes the response by specified rules. For example, the data can be parsed by selectors from HTML content. Then, it returns the data to the database or exports them into the file. The following requests are made similarly to scrape the entire content from the webpage.

Further in the section are described all spiders separately. They have a process flow presented in figure 2.3 and use *Vulnerabiliy Item* to manage data the same. Furthermore, they all use keywords to classify dataset categories and calculate Internet of Things score. However, they differ in methods of gathering data from web pages or extracting data from responses.

2.6.2.1 CVE_NIST_API

We used the available API to scrape vulnerabilities from the CVE NIST database on *services.nvd.nist.gov*. It returns data in JSON format. To scrape primarily the vulnerabilities associated with the Internet of Things, we used the API keyword searching function with keywords from created lists described Keywords.

The response from the API contains all vulnerabilities associated with the given keywords. Therefore, vulnerabilities are parsed from the response and then processed individually. First, during the single vulnerability processing, the data about the scraping request, like used keywords or URL, are saved into the *Vulnerability Item*. Then, the vulnerability data are processed. Due to the structured response in JSON, it is effortless to extract and parse data. Furthermore, the vulnerability data are mostly consistent, and values are not missing, unlike in other scrapers.

The methodology of processing vulnerability is the following. First, the CVSS score is extracted and converted if needed. Second, the vendor is saved from the vulnerability's CPE and vulnerability type category specified based on the assigned CWE. Next, the description of the scraped vulnerability is processed by method *process_description* described in section Description Processing. The missing values of *Item Vulnerability* are added from the processed description using keywords. Finally, the IoT score is calculated based on keywords and processed description.

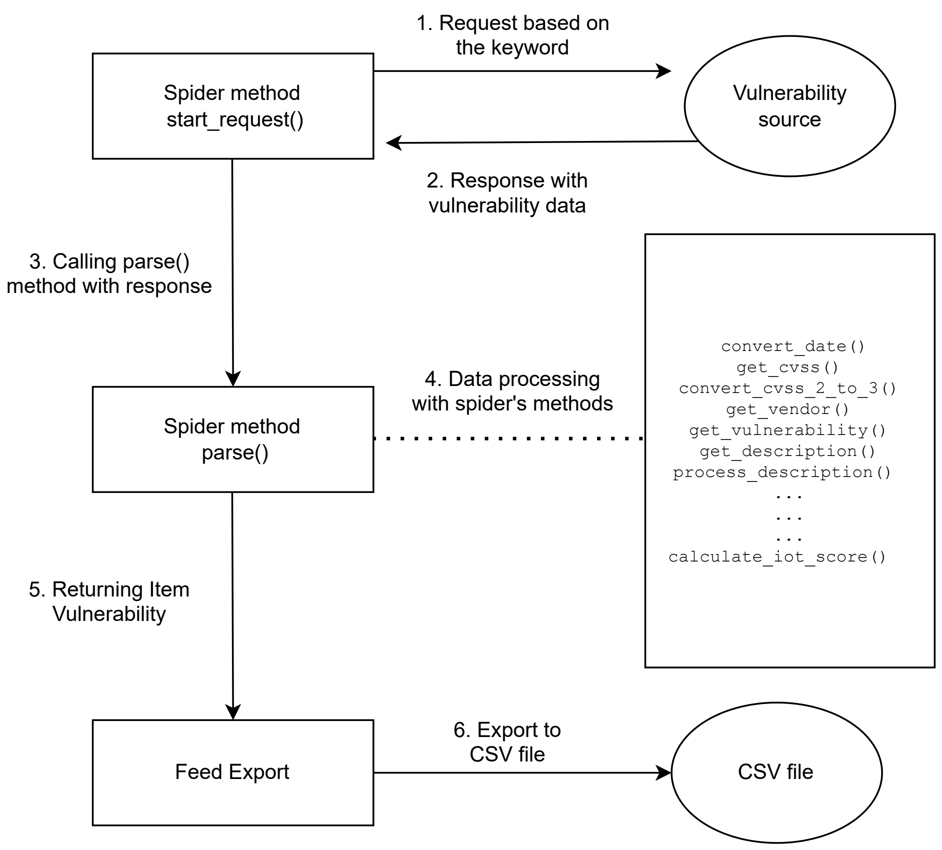

Figure 2.3: Spider processing flow. The figure introduces the scraping flow of Spider from the start method call to the final exporting of the processed data. It also shows communication with the source and the parsing process of received data.

2.6.2.2 EXPLOIT_DB_WEB_PLAYWRIGHT

This spider uses as a source webpage *exploit-db.com*. Due to JavaScript on the page, the Playwright was used in addition to Scrapy to be able to scrape and process the JavaScript content. The vulnerabilities related to the Internet of Things were searched based on the *Platform* parameter on the webpage and proposed keywords.

Due to the nature of the webpage, some of the attributes of the *Vulnerability Item* were mainly missing. For example, the webpage does not offer a CVSS vector or score. Furthermore, CVE does not have to be assigned or exist for the exploit's vulnerability. Regardless, we implemented methods to attempt to extract some value types if they exist.

Because of the use of JavaScript, the Playwright handler makes the start request. Then, the *parse* method is called with the response as an input. To get the required data about vulnerabilities, the *asyncio* library must be used to be able to wait for the loading of JavaScript content. In response is the part of the exploit list from which the spider extracts exploit indexes for further use. Then if possible, it continues to another part of the list. After getting and extracting data, the following request can be sent using the basic Scrapy request handler. The whole approach is shown in figure 2.4.

The procedure of processing scraped vulnerabilities is analogous to the previous spider. First, information about scraping is saved. Next, spiders attempt to extract CVE. As a description of vulnerability is used title and code of the exploits. We mostly used comments in the code to extract desired data. Due to the frequently lengthy code of exploit, we decided to save only the first 500 characters of the original description. Finally, the vulnerability details were extracted based on the processed description by the method *process_description*, and the IoT score was calculated.

2.6.2.3 EXPLOIT_PACKETSTORM_WEB

This spider has as a source webpage *packetstormsecurity.com* . We scraped data from the *file* section containing, for example, exploits or advisories. Searching on the webpages with keywords is used for collecting mostly Internet of Things vulnerabilities.

Vulnerability data are extracted from the title and description of the found file. Usually, it is very short. However, it mostly contains enough information to distinguish at least vulnerability and device type by searching keywords from the created wordlists. Similarly to the previous spider *EXPLOIT_DB_WEB_PLAYWRIGHT*, some required data are missing. For example, the CVSS is not added to file descriptions, and CVE occurs occasionally.

The response from the source contains a list of the vulnerability files. Foremost, the list is parsed, and file information is extracted. Then spider iterates through all gathered files. The process is almost identical to previous spiders. First, it is checked that the scraped file has the expected tag—exploit, vulnerability, or advisory. Second, details about the scraping are saved into the *Item*. Then, the spider attempts to extract information about the vulnerability based on the processed description by the method *process_description*. Also, the Internet of Things score is calculated based on description and keywords. Finally, the *Vulnerability Item* is exported and saved into the CVS file.

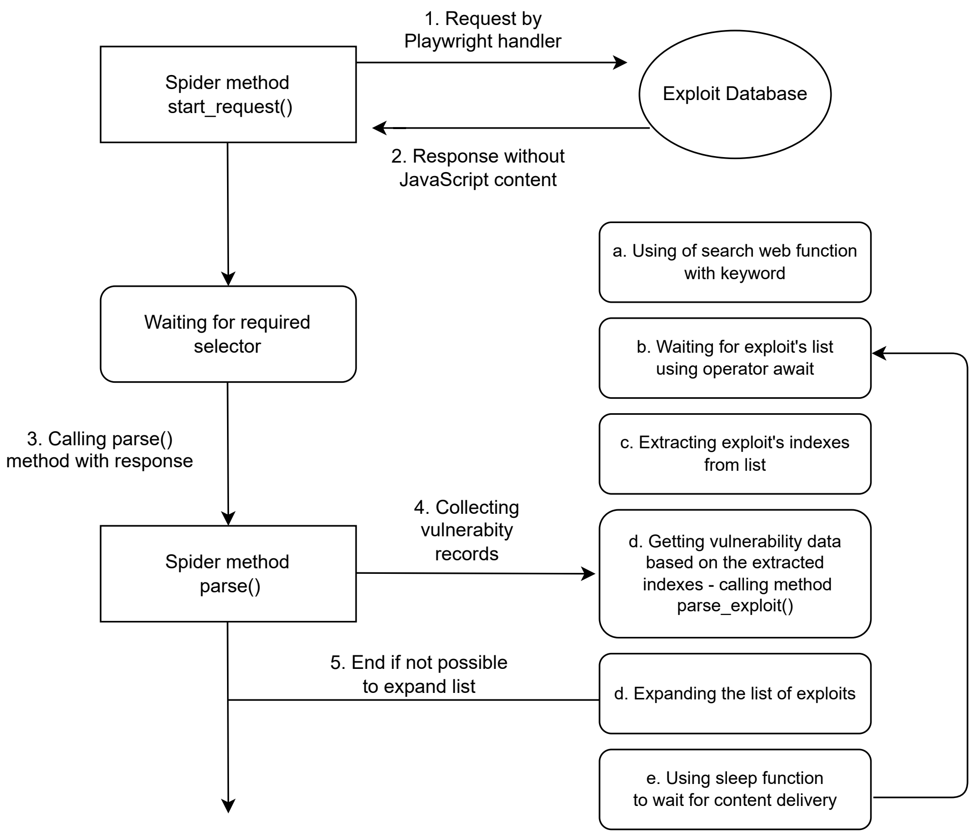
1. Request by Playwright handler
Spider method start_request()
Exploit Database
2. Response without JavaScript content
a. Using of search web function with keyword
Waiting for required selector
b. Waiting for exploit's list using operator await
3. Calling parse() method with response
c. Extracting exploit's indexes from list
4. Collecting vulnerabity records
Spider method parse()
d. Getting vulnerability data based on the extracted indexes - calling method parse_exploit()
5. End if not possible to expand list
d. Expanding the list of exploits
e. Using sleep function to wait for content delivery

Figure 2.4: EXPLOIT_DB_WEB_PLAYWRIGHT spider processing flow. The figure presents the scraping process scraping using Playwright due to JavaScript content.

2.6.3 Shared Functions

This section describes functions shared between the spiders or methods implemented in each spider dealing with the same issue. They mainly resolve the challenges presented in section Challanges.

2.6.3.1 CVSS Converter

To preserve the consistency in the dataset, we decided to work only with CVSS in version 3.1. However, the CVE NIST database has only CVSS 2 score and vector in older vulnerability records. Due to that reason, we propose the schema to convert the base metrics group of CVSS 2 to CVSS 3.1.

In this work dealing with the CVSS conversion [32], they presented machine learning using 50 words from the vulnerability description to convert the CVSS version. We aimed to use a simpler method to quickly process large

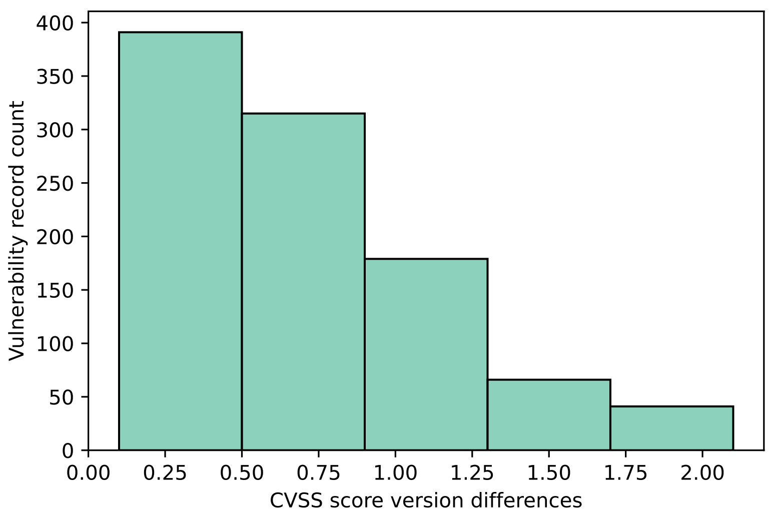

Figure 2.5: Differences of CVSS scores between versions. Histogram of differences between original CVSS 2 and calculated values based on proposed converter in CVSS 3.1.

amounts of data. Therefore, we implemented a direct conversion based on assigned values.

For example, the metric *Authentication* in CVSS2 was replaced by equivalent metric *Privileges Required*. We transformed values based on the parallel of values:

- None(N)→None(N)
- Single(S)→Low(L)
- Multiple(M)→High(H)

We decided to keep the previous value of CVSS in case of an erroneous result of the conversion. As a result, the *Vulnerability Item* has three fields associated with CVSS score: cvss31, cvss_original, cvss_vector. Therefore, it is possible to work with the previous value if needed. However, based on the scraped data, the average difference between the original and calculated values is 0.6, and the maximal difference is 2.1. The histogram of differences is displayed in a plot in figure 2.5.

The function used for conversion is called *convert_cvss_2_to_3* and occurs in the file *cvss_converter.py*. It uses the Python package *cvss*, which presents functions related to CVSS score, like calculating the score based on the given vector. The function takes as an input vector in version CVSS 2 and returns the transformed vector in version 3.1 and the associated score.

