TU WIEN Informatics

Internet of Things Testbed

Ein modulares Framework um Schwachstellen zu finden

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Kevin Gufler, BSc
Matrikelnummer 11775815

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Mitwirkung: Univ.Lektor Dipl.-Ing. Dr.techn. Georg Merzdovnik, BSc
Univ.Lektor Dipl.-Ing. Christian Kudera, BSc
Dipl.-Ing. Michael Pucher, BSc

Wien, 8. Mai 2024

Kevin Gufler | Edgar Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Internet of Things Testbed

A modular Framework to find vulnerabilities

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering/Internet Computing

by

Kevin Gufler, BSc
Registration Number 11775815

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar Weippl
Assistance: Univ.Lektor Dipl.-Ing. Dr.techn. Georg Merzdovnik, BSc
Univ.Lektor Dipl.-Ing. Christian Kudera, BSc
Dipl.-Ing. Michael Pucher, BSc

Vienna, May 8, 2024

Kevin Gufler | Edgar Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Kevin Gufler, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 8. Mai 2024

Kevin Gufler

Danksagung

Ich möchte diese Gelegenheit nutzen, um allen zu danken, die mich während der Erstellung dieser Abschlussarbeit und der Reise durch das Studiums unterstützt und begleitet haben.

Zunächst gilt mein tiefster Dank meiner Familie. Ohne ihre stetige und bedingungslose Unterstützung wäre diese Arbeit nicht möglich gewesen. Sie waren immer für mich da und haben mir die Kraft und den Rückhalt gegeben, die ich benötigt habe, um mein Studium erfolgreich zu meistern.

Ein besonderer Dank geht an meine Tutoren. Ihre fachliche Unterstützung und die Bereitschaft, sich Zeit für meine Fragen und Anliegen zu nehmen, waren für mich während des gesamten Forschungsprozesses von unschätzbarem Wert.

Ich möchte auch die Geschwister Moritz und Luise Ilg erwähnen, mit denen ich gemeinsam die Höhen und Tiefen des Studiums erlebt habe. Eure Freundschaft und Zusammenarbeit waren eine große Bereicherung für meine akademische und persönliche Entwicklung.

Nicht zu vergessen sind meine Freunde und Freundinen, die stets ein offenes Ohr für mein Jammern hatten, wenn mal etwas nicht nach Plan lief. Eure Geduld und euer Verständnis waren eine enorme Hilfe und haben mir immer wieder neuen Mut zugesprochen.

Ich bin jedem Einzelnen von euch zutiefst dankbar für die Unterstützung, Motivation und die vielen kleinen Dinge, die diesen wichtigen Abschnitt meines Lebens bereichert haben.

Kurzfassung

Die sich ständig verändernde digitale Landschaft hat zu einem sprunghaften Anstieg der Nutzung von Geräten des Internets der Dinge (IoT) sowohl im privaten als auch im beruflichen Umfeld geführt. Dies hat den dringenden Bedarf an starken Cybersicherheitsmaßnahmen unterstrichen, um diese Geräte vor ihren zahlreichen Schwachstellen zu schützen. Mit dem Fortschreiten der IoT-Technologie wird diese zunehmend mit dem täglichen Leben verwoben, so dass die Sicherheit dieser Geräte nicht nur eine Option, sondern eine Notwendigkeit ist. Diese Arbeit zielt darauf ab, im Bereich der IoT-Cybersicherheit eine Vorreiterrolle einzunehmen, indem sie ein modulares IoT-Testbed entwickelt, bewertet und Verbesserungen vorschlägt, um die kritischen Cybersicherheitsherausforderungen für IoT-Geräte zu bewältigen. In einer Welt, in der IoT-Geräte immer mehr in unser tägliches Leben integriert werden, ist der Schutz dieser Geräte vor potenziellen Bedrohungen und Schwachstellen von größter Bedeutung. Das durch diese Forschung geschaffene Testbed bietet eine zuverlässige Plattform für automatisierte Penetrationstests, die Sammlung von Informationen und die Darstellung der Ergebnisse in einem klaren Format.

Die erste Phase dieser Forschung konzentrierte sich auf die Architektur des IoT-Testbeds und betonte dessen modularen Aufbau, der die Integration von Tools zur Durchführung umfassender Sicherheitsbewertungen erleichtert. Die Fähigkeit des Testbeds, sowohl ein- als auch ausgehenden Netzwerkverkehr zu erfassen und zu analysieren, wurde ebenso unter die Lupe genommen wie die Implementierung von automatisierten Testpipelines, die die Effizienz der Schwachstellenerkennung erheblich steigern.

Die Bewertung des Testbeds zeigte ihre Stärken bei der Identifizierung und Ausnutzung bekannter Schwachstellen in IoT-Geräten, insbesondere durch den Einsatz von Tools wie Hydra für Brute-Force-Angriffe und Nmap für Netzwerk-Scans. Allerdings wurden Einschränkungen bei der Erfassung des eingehenden Datenverkehrs und der Bedarf an einem breiteren Toolset als Bereiche identifiziert, die eine weitere Entwicklung erfordern. Die Verwendung von HTML-Jinja-Vorlagen für die Ergebnispräsentation wurde als Schlüsselmerkmal hervorgehoben, das die Umwandlung von Rohdaten in informative und ästhetisch ansprechende Formate ermöglicht.

Mit Blick auf die Zukunft beschreiben wir einige Ideen, welche die Verbesserung der Fähigkeiten des Testbeds zur Informationserfassung, Erweiterung der Testwerkzeuge und die Erforschung der Integration von maschinellen Lernverfahren zur automatischen Erkennung von Anomalien umfassen und die nötigen Rechte des Testbeds.

Abstract

The ever-changing digital landscape has seen a surge in the use of Internet of Things (IoT) devices in both personal and professional settings. This has underscored the urgent need for strong cybersecurity measures to protect these devices from their numerous vulnerabilities. As IoT technology advances, it becomes increasingly intertwined with everyday living, making the security of these devices not just an option but a necessity. This thesis aims to lead the way in IoT cybersecurity by developing, evaluating, and suggesting improvements for a modular IoT Testbed designed to tackle the critical cybersecurity challenges IoT devices face. In a world where IoT devices are becoming more integrated into our daily lives, safeguarding them from potential threats and vulnerabilities is paramount. The Testbed created through this research provides a reliable platform for automated penetration testing, information gathering, and presenting findings in a clear format.

The initial phase of this research focused on the architecture of the IoT Testbed, emphasizing its modular design that facilitates the integration of a diverse array of tools for conducting comprehensive security assessments. The Testbed's capability to capture and analyze both inbound and outbound network traffic was scrutinized, alongside the implementation of automated testing pipelines that significantly enhance the efficiency of vulnerability detection.

Evaluation of the Testbed revealed its strengths in identifying and exploiting known vulnerabilities within IoT devices, particularly through the use of tools such as Hydra for brute force attacks and Nmap for network scanning. However, limitations in capturing incoming traffic and the need for a broader toolset were identified as areas requiring further development. The Testbed's use of HTML Jinja templates for result presentation was highlighted as a key feature, enabling the transformation of raw output into informative and aesthetically pleasing formats.

Looking forward, we outline a roadmap for future work that includes enhancing the Testbed's information acquisition capabilities, expanding its suite of testing tools, exploring the integration of machine learning techniques for automated anomaly detection and permission management for this application.

Contents

CHAPTER 1

Introduction

The Internet of Things (IoT) growth represents a paradigm shift in the interaction between the digital and physical worlds, affecting a broad spectrum of sectors, including agriculture, healthcare, and urban development. IoT devices, by offering real-time monitoring and control capabilities, have the potential to significantly enhance productivity in agriculture through environmental monitoring, improve quality of life with intelligent healthcare solutions [21], and increase the sustainability of urban environments through the implementation of 'Smart Cities' technologies. These technologies facilitate noise pollution monitoring, efficient traffic light management, and sophisticated waste management systems [23]. Currently, 17.08 billion IoT devices are in use, and this number is expected to grow to 29.42 billion by 2030 [1].

Despite these advancements, integrating IoT technologies brings forth substantial security concerns, particularly considering the grave implications of a compromised device. The automotive industry, for example, has witnessed life-threatening incidents due to sophisticated cyber-attacks, leading to sudden and unexpected vehicle stops on highways [15]. Likewise, in the healthcare domain, a breach in the security of smart devices could pose severe risks to patient safety. Examples of this have already been seen with different hospitals being targeted by Ransomware; in 2022 alone in the USA, 66% of hospitals have been targeted by a ransomware attack [2]. Poorly secured IoT devices might function as a jump host to allow such attacks [8] or steal valuable information. Furthermore, the infrastructure of Smart Cities, despite its innovative approach to urban management, is susceptible to significant security threats, given the extensive array of services it encompasses and the vastness of its network [23].