2.6.3.2 Description Processing

The descriptions of the scraped vulnerabilities are very diverse. They are primarily dependent on the publisher of the vulnerability record. As explained in the section Challanges, no specific rules exist for writing descriptions or titles. Therefore, the spiders cannot depend on any specifications and patterns in processing the descriptions. To be able to work with them and apply the keywords for classification, we needed to standardize and unite their form.

We used the library *nltk* to process the description. First, we applied tokenizer *RegexpTokenizer* to split the description into tokens— words, digits, and other symbols. Second, the stopwords were removed based on the corpus *stopwords.* Further, the numbers were extracted away. Finally, we applied *WordNet Lemmatizer* to get words into the inflected form.

This process is applied in method *process_description.* Each spider has it implemented separately. The method takes as input the description in the string format and returns the processed description that is further used for extracting keywords. Example of the processed description saved as a list: *['multiple', 'unspecified', 'vulnerability', 'camera', 'life', 'attacker', 'denial', 'service', 'unknown', 'vector'].*

2.6.3.3 Internet of Things Score

To automatically identify the Internet of Things vulnerability, we proposed the scoring schema that assigns the weight to keywords based on the association with the Internet of Things. The keywords more connected to the Internet of Things have a higher score. The score ranges from 0 to 10, and the additional value -1 is set when the score cannot be calculated. The result score is the mean of keywords weights.

The score is implemented as a Python dictionary. The key is score weight, and the value is a list of keywords—for example, the score with weight 6: *6: ['smart', 'device'].* We decided on this structure because it can be easily modified and applied to the vulnerability data.

The score is calculated in method *calculate_iot_score*, which each spider has implemented slightly differently. However, the whole methodology is identical. The method takes as input the vulnerability description and, based on the spider, other vulnerability information like vendor or device type. Then the score dictionary is applied to input data. Therefore, the keywords are found, and their weights add up. Finally, the mean of found keywords weights is calculated.

The primary purpose of the score is to enable automatized classification of the vulnerabilities. However, it only provides approximate outcomes due to many factors. For example, the score can be affected by the length of the description. If the description consists only of a few words or is overly verbose, it can distort the calculated score. We mainly operated with descriptions

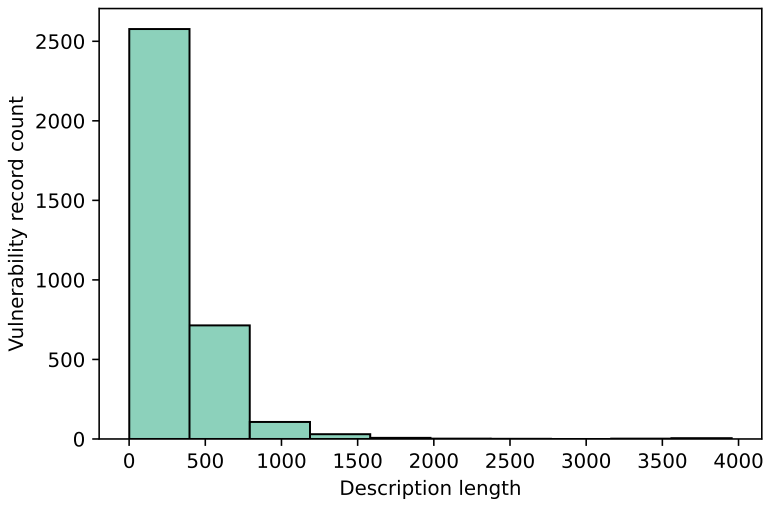

Figure 2.6: Lenghts of the descriptions from CVE NIST database. Histogram presenting description lengths from obtained CVE NIST vulnerabilities records.

shorter than 500 characters, but occasionally, descriptions were longer than 4000 characters, as displayed in the histogram in figure 2.6.

Due to the distortions, we cannot rely solely on the Internet of Things score in labeling vulnerabilities. Further classification must be done manually—however, the scoring significantly helps with manual annotations. First, it reduces the amount of data that needs to be manually processed. The vulnerability records with high scores can be automatically labeled as smart device vulnerabilities. Similarly, we can assume that vulnerabilities with a lower score are not the Internet of Things associated. Second, the score enables us to group the vulnerabilities by their probability of belonging to the Internet of Things.

Chapter 3

Dataset

This chapter describes the dataset structure and process of creation. First, the methodology is presented, and dataset columns are defined. Then, the vulnerability categories are proposed with descriptions, security impacts, and defense methods. Finally, in the last part of the chapter, we introduce the device categories.

3.1 Methodology of Dataset Creation

The dataset was created from 3 data sources described in section Data Sources. We used scrapers defined in the previous chapter Scrapers to scrape the source web pages and automatically process the obtained data. During scraping, nearly 16000 vulnerability records were collected that could be potentially related to the Internet of Things.

After acquiring and automatically processing all data, manual control and editing were performed. First, the vulnerability records were labeled based on the Internet of Things score. Second, the device types and vulnerability categories were checked and corrected if needed. Finally, the missing values of devices or vulnerabilities were added.

From the manually revised data, duplicates were removed based on the same CVE or description. Then, only vulnerability records labeled as Internet of Things related were extracted to create the dataset.

The final dataset comprises 4532 vulnerabilities extracted from nearly 16 000 scraped records. The data are mainly from the CVE NIST database. The ratio of vulnerability records source is pictured in a graph 3.1. We used records without any time restriction.

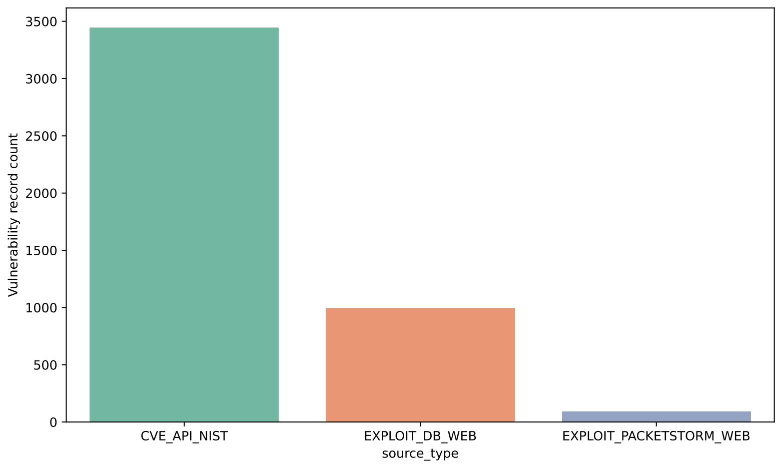

Figure 3.1: Number of vulnerability records per source. This figure shows the number of gathered records from 3 data sources—CVE NIST, Exploit Database, and Packet Storm.

3.2 Dataset Columns

Each vulnerability record in the created dataset has 18 columns describing details about scraping and vulnerability. In the following sections that are divided by information type, the columns are listed and described.

3.2.1 Data Scraping Columns

In this section are described five columns. They consist of information about the scraping process and sources.

- **date_scraped:** date of the scraping from the webpage in format `YYYY-MM -DD HH:MM:SS`
- **date_vuln_published:** date of publishing on the website in format `YYYY-MM-DD HH:MM:SS`
- **searched_keywords:** keywords used to search a vulnerability on the website
- **source_type:** scraped webpage label
  - **CVE_API_NIST:** services.nvd.nist.gov
  - **EXPLOIT_DB_WEB:** exploit-db.com
  - **EXPLOIT_PACKETSTORM_WEB:** packetstormsecurity.com
- **source_url:** URL of scraped webpage

3.2.2 Vulnerability Columns

The following columns consist of details about the scraped vulnerability.

- **cve:** assigned CVE if exists
- **cvss31**: CVSS score in version 3.1
- **cvss_original:** CVSS in the previous version if new one in CVSS 3.1 needed to be calculated
- **cvss_vector:** CVSS vector if exists
- **description:** description of vulnerability obtained from the scraped webpage
- **device_group**: category of the device (additionally explained in section Device Categories)
- **device_type:** exact type of device
- **vendor:** vendor of the vulnerable device
- **is_iot:** label if a vulnerability is considered as the Internet of things
- **iot_score:** calculated Internet of Things score (further described in section Internet of Things Score)
- **iot_keywords:** keywords used for calculation of Internet of Things score
- **vulnerability:** category of the vulnerability (more in section Vulnerability Categories)
- **vuln_description:** description associated with vulnerability type on the scraped webpage

3.3 Vulnerability Categories

In the dataset, we propose the following 17 categories of vulnerability categories. First, our primary goal was to cover all possible vulnerabilities in smart devices. Therefore, we primarily used the CWE list [9] for reference and classification creation. Second, we tried select groups of vulnerabilities for the best automatic classification result and low rate of false positives.

3.3.1 Access Control Problem

Problems with access control enable an attacker to access resources that should be restricted or hidden. Therefore, during the attack, the malicious actor can bypass protection and access the forbidden resource. [33] This unwanted behavior can lead to data breaches or misuse of functionalities by the attacker.

Access control problems arise mainly from poor design or errors in development. Therefore, analyzing permissions and privileges in a system and correctly setting up access control lists is essential. [34] The system setting also should not be left in the default state setting. Furthermore, for the system security is crucial to implement user authentication properly. Before taking action on the device, firstly user must be identified and verified. Secondly, the user's permissions must be checked and compared with set-up permissions for the requested operation. Otherwise, a malicious actor could access resources and attack the system. Also, after permitting access to the resource, the operation should be logged for security monitoring and eventual security incident investigation.

In this category belongs numerous vulnerabilities. First, it contains vulnerabilities associated with missing or improper set authentication. Thus, the attacker is not required to prove his identity before accessing the resource. Next, vulnerabilities arise from incorrect permissions and privileges settings. For example, the permissions are incorrectly set if a logged user to internet banking can access sensitive data from other users, like account balance or list of payments, as displayed in figure 3.2.

The access control problem can lead to numerous critical security incidents. For example, a company's reputation can be badly affected if the attacker gains and leaks sensitive data like personal or health records. Also, it can cause financial loss. Similarly, if the malicious actor attains high privileges, he can control the whole system. Then, based on the goal, he can damage or misuse it for the following attack.

We chose this category based on CWE-284: *Improper Access Control*, and high placements in OWASP TOP 10 for the web applications [2].

3.3.2 Cross-site Request Forgery

In the Cross-site Request Forgery attack, the attacker manipulates the authenticated victim into sending his malicious request to the vulnerable webpage. As a result, the malicious actor bypasses the necessity to be authenticated directly into the system as the victim. [35] Thus, he does not need to know the user's password, session cookie, or another secret to perform malicious action via request.

First, the attacker crafts a request for an adversary operation, such as altering the victim's settings, changing the password, or performing an action on the vulnerable web page on the victim's behalf. For example, in internet

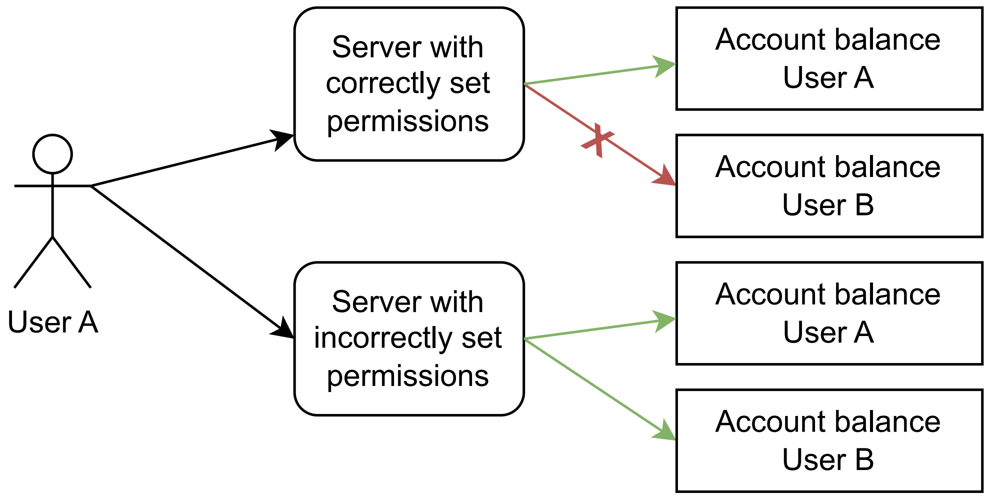

Figure 3.2: Access control. This figure shows two servers with differently set access control. The first server is poorly set. Therefore, the user can incorrectly access the data about different users.

banking, the attacker could make a payment as a victim to the desired bank account. Second, he delivers a malicious link. The victim can receive an e-mail with the link or click a button on the attacker's website. The attacker usually uses social engineering to force the victim to send the malicious request.

In CSRF, the malicious actor abuses parameters added to the request by the browser—for example, session cookies. The parameters are added automatically to the fraudulent request because the legitimately logged victim performs the action, not the attacker. Thus, the sent request to the web application cannot be detected as adversarial. [36]

Nowadays, various methods exist to prevent Cross-site Request Forgery attacks. Primarily, the CSRF token should be used in web applications. [35] Furthermore, the *SameSite* cookie flag should be set to prevent sending session cookies with a request from another webpage.