The risk extends beyond physical damage or disruption of services; the digital realm is equally vulnerable. This was starkly illustrated by the Mirai Botnet incident, wherein over half a million IoT devices with inadequate security were commandeered to execute large-scale Distributed Denial of Service (DDoS) attacks, affecting not just individual

devices but the broader digital ecosystem [7, 19]. Incidents like this Botnet happen because of poorly secured IoT devices, e.g. through the use of standard credentials.

A study showed that 70% of IoT devices have security flaws. Further claiming that each device contains an average of 25 flaws [8]. The fundamental challenge lies in the intrinsic characteristics of IoT devices. With limited computational resources and aggressive pricing strategies, these devices often come equipped with software poisoned with vulnerabilities [8]. Such weaknesses render the devices a target for exploitation by malicious entities. As the IoT landscape continues to expand, these vulnerabilities present an escalating threat, posing risks to individual devices and the integrity of broader networks to which they are connected, potentially compromising more sensitive systems and information[8]. By performing passive network traffic analysis, a device might identify a different IoT device through their traffic; this allows for exploiting a known vulnerability for a specific device/software version [6]. These vulnerabilities are not only used by an attacker trying to build big botnets to steal knowledge worth millions of dollars [5], but advertisers might also use them to listen to consumers [4].

In this thesis, we show the design, implementation, deployment, and expansion of a new modular IoT Testbed that allows its users to quickly set up their private Testbed and use it to test devices for vulnerabilities by starting different tools or pipelines. Since the landscape of IoT devices advances so fast, some tools might become obsolete, or newer tools will be needed. We address this issue by allowing for a modular toolkit design so it can easily be expanded to allow for a new threat exploit. Tools in this field often do not allow an easy entry for users to check for vulnerabilities in their smart home appliances by demanding a good foundation about IoT Devices or how they work. We also address this issue by enabling users to add new tools easily and receive generated reports showing problems on one collective PDF.

Based on this existing research, we formulated the research questions alongside the contributions that are made to answer them:

(**R0**): Can a device be identified by its traffic in a non-ML way and is user information being leaked?
To answer this, the Testbed that is being developed with this thesis is going to be used to check for any leaked data in the network traffic of the test device set.

(**R1**): Can tools be collected to a pipeline to share information and create an attack workflow?
This work will contribute to this question by making a small set of tools showcasing the Testbed's extensibility and allowing for pipelines (chained tools).

(**R2**): Is there a way to allow for dynamically generated PDF exports where the design is easily changeable?
By finding a way to classify tool executions automatically by several factors and collecting them to create a report that supports templating.

CHAPTER 2

Related Work

The world of IoT security has witnessed significant attention aimed at identifying potential vulnerabilities within devices and analyzing their firmware components or their traffic [24, 17, 18, 22, 6]. This pursuit is paralleled by efforts to construct specialized Testbeds designed for the examination of IoT devices or even other devices such as wearables [16].

This chapter will now list different approaches in those fields, starting with manual analysis and then showing a list of different Testbeds and frameworks to automate the testing. After that, some notable papers are shown in the field of device detection, which uses mainly machine learning approaches. Lastly, it covers some works in the field of malicious traffic detection and manipulation of the IoT devices' traffic.

2.1 Manual Analysis

There are several different approaches to attacking and analyzing IoT devices without using a Testbed. The work of Mohd Bakry et al. [38] employs three tools to analyze and attack a Raspberry PI. The attacks are a brute force, a Man in the Middle (MiTM), and a DoS attack with the tools Nmap, Bettercap, and Xerosploit. The setup is a Raspberry PI running SSH and VNC which is connected to a router. The attacker connects itself to the same router and runs these tools. All of the attacks succeeded, and they argue that future research should be done with those attacks on different IoT devices since the test with the Raspberry PI reveals that vulnerabilities could be found in most embedded or IoT-related controllers such as drones, smart bulbs, smart locks, and other IoT devices.

The following work by Ronen et al. [39] analyzed smart bulbs security issues by attempting to gain control over the bulb from 100 meters away with their own receiver. In the test set are high-end and lower-end smart light systems, ranging from an expensive Philips HUE system to a cheap system manufactured by LimitlessLED. They conducted two different tests; the first one concentrated on using smart lights as a covert LIFI communication

tool to read data from an office building 100 meters away. The second test focused on gaining control of a light bulb and letting it strobe at a specific frequency that may trigger seizures in people who have photosensitive epilepsy.

Another interesting work in this field comes from Seralathan et al.[40], where they tested an Anip smart camera for vulnerabilities. As tools, they used Nmap to brute force port RTSP to get a video stream and a Man in the Middle (MITM) attack to analyze the traffic with Wireshark. In this process, they were able to find real-time streams and credentials stored in clear text in the mobile application.

Another approach has been done by Akhilesh et al. [11] by using a Testbed with a Kali Linux server, a WLAN Access Point, and a test set of 5 IoT devices. The goal of this research was to find the most common vulnerabilities that appear in the test set. They further created a new evaluation score, which encompasses more metrics than the current CVSS score to indicate the impact of the given vulnerabilities.

Wood et al. [47] created a paper that introduced a method to capture traffic from medical IoT devices, which then gets analyzed in an automated way, and cleartext information that may reveal sensitive medical conditions and behaviors can be detected. Using two smart scales, two blood pressure monitors, and a blood sugar monitor in this analysis, they found cleartext identification and medical information using a dictionary list of the 100 most common medical terms. One blood pressure monitor even leaked its device type as well as the user behavior in using it.

2.2 Frameworks

By creating a framework those functionalities and routines can be encapsulated and even automated. Efforts in this direction is being demonstrated by the following papers and projects.

One Testbed is SAFER, proposed by Oser et al.[10] it can be used to determine a risk assessment score by analyzing the firmware and using the National Vulnerability Database (NVD). This work used a set of 38 devices to show that their framework can give a good evaluation of the current risk and even further can predict the future risk of a device by analyzing indicators such as response time of fixes for security patches.

Also, in the enterprise area, different approaches have been taken to create a Testbed to pentest IoT devices. One of these projects is the FIT IoT-Lab [9]. This platform contains an extensive network with over 1500 nodes that facilitate a wide range of experiments and tests. Its primary emphasis, however, is on evaluating networking capabilities within small wireless sensor devices and a variety of communicating objects, thus limiting its focus on direct vulnerability assessment within devices.

National Instruments posed another solution [20] where researchers there developed a Testbed tailored for the research of security vulnerabilities in IoT devices, employing methodologies such as port scanning, process enumeration, device fingerprinting, and

vulnerability scanning. Their focus spanned across an array of devices, including smart home appliances, sensors, and wearables some of those devices are a Nest Cam, Philips Hue, Amazon Echo and SENSE Mother. Since this is a proprietary and closed-source National Instruments TestStand, it is not open for normal consumers to use this platform to quickly check for vulnerabilities in their smart home. Further, its closed sources prohibit developers from just creating a new tool in the Testbed if the need exists.

A further contribution to the field is represented by the SecuWear tool [16], created for the extraction of vulnerabilities and information from wearable devices. This approach is characterized by an exhaustive data collection phase followed by a set of targeted attacks on wearables. Despite its innovative approach, the tool's application is predominantly limited to wearables, leaving its utility for a broader spectrum of IoT devices as an area for further exploration.

An additional notable development in this space is the introduction of Zingbox IoT Guardian by Paloalto [34], a comprehensive platform designed for the lifecycle management of IoT devices. This solution not only facilitates initial device setup and robust security enforcement but also provides an intuitive interface for data interaction. It enables users to automate the identification of threats and the execution of corresponding mitigation strategies. Despite these advanced features, as it is with the tool from National Instruments, this tool is also closed-source, and its model of subscription-based access may limit its reach.

An exemplary project of notable significance within the realm of IoT Testbeds is PentOS [37], which pioneers the introduction of an advanced Testbed infrastructure. This Testbed is distinguished by its support for diverse connectivity options, including WLAN and Bluetooth, and with future changes, the researchers also wants to extend its compatibility to ZigBee technologies. PentOS is configured to support a predefined suite of tools and does not allow it to expand its toolset.

Another framework comes from Tekeoglu et al.[41], who, analyzed captured packets from network layers 2 and 3 by using a hub connecting to two access points and a Kali Linux machine. Further, they ran Ubertooth on a different machine and also analyzed the Bluetooth packets with Wireshark. Their test set consisted of HDMI sticks, drones, wireless cameras, activity trackers, and smartwatches, and to control those devices, they used a smartphone with dedicated apps on it. By using Nmap and employing weak password checks or brute force checks, they found a couple of vulnerabilities in the test devices. However, the whole Testbed is not automated, meaning that attacks can be ran one after another but cannot be chained to create workflows.

The penetration testing platform from Abu Waraga et al. [42], which assesses risks and vulnerabilities of their test set of a smart bulb and an IP camera, shows great potential as it is divided into five modules: GUI, Testing, Network, Monitoring, Reporting, and Storage. The Testing module utilizes port scanning, vulnerability scans, downgrade attacks, brute forcing directories, and testing SSL configurations to gather information about the device or test the vulnerabilities with the following tools, which is not a

Name	Contributions	Problems
Safer [10]	Risk Assessment Score and NVD.	No modular design. Does not attack the device. with different tools.
FIT IoT-Lab [9]	Network with over 1500 nodes.	Not a real Testbed more an environment to test IoT devices behavior in a network.
NI: Let the Cat Out of the Bag [20]	Testbed with different approaches to pentest a device.	Closed-source.
SecuWear [16]	Testbed that allows to scan mostly wearables.	Only wearables so different IoT devices can not be scanned.
ZingBox [34]	Testbed that covers the whole lifecycle of a device and allows for a generated export.	Closed-source and subscription-based.
PentOS [37]	A Testbed for devices connecting via LAN, WLAN, and Bluetooth having multiple tools.	PentOSs design is modular but mostly supports only its predefined suite of tools.
Testbed for Security and Privacy [41]	This Testbed uses a collection of tools and tests network and bluetooth devices.	Framework is not automated.
IoT Security Testbed [42]	A modular Testbed that uses a varity of tools to test for vulnerbailties.	Small number of devices that were being tested.

Table 2.1: Testbed solutions

complete collection of the ones being used in this project: Nmap, Metasploit, Tshark, SQLmap, SSLStrip and Nikito. The results showed that the IP camera was vulnerable to a couple of attacks.

To summarize each project with their problems is listed in Table 2.1. Most of the solutions are either not modular and do not allow for enhancement of its toolset, are not open source, or are subscription-based and thus do not allow for a quick check of all the IoT Devices in a household.

2.3 Device detection

In the field of device reconnaissance, some studies have been conducted; for example, the authors of the paper by Shahid et al. [6] created an experimental smart home network where they analyzed streams of packets from four different devices. Further, six different classification algorithms (Random Forest, Decision Tree, SVM (with rbf kernel), k-Nearest Neighbors, Artificial Neural Network (ANN) and Gaussian Naive Bayes) have been used to guess the name of the IoT device that is generating traffic having the Random Forest as their winning classifier. They achieved a minimum accuracy as high as 95%. However, they had problems classifying devices if they come from the same vendor, as they could have very similar traffic.

Further, Acar et al.[12] also developed a machine learning approach to detect devices by their traffic, encrypted or plain. They claim that an adversary passively sniffing the traffic can achieve a high accuracy of 90% with their model trained from 22 devices. Stating that the system loses accuracy in a multi-user environment as opposed to a single-user environment where it is easier to track the behavior of a device.

Another work in this area is the work by Sivanathan et al. [13], who also created a machine learning project that can identify IoT devices with over 99% accuracy in an environment of 28 devices.

ProfilIoT [14] is a project very similar to the others, again a machine learning approach to identify IoT devices by their network traffic. For this, they used a test set of nine IoT devices and got an overall classification accuracy of 99.281%. They even claim to be able to identify a device down to the model and not just its brand.

To summarize, each paper is listed in Table 2.2. As we can see, each solution already provides a good accuracy while detecting different devices of different brands. Difficulty arises from similar devices from the same brand as they often use a very similar software. Almost all of the solutions use a AI driven approach to guess the type and brand of the device they belong to. Those approaches often need a large dataset of devices to train the underlying model and a large model might bring latency.

2.4 Malicious traffic detection and manipulation

Additional work has been done in the field of mailicous device detection, where the goal is not to identify the device type, brand, or model but instead to identify devices that generate malicious or strange traffic.

One of those projects is Ghost[43]. Ghost is a project that can extract meaningful information from a live traffic capture and detect abnormal situations in the traffic while also being hardware agnostic so that a vast collection of people can use it. They also released network traffic data sets that can be used for future analysis.

Name	no. of devices	accuracy	Note
Shahid et al. [6]	4	95%	Hard to distinguish similar devices of the same brand.
Acar et al. [6]	22	90%	Losing accuracy in a multi-user environment.
Sivanathan et al. [13]	28	99%	
ProfilIoT [14]	9	99.28%	Can distinguish different models of the same brand.

Table 2.2: Network traffic analysis

Another project is IoT-Keeper[44], which uses Gated Recurrent Units (GRUs) to analyze network traffic and detect DOS attacks with a high success rate (TPR = 0.89) and scanning attacks with a TPR = 1.0. Those high rates, however, come with a high resource footprint, which means that resource-constrained devices cannot employ this solution. They compared it with another project and were also able to detect other network attacks, such as MitM attacks, with their solution.

Another solution by Subahi and Theodorakopoulos [45] that concentrates on the phone, which gives commands to the IoT devices mainly in the direction of App-to-cloud. They were able to classify each user's interaction with the device, label them with a sensitivity level, and label them with the information that can be found in the package, such as credentials or user location. While doing so, they achieved a detection for the interaction labeling rate of 99.4%, they detect packets carrying sensitive information by 99.8% and further can identify the content type of those sensitive information packets with a success rate of 99.8%.

Another example is the project of Bachy et al. [46], who analyzed smart TVs traffic and apps and launched attacks on them. They did this by intercepting channels or attacking those apps that are running on the TV with four different smart TV types. They extracted the firmware of those TVs and applied XSS attacks on their web browser to gather information or gain control over the TV. They found that some TVs were being updated over an unsecured channel, which made them vulnerable to firmware modification attacks.

CHAPTER 3

Background

The following chapter delves into fundamental knowledge essential to IoT Testbeds and devices. Its purpose is to equip readers with a grasp of the subject matter, enabling an understanding of the topics covered in this thesis. Furthermore, it sheds light on the reasoning behind crucial decisions made throughout the thesis, including those concerning the selection of the operating system.

3.1 IoT Devices

The Internet of Things (IoT) introduces a new era where everyday objects are interconnected to form a network of intelligent devices. These devices vary in complexity and capability, from basic sensors to advanced industrial equipment. They are changing the way people interact with the physical and digital world. IoT devices can communicate and function over networks, particularly the internet, which facilitates the autonomous exchange and processing of data.

3.1.1 Diverse Hardware Configurations and Applications

The landscape of IoT (Internet of Things) devices is diverse, reflecting the wide-ranging applications they are designed for. This diversity highlights the adaptability of IoT devices to perform specific tasks. For example, a traditional smart thermostat is equipped with sensors, processors, and communication modules specifically designed for environmental monitoring and control. In contrast, advanced IoT devices like smart vehicles incorporate GPS modules, advanced computing hardware, and various sensors designed to perform multiple tasks, including navigation and safety management.

The design philosophy of IoT devices is centered on optimizing task-specific performance, emphasizing energy efficiency, compactness, and cost-effectiveness. These principles are critical for ensuring the scalability and sustainability of the IoT ecosystem, which is

rapidly becoming a cornerstone of modern technological infrastructure. However, while IoT devices can gather an enhanced understanding of their environment, they are also vulnerable to unintentionally disseminating information to malicious entities.

3.1.2 Connectivity and Communication

The functionality of IoT devices is centered around their connectivity, which allows them to seamlessly integrate into larger networks. Different technologies, such as WLAN, Bluetooth, Zigbee, and cellular networks, enable this connectivity based on the specific requirements of each IoT application. Factors such as operational range, data bandwidth, and energy consumption are vital in determining the appropriate connectivity method.

In addition, IoT devices often operate within larger systems, using gateways to access the internet or connect to centralized servers. This interconnectedness enhances the usefulness of IoT devices but can also introduce complications in managing network traffic and ensuring secure data transmission.

This Testbed is specifically designed to integrate with an existing network infrastructure. If the WLAN is already operational within the network, it can facilitate the connection of devices to the Testbed and LAN. While some IoT devices on the market can be easily used as standalone devices and only need to be connected to a network with a working network connection, some other devices, such as Zigbee, need a hub to work. While the testbed planned in this thesis does not support Zigbee directly, a Zigbee station can act as gateway that is then connected to the Testbed. When using a hub, such as Zigbee, the devices that are being connected to the smart home network will not directly access the internet; instead, they will use the route to the hub to access services in the cloud.