Even though CSRF is a web application vulnerability, many smart devices use webpages for configurations, settings, and other functionalities. Thus, it is also relevant to The Internet of Things. This category covers CWE-352: *Cross-Site Request Forgery (CSRF).*

3.3.3 Cross-site Scripting

Cross-site scripting is a web application vulnerability when the attacker injects malicious code or script into a legitimate web page running in the victim's browser. The attacker can change with injection the form of the webpage. [37] For example, he can craft a fake login screen on the vulnerable site and steal the victim's credentials. Furthermore, the attacker can gain sensitive data like the user's session cookie and impersonate the victim.

In XSS, the attacker abuses the webpage's running code in the victim's browser. He injects his code into an HTML element and makes browsers execute it on behalf of the victim. There are several ways how to place malicious code into the webpage. According to the code injection method, various types of XSS exist, for example, stored, reflected, or DOM. [38] This is an example of the payload which triggers an alert on the webpage and print session cookie: `<script>alert(document.cookie)</script>`.

Stored cross-site scripting occurs when the attacker saves the code into the webpage. Then the victim must perform the required action to execute the code. For example, the attacker can post a comment on a vulnerable webpage with malicious code. Thus, the comment with payload is then permanently saved into the page. When the victim visits the webpage and views the comment, the code is executed, and the attack is performed successfully.

In reflected cross-site scripting, the malicious code is injected into the webpage only temporarily. The attacker must craft a webpage link and then make the victim click on it to execute the attack. The victim can be convinced to click on the malicious link through social engineering.

The DOM cross-site scripting arises when the attacker injects a malicious code into the Document Object Model (DOM). The payload can be delivered to the victim via the link as in the previous type of attack.

This vulnerability arises due to improper input validation from the user. To prevent this attack, controlling and validating all input is necessary. For example, special symbols can be escaped or removed based on the input data type. However, security measures do not have to relate only to input. The headers in the response, like *Content Security Policy*, should be adequately and correctly set. Also, suitable flags of cookies can prevent session data from stealing.

The consequences of the attack depend on the webpage content and applied protection measures. However, the XSS can lead to information disclosures and or the compromisation of user accounts.

This category was chosen for similar reasons to the previous class Cross-site Request Forgery.

3.3.4 Denial of Service

The attacker's primary goal is to make the system unusable in the Denial on Service attack. Thus, the attacked system cannot run correctly and be appropriately used by legitimate users. The system during the DoS attack is usually running slow, and functionalities are not working correctly. In some cases, the system can be shut down by the attacker.

The Denial of service can be caused in several ways. Firstly, the attacker can be sent an enormous amount of network traffic to the targeted system.

This behavior can lead to the consumption of the victim's resources. Thus the system cannot establish new connections and process more demands. [39] In this category belongs attacks like ping flood, SYN flood, or Slow Loris. Second, the malicious actor can craft a malformed request that the system cannot correctly proceed.

This category also includes Distributed Denial of service (DDoS). The attacking approach is analogous to DoS, but the attacker abuses numerous sources to generate traffic for flooding the victim. [40] The misused devices are often part of the botnet— an automatized group of infected machines frequently containing the Internet of Things devices. The attacks are portrayed in figure 3.3.

Nowadays, it is essential to protect publicly available systems before DoS attacks. The system should be able to handle extensive incoming traffic and respond to it with defined procedures. [40] Numerous devices and solutions focusing primarily on DoS protection are available on the market —for example, IPS systems, firewalls, routers, or proxies. Also, the attack surface should be minimized, and unnecessary system functionalities should be suppressed from the public.

The DoS system can have various consequences. For companies, it can be reputation and financial losses due to the unusability of the service used by their consumers. Furthermore, the unusable system can have various critical outcomes in specific areas like healthcare.

We proposed this category because the Denial of service is a widespread threat. For example, according to the report from Cloudflare, in the third quarter of 2022, amount of DDoS attacks increased compared to the previous year [41].

3.3.5 Execution of Malicious Code

The execution of malicious code in the victim's system is a threat with severe consequences. It can cause significant damage to the system or even enable the attacker to take over control of it. [42] This broad category contains vulnerabilities enabling attackers to run malicious code or commands in the targeted system.

There are many possible attack vectors on how the attacker can force the system to execute his code. Sometimes is possible to do it remotely, in other cases, the attacker must have physical access to the device. Furthermore, the attacker can execute his malicious program or inject code into the running process. For example, the attacker can misuse the improper input check in the system. If the vulnerable system does not control input sufficiently and executes it without restrictions, the attacker can insert input with malicious code and attack the system.

There are multiple methods how to protect the system from letting the attacker execute the malicious code. The antivirus can be used to catch hostile

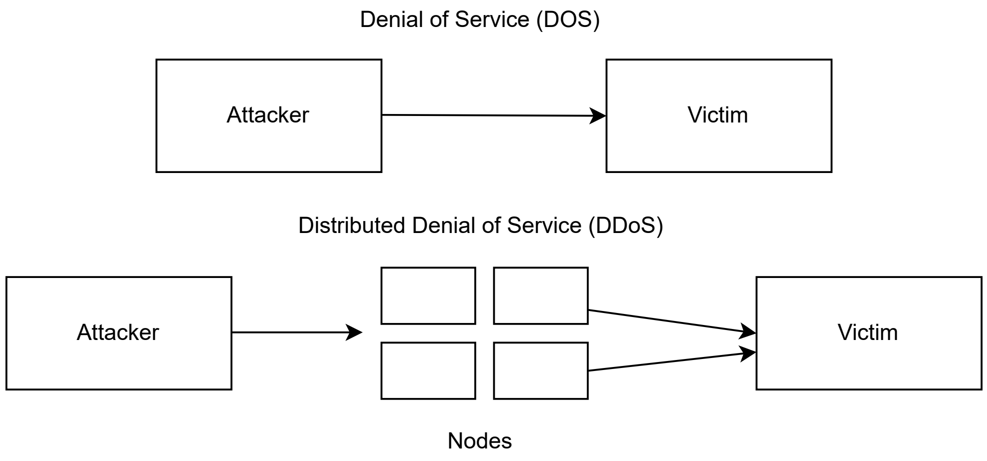

Figure 3.3: Denial of service/Distributed Denial of Service. The figure presents the difference between Denial of Service and Distributed Denial of Service. During DDoS, numerous machines are misused to attack the victim.

programs. Also, proper monitoring and logging should be implemented to detect an attack. Furthermore, all system input should be validated before further processing, and the permissions in the system should be adequately set and enforced to prevent code execution. Many security measures can also be taken in the system architecture. For example, the data can be labeled executable or non-executable based on their purpose. Also, integrity checks can be done. [43]

The execution of malicious code in the system can have severe consequences. For example, the attacker could run malware like ransomware or virus to damage the system. Furthermore, executing the code can enable the attacker to control the system and perform adversary operations. Also, the attacker could elevate his privileges or access sensitive data via executing code.

This category was classified based on the severity of the execution of malicious code in the system. It groups multiple vulnerabilities with CWE like CWE-78: *Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')* or CWE-94: *Improper control of generation of code ('code injection').* Other proposed categories like *SQL injections* or *Cross-site scripting* could be considered a subcategory of this. However, we decided to separate them based on the specification of the attack.

3.3.6 Improper Data Handling

This category contains all vulnerabilities associated with file manipulation and data handling in the system. The attacker can abuse poorly implemented data management and gain control over the victim's system or access sensitive data. Thus, it is essential to correctly manipulate data and perform secure operations with them to prevent security incidents and data leaks.

One of the common weaknesses in the system is an unrestricted file upload. With insufficient restrictions, the attacker can upload a problematic file format for the system. [44] For example, if the system prompts the user to upload an image, other formats, like executable programs, should be rejected. Instead, only file types typical for images like JPEG or PNG should be accepted as input from the user. If the system does not restrict the upload of a hostile file, the attacker could potentially upload malware or any other desired file and attack the vulnerable system. Also, parameters like file size should be limited.

Systems usually contain various files and process numerous data. Therefore, it is crucial to manipulate them carefully. It should be appropriately specified how to manipulate sensitive data. For example, sensitive information should not be stored in logs. Furthermore, sometimes information like personal data does not have to be permanently stored in the system. Also, the file system should be adequately implemented in the system. For example, only specified users should have the right to access and modify files. Also, sensitive files should be treated carefully. [45]

Vulnerabilities in this category can have various impacts. First, the malicious actor could misuse the vulnerability to access restricted data. Thus, it could lead to a leak of sensitive data like personal information, which can be critical for companies and organizations. Second, the attacker could execute malicious code and harm the vulnerable system. Furthermore, the vulnerability could lead to file modification or corruption.

As mentioned above, the protection from vulnerabilities in this category is proper file and data management design. Thus, only files with permitted types and parameters should be accepted and saved into the system. Furthermore, sensitive data should be treated carefully and saved only if needed. [45]

This category is based on CWEs like CWE-434: *Unrestricted Upload of File with Dangerous Type* or CWE-59: *Improper Link Resolution Before File Access ('Link Following').* It was proposed to cover all vulnerabilities related to file and data management. However, it overlaps with categories *Improper input validation* and *Problematic cryptography.*

3.3.7 Improper Input Validation

Insufficient input validation can lead to many severe security threats. If the input is not validated correctly, the attacker can construct a malicious input payload and attack the system. As a result, the system's integrity, confidentiality, and availability could be violated. This category includes vulnerabilities associated with input validation issues in the systems.

It is necessary to check and validate all input data from the user and untrusted sources before further processing it in the system. If the validation is improper, the input data from the attacker can trigger an adversary action in the system and lead to various security issues. [46] Therefore, the system should always operate with data that are considered safe and correct.

Various parameters and requirements should be checked based on the data input type—for example, input length, data type, object structure, or special symbols. Furthermore, the correctness of the data should be validated, like correct date values or birth number form requirements.

For illustration, the user inserts the amount of money to transfer in the banking system. The input does not have to consist of letters or special characters. Thus, the system should check that input consists only of digits. If the attacker attempts to insert the malicious payload with special symbols, the system will not accept the input and stop the attack. Also, the system should check that the required amount of money can be transferred. Hence, the user balance in the bank account is sufficient, and the input value is not negative.

Various protection methods exist to prevent the attacker from inserting malicious input into the system. First, the system should never trust the user input, and control should always be performed. Second, restricted characters can be escaped or removed based on the system's requirements. Also, regular expressions or allowlists/blocklists can define the correct input structure.

Incorrect or insufficient input control can lead to various security breaches. The specifics of the attacks and consequences depend heavily on the system type. However, the attacker can misuse input vulnerabilities to, for example, execute malicious code, cause Denial of Service, or access sensitive data.

Many of the proposed categories of vulnerabilities could be considered a subcategory of this one. For example, *SQL injection* or *Cross-site scripting.* However, we divided them into separate categories based on the attack methods and impacts to achieve more complex classifications. Therefore, this category was created to cover vulnerabilities similar to CWE-20: *Improper Input Validation* and CWE-129: *Improper Validation of Array Index.*

3.3.8 Improper Use of Memory

Vulnerabilities listed in this category affect the security of memory. They usually occur from programming bugs or incorrect memory management. As a result, the attacker can misuse memory vulnerabilities in various forms to attack the system and cause several security breaches. For example, the malicious actor can cause a crash, corrupt the data in memory or execute malicious code.

Nowadays, many programming languages like C# or Java have *memory safety.* That means they provide protection in development, prevent from bugs in code leading to memory vulnerabilities and enable automatic memory functions like *garbage collectors.* Therefore, it should increase the memory's security. However, languages like C or C++ are not memory save. Thus, they do not protect from errors, and memory management is mainly on the programmer. [47]

Many vulnerabilities can occur during memory allocation and release. For example, the same allocated memory can be faulty freed two times. Furthermore, the released address in memory can be addressed in the program again, causing unpredictable behavior. Also, it is not secure to not free memory when it is no longer needed. This can be misused by the attacker, for example, for Denial of the service.

The manipulation with pointers can also lead to security threats. [47] For example, performing arithmetics with pointers can cause reaching in the memory out of bounds. Thus, possibly enabling the attacker to perform the attack. Furthermore, attempting to access a resource with a NULL pointer also impacts the system's security.

The protection from this type of vulnerability is to use *memory safety* programming languages if possible. Furthermore, the correct and safe function should be implemented to perform operations with memory. Also, the code should be appropriately checked and tested to detect and mitigate bugs.

The vulnerabilities in memory can be critical for system security. First, the attacker could gain access to sensitive data, which can be critical based on the system type. Second, he could modify or damage the data stored in the memory. Furthermore, the attacker could execute code and take control of the system. Also, the crush of the system can arise.

We proposed this category due to the severity and widespread memory-related vulnerabilities. Furthermore, vendors often use memory-unsafe programming languages like C or C++ in the Internet of Things devices. This category contains CWEs related to memory vulnerabilities like CWE-415: *Double free* or CWE-416: *Use after free.* The overflow vulnerabilities are separated in category *Overflow* due to the high occurrence.

3.3.9 Insecure Design/Design Flaw

This category includes vulnerabilities arising from design flaws and incorrect programming techniques. Therefore, the malicious actor misuses development bugs in the system to perform the attack. As a result, the flaws can cause severe damage based on the severity of the error and the parameters of the system.