3.2 Kali Linux

Kali Linux[31] has emerged as a leading distribution in the field of cybersecurity. It has been specifically designed for penetration testing and security auditing. This Debian-based platform is known for its extensive suite of penetration testing utilities. These tools are carefully selected to cover all aspects of cybersecurity assessments, including network scanning, vulnerability discovery, wireless network evaluation, and password integrity testing. The distribution is continuously updated to keep the tools effective against evolving cybersecurity threats. Kali Linux's arsenal is a critical asset for anyone looking to conduct thorough security assessments.
In summary the keypoints of Kali Linux are:

- **Open-Source and Accessible**: Kali Linux is open-source and free for users. This accessibility ensures that the Testbed can leverage a state-of-the-art operating system without incurring licensing fees, thereby reducing operational costs and facilitating widespread adoption.

- **Pre-configured Tools**: Secondly, Kali Linux comes with pre-configured permissions, allowing immediate access to a wide range of tools without the need for additional permissions or configurations. This readiness significantly streamlines the setup process, enabling the Testbed to be operational with minimal setup time.
- **Comprehensive Security Toolkit**: Lastly, Kali Linux offers a collection of ∼600 tools which can easly be used in the Testbed. Two of the tools that are being used in this thesis are:
  - **Nmap** is a tool for discovering network topologies and auditing network security. It is widely used for mapping network environments and identifying potential vulnerabilities [26].
  - **Hydra** is a parallelized login cracker which supports numerous protocols to attack. It is fast and flexible, and new modules are easy to add [27].

Kali Linux's integration capabilities with a variety of tools make it a valuable resource for an IoT Testbed environment. Its collection of tools can be easily incorporated into the Testbed by writing a wrapper for them, providing a robust framework for conducting comprehensive security assessments of IoT devices.

3.3 Domain Name System

The Domain Name System (DNS) is like a telephone directory for the internet. It translates user-friendly domain names into numerical Internet Protocol (IP) addresses, which are used to locate and identify computer services and devices within network protocols. This service is crucial to the internet's usability, as it enables the use of memorable domain names instead of the complicated numerical IP addresses that computers use to communicate over the network.

3.3.1 Operational Mechanics of DNS

DNS resolution is a complex process initiated when a user enters a website address into a browser. The process is designed to locate the IP address corresponding to the website address. The operational flow of DNS involves several steps:

1. The browser begins by querying a DNS resolver, usually provided by the user's Internet Service Provider (ISP). The DNS resolver serves as the initial point of contact for the DNS lookup.
2. If the DNS resolver does not have the required IP address in its cache, it reaches out to other DNS servers across the internet to find the necessary information.
3. This inquiry navigates through a hierarchy of DNS servers, starting from the root name servers, then proceeding to the top-level domain (TLD) name servers (such

as .com, .net, .org), and finally culminating at the authoritative name servers for the queried domain.

4. After locating the IP address, it is relayed back to the browser, which can request the desired web page from the corresponding web server.

This process exemplifies the role of DNS in enabling easy website access through domain names instead of IP addresses.

3.3.2 DNS Records

DNS servers maintain a set of structured data known as DNS records, which are fundamental for the system's functionality. These records contain the following information:

- **A Records (Address Records)**: The core DNS records associate domain names with IPv4 addresses.
- **AAAA Records**: These records are similar to A Records but are used to map domain names to IPv6 addresses, which are necessary due to the internet's expansion.
- **CNAME Records (Canonical Name Records)**: These records enable the association of an alias name with a valid domain name, allowing for resource redirection.
- **MX Records (Mail Exchange Records)**: These records specify the mail exchange servers that are responsible for routing emails to a particular domain.
- **NS Records (Name Server Records)**: These records identify the authoritative name servers for a domain, which is a crucial element in the DNS lookup process.

3.3.3 Security within DNS

DNS is not immune to security threats. DNS spoofing and cache poisoning are some of the vulnerabilities that can put users at risk by redirecting them to malicious sites without their knowledge. To address these concerns, DNSSEC (DNS Security Extensions) has been developed to provide an extra layer of security and ensure the authenticity and integrity of DNS data.

As cyber threats continue to evolve, measures to secure DNS are also evolving. Since Technitium [30] is being used as the DNS server, it is possible to mitigate potential security threats by configuring records that redirect specific malicious addresses to non-reachable IP addresses. This helps to obstruct access to harmful domains.

3.4 Dynamic Host Configuration Protocol (DHCP)

The Dynamic Host Configuration Protocol (DHCP) is a protocol for managing IP networks. It automates assigning IP addresses, subnet masks, gateways, and other IP parameters to network devices, making network configuration more effortless.

3.4.1 Operational Mechanics of DHCP

The DHCP protocol uses a client-server architecture to simplify network configuration. When a device connects to a network, it sends a broadcast request to find a DHCP server. The server has a pool of IP addresses and relevant configuration settings and offers the client an IP address lease with necessary network settings for a specific duration.

The process of assigning an IP address to a client device using DHCP involves four steps:

- **DHCPDISCOVER**: The client broadcasts a request across the network to find available DHCP servers.
- **DHCPOFFER**: DHCP servers respond to the client's request by offering available IP addresses and network configuration parameters.
- **DHCPREQUEST**: The client selects one of the offers and formally requests the proposed configuration.
- **DHCPACK**: The server acknowledges the client's request and finalizes the network settings for the client's use by leasing the IP address to the client.

3.4.2 The Strategic Role of DHCP

DHCP is an important protocol that helps improve network management efficiency by eliminating manual IP address assignments. It is especially useful in dynamic network environments where devices often connect and disconnect and in large-scale networks where manual configuration can lead to errors and inefficiencies. DHCP can assign IP addresses, specific DNS servers, and gateway configurations, which is a valuable feature in IoT Testbeds. This allows for the redirection of all DNS requests and network traffic through the host machine, which enables traffic capture and analysis. Such a configuration is essential for monitoring network interactions of IoT devices, diagnosing issues, and ensuring robust network security measures are in place.

Ensuring that only one DHCP server is operational during the Testbed's use is essential. This is necessary because, during the DHCPDISCOVER phase, a pre-existing DHCP server within the network might issue a DHCPOFFER. Such an occurrence could result in devices maintaining communication capabilities with other gateways.Then, critical parameters such as the standard gateway or DNS server may not be accurately configured in alignment with the Testbed's requirements. This means that the Testbed will not be able to capture all the traffic it would otherwise.

3.5 Standard Gateways

A standard gateway acts as the bridge that allows data to flow between different network environments, such as a local area network (LAN) and the wider internet. This device is essential in directing traffic to and from workstations within the network, enabling smooth data exchange across various network segments.

3.5.1 IoT Testbeds as Standard Gateways

In Internet of Things (IoT) research and development, an IoT Testbed can be specially configured to act as the main gateway for various IoT devices connected within the Testbed environment. This unique configuration allows the Testbed to play a central role in managing and monitoring the data traffic flowing to and from the IoT devices. The benefits of using an IoT Testbed as a standard gateway are numerous, especially in terms of data analysis and security evaluation:

- **Comprehensive Data Capture and Analysis**: The Testbed has a crucial role as a gateway in the IoT ecosystem. It is uniquely positioned to capture, record, and analyze the data packets transmitted by IoT devices, enabling a deeper understanding of their behavior under different conditions and diverse test scenarios. The analysis can be conducted manually using Wireshark or automated tools like BruteShark, which can be used to analyze traffic for credentials systematically.
- **Advanced Security Assessment**: Furthermore, the Testbed's gateway position makes it an essential security management component in the IoT ecosystem. By analyzing the data traffic, the Testbed can detect and evaluate potential security vulnerabilities, such as unauthorized access or anomalous data patterns that could signify underlying security threats. This proactive approach to security analysis is critical for the early detection of vulnerabilities, allowing for timely remediation measures to be implemented and ensuring the integrity of the IoT infrastructure.

CHAPTER 4

IoT Framework Implementation

The IoT Testbed allows for comprehensive security testing of IoT devices, from initial connection to detailed vulnerability assessment and reporting. The following outlines the typical journey of a device as it undergoes scrutiny within this environment:

1. Connecting a device to the Testbeds network
2. Discover the device in the Testbed
3. Examine Device Detail and Customization
4. Vulnerability Testing
5. Log Review and Classification
6. Report generation

By looking at this lifecycle, we can see that the Testbed is structured into six main parts, each responsible for its own specific task. They have little to no coupling, so the modular design allows for an easy exchange or extensibility of these parts. The system to support this lifecycle can be seen in Figure 4.1, that shows the structure of the Testbed with all its components and what runs on them.