During development, many bugs can arise from poor programming practices or laxness. For example, in programming languages such as C or C++, errors can appear from incorrect type conversions or initialization of the objects, leading to unexpected behavior that the attacker can misuse. Furthermore, errors can occur from inaccurate arithmetic operations like float multiplication. Also, if the values are not properly checked, the error arises when the number is divided by zero value. Similarly, overflow or underflow of value occurs when the operation result has a form the system cannot store correctly. [48]

Problematic are also redundant fragments of code and comments in the program. The unnecessary code increases the attack surface, and the malicious actor can misuse it to perform the attack. Furthermore, the comments can be problematic if they contain sensitive information such as passwords or API keys. Also, they can provide information about system logic to the attacker. [48]

The developers should focus on system architecture and proper system design to prevent vulnerabilities in the category. Also, emphasis should be put on correct programming practices during development. Implemented code should be appropriately checked and tested to discover bugs. Thus, minimizing the attack surface for the attacker.

Vulnerabilities in this category can lead to numerous security risks based on the system type and flaw effects. For example, if the attacker misuses the infinite loop, he could cause the Denial of the Service and damage the system. Furthermore, executing malicious code and taking control of the system would be possible. Also, the malicious actor could access sensitive data or any other restricted resource.

We proposed this category to categorize vulnerabilities occurring from poor programming practices. This category is based on many CWEs like CWE-489: *Active Debug Code* and CWE-704: *Incorrect Type Conversion or Cast.*

3.3.10 Other Vulnerabilities

The data in the dataset are labeled automatically. Therefore, we created this category to assign a label to vulnerabilities that do not belong to any proposed category. Usually, it considers vulnerabilities with a low occurrence rate. Thus, we decided not to propose a separate category for them. Furthermore, records with uncertain vulnerability types or covering a wide range of vulnerabilities are put in this category.

3.3.11 Overflow

The overflow in memory occurs when the data saved into the buffer are written over the buffer's boundary. The vulnerability caused by this improper behavior usually leads to the execution of malicious code by the attacker. [49] Furthermore, it can crash a system or lead to faulty program outcomes. Thus, it is a severe vulnerability that can cause extensive damages to the vulnerable system.

The buffer is a part of the memory reserved for temporary data storage. It is used mainly during the transportation of the data from the input. [50] Thus, the malicious actor can prepare the adversary input payload and attack the system over overflow vulnerability. However, there is a wide range of types of overflow attacks that differ in many aspects. For instance, it can vary in memory placement— stack or heap. [51]

The scenario of the attack may be as follows. First, the attacker crafts the input larger than the buffer size. Therefore, after inserting the input, the overflow in the buffer occurs. This behavior is shown in figure 3.4. If the system has no protection, the data in the memory out of the buffer are rewritten based on the attacker's input, and the program continues without detecting malicious behavior. For example, the attacker could rewrite the function's return address in memory with his address value where the malicious code is prepared. Therefore, when the function is finished, the system uses the attacker's malicious address and fetches instructions with the adversarial code.

The overflow is still a common vulnerability. It occurs mainly from incorrect system design or faulty implementation of the program. Thus, inappropriate functions are used, and the input is not adequately controlled. Also, the input size for the buffer is not checked correctly. This vulnerability is frequently in programs written in C or C++. Therefore, it is a relevant attack for the Internet of Things devices because the following languages are often used for implementation.

There are multiple mechanisms to protect the system from overflow attacks. For example, the canary values can be used. The system detects overflow and stops undesirable actions if the canary value is overwritten in memory. Next, the part of the memory can be marked as unexecutable, so data in the buffer are treated as data and not instructions. [51]

This category could be considered as a subcategory of *Improper use of memory*. However, we proposed it separately based on the high occurrence of overflow vulnerabilities in the Internet of Things devices. Category *Overflow* consists of multiple CWEs, for example, CWE-787: *Out-of-bounds Write* or CWE-120: *Buffer copy without checking size of input ('classic buffer overflow')*.

3.3.12 Path Traversal

The path traversal vulnerability enables the attacker to access the file system's parts unauthorizedly. Therefore, the attacker can browse directories and read files that should be inaccessible to him. [52] This undesirable behavior can cause data breaches and disturb the system's confidentiality. Path traversal vulnerability occurs from incorrect input validation and sanitization.

Firstly, in a path traversal attack, the malicious actor must find the vulnerable entry point to the file system. Second, he crafts a malicious input payload that creates a pathname in the file system for the desired file. The malicious payload usually contains special symbols based on system type. For example, adversarial input for reading file *passwd* in the Unix operating system can look like this: `../../etc/passwd`.

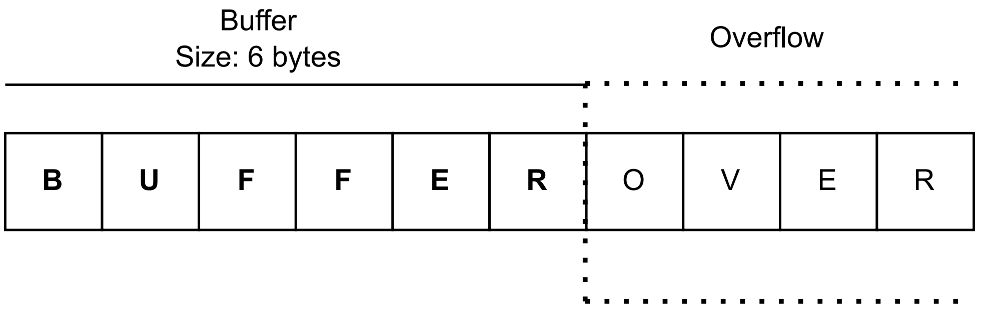

Figure 3.4: Buffer overflow. In the figure is displayed a buffer with a size of 6 bytes. The input 'Overflow' is larger than the given size. Therefore, the overflow occurs, and data are written outside the buffer.

Because the system does not correctly sanitize the given file name from the attacker, the file system receives the whole path and returns the requested file. Therefore, it allows the attacker access to the file system outside the permitted bounds. As a result, the malicious actor can gain potentially sensitive data with a successful path traversal attack.

The correct input validation should be implemented in the system to prevent path traversal. Thus, escape special characters that are usually not in the file names. Furthermore, the system should not return the file to the attacker without providing the required permission for access.

The path traversal vulnerability is relevant for the Internet of Things because smart devices also contain web applications. We proposed this category based on the specifications of the attack and high occurrence in Internet of Things devices. The group is based mainly on CWE-22: *Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal').*

3.3.13 Problematic Authentication/Session Handling

All vulnerabilities in this category are caused by incorrect authentication mechanisms or improper session management. These vulnerabilities usually arise from poor design and incorrect system implementation, leading to various security violations like bypassing authentication or stealing the user's session.

The main aim of the authentication process is to verify the identity of the user. This process is essential for the system's security because the user's identity should be checked before accessing restricted resources. [53] The user must provide during the authentication process login credentials, including his secrets like password, PIN, or key. Based on those identifiers, the system checks the user. If the authentication is successful, the user usually receives a token to further identity checks before accessing resources, executing operations, etc.

The session is established after the successful authentication when the session parameters are set and the tokens are created for the identified user. The tokens, such as cookies or JWT tokens, are usually used for user identification during the entire session. [54] Therefore, they should be appropriately secured, and only the authenticated user should have access to them. However, if the attacker can get them, it can lead to several security problems. For example, the malicious actor can hijack the session or recreate parameters that do not have high entropy.

Furthermore, the whole session should be terminated appropriately. [54] The tokens should be invalidated after a specified time or based on the user's request. If the session can not be ended on demand, it can be critical if the attacker gains the secret tokens.

All vulnerabilities in this category can lead to various security incidents. For example, the attacker can bypass vulnerable authentication processes and gain access to the system without knowing or providing correct login credentials. Furthermore, authentication is often missing in the systems, so resources are freely available. Finally, in the case of session management, the malicious actor can steal the user session and impersonate the victim. This attack can be critical for systems processing sensitive data like internet banking.

The authentication process should be correctly implemented to avoid vulnerabilities in this category. Thus, multifactor authentication and known protocols like OAuth should be used. [55] Also, authentication should be required before accessing nonpublic resources. In addition, for session management, the tokens should be protected from malicious actors. For example, the cookies should have flags like *Secure* and *HttpOnly* to protect them from stealing.

A similar category is repeatedly highly placed in OWASP's TOP 10 list, even though proper authentication and session management are essential for the system's security. Thus, we proposed this category to cover corresponding vulnerabilities due to their severity and possible impact. Some of the Associated CWEs are CWE-278: *Improper authentication*, CWE-384: *Session fixation*, or CWE-306: *Missing authentication for critical function.*

3.3.14 Problematic Cryptography

This category covers all vulnerabilities associated with lacking or insufficient implementation of cryptographic mechanisms in the system. Cryptography is primarily used to secrete data from the access of unauthorized actors. [56] However, poorly implemented cryptographic functions can lead to data leaks and several security breaches. Therefore, it is crucial to secure data during transportation and storage.

Sometimes when data are encrypted, weak or incorrectly implemented algorithms are used. This incorrect behavior can lead to disclosure issues or help the malicious actor further in the attack. For example, if the attacker can access unencrypted data from the network traffic or system logs, he can use gained information like IP addresses in the next part of the attack. In addition, although many weak algorithms were considered strong before, flaws were found in their design, or the computers now have enough computation power to crack them. So nowadays, for example, RC4 and DES are considered weak algorithms. [57]

This category also includes vulnerabilities associated with certificate manipulation and insufficient entropy.

Poor cryptography can lead to numerous security risks and issues. As mentioned above, the attacker can misuse gained data in the following attacks. Furthermore, the malicious actor can leak stolen data like personal or health records to the public and cause harm to the attacked company.

To prevent vulnerabilities in this category, emphasis should be placed on selecting the correct and strong algorithms while designing the system architecture. Furthermore, the system should be prepared for a possible change of cryptographic algorithms in case of failure of their security requirements. Also, developers should never design and implement their security algorithms. Instead, the verified and standard ciphers should be used, for example, AES and RSA.

As mentioned above, using solid cryptographic algorithms is essential for system security. Insufficient encryption can lead to several threats to Internet of Things devices. Therefore, we proposed this category to classify all vulnerabilities associated with encryption. In this category are many CWEs like CWE-295: *Improper Certificate Validation* or CWE-326: *Inadequate encryption strength.*

3.3.15 Problematic Password Management

Following good practices with password manipulation in development is essential to system security. The attacker can misuse improper implementation or another flow in the system to gain the user's passwords. This category contains a wide range of vulnerabilities that arises from poor password management.

Weaknesses can occur already during password creation. First, the sufficient requirements of passwords should be enforced. For example, the minimal length of the password should be set, and passwords from known wordlists should be forbidden. The password wordlists usually consist of popular and often-used passwords that leaked from data breaches. By *SecLists*, the most popular passwords are the following [58]:

1. 123456
2. password
3. 12345678

4. qwerty
5. 123456789

If the login is not adequately secured, the attacker could perform a brute force attack to log in as a victim.

Furthermore, the user should have the option to change login credentials. Often in the Internet of Things devices, the passwords are hard coded. Thus, even when the password is exposed, the user can not change it, and the device is not correctly protected. The malicious actor could, for example, bypass the authentication and misuse the device in the botnet.

Next, the set passwords should be securely stored in the system. Thus, the correctly implemented cryptography method should protect passwords from attackers. For instance, the attacker can attempt to crack a weak hash and gain a user password in clear text. [59]

This category contains vulnerabilities associated with poor password management, as described above. Some vulnerabilities could be considered part of the category *Problematic cryptography* due to encryption issues. However, we placed them in this category because of their connection with password management. This category covers multiple CWEs like CWE-522: *Insufficiently Protected Credentials* and CWE-798: *Use of Hard-coded Credentials.*

3.3.16 Race Condition

The race condition occurs when at least two entities simultaneously attempt to operate with the same shared resource. [60] Thus, it violates the expected behavior of sequential processing of that resource. Race conditions lead to unexpected results and unpredictable behavior. Therefore, vulnerabilities arise, and the adversarial actor can misuse them to attack the system.

The race condition can lead to numerous consequences. First, the vulnerable system can crash because of the attack and be unable to use. Second, the attacker can force the system to perform an adversarial operation. Next, the malicious actor can misuse the incorrect behavior and modify the processed resource—like variables, objects, or files. [61] For example, in a vulnerable banking system, the attacker can send malicious requests to raise a race condition and force the system to modify the account balance to the required value or send money incorrectly. This conduct is pictured in figure 3.5.

Prevention of the attack is to design and implement the system properly. Thus, the sequence of the resource operations should always be strictly given and synchronization mechanisms like locks should be used. Furthermore, the developers should try to avoid shared states as much as possible and thereby prevent potential race conditions.

This category was proposed based on the specific behavior leading to race condition vulnerabilities. It includes CWEs like CWE-336: *Race Condition*

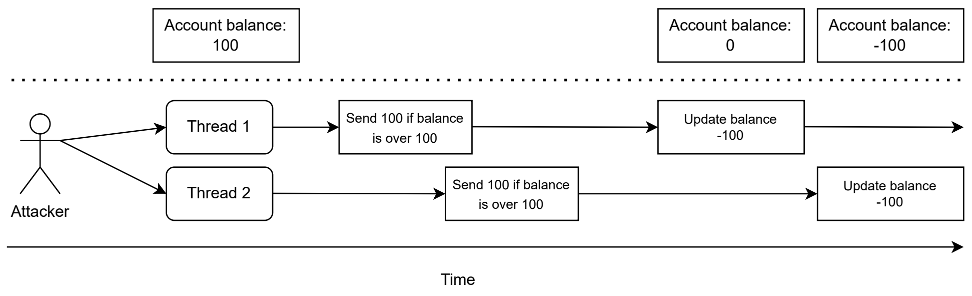

Figure 3.5: Race Condition. The figure describes race conditions in a vulnerable banking system. The attacker can withdraw the same value multiple times and obtain more money than is available.

within a Thread and CWE-367: *Time-of-check Time-of-use (TOCTOU) Race Condition.*

3.3.17 SQL Injection

SQL injection, also known as SQLI, occurs when the attacker injects an adversary payload into a SQL statement and tricks the database into executing it. [62] Thus, the malicious actor misuses improper system validation and neutralization of the part of the SQL statement given by the user. As a result, the database is forced to run the malicious command crafted by the attacker. This attack allows the attacker to read data or take control of the vulnerable database.

SQL is a language used for the manipulation of data in databases. With SQL commands, the user manages numerous records without describing an exact method of processing data in the database. [63] For example, this simple statement `SELECT * FROM computer WHERE computer.os = 'linux'` picks all information from database storage about computers in table *computer* running the operating system Linux.

There are many ways how to accomplish a successful attack. Based on its execution, many types of SQL injection exist, such as in-band (also called classical) or blind. In the in-band injection, the attacker injects a malicious payload into the SQL statement and gets data from the database clearly in response. Thus, it is the most straightforward way of this attack. Another type is blind SQL injection. The attacker crafts the requests with SQL statements the way that the database response *True* or *False*. Then, based on the responses, the attacker can construct the data from the database. [62]

The SQLI attack can have various consequences. The malicious actor can gain data from the database that should not be accessible. The severity depends on the nature of the stolen data. It can be critical for companies in case of stealing personal records or login credentials. Furthermore, the attacker could modify the database content or even destroy it.

The primary protection from SQL injection attacks is proper input validation. Thus, control all data the users give and neutralize all potentially problematic characters. Second, the prepared statements should be used in the database. Furthermore, the SQL statements formed from the user input should be executed with the lowest permissions possible.

We proposed this category based on CWE-89: *Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*. Because of the use of databases in smart devices, SQL injection is a relevant and dangerous threat to the Internet of Things. Furthermore, we decided to separate this category from *Execution of malicious code* due to the specificities of the attack.

3.4 Devices Categories

The Internet of Things in the last few years extended into many fields. Smart devices nowadays are used in smart buildings, multiple industries, and more. Their spread will continue into numerous other areas, and new devices will be invented in the following years. Therefore, the number of categories will likely increase in the future.

We propose categories based on current common-use areas of Internet of Things devices. We even created categories for specific smart devices like cameras or temperature gadgets. Also, some categories may overlap because we picked groups of devices with more specific uses from general categories. For example, due to the extensity and specificity of some smart devices, we extracted the automotive and healthcare industry into their separate categories from the category Industry devices.

3.4.1 Cameras

Cameras are one of the most spread Internet Of Things devices. There are used for multiple purposes. For example, they can be placed in households or buildings for security monitoring. Hence, we proposed this category for the specific device as a camera due to its broad usage.

3.4.2 Car Devices

The Automotive Internet of Things consists of smart devices used in vehicles—for example, sensors, detectors, or monitors. According to the forecast [64], it will be fastly increasing, and by 2030, it should expand by a quarter. Therefore, we created this category of smart devices.

3.4.3 Healthmedical Devices

The following category was proposed to cover devices in The Medical Internet of Things. The range of use in healthcare is broad—for example, monitoring systems for glucose or heart, smart peans, or wearable devices.

3.4.4 Industry Devices

Internet of Things devices are used in many industries like energy, chemical, or mining. Primarily it is devices such as sensors, actuators, monitors, or controllers. We classified the industries with the most extensive representation into their own separate categories.

3.4.5 Lightning

Smart lighting devices have been very popular recently. Thus, we proposed the category only for them. This category includes smart light switches, bulbs, or light strips.

3.4.6 Multiple

In this category belongs the vulnerability records with multiple associated diverse devices, for illustration, if the vulnerability from the same vendor is relevant for camera devices and wearable gadgets.

Example of vulnerability description with multiple affected device groups: *Memory corruption due to use after free issue in kernel while processing ION handles in Snapdragon Auto, Snapdragon Compute, Snapdragon Connectivity, Snapdragon Consumer Electronics Connectivity, Snapdragon Consumer IOT, Snapdragon Industrial IOT, Snapdragon Mobile, Snapdragon Voice & Music, Snapdragon Wearables.*

3.4.7 Network Devices

We suggested this category because network devices also belong to the Internet of Things. This includes all devices providing network communication, such as routers, switches, or gateways.

3.4.8 Smart Buildings

Smart buildings use the Internet of Things devices for operating functionalities and automated management. For example, temperature, humidity, electricity, or lighting can be regulated and monitored through smart devices.

3.4.9 Smart Home Appliances

The popularity of smart home appliances has increased in recent years. As a result, many households use smart appliances such as fridges, vacuums, coffee makers, or televisions. Therefore, we suggested this category.

3.4.10 Smart Small Home Appliances

This category contains small smart devices used in households—like smart plugs, power strips, sockets, alarms, doorbells, or locks. We proposed this based on the vast popularity of smart small home devices.

3.4.11 Temperature Control

The devices for temperature control are broadly used in smart households and buildings. It is The Internet of Things devices like thermostats, air conditioning, heaters, or weather stations.

3.4.12 Voicesound Devices

This category includes all smart devices associated with voice and sound-for example, speakers or voice assistants. We proposed this category because speakers are among the most popular Internet of Things devices.

3.4.13 Wearable Devices

Wearable devices are a broad category. It includes many types of gadgets with various uses, for example, health or fitness. This category includes devices such as fit bands, gloves, glasses, rings, or other jewelry.

Chapter 4

Ranking Lists

This chapter presents numerous ranking lists created from the dataset introduced in the previous chapter Dataset. First, we propose a ranking list of the top 10 vulnerabilities in the Internet of Things and the creation methodology. Second, the ranking list of the top 10 vulnerabilities focusing on camera devices is presented. Finally, we introduce the top vulnerability for each device category.

4.1 Top 10 Vulnerabilities

This ranking list shows the top 10 vulnerabilities in Internet of Things devices. It was formed from the collected vulnerability records. The data was gathered from 3 sources—CVE NIST database, Exploit Database, and Packet Storm, and for their collection, scrapers implemented in the Scrapy framework were used. The raw collected data were after scraping processed automatically and manually. A detailed description of the whole method of dataset creation is shown in section Methodology of Dataset Creation.

The order of the ranking list is set by the number of occurrences of specific vulnerability categories. We did not use any other metrics like CVSS in the computation of the final order due to the lack of it in numerous vulnerability records from the Exploit Database and Packet Storm. The result would not be accurate and consistent if some values were missing.

The final ranking list of the top 10 vulnerabilities in the Internet of Things is presented in the table 4.1. Based on the result, we can assume that Internet of Things devices are most vulnerable due to exposures associated with memory. Also, poor access control and password management widely threaten smart devices due to their high ranking. Furthermore, it briefly corresponds with the ranking list created by OWASP for smart things devices in 2018, even though the methodology of categorizing vulnerabilities into groups slightly differs.

Table 4.1: Ranking List of Top 10 Vulnerabilities in the Internet of Things

Position	Vulnerability category	Occurences
1	Overflow	1116
2	Improper use of memory	624
3	Access control problem	463
4	Execution of malicious code	374
5	Problematic password management	340
6	Problematic authentication/session handling	261
7	Improper input validation	255
8	Denial of service	210
9	Problematic cryptography/certificate manipulation	155
10	Insecure design/design flaw	111

The presented results from the ranking list are similar to what we would expect based on our observations and knowledge. However, the actual state of security in the Internet of Things devices used in organizations and households can differ due to insufficient reporting from vendors and lacking disclosure policies. Nevertheless, the ranking list gives an idea what are most common and threatening vulnerabilities for the Internet of Things devices. It can help vendors during development or security experts with securing smart devices.

4.2 Top 10 Vulnerabilities in Camera Category

This ranking lists present the top 10 vulnerabilities in camera devices. The methodology of creation is identical to the previous ranking list. Therefore, the order is given by the number of occurrences of specific vulnerability types. We proposed this list due to the wide use and popularity of camera devices. The created ranked list is shown in the table 4.2.

The result in the ranking list defined only for cameras varies from common ranking lists. The first place has a category *Problematic password management*, which can be expected due wide use of default or hard-coded credentials in cameras. In the common ranking list, it is placed in fifth place. The ninth and tenth positions in the list are vulnerabilities associated with web applications. That is caused because we also consider vulnerabilities found in web configuration pages for cameras and other smart devices. This list also corresponds with the OWASP Internet of Things 2018 ranking lists. The first place shared similar categories associated with password management.

Table 4.2: Ranking List of Top 10 Vulnerabilities in Camera Devices

Position	Vulnerability category	Occurences
1	Problematic password management	96
2	Access control problem	94
3	Overflow	61
4	Execution of malicious code	58
5	Problematic authentication/session handling	47
6	Improper data handling	24
7	Improper use of memory	23
8	Problematic cryptography/certificate manipulation	18
9	Path traversal	17
10	Cross-site Request Forgery	15

4.3 Top Vulnerability per Device Category

The following list displayed in table 4.3 presents all categories with their most common vulnerability. Like in previous ranking lists, the top vulnerability is determined by the occurrences of a specific vulnerability category. The result in some device categories can be distorted by a low rate of vulnerability records. The amount of vulnerability records for each device group is displayed in figure 4.1.

The categories mostly repeat in the first place of rankings lists for device categories. The most common is the category *Access control problem.* Then it is followed by two categories with the same number of occurrences—*Overflow* and *Problematic authentication/session handling.* Also, categories associated with memory, device design, and authentication appear in the presented table.

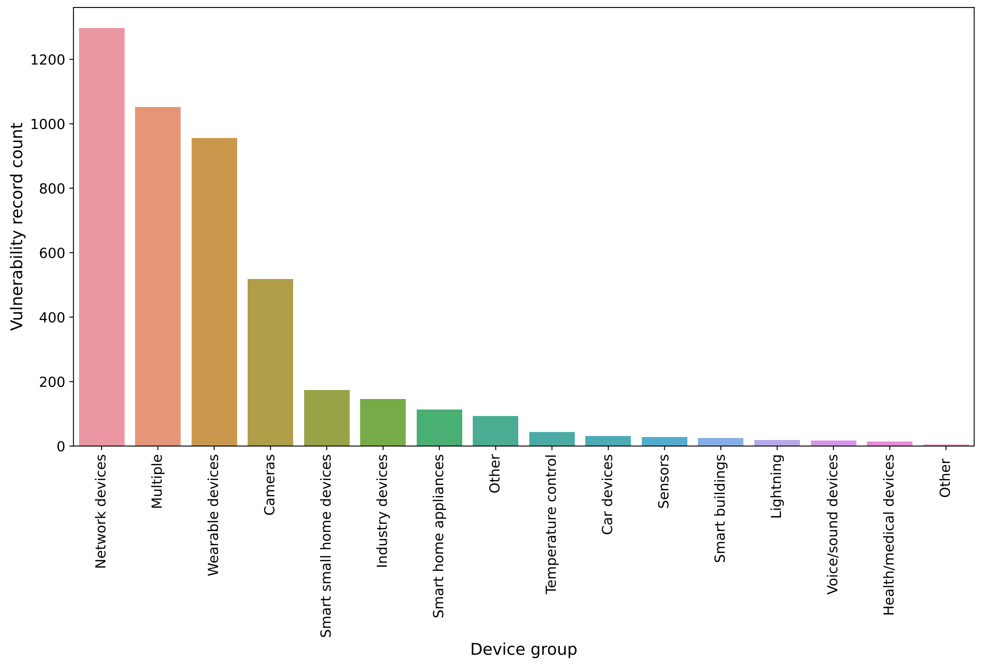

Figure 4.1: This graph shows the number of gathered vulnerability records from multiple sources. Each bar represents a different device group.

Table 4.3: List of Top Vulnerability per Device Category

Device category	Top vulnerability category
Cameras	Problematic password management
Car devices	Problematic authentication/session handling
Health/medical devices	Problematic authentication/session handling
Industry devices	Access control problem
Lightning	Access control problem
Multiple	Overflow
Network devices	Overflow
Other	Access control problem
Sensors	Access control problem
Smart buildings	Access control problem
Smart home appliances	Access control problem
Smart small home devices	Overflow
Temperature control	Problematic authentication/session handling
Voice/sound devices	Insecure design/design flaw
Wearable devices	Improper use of memory

Chapter **5**

Vulnerability Analysis

This chapter presents three chosen vulnerability categories and their analysis in the context of the Internet of Things. We selected those categories based on occurrence and placement in the created ranking lists. The first described category of vulnerabilities is the access control problems. It takes first place in numerous ranking lists separately made for categories of devices. Then overflow vulnerability analysis is presented. The category *Overflow* is the most common threat for Internet of Things devices based on the created ranking lists. Finally, the last section introduces vulnerabilities caused by poor password management. This category is first placed in ranking lists composed of camera devices.

5.1 Access Control Problem

Th access control problems are a widespread threat to Internet of Things devices. This category takes third place in the created ranking list of the top 10 vulnerabilities in smart devices shown in table 4.1. Furthermore, it is in second place in the list explicitly proposed for the cameras presented in table 4.2. Also, access control problems are frequently identified as the most occurred vulnerability category for specific device category. These statistics are presented in the table 4.3.