- **Network Connection**
  The detection step is the first of the lifecycle of a device in the Testbed. When looking at the Figure 4.1, those devices might be a Smart Bulb and a Raspberry Pi that got connected to the local network. This could be over WLAN or LAN via a wired connection to the router to which the Testbed is connected. Ensure the absence of other DHCP servers within the network to prevent conflicts and

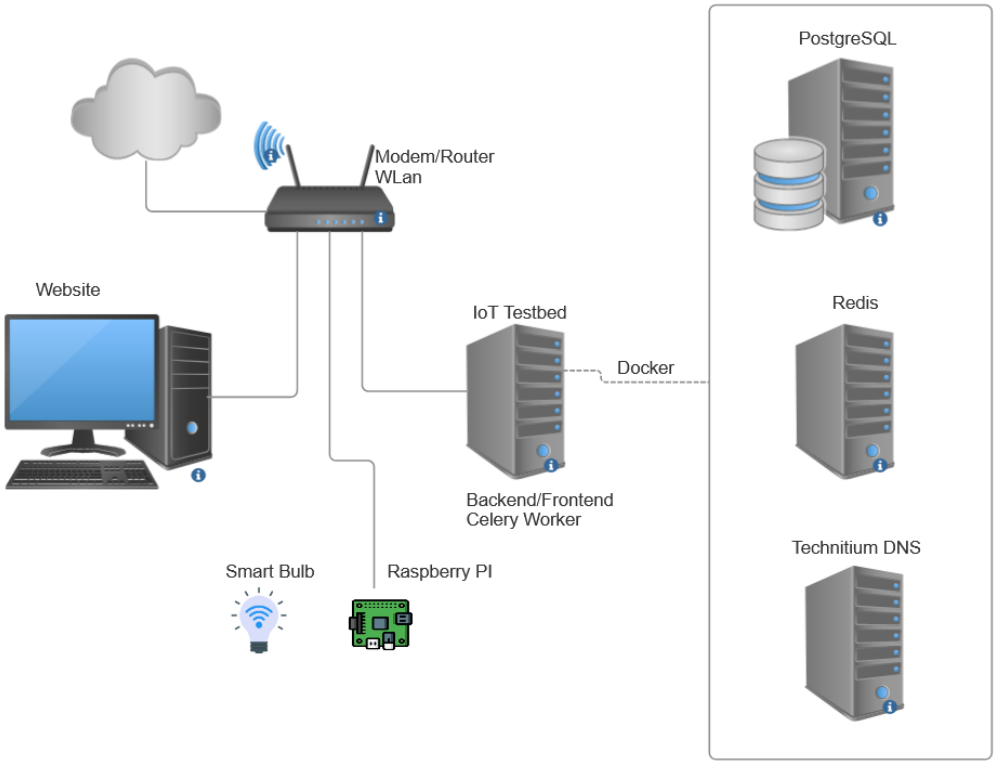

Figure 4.1: General project structure

ensure the device is correctly connected to the same network as the Testbed since the devices get polled from the DHCP server. It is important to note that modern devices often possess the capability to alter their MAC addresses, which can create challenges in consistently tracking a device's activity across multiple sessions. To mitigate this issue and maintain accurate device tracking, it is advised to turn off any such features that allow for MAC address randomization before connecting the device to the Testbed.

- **Device Discovery**
  These connected devices will then get an IP address assigned by the Technitium DNS server which also offers a DHCP service. By using the "Scan connected devices" button on the Testbed's homepage, devices can be imported. A notification will confirm the successful detection of a new device.

- **Show Device Detail and Customization**
  Access the newly discovered device's detail page to review its information. This step allows the user to rename the device for ease of identification throughout the testing process. This is the step where a first simple reconnaissance happens. Information that can be gathered in this stage is basic networking information such

as ip address, host-name and through its mac address a quick vendor lookup can be facilitated.

- **Vulnerability Testing**
  The next step is to run tools against the device. When doing so, the IoT Testbed creates a job with the job description and sends it to Redis, the message broker. The Celery worker then picks these jobs up and runs them. These tools might also be chained to pipelines, so a sequential execution of different tools is possible.
- **Log Review and Classification**
  After a tool or pipeline has been run, a success check must be performed to determine whether the tool found any vulnerabilities or whether the device is secure against a certain attack. This should be done automatically, but the user still has the possibility of changing the success status if a missmatch occurs.
- **Report generation** At the end of the test's, a report should be generated that gives an overview of what tests have been run and also shows if a vulnerability has been found, along with further information.

All this information about devices and tools will be persisted in the Postgresql server seen in Figure 4.1.

The following chapter elaborates on the implementation of the IoT Testbed, describing those six steps and explaining the methods and technologies employed. It starts with the foundation on which the Testbed is running, then describes how each part is implemented.

4.1 System and Tools selection

This section details the selection process for the underlying operating system and the programming language which were used to program the framework. Further the database system in use is described and why the choice fell on the chosen technology and concludes with the reasoning behind using Docker.

4.1.1 Operating System selection

The operating system selection involved a few different distributions: Kali Linux, Parrot OS, and BlackArch.

Each of those three options is being used in the penetration testing and hacking community, and each has its own toolset and capabilities.

- **Kali Linux**
  Kali Linux [31] is a Debian-based distribution that aims to use free software and thus comes with ∼600 tools for pen testing. As such, it builds on an existing, very strong foundation, Debian, and creates a layer on top of its foundation created specifically for hacking, concentrating on the improvement of such.

- **Parrot OS**
  Parrot OS [32] presents another possible option since it does things very similar to Kali and is also a Debian-based system. ParrotOS is further compatible with more systems, such as Kali Linux, and, as such, would allow a broader community to use it. It offers mostly all the standard tools that Kali offers and further comes with its own set of tools such as AnonSurf, Wifiphisher, and Airgeddon and, as such, accumulates a collection of ∼700 tools.

- **BlackArch**
  Blackarch [33], is an Arch-based distribution and comes with a collection of ∼2600 pre-installed tools, and with the Arch User Repository (AUR), offers a lot more user-contributed packages. Further, it needs fewer resources than Kali Linux and Parrot OS to run and is overall faster.

The decision, however, fell to Kali Linux as it is a robust operating system, offers many different tools that are needed, and if something goes wrong, troubleshooting is easier since the community behind it is bigger than with Parrot OS or BlackArch. Further, the developer's familiarity with Kali Linux is greater than that of the other operating systems.

Although the Testbed has been tested and developed with Kali Linux as its operating system, it should be usable with ParrotOS since they are very similar in terms of architecture and tools they offer. It should also run with BlackArch or other different operating systems.

Python and the Django web framework are the core components of this technological framework, and each brings its own distinct benefits that enhance the Testbed's development and operational efficiency.

4.1.2 Python

Python has been chosen as the primary programming language for developing the IoT Testbed for various reasons that make it a suitable choice for this project:

- **Comprehensive Library Support**: Python has a vast ecosystem of third-party modules and tools, including support for networking operations, cybersecurity tasks, and interaction with hardware devices. Python's libraries, such as Nmap and Metasploit, provide robust functionalities for network scanning and vulnerability exploitation, which are crucial for the Testbed's operations. Additionally, Python offers numerous tools and libraries relevant to IoT security and network analysis, making it well-suited for the project.

- **Developer Familiarity**: Python is a popular scripting language in the information security community, which the development team is proficient in, which makes it easier to work with the language and focus on implementing sophisticated features

and functionalities rather than navigating the learning curve associated with less familiar languages.

- **Versatility and Readability**: Python's syntax is designed for clarity and simplicity, making it an ideal choice for projects that prioritize rapid development and maintainability. Its versatility across different programming paradigms and platforms ensures that Python can adapt to the needs of the IoT Testbed project.

4.1.3 Django

Django is a Python web framework known for its clean and pragmatic design philosophy, enabling rapid development. It follows the model-view-template (MVT) architectural pattern, providing a scalable and maintainable component-based architecture [35]. The selection of Django was influenced by several key attributes, including:

- **Rapid Development Cycle**: Django aims to expedite the transition from concept to completion by providing developers with high-level abstractions, shortcuts for common patterns, and clear conventions. This aligns with the project's objective of quickly developing a fully functional IoT Testbed.
- **Comprehensive Database Support**: Django comes with a database API that simplifies database operations. The framework's database abstraction layer automatically generates database schemas from high-level Python data types, simplifying the process of integrating and managing the Testbed's data.
- **Adaptability to Project Requirements**: Django's modular architecture makes it flexible and adaptable to meet the specific requirements of the IoT Testbed project.

In summary, the implementation of the IoT Testbed leverages the strengths of Python and Django to create a robust, flexible, and efficient development environment. The combination of Python's extensive libraries and developer familiarity with Django's rapid development capabilities and database support provides a solid foundation for building and operating the IoT Testbed.

4.1.4 PostgreSQL

PostgreSQL, also known as Postgres, was chosen for the IoT Testbed project's relational database management system (RDBMS). Its open-source nature, extensive feature set, and adherence to technical standards make it a popular choice. PostgreSQL is designed to manage diverse workloads, from modest applications on single machines to complex operations across expansive data warehouses or web services catering to numerous concurrent users. It is widely deployed and available across major operating systems like Linux, FreeBSD, OpenBSD, and Windows, underscoring its versatility and robustness [36].