One of the main uses of The Internet of Things devices is data manipulation. Due to the wide adoption of smart devices in numerous areas like healthcare, industries, and even households, the handled data can be of sensitive nature. Therefore, it is essential to put emphasis on securing them from unauthorized actors with a properly implemented access control system. However, that can be challenging due to the heterogeneous nature of the Internet of Things. The smart devices in a maintained network are often from multiple different vendors and use various technologies and protocols. Furthermore, new devices are easily attached to the existing system, which raises numerous security challenges, including implementing secure access control management.

The general information about access control management and protection is already shown in a section describing vulnerability category Access control problem. In the following part of the thesis, we introduce access control management specifically for Internet of Things devices. Finally, we present known attacks associated with access control vulnerabilities.

5.1.1 Access Control Architecture

There are numerous approaches to the access control architecture design and implementation in the Internet of Things. First is the *centralized architecture* where one central entity handles all operations associated with access control. Therefore, all smart devices rely on the specified entity and leave all access control management on it. This approach has several weaknesses. If the central entity is vulnerable or attacked, the security of all smart devices is compromised. Similarly, if the central object is unavailable, the access control is not enforceable because the smart devices do not have the capability. Therefore, the attacker can misuse the single point of failure to attract the Internet of Things devices. [65]

The next type of architecture is *hybrid architecture.* It is similar to the previous one, but smart devices can participate in access control operations. In the last presented architecture *distributed architecture* is not central objects managing access control operations. Instead, the devices communicate and share data about access control between themselves. It removes a single point of failure problem from previous architectures. [65] The form of architecture is displayed in figure 5.1.

5.1.2 Access Control Models

The process of providing user access to the required entity can be implemented by numerous access control models. In the Internet of Things, smart devices usually use models derivated from known access control models as role-based or attribute-based. [66] The role-based model permits or restricts user access by assigned roles and set policies. Each role has set the rights allowing access to specified entities. Users can have assigned more roles which enables a more straightforward setting of rights to the specific user. However, the model administration can be unbearable in large networks due to dynamic of Internet of Things devices. [65]

In the attribute-based model, the user is granted permission by satisfying the required attributes. [66] The range of attributes that can be set and enforced is wide. For example, the attributes associated with the smart objects like device ID or environment attributes like location. [65]

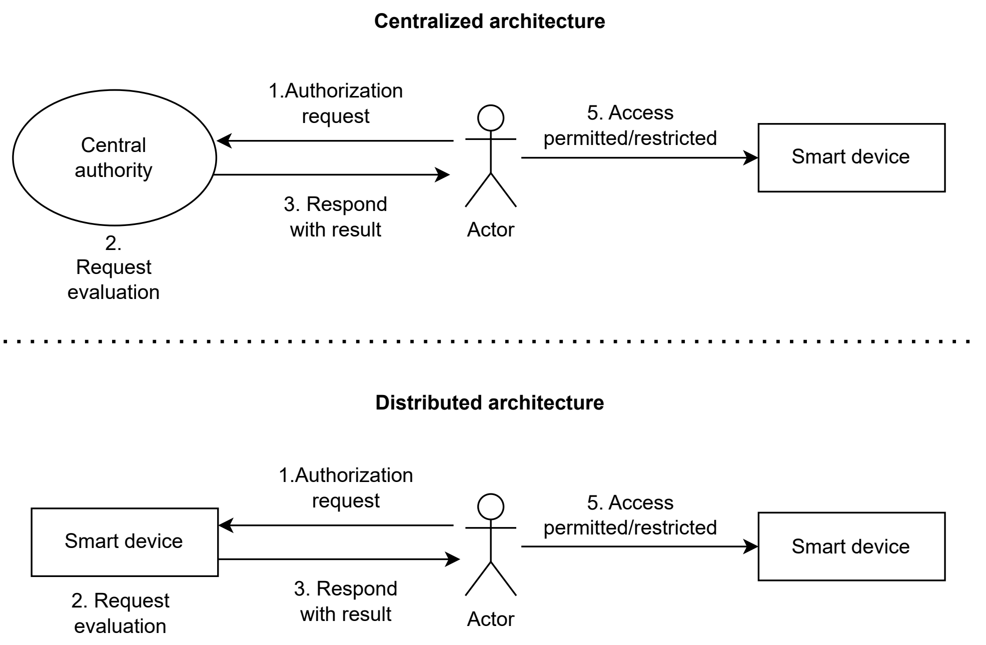

Figure 5.1: Central and distributed access control architecture [65]. This figure shows the differences between the two architectures. In central architecture, the main authority responsible for all actions exists. In a distributed architecture, the devices communicate with each other to execute requested actions.

In this published work [67], the authentication and access control model for the Internet of Things devices is proposed. It is built on centralized architecture and uses a role-based model for access control. The roles representing a position in an organization are put into a hierarchical structure. That should ensure consistency in the network and enable the management of a large number of smart devices efficiently. Furthermore, the model should protect from, for example, replay attacks or eavesdropping attacks. This publication [65] introduces a model suitable for Internet of Things devices. It uses tokens to permit access to the requested resource. This approach enabled work with constrained devices because some devices might not support the required cryptographic algorithms or have insufficient computing power.

Work [66] uses smart contract technologies to implement a framework enabling access control for Internet of Things devices. Specifically, the Ethereum smart contract platform is used. However, smart devices do not perform smart contract operations due to their limitation of computation power. Instead, they are connected to the gateways responsible for Ethereum operations enabling running access control systems via smart contracts. Similarly, the work [68] proposes a schema built on an attribute-based access control model using the Ethereum platform. It aims to provide a reliable access control mechanism and solve issues associated with the Internet of Things—the dynamic behavior and distributed structure.

5.1.3 Related Attacks

There are many known vulnerabilities in Internet of Things devices caused by poor access control implementation. For example, in 2016, numerous vulnerabilities were found in the baby heart monitor produced by Owlet. One of the vulnerabilities was caused by missing access control. The base station and sensor linking did not require authentication. Therefore, the malicious actor could access the device and sensitive data. [69] In 2017, an access control vulnerability was discovered in St. Jude pacemakers. The malicious actor could remotely change the commands and influence the behavior of the pacemaker. For illustration, that could lead to battery drain or modifying pacing. This vulnerability could have severe outcomes as it involves the devices implemented in the patient's body affecting health. The solution to it was to update the firmware by a healthcare provider. [70] Furthermore, an access control flaw was found in Trendnet cameras. Anyone who known the camera's IP address could access the footage, even on the devices that had set up the password. Many vulnerable cameras were found online, and footage from them was published. Therefore, the privacy and security of many people or organizations were disrupted. The vendor published a firmware update to fix the vulnerability in the affected cameras. [71]

5.2 Overflow

The buffer overflow vulnerability is a serious threat. It may allow the attacker to execute malicious code or gain sensitive data. Therefore, it is necessary to protect smart devices against it. However, this category is in first place in the created ranking lists presenting the top 10 vulnerabilities in the Internet of Things in table 4.1. Furthermore, it takes third place in the list introducing vulnerabilities in camera devices proposed in table 4.2. Also, it is placed in the first position in numerous lists explicitly created for the device categories.

Many Internet of Things devices have components implemented in C or C++. Unfortunately, these languages are considered memory unsafe. Therefore, the memory management is mainly on the programmer, which can lead to bugs and buffer overflow vulnerabilities. Securing smart devices against the buffer overflow can be challenging. The standard method may not be sufficient in some cases due to the computing ability and restricted resources of the Internet of Things devices. [72]

The principles of the buffer overflow attack were already described in section Overflow, which introduces the vulnerability category. This section further presents the types of buffer overflow attacks and the protection from them

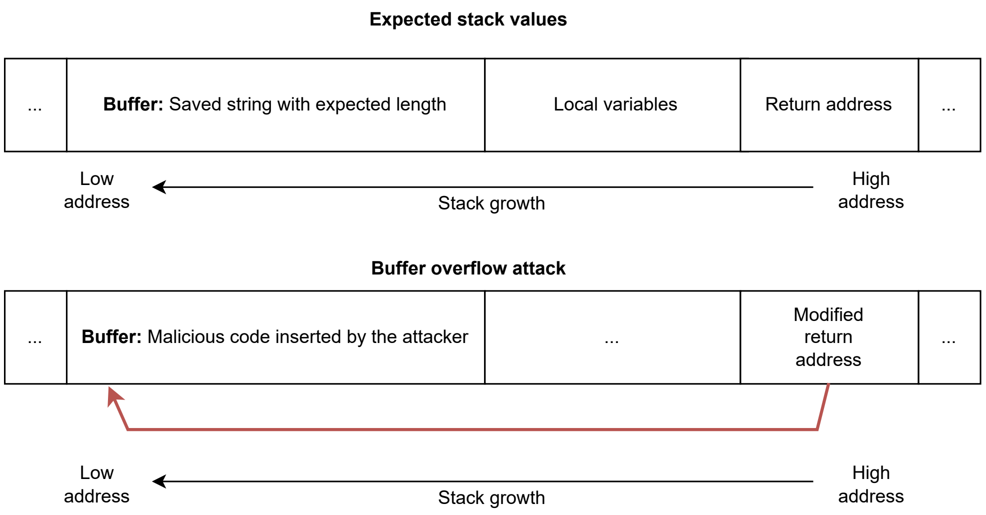

Figure 5.2: Stack buffer overflow attack [74]. In the figure are displayed buffers on the stacks. The first stack shows the expected layout and data storage. In the second one, the buffer overflow attack is pictured.

in Internet of Things devices. Then, known buffer overflow vulnerabilities are introduced.

5.2.1 Buffer Overflow Types

For a successful buffer overflow attack, the malicious actor usually must prepare an advisory code and modify the memory so the system is forced to run the prepared code. Based on the method of attack implementation, the types of buffer overflow exist. In addition, the buffer overflow attacks can be divided by the type of memory. Therefore, heap and stack buffer overflow exist.

In this section, we introduced two methods of executing the buffer overflow attack. In the first technique, the attacker injects the input, including malicious code and the address of the first adversary instruction. The injected address rewrites the return address saved in memory. [73] An example of adversarial input saved memory is shown in figure 5.2. Thus, the system executes malicious code when the return address is fetched from memory and called. Various defense methods exist for this technique and are described in the following section.

To avoid protection, return-oriented programming can be used. As in the previous method, the attacker must force the system to execute the instruction in the desired place. However, the attacker does not inject the malicious code, but it is composed of the parts of already existing code occurring in memory.

The parts are called *gadgets*. By composing the gadgets in sequence, the attacker can perform various operations in the vulnerable system. [74]

5.2.2 Buffer Overflow Protection

A wide range of protection methods from buffer overflow exists. Usually, the combination of them is implemented to protect the device or system from buffer overflow attacks. First, the save programming language should be used. However, in the case of Internet of Things devices, it can be problematic due to their nature. For example, some devices have low computing power. [72] Therefore, the programming language as C must be used for implementation.

In the case of memory-unsafe programming languages, at least the safe version of functions should be used. For example, the function *gets* in programming language C reads data from the input and saves it into the array specified by a given pointer. However, it is considered unsafe because it does not check the input size and boundary limits. Thus, it is vulnerable to a buffer overflow, and the attacker can misuse it for code execution, for example. The function *fgets* should be used as a replacement. It fixes the issues in the function *gets* because it must have the specified allowed input data size as a function parameter.

In addition, the proper programming techniques should be followed by developers to avoid buffer overflow vulnerability. Therefore, during the data manipulation, the data should always be checked if they fulfill the required length based on the buffer size. Furthermore, static or dynamic analysis should be performed of the source code to discover vulnerable parts. [75] However, these analyses do not directly protect from the buffer overflow but can detect vulnerable code and minimize risks.

Many protections from buffer overflow can be applied during the run of the smart device. One of them is address space layout randomization. In this method, the operating system places the segments of the executed program in random places in the memory. Therefore, the attacker cannot predict the address during the attack. However, it can be insufficient protection in Internet of Things devices using 32-bit architecture due to lack of randomization space. [74] The next approach is data execution prevention when certain parts of the memory used for data storage cannot be executed. This protection can be bypassed by return-oriented programming. [74]

These previously described protection methods attempt to prevent buffer overflow attacks. The following techniques differ because they try to detect and stop the ongoing attempt. It can be done, for example, by checking the integrity of the pointers and return addresses. Thus, the system can detect the modification caused by the attacker.

The canary is a value placed into the memory before the return address. Figure 5.3 displays the memory layout with canary value. Suppose the malicious actor misuses buffer overflow vulnerability and attempts to rewrite the

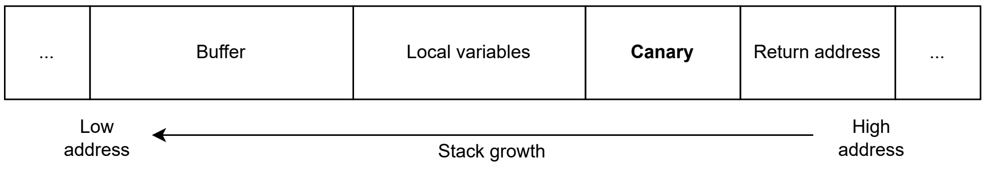

Figure 5.3: Stack layout with canary value [74]. The figure shows a stack layout with a buffer and protection mechanism *Canary.*

return address. In that case, the system detects the canary value modification and restricts the jump to a corrupted return address. There are many different types of canary values. For example, it can consist of null terminators, which protect the functions that detect the end of the string based on that. Furthermore, the canary can be a random value which is also saved in a global variable to further comparison. Due to the randomness, the attacker cannot predict the canary value in advance. [76]

Many works focus directly on protecting the Internet of Things devices from buffer overflow. For example, the work [74] presents a method to protect embedded devices. Therefore, it considers their restricted resources and computing power. The proposed method prevents address modification by the malicious actor. In work [77] also focus on embedded devices protection from buffer overflow attack. The protection comprises two components—hardware boundary checking and integrity of the pointer controlling.

5.2.3 Related Attacks

There are numerous published and described overflow vulnerabilities in Internet of Things devices. For example, the TrendNet camera with a specific firmware version is vulnerable to overflow due to improper parsing of *Authorization header*. The header's value is saved in memory without a check of the length. Therefore, the malicious actor could potentially rewrite memory out of the bound of the buffer and attack the device. Furthermore, the attacker even does not need to authenticate to perform a malicious operation. [78] Similarly, the overflow vulnerability was found in the baby monitor produced by the vendor Victure. The attacker could also misuse the parsing of *Authorization header*. In addition, in the device were discovered multiple other vulnerabilities like missing access control or hard-coded credentials. [79]

Next, in the Sonos speaker, the overflow vulnerability was found. It occurs during the parsing of audio content. The attacker can misuse it to code execution as a root. That is a critical vulnerability because it does not require the authentication of the malicious actor. Therefore, it has a CVSS score of 8.8. The vendor recommends updating the software to a specific version to protect the device from vulnerability. [80]

5.3 Problematic Password Management

Password management is frequently a weak security component in Internet of Things devices. This vulnerability category ranks fifth in the created ranking list of most common vulnerabilities presented in table 4.1. In addition, it has first place in the ranking list specified directly on the camera devices from table 4.2. A similar category appears in first place in the ranking list of the Internet of Things vulnerabilities from 2018 created by OWASP.

The vulnerabilities associated with poor password management are mainly caused by a lack of emphasis on the security by Internet of Things vendors. They often aim to produce devices fast and cheaply. Therefore they do not focus enough on designing proper password management and implementing security mechanisms. In many cases, they even knowingly use hard-coded credentials. That is a significant security issue because many smart devices are used in critical infrastructure and networks. It can lead to numerous severe security incidents like sensitive data breaches or the attacker taking control of the system.

Password management was briefly introduced in section Problematic password management. It describes the vulnerability category of the same name. In the following sections, we provide more details about password setting principles and the correct storage of passwords in smart devices. In addition, default and hard-coded credentials are discussed.

5.3.1 Password Setting

The principles of setting and manipulating password in the Internet of Things is similar to password management in web applications or other systems. The smart device user should be able to set and change a password. In addition, the password should fulfill multiple parameters to protect the system's security. For example, the following password parameters can be enforced:

- password length
- use of numeric characters
- use of special symbols
- use of uppercase and lowercase letters
- does not contain personal information like date of birth
- does not belong in known-password lists
- entropy

Based on those parameters, the system can calculate password strength. Thus, how challenging it is for an attacker to guess the password. The tools used for calculation are named password strength meters. They perform an analysis of the given password and can deliver feedback on the password format to the users. Unfortunately, some password strength meters may be implemented insufficiently. By a study [81], many password strength meters operated by popular companies can produce misleading results on password strength.

Furthermore, the system should check if the user's password is not in the list of known passwords. Those lists are usually freely available online and come from data breaches. For example, the famous list of passwords *RockYou* named after the company where the data leak occurred. Over 32 million user accounts were compromised because the company did not use encryption for user data. From the gathered leaked password, the word list was created and is now widely used by security experts and even attackers. For example, the malicious actor can misuse it for a dictionary attack. [82]

The system should force the user to set a strong password following the specified policies to prevent security incidents caused by weak passwords. Furthermore, the device should limit the amount of failed login attempts, and it should not matter if the tries are performed remotely or if the attacker has physical access to the device. These described mechanisms prevent brute force or dictionary attacks from the malicious actor trying to gain access to the system.

5.3.2 Password Management

The password must be manipulated and stored appropriately. Therefore, the vendors must focus on it sufficiently during device design and implementation. First, encrypted configuration files or databases must be used for storage in the Internet of Things devices. These objects must be adequately protected from unauthorized users due to sensitive content. Only essential entities should be able to access the data for necessary operations. Second, the password cannot be saved in plain text. Strong and proper cryptography algorithms should be used to protect the password from the attacker. [83] If inadequate algorithms are implemented, the attacker can crack the password and gain plaintext form.

5.3.3 Hard-coded/Default Credentials

Frequent security issues in the Internet of Things devices are hardcoded credentials—sensitive login data like usernames and passwords accessible in the source code of the smart device. Many vendors of smart devices use this approach due to simplicity and low costs. They do not consider security aspects. [84]

The default credentials are login data set by the vendor of a device. Usually, there are shared between all the same types of devices. It is on the consumers to change the device credentials to secure ones. However, many users do not modify it, leaving the device insecure.

On the Internet are accessible numerous revealed hard-coded/default login credentials for specific smart devices. Therefore, it is usually easy for the attacker to find the required credentials for the vulnerable devices. Then, the malicious actor can them to login into the victim's device. That can lead to numerous security incidents. In addition, if the demanded login data are not accessible online, the attacker can use reverse engineering methods to attempt to gain data from the source code.

The hard-coded/default credentials in devices are frequently misused by botnets. The botnet is a network of infected machines that the attacker controls. They are often misused in Denial of Services attacks or for sending spam messages. The Internet of Things devices are a common target of the attacker forming the botnet due to their wide use and lack of security.

One of the most famous botnets is Mirai. It possesses mostly the Internet of Things devices. To access the device and take control of it, Mirai uses hard-coded credentials. [83] The impacts of Mirai attacks are noticeable worldwide. For example, in 2016, Mirai raised the Distributed Denial of Service attack against the DNS provider *Dyn*. It led to the inaccessibility of many popular websites like Twitter, Github, and Reddit. [85]

To mitigate security risks associated with the default password, the user must change it before using the device. If the device does not support password change, it should not be used because it is not secure by default. Using knowingly vulnerable devices in the network is an enormous security risk. For example, the attacker can use it as an access point to the company or household network.

Conclusion

This thesis's main goal was to create a vulnerability ranking list for the Internet of Things with transparent methodology and public data sources. Therefore, we used three web pages and APIs containing Internet of Things vulnerabilities to obtain the data. To gather the data automatically, we implemented the scraping tool in the framework Scrapy. During the implementation, we had to deal with numerous challenges associated with scraping content processed by JavaScript and CVSS score unification. Furthermore, we needed to develop a mechanism to categorize scraped vulnerabilities and identify if they belong to the Internet of Things area.

After collecting data, we processed them automatically and manually. The automatic processing was handled by the scraping tool. Further annotation and edits were performed manually. To simplify the manual work, we proposed the *Internet of Things score* to determine if a scraped vulnerability is associated with the Internet of Things.

The cleaned and processed data were further used for dataset creation. It contains of 4532 vulnerability records. Based on the dataset, we proposed various ranking lists for smart devices. First introduced is the ranking list of the top 10 vulnerabilities occurring in the Internet of Things. Then, the top 10 vulnerabilities in the camera devices ranking list is presented. Also, we introduce the most common vulnerbaility for each device category.

From the formed common ranking list, we can assume that the most common vulnerabilities are related to the memory of Internet of Things devices. For cameras, the most common vulnerabilities are associated with password management. A similar category was first in the OWASP ranking list for smart devices created in 2018. However, the classification of categories slightly differs from our created categories, so it cannot be directly compared.

The created ranking lists can be helpful tools for security experts, organizations, and device vendors. It shows the most common threat to the Internet of Things devices. Therefore, it enables prioritizing more spread vulnerabilities and threats first. That can lead to preventing security incidents and severe impacts. Also, ranking lists are valuable for risk management.

In the last part of the thesis, the vulnerability analysis in the context of the Internet of Things was performed. We selected three vulnerability categories in the final ranking lists based on their occurrence. Then, we analyzed them and described protection from malicious actors and possible attacks. The chosen categories were *Access control problem*, *Overflow*, and *Problematic password management*.

In future work, the scraping tool can be expanded in multiple ways. For example, new data sources for vulnerabilities can be added. Next, categories of devices and vulnerabilities can be expanded or divided into more detailed categories. It may even be necessary due to the increasing use of Internet of Things devices and vulnerability reporting. Furthermore, the vulnerability record category classification mechanism can be improved to reduce the error rate and simplify the manual processing and labeling of data. Similarly, the *Internet of Things score* can be edited or replaced with different heuristics.

In addition, the scraping tool can be repeatedly used to collect data for new ranking list creation in the future. Also, it can help with automatic data processing with implemented tools and mechanisms.

The result of this thesis is expected to be processed and published as an academic paper.

Bibliography

[1] Foundation, I. S. IOT Security Foundation announces Fifth Report on consumer IOT vulnerability disclosure policy status. [Visited 25-03-2023]. Available from: `https://www.iotsecurityfoundation.org/iot-security-foundation-announces-fifth-report-on-consumer-iot-vulnerability-disclosure-policy-status/`

[2] OWASP. Owasp Top Ten. [Visited 27-03-2023]. Available from: `https://owasp.org/www-project-top-ten/`

[3] OWASP. Owasp internet of things project. [Visited 27-03-2023]. Available from: `https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project`

[4] Rose, K.; Eldridge, S.; et al. The internet of things: An overview. *The internet society (ISOC)*, volume 80, 2015: pp. 1–50.

[5] Xia, F.; Yang, L. T.; et al. Internet of things. *International journal of communication systems*, volume 25, no. 9, 2012: p. 1101.

[6] Chen, Y.-K. Challenges and opportunities of internet of things. In *17th Asia and South Pacific design automation conference*, IEEE, 2012, pp. 383–388.

[7] Jay, A. Number of internet of things (IOT) connected devices worldwide 2022/2023: Breakdowns, Growth & Predictions. Mar 2023, [Visited 02-04-2023]. Available from: `https://financesonline.com/number-of-internet-of-things-connected-devices/`

[8] OWASP. Vulnerabilities. [Visited 24-03-2023]. Available from: `https://owasp.org/www-community/vulnerabilities/`

[9] Mitre. Common weakness enumeration. [Visited 28-03-2023]. Available from: `https://cwe.mitre.org/`

[10] Mitre. Common weakness enumeration - About. [Visited 28-03-2023]. Available from: https://cwe.mitre.org/about/index.html

[11] CVE. Website. [Visited 28-03-2023]. Available from: https://www.cve.org/About/Overview

[12] NIST. Website. [Visited 28-03-2023]. Available from: https://nvd.nist.gov/products/cpe

[13] Mell, P.; Scarfone, K.; et al. Common Vulnerability Scoring System. *IEEE Security & Privacy*, volume 4, no. 6, 2006: pp. 85–89, doi: 10.1109/MSP.2006.145.

[14] FIRST. Website. [Visited 27-03-2023]. Available from: https://www.first.org/cvss/v2/guide

[15] FIRST. Website. [Visited 27-03-2023]. Available from: https://www.first.org/cvss/v3.0/specification-document

[16] FIRST. Website. [Visited 27-03-2023]. Available from: https://www.first.org/cvss/v3.1/specification-document

[17] Dugal, D.; Rich, D. Common vulnerability scoring system - The State of CVSS to Come. [Visited 27-03-2023]. Available from: https://www.first.org/resources/papers/sig-may-jun2021/CVSS-v4-FIRST-SIG-Update-2021.pdf

[18] OWASP. Owasp. [Visited 27-03-2023]. Available from: https://owasp.org/projects/

[19] MITRE. Common weakness enumeration - TOP 25. Available from: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html

[20] NIST. NVD - General. [Visited 30-03-2023]. Available from: https://nvd.nist.gov/

[21] security, O. Exploit Database - Exploits for Penetration Testers, Researchers, and Ethical Hackers. [Visited 30-03-2023]. Available from: https://www.exploit-db.com/

[22] storm, P. Packet Storm. [Visited 30-03-2023]. Available from: https://packetstormsecurity.com/about/

[23] Richardson, L. Beautiful soup documentation. 2007, [Visited 30-03-2023]. Available from: https://beautiful-soup-4.readthedocs.io/en/latest/

[24] Selenium. Selenium with Python - Selenium Python Bindings 2 documentation. [Visited 31-03-2023]. Available from: `https://selenium-python.readthedocs.io/`

[25] Playwright. Playwright Python. [Visited 31-03-2023]. Available from: `https://playwright.dev/python/docs/intro`

[26] Scrapy. Scrapy 2.8 documentation - Scrapy 2.8.0 documentation. [Visited 31-03-2023]. Available from: `https://doc.scrapy.org/en/latest/index.html`

[27] Scrapy. Scrapy-plugins/scrapy-playwright: playwright integration for Scrapy. [Visited 31-03-2023]. Available from: `https://github.com/scrapy-plugins/scrapy-playwright`

[28] NLTKproject. NLTK - Natural Language Toolkit. [Visited 31-03-2023]. Available from: `https://www.nltk.org/`

[29] Python. Asyncio — Asynchronous I/O. [Visited 31-03-2023]. Available from: `https://docs.python.org/3/library/asyncio.html`

[30] Scrapy. Items - Scrapy 2.8.0 documentation. [Visited 1-04-2023]. Available from: `https://doc.scrapy.org/en/latest/topics/items.html`

[31] Scrapy. Spiders - Scrapy 2.8.0 documentation. [Visited 1-04-2023]. Available from: `https://doc.scrapy.org/en/latest/topics/spiders.html`

[32] Nowak, M.; Walkowski, M.; et al. Machine learning algorithms for conversion of CVSS base score from 2.0 to 3. x. In *Computational Science–ICCS 2021: 21st International Conference, Krakow, Poland, June 16–18, 2021, Proceedings, Part III*, Springer, 2021, pp. 255–269.