Integration with Django

PostgreSQL's integration with Django further justifies its selection as the primary datastore for the IoT Testbed project. The utilization of Django's Object-Relational Mapping (ORM) system in conjunction with PostgreSQL enables the creation of data models that are automatically translated into database schemas.

4.1.5 Docker

The Testbed is comprised of three containers - Technitium, Postgres DB, and Redis as can be seen in Figure 4.1. These containers are used for the configuration and deployment process, enabling efficient management and scalability of the DNS/DHCP services, data storage, and message brokering functionalities.

The IoT Testbed has adopted Docker containers for critical components, offering several advantages:

- **Isolation**: Containers provide a segregated environment for each service, minimizing the risk of conflicts between dependencies and streamlining the troubleshooting process.
- **Portability**: Docker ensures that each containerized service can be easily deployed across different environments, maintaining consistency in operation regardless of the underlying infrastructure.
- **Rapid Deployment**: The use of Docker containers significantly reduces the setup and deployment time for the Testbed, enabling rapid provisioning of services and iterative development cycles.

In summary, the integration of Docker into the IoT Testbed's architectural design enhances the system's modularity, scalability, and operational efficiency. By containerizing the Technitium DNS/DHCP service, the PostgreSQL database, and the Redis message broker, the Testbed achieves a high degree of service isolation, streamlined management, and robust performance.

4.2 Network Connection

After a device has been added to the Testbed via LAN or WLAN, the DNS logging happens on the Technitium DNS server; however, some devices might bypass the DNS logging, as mentioned in the evaluation chapter 6, so the Testbed functions as a gateway to capture more data about the communication.

Configuring the Standard Gateway in the IoT Testbed

The IoT Testbed serves as a standard gateway for connected IoT devices. To allow the gateway to function correctly, specific configurations within the underlying Kali Linux system are necessary to manage network traffic routing. These configurations are centralized within the `testbed/helper/routing.py` script, which encapsulates the essential operations for initializing and cleaning up network routing settings.

Critical Routing Methods

The script provides two critical methods that facilitate the manipulation of network traffic routing:

- **init_routing()**: This method is fundamental for enabling the Kali Linux machine to route traffic from connected IoT devices through to the internet or other network segments. Upon invocation, it executes critical system commands to modify network settings. Specifically, it runs the following commands:

  ```
  sysctl -w net.ipv4.ip_forward=1
  iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
  ```

  The first command enables IP forwarding, and the second configures NAT (Network Address Translation) via iptables. These changes allow the Testbed to function as a gateway.

- **cleanup_routing()**: This method disables IP forwarding and clears the NAT configuration established by iptables. It's used to revert the system to its default state or to clean up after a test session. Specifically, it runs the following commands:

  ```
  sysctl -w net.ipv4.ip_forward=0
  iptables -t nat -F
  ```

  This process ensures that the system's network settings are reset, preventing unintended routing or security implications outside of testing scenarios.

Adaptation to Different Systems

Although the provided commands are optimized for Debian/Kali Linux, which is the default operating system of the Testbed, it is important to note that adjustments may be necessary if the Testbed is deployed on a different system. The commands for enabling IP forwarding and configuring iptables may differ across various Unix/Linux distributions. Therefore, if the IoT Testbed is implemented on an alternate platform, these commands must be modified accordingly to ensure compatibility and maintain the Testbed's gateway functionality.

To simplify these adjustments, the IoT Testbed interface includes a settings tab, which provides an easy way to update the routing commands as well as the package manager that should be used to install tools and to meet the system's requirements. In essence, the `testbed/helper/routing.py` script and its methods, *init_routing()* and *cleanup_routing()*, are essential to the IoT Testbed's operation as a standard gateway. By managing the system's routing configurations, the Testbed can effectively channel IoT device traffic, supporting comprehensive network analysis and security assessments within a controlled testing environment.

4.3 Device Discovery and First Reconnaissance

The Testbed's design incorporates Technetium DNS, a dynamic DNS server known for intercepting and analyzing DNS requests. Additionally, Technitium serves as the DHCP server, facilitating the acquisition of DHCP leases from which connected devices and their MAC addresses can be identified. This capability enables preliminary reconnaissance of the device and its vendor by MAC address.

A specialized connector facilitates seamless interaction between the Testbed and the Technetium DNS server. Utilizing HTTP REST requests, this connector enables the querying of DNS requests made by connected IoT devices.

4.3.1 Technetium DNS in the Testbed

Technetium DNS [30] offers a versatile set of features that extend beyond the basic functions of a DNS server. Its capabilities include DNS request logging, traffic analysis, and providing detailed insights into the network behavior of connected devices. While the primary focus within this thesis is on capturing and analyzing DNS requests, the broader potential applications of Technetium DNS in network management and security analysis are noteworthy since it offers a lot more addons than the one that is being used in the scope of this project. The one addon that is currently being used with Technitium is *Query Logs (Sqlite)* which is responsible to store the DNS queries that are being generated by the devices. By doing so it allows us to retrieve DNS requests that are being created by a device to review it in a later moment and download them.

Both the DNS and DHCP functionalities are accessible through the implementation script located at `testbed/helper/dns/technitium.py`. Essential parameters for initializing these services are modifiable within this script and can also be adjusted through the Testbed's settings interface. A summary of the DNS and DHCP parameters, including their configuration and management, is provided in Table 4.1 and Table 4.2, respectively. The address space allocated for these services is initially set to accommodate 104 addresses, with the option for expansion through the Testbed's settings.

Parameter	Value
Name	technitium
Secret Token	
User	admin
Password	password
Host	http://127.0.0.1
Port	5380

Table 4.1: DNS Settings

Parameter	Value
Starting Address	192.168.0.150
Ending Address	192.168.0.254
Subnet Mask	255.255.255.0
Server Address	1.1.1.1
Router Address	192.168.0.1

Table 4.2: DHCP Settings

4.3.2 DNS Data Capture

As mentioned before the data captured for each request encompasses several critical components:

- **Client IP Address**: This identifies the device making the DNS request, enabling the tracking of requests back to the originating device and facilitating device-specific network behavior analysis.
- **Requested Domain Name**: The specific domain name requested by the device, offering insights into the external services and resources the device is attempting to access.
- **DNS Record Type**: The type of DNS record requested (e.g., A for IPv4 addresses, AAAA for IPv6 addresses), providing context on the nature of the request and the information being sought.
- **DNS Response**: Includes resolved IP addresses as part of the response, delivering detailed information on how domain name requests are resolved and the ultimate destination of outbound device traffic.
- **Protocol Utilized**: The protocol (UDP, TCP) used for the request offers insights into the communication methods preferred or required by the device.
- **Response Type**: Indicates the nature of the DNS response—whether the request was served from cache, blocked, or resolved through recursive queries—shedding light on the efficiency and security aspects of DNS resolutions.

Access to the data is made possible through the Technitium web user interface, which can be accessed via the URL: http://<ip_of_testbed>:5380/. This interface provides a user-friendly portal for direct interaction with the DNS data. Additionally, the IoT Testbed simplifies the process of retrieving DNS logs through a download feature, making it easier to analyze the data.

4.4 Vulnerability Testing

The IoT Testbed can accommodate a wide range of tools, making it a versatile platform for conducting various security analyses on IoT devices. There are no strict criteria for including tools; any utility processing input and producing output can be seamlessly integrated as an add-on Python file. The process of creating and integrating these add-on tools will be discussed in detail in this subsection 5.2.1.

4.4.1 Tools

The IoT Testbed incorporates several tools to showcase its capabilities and provide a foundation of functionalities. These tools exemplify the kind of add-ons that can enhance the Testbed's utility. The following is a list of tools that have already been incorporated:

- **Nmap** [26]: A network exploration tool and security scanner that can identify open ports on devices, which can potentially uncover entry points for exploitation.
- **Hydra** [27]: A tool that facilitates testing of credential strength across services by conducting rapid dictionary attacks against various protocols. Within the stock configuration of the IoT Testbed, Hydra is configured to target:
  - SSH
  - Postgres
- **tcpdump** [28]: A tool instrumental in capturing network traffic traversing a specified interface. Devices within the Testbed network are configured to route traffic through the Testbed as their standard gateway. Tcpdump can effectively log outgoing traffic, although it may not capture all incoming data.
- **Bruteshark** [29]: This tool analyzes network dumps to extract credentials and other sensitive information acting on data captured by tcpdump. It offers a layer of analysis that complements the data acquisition capabilities of tcpdump.

4.4.2 Pipelines

In the IoT Testbed, pipelines automate the security testing process by orchestrating a sequence of tool executions. This automation helps streamline the penetration testing workflow, making security assessments more efficient. An example pipeline, preconfigured in the Testbed, illustrates the concept:

1. The pipeline starts with Nmap, which identifies open ports on target devices.

2. Based on Nmap's findings, the pipeline triggers parallel attacks:

   - A PostgreSQL-focused Hydra attack.
   - An SSH-targeted Hydra attack.

This pipeline exemplifies the Testbed's ability to automate sequences of security tests, showcasing its potential to conduct comprehensive assessments with minimal manual intervention. Further discussions on custom pipeline development will be provided in section 5.3.

4.4.3 Optimizing Task Execution with a Dedicated Worker Mechanism

Those tools and programs that can be run either as a standalone tool or as a link of a chain in a pipeline will be passed on as a job to the Redis message queue and will be picked up by the Celery worker. The worker mechanism uses Celery, an asynchronous task, and job queues based on distributed message passing on Redis. This allows the system to efficiently manage and execute tasks outside of the Testbed's primary backend environment [25].

Integration of Celery with Redis

Celery is used as the backbone for task execution within the IoT Testbed, and Redis acts as the message broker. This setup creates a communication channel between the system's backend and the Celery worker. It helps to dispatch and manage tasks, particularly those that involve tool execution for device analysis.

Execution and Permission Management

To initiate the operation of the Celery worker, the following command is used:

```
sudo celery --app "testbed" worker --loglevel=info -P prefork
```

Launching the Celery worker with **sudo** permissions is essential to execute specific tasks requiring administrative/elevated network rights, such as network traffic capture using *tcpdump*. Running the worker with elevated privileges ensures that it can perform operations that require direct interaction with system hardware or privileged system resources.

Benefits of the Worker Mechanism

The implementation of a dedicated worker mechanism through Celery and Redis offers several advantages to the system:

- **Separation of Concerns**: By separating task execution from the core backend logic, the system becomes more clear and maintainable. This separation ensures that the backend remains responsive and stable, even under heavy computational loads.
- **Enhanced Scalability**: The worker mechanism allows for the dynamic scaling of task execution capabilities based on the current load. This facilitates efficient resource utilization and minimizes response times for task completion.
- **Security and Stability**: Running tasks with the necessary permissions ensures that all operations, even those requiring elevated rights, are executed securely and reliably. This approach mitigates potential security risks associated with privilege escalation within the main application.

After a tool has been run, an entry will be created in the framework where the user can check the success of the previously ran tools.

4.5 Log Review and Classification

To classify the success of tools in an automatic manner for a faster reporting system or use those results in a pipeline of chained tools, it is essential to find a way to do this in a dynamic way since not only the return code, for example, should be used but more over the whole output content of a tool. For this, the IoT Testbed uses the tool Python file, which should contain a function to check for the success of a tool automatically. More on this in the evaluation chapter 6.

The *check_success* function in the Python file which is located in the addons folder structure will be called by the framework to check for the success of a tool run. The Hydra tool is a simple example which uses a short regex to check if the tool was able to extract credentials as seen in Listing 1

Since the return code is not enough this function receives the stdout, stderr and maybe even extra content besides the return code of the tool run depending on how the Python wrapper file of the tool is programmed.

4.6 Report generation

After conducting several tests on a device in the IoT Testbed, a detailed report summarizing the findings can be generated. This report allows the identification and addressing of any potential vulnerabilities effectively.

```
def check_success(return_code, output_content_stdout, output_content_stderr,
↪  output_content_extra):
  # Regular expression to match lines indicating that valid passwords were
  ↪  found
  valid_password_pattern = re.compile(r'\d+ valid password(s)? found')

  return bool(valid_password_pattern.search(output_content_stdout))
```

Listing 1: Check success of tool

4.6.1 Generating a Device Report

A device report can be generated with the following steps:

1. When starting from the landing page of the testbed a device for which the report should be generated has to be selected by clicking onto it. Now a detail page about the device is being shown.
2. Now the *Download Report* button has to be clicked to start the report generation process. Now a different page is shown where all previously performed test that have been linked to the device or did not have any device selected at all will be shown in this page.
3. The tool executions have to be selected that should be included in the report. By doing so, either all tests can be collected in one big report or different smaller PDFs can be created to give the report on specific security sections of a device.
4. When all tests have been selected that should be included in the report the next step is to click on *Generate Report*. The IoT Testbed will then compile the chosen test results into a structured PDF document. An example of such a report can be found in chapter 6.

The generated report consists of several key sections that provide insights into the tested device and the outcomes of the various security assessments:

- Title Page: Provides an overview of the device, including essential identification information.
- Index Page: Lists the included tests and serves as a navigational aid for reviewing the report's contents.
- Detailed Test Reports: Presents in-depth findings from the selected tests. The level of detail and presentation style for each test is guided by the configuration of the corresponding tool within the Testbed, particularly concerning the handling of test outcomes.

4.6.2 Handling of Test Outcomes

The report's treatment of individual test results is based on the tool's configuration specified by some variables that can be found in the the wrapper file for the tool, notably the *SHOW_OUTPUT_ON_FAILURE* setting:

- SHOW_OUTPUT_ON_FAILURE = True
  - if addon offers *<addon_name>.addon_failure.html* as a template, then this will be displayed in the report.
  - else shows the normal output template for this addon
- SHOW_OUTPUT_ON_FAILURE = False
  - omits this addon from the detailed log page

This allows for a dynamic way to include each tool differently in the report PDF, depending on the success of the run.

CHAPTER 5

Extending the Framework

This chapter shows how to add more functionalities to the Testbed in the form of new tools that can be either pre-existing or newly developed. Further it will dive into chained tools in the form of pipelines, how to create, export or import them. As well as other parts that can be changed or exchanged and how this process would look like.

5.1 Incorporating Alternative DNS/DHCP Services

While the IoT Testbed is configured to utilize Technitium as its default DNS/DHCP service provider, due to its open-source nature [30], the flexibility to integrate alternative DNS/DHCP services exists. This adaptability, however, comes with considerations, primarily because some specific functions have to be written into a helper file to ensure a functioning Testbed.

5.1.1 Implementing a New DNS Helper

To implement a different DNS/DHCP service, a new DNS helper file within the designated directory has to be created: testbed/helper/dns/<dns_helper_file>.py. This file must include the following essential methods tailored to interact with the new service:

- The function **get_default_settings()** is designed to provide a structured representation of the default configuration parameters for two critical network services: DNS (Domain Name System) and DHCP (Dynamic Host Configuration Protocol). Upon invocation, this function returns a list containing two dictionaries, each corresponding to the configuration settings for DNS and DHCP services, respectively. This configuration is necessary for all further functions.
- The **login(settings: Dict[str, str]) -> None** function is responsible for authenticating the user to the DNS server via the Technitium API using specified

credentials in the get_default_settings() and subsequently updating the provided settings dictionary with a secret token acquired during the authentication process. This function embodies a critical component for communication with the DNS server and is needed for subsequent operations requiring authentication, such as modifications to DNS records, querying DNS logs, and in Technitium's case, setting the DHCP scope or deleting existing scopes. Since the IoT Testbed is implemented to use Technitium as the choice of DNS and DHCP server and is offering an API to talk to it, the new DNS server which should replace Technitium should also offer API endpoints to request the needed information.

- The function **list_all_dhcp_scopes(settings: Dict[str, str]) -> List[Dict]** is designed to retrieve a comprehensive list of Dynamic Host Configuration Protocol (DHCP) scopes from a designated external API, leveraging a set of predefined settings for authentication. This function illustrates a pragmatic application of interfacing with network configuration APIs, facilitating the dynamic acquisition of network scope data crucial for network management and configuration tasks.

- The function **setup_dns(settings: Dict[str, str]) -> None** plays an important role in configuring the Domain Name System to ensure its operational readiness. In the current state, this means receiving a valid authentication token and then using the token to install the log apps of Technitium DNS, which is crucial for enabling the persistent storage of DNS data.

- The last needed function is **setup_dhcp(dns_settings: Dict[str, str], dhcp_settings: Dict[str, str]) -> None** and it is responsible for setting up the DHCP. Currently, this function checks for a valid token, removes all existing DHCP scopes, and creates a new scope that is needed to ensure the network is working as intended.

To ensure seamless integration and functional parity with Technitium's capabilities, when implementing DNS/DHCP methods, it is important to ensure that they comply with the selected service's operational requirements and API specifications.

5.1.2 Integration Process

The integration process of an alternative DNS/DHCP service into the IoT Testbed is facilitated through dynamic module importation and verification. This process is demonstrated in Listing 2.