[33] Beckerle, M.; Martucci, L. A. Formal definitions for usable access control rule sets from goals to metrics. In *Proceedings of the ninth symposium on usable privacy and security*, 2013, pp. 1–11.

[34] Sandhu, R. S.; Samarati, P. Access control: principle and practice. *IEEE communications magazine*, volume 32, no. 9, 1994: pp. 40–48.

[35] Siddiqui, M. S.; Verma, D. Cross site request forgery: A common web application weakness. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, IEEE, 2011, pp. 538–543.

[36] Barth, A.; Jackson, C.; et al. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 75–88.

[37] Gupta, S.; Gupta, B. B. Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, volume 8, 2017: pp. 512–530.

[38] Hydara, I.; Sultan, A. B. M.; et al. Current state of research on cross-site scripting (XSS)–A systematic literature review. *Information and Software Technology*, volume 58, 2015: pp. 170–186.

[39] NCSC. Denial of service (DOS) guidance. [Visited 15-03-2023]. Available from: `https://www.ncsc.gov.uk/collection/denial-service-dos-guidance-collection`

[40] Carl, G.; Kesidis, G.; et al. Denial-of-service attack-detection techniques. *IEEE Internet computing*, volume 10, no. 1, 2006: pp. 82–89.

[41] Yoachimik, O. Cloudflare ddos threat report 2022 Q3. Nov 2022, [Visited 27-03-2023]. Available from: `https://blog.cloudflare.com/cloudflare-ddos-threat-report-2022-q3/`

[42] Wikipedia. Arbitrary code execution. Mar 2023, [Visited 16-03-2023]. Available from: `https://en.wikipedia.org/wiki/Arbitrary_code_execution`

[43] Fiskiran, A. M.; Lee, R. B. Runtime execution monitoring (REM) to detect and prevent malicious code execution. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.*, IEEE, 2004, pp. 452–457.

[44] Pooj, K.; Patil, S. Understanding File Upload Security for Web Applications. *International Journal of Engineering Trends and Technology*, volume 42, no. 7, 2016: pp. 342–347.

[45] Tayan, O. Concepts and tools for protecting sensitive data in the it industry: a review of trends, challenges and mechanisms for data-protection. *International Journal of Advanced Computer Science and Applications*, volume 8, no. 2, 2017.

[46] Scholte, T.; Robertson, W.; et al. Preventing input validation vulnerabilities in web applications through automated type analysis. In *2012 IEEE 36th annual computer software and applications conference*, IEEE, 2012, pp. 233–243.

[47] Azevedo de Amorim, A.; Hriţcu, C.; et al. The meaning of memory safety. In *Principles of Security and Trust: 7th International Conference, POST 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings 7*, Springer, 2018, pp. 79–105.

[48] OWASP. Owasp secure coding practices-quick reference guide. [Visited 17-03-2023]. Available from: `https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/`

[49] Lhee, K.-S.; Chapin, S. J. Buffer overflow and format string overflow vulnerabilities. *Software: practice and experience*, volume 33, no. 5, 2003: pp. 423–460.

[50] Wikipedia. Data buffer. Mar 2023, [Visited 17-03-2023]. Available from: `https://en.wikipedia.org/wiki/Data_buffer`

[51] Larochelle, D.; Evans, D. Statically detecting likely buffer overflow vulnerabilities. In *2001 USENIX Security Symposium, Washington, DC*, Version: http://www. usenix. org/events/sec01/larochelle. html, 2001, pp. –.

[52] Flanders, M. A simple and intuitive algorithm for preventing directory traversal attacks. *arXiv preprint arXiv:1908.04502*, 2019.

[53] Ometov, A.; Bezzateev, S.; et al. Multi-factor authentication: A survey. *Cryptography*, volume 2, no. 1, 2018: p. 1.

[54] Vlsaggio, C. A.; Blasio, L. C. Session management vulnerabilities in today's web. *IEEE Security & Privacy*, volume 8, no. 5, 2010: pp. 48–56.

[55] OWASP. Authentication cheat sheet. [Visited 18-03-2023]. Available from: `https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html`

[56] Rivest, R. L. Cryptography. In *Algorithms and complexity*, Elsevier, 1990, pp. 717–755.

[57] OWASP. WSTG - Testing for Weak Encryption. [Visited 18-03-2023]. Available from: `https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/04-Testing_for_Weak_Encryption`

[58] Miessler, D. Danielmiessler/Seclists. [Visited 18-03-2023]. Available from: `https://github.com/danielmiessler/SecLists`

[59] Wikipedia. Password policy. Jul 2022, [Visited 18-03-2023]. Available from: `https://en.wikipedia.org/wiki/Password_policy`

[60] Netzer, R. H.; Miller, B. P. What are race conditions? Some issues and formalizations. *ACM Letters on Programming Languages and Systems (LOPLAS)*, volume 1, no. 1, 1992: pp. 74–88.

[61] CWE. Common weakness enumeration. [Visited 19-03-2023]. Available from: `https://cwe.mitre.org/data/definitions/362.html`

[62] Halfond, W. G.; Viegas, J.; et al. A classification of SQL-injection attacks and countermeasures. In *Proceedings of the IEEE international symposium on secure software engineering*, volume 1, IEEE, 2006, pp. 13–15.

[63] NIST. [Visited 20-03-2023]. Available from: `https://www.itl.nist.gov/div897/ctg/dm/sql_info.html`

[64] VerifiedMarketResearch. IOT in automotive market size, share, trends, Opportunities and Forecast. Jan 2023, [Visited 27-03-2023]. Available from: `https://www.verifiedmarketresearch.com/product/iot-in-automotive-market/`

[65] Andaloussi, Y.; El Ouadghiri, M. D.; et al. Access control in IoT environments: Feasible scenarios. *Procedia computer science*, volume 130, 2018: pp. 1031–1036.

[66] Zhang, Y.; Kasahara, S.; et al. Smart Contract-Based Access Control for the Internet of Things. *IEEE Internet of Things Journal*, volume 6, no. 2, 2019: pp. 1594–1605, doi:10.1109/JIOT.2018.2847705.

[67] Liu, J.; Xiao, Y.; et al. Authentication and Access Control in the Internet of Things. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, 2012, pp. 588–592, doi:10.1109/ICDCSW.2012.23.

[68] Song, L.; Li, M.; et al. Attribute-based access control using smart contracts for the internet of things. *Procedia computer science*, volume 174, 2020: pp. 231–242.

[69] Thomson, I. Wi-Fi Baby Heart Monitor may have the worst IOT security of 2016. Oct 2016, [Visited 14-04-2023]. Available from: `https://www.theregister.com/2016/10/13/possibly_worst_iot_security_failure_yet/`

[70] Davis, J. FDA to patients with St. Jude Pacemakers: Update needed to keep hackers out of devices. Aug 2017, [Visited 14-04-2023]. Available from: `https://www.healthcareitnews.com/news/fda-patients-st-jude-pacemakers-update-needed-keep-hackers-out-devices`

[71] Zetter, K. Flaw in home security cameras exposes live feeds to hackers. Feb 2012, [Visited 14-04-2023]. Available from: `https://www.wired.com/2012/02/home-cameras-exposed/`

[72] Mullen, G.; Meany, L. Assessment of Buffer Overflow Based Attacks On an IoT Operating System. In *2019 Global IoT Summit (GIoTS)*, 2019, pp. 1–6, doi:10.1109/GIOTS.2019.8766434.

[73] Fu, D.; Shi, F. Buffer Overflow Exploit and Defensive Techniques. In *2012 Fourth International Conference on Multimedia Information Networking and Security*, 2012, pp. 87–90, doi:10.1109/MINES.2012.81.

[74] Habibi, J.; Panicker, A.; et al. DisARM: mitigating buffer overflow attacks on embedded devices. In *Network and System Security: 9th International Conference, NSS 2015, New York, NY, USA, November 3-5, 2015, Proceedings 9*, Springer, 2015, pp. 112–129.

[75] Fu, D.; Shi, F. Buffer Overflow Exploit and Defensive Techniques. In *2012 Fourth International Conference on Multimedia Information Networking and Security*, 2012, pp. 87–90, doi:10.1109/MINES.2012.81.

[76] Cowan, C.; Wagle, F.; et al. Buffer overflows: Attacks and defenses for the vulnerability of the decade. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, IEEE, 2000, pp. 119–129.

[77] Shao, Z.; Zhuge, Q.; et al. Defending embedded systems against buffer overflow via hardware/software. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, 2003, pp. 352–361, doi:10.1109/CSAC.2003.1254340.

[78] Munawwar. TrendNet wireless camera buffer overflow vulnerability. Dec 2022, [Visited 18-04-2023]. Available from: `https://payatu.com/blog/trendnet-wireless-camera-buffer-overflow-vulnerability/`

[79] Bitdefender. Security cracking the victure IPC360 monitor - bitdefender. [Visited 18-04-2023]. Available from: `https://www.bitdefender.com/files/News/CaseStudies/study/402/Bitdefender-PR-Whitepaper-VictureIPC-creat5590-en-EN.pdf`

[80] ZeroDayInitiative. ZDI-23-449 — Zero Day Initiative. [Visited 18-04-2023]. Available from: `https://www.zerodayinitiative.com/advisories/ZDI-23-449/`

[81] de Carné de Carnavalet, X.; Mannan, M. From very weak to very strong: Analyzing password-strength meters. In *Network and Distributed System Security Symposium (NDSS 2014)*, Internet Society, 2014, pp. –.

[82] Wikipedia. Rockyou. Dec 2022, [Visited 14-04-2023]. Available from: `https://en.wikipedia.org/wiki/RockYou`

[83] Singh Verma, R.; Chandavarkar, B. Hard-coded credentials and web service in IoT: issues and challenges. *International Journal of Computational Intelligence & IoT, Forthcoming*, volume 2, no. 3, 2019.

[84] Chandavarkar, B. R. Hardcoded Credentials and Insecure Data Transfer in IoT: National and International Status. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2020, pp. 1–7, doi:10.1109/ICCCNT49239.2020.9225520.

[85] Wikipedia. Mirai (malware). Mar 2023, [Visited 14-04-2023]. Available from: `https://en.wikipedia.org/wiki/Mirai_(malware)`

Appendix A

Acronyms

AES	Advanced Encryption Standard
API	Application Programming Interface
CPE	Common Platform Enumeration
CSRF	Cross-site Request Forgery
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DES	Data Encryption Standard
DOM	Document Object Model
HTML	Hypertext Markup Language
IP	Internet Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
NIST	National Institute of Standards and Technology
NLTK	Natural Language Toolkit
OWASP	Open Worldwide Application Security Project
SQL	Structured Query Language
XML	Extensible Markup Language

XSS	Cross-site scripting
URL	Uniform Resource Locator

Appendix B

Created Ranking Lists

Table B.1: Ranking List of Top 10 Vulnerabilities in the Internet of Things

Position	Vulnerability category	Occurences
1	Overflow	1116
2	Improper use of memory	624
3	Access control problem	463
4	Execution of malicious code	374
5	Problematic password management	340
6	Problematic authentication/session handling	261
7	Improper input validation	255
8	Denial of service	210
9	Problematic cryptography/certificate manipulation	155
10	Insecure design/design flaw	111

Table B.2: Ranking List of Top 10 Vulnerabilities in Camera Devices

Position	Vulnerability category	Occurences
1	Problematic password management	96
2	Access control problem	94
3	Overflow	61
4	Execution of malicious code	58
5	Problematic authentication/session handling	47
6	Improper data handling	24
7	Improper use of memory	23
8	Problematic cryptography/certificate manipulation	18
9	Path traversal	17
10	Cross-site Request Forgery	15

Table B.3: List of Top Vulnerability per Device Category

Device category	Top vulnerability category
Cameras	Problematic password management
Car devices	Problematic authentication/session handling
Health/medical devices	Problematic authentication/session handling
Industry devices	Access control problem
Lightning	Access control problem
Multiple	Overflow
Network devices	Overflow
Other	Access control problem
Sensors	Access control problem
Smart buildings	Access control problem
Smart home appliances	Access control problem
Smart small home devices	Overflow
Temperature control	Problematic authentication/session handling
Voice/sound devices	Insecure design/design flaw
Wearable devices	Improper use of memory

Appendix C

Structure of Enclosed Files

```
README.txt......................the file with attched content description
├── datasets.................................folder with created datasets
├── jupyter_notebooks.........................attached Jupyter notebooks
├── scraper.................................Python code of scraping tool
└── thesis_text............................folder with thesis source code
```