This code dynamically identifies and imports the newly created DNS helper module based on the Testbed configuration. The *check_dns_helper_ok* function verifies the compatibility and readiness of the helper module, ensuring that all necessary methods are implemented correctly and operational.

```
dns_settings['name'] = get_dns_helper_file(
os.path.join("testbed", "helper", "dns"))
module_name = "testbed.helper.dns." + str(dns_settings['name'])
dns_helper = importlib.import_module(module_name)
okay, mess = check_dns_helper_ok(dns_helper)
```

Listing 2: Load DNS Helper

Considerations and Recommendations

When integrating an alternative DNS/DHCP service, ensuring that the new service aligns with the Testbed's functionality is important. Although transitioning to a different DNS/DHCP service within the IoT Testbed is feasible, it requires careful planning, implementation, and validation. Some points that need extra checking might be:

- It is important that the new DNS server supports API requests so
  - the Testbed can log in to the server
  - the DNS queries can be requested to a specific IP
  - the DHCP leases can be queried as well as the DHCP scopes.
- If DNS and DHCP get offered by a different server, the DHCP part needs a new wrapper.
- It is essential that the new DNS server persists previously made requests so the testbed can request them over the API.

5.2 Extending the IoT Testbed with Custom Tools

The following section will give an overview of what is needed to create a new tool and expand the tool set of the IoT Testbed. First an implementation overview is given so that new tool wrappers can be developed.

5.2.1 Implementing a New Tool

To ensure compatibility and operational integrity, a structured process is in place for adding new tools to the IoT Testbed ecosystem. Here is a detailed guideline for integrating a new tool, using Hydra as an example:

Tool Integration Structure

To integrate a new tool, such as Hydra, into the IoT Testbed, the directory structure seen in Listing 3 should be established within the addons folder.

```
hydra
|-- hydra.py  <- The Addon file, has to have the same name as addon folder
|-- templates
|   |-- hydra_failure.html
|   |-- hydra_stderr.html
|   |-- hydra_stdout.html
|   `-- hydra_dashboard.html
```

Listing 3: Addon folder structure

This structure separates the tool's executable logic (*hydra.py*) from its output presentation layer (the templates directory), promoting modularity and ease of maintenance.

Essential Components of a Tool Addon

A tool addon file, such as *hydra.py*, must define specific global variables and implement key functions to ensure seamless integration and functionality within the Testbed. These include:

- **ADDON_NAME**: Identifies the addon, typically set to the tool's name or its executable function within the system.
- **SHOW_OUTPUT_ON_FAILURE**

  This variable determines the behavior of a tool on failure. Setting it to *False* means that the tool will not generate a report in the PDF if it fails or does not find any vulnerabilities. Setting it to *True* allows the addon to add its output to the report, if a template is present that handles the failure output. Otherwise, the next variable tells the system which output to use for true negative cases. This is described in more detail in subsection 4.6.2.
- **FAILURE_CHANNEL**

  This variable tells the IoT Testbed which channel to use on failure since some tools might deliver the information on stdout, some on stderr, and some on a specified output path which in our case is called extra. This variable can be one of the following:

  - stdout: normal stdout channel
  - stderr: normal stderr channel
  - extr: extra channel that can be specified in the tool Python file

Key Functions for Tool Operation

The following functions describe the interface for integrating the operation of the IoT Testbed addon and its integration within the Testbed:

- **get_file_ext(filetype)** This Python function in Listing 4 accepts a single

```python
get_file_ext(filetype: str) -> str:
extensions = {"stdout":"txt", "stderr":"txt", "extra":"txt"}
return extensions[filetype]
```

Listing 4: Retrieve file extensions per addon

  argument, *filetype*, which specifies the type of output file (e.g., *stdout*, *stderr*, or *extra*). It then returns the corresponding file extension as a string, based on a predefined dictionary mapping each output file type to its appropriate file extension.

  By default, all file types are assigned a *.txt* extension, which is suitable for textual data. However, this function can be easily adapted to accommodate specific file formats unique to certain tools. For instance, the *tcpdump* tool generates output files in the *.pcap* format, and this function can be tailored to return a *.pcap* extension for output files generated by *tcpdump*, enhancing the usability and compatibility of the captured data.

- **get_options()**

  The *get_options()* function is used for customizing and configuring tools within the IoT Testbed, providing a structured approach to specifying the default operational parameters of a tool. It outlines the flags, arguments, and positional arguments necessary for invoking the tool with its desired functionality. The significance of this function lies in its ability to dynamically tailor tool execution to specific testing scenarios, thereby enhancing the Testbed's adaptability and efficiency.

  The implementation of *get_options()* is designed to return a dictionary that encapsulates the default execution parameters for the tool. The options for the Hydra tool are shown in Listing 5, which is divided into three sections.

  To show an example, these three parts from the options are explained with the example of Hydra[27] which can be seen in the following list.

  – **Flags**: An array that may contain default flags to be used during tool execution. This section allows for simple binary options that modify the tool's behavior without requiring additional data.

  – **Arguments**: A dictionary mapping argument names to their values. This structure supports more complex configurations, such as file paths or numerical values, that influence the tool's operation. In this example, three arguments

```
def get_options() -> Dict[str, Any]:
options = """{
  "flags":[""],
  "arguments":{
    "L":"path_to_file(default=<addons/.../wordlist.txt>)",
    "P":"path_to_file(default=<addons/.../wordlist.txt>)",
    "t":"4"
  },
  "pos_arguments":{
    "protocoltarget":
    "protocol_to_attack(default=<ssh>)://ip_of_device"}
}"""
return options
```

Listing 5: Retrieve options per addon

are present. *L* and *P* are a username and a password file.
This follows a certain structure if there is the need to provide the user with an interface to change this argument. If it is a fixed path, it is better to just specify the path; however, in this example, it would be nice for the user to upload a specific user/password file that fits the password specification for a specific device, for example. The *path_to_file* tells the IoT Testbed that a user can choose a path here, and this should be displayed as a file input on the website. If the user does not specify a file that needs to be used, the default value is being used specified by *default=*.

– **Positional Arguments**: These are attached to the end of the command and offer the same variability that we saw in Arguments. In this example, we have two keywords. One is *protocol_to_attack* which will be a text input on the website, with the default value of *ssh* and the *ip_of_device* which gets replaced with the actual ip of the device that is registered in the IoT Testbed.

In summary, the *get_options()* function exemplifies the modular and configurable nature of the IoT Testbed, enabling the seamless integration of tools with diverse operational requirements. Through this function, the Testbed allows users to tailor tool execution to the specific needs of their security assessments, thereby maximizing the effectiveness and precision of IoT device evaluations.

Moreover, the structured return value of *get_options()* facilitates the automatic generation of user interfaces for tool configuration on the Testbed's web platform. This means that the options defined within the function can be utilized to build a user interface, making tool configuration more user-friendly.

For the hydra example, the run page might look like Figure 5.1:

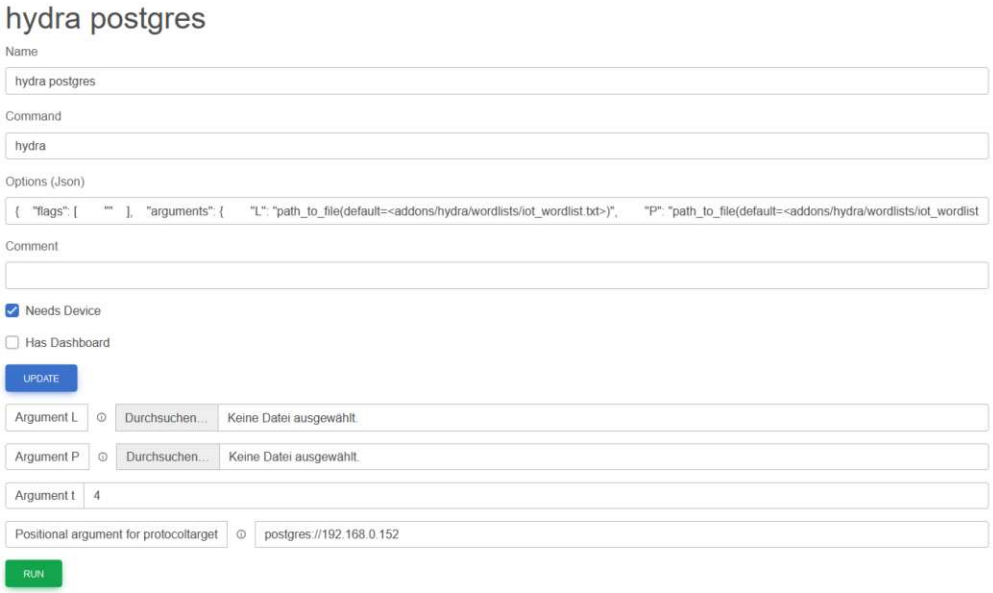

Figure 5.1: Shows the run page for hydra on a device

- **install_addon(package_manager)**

  The **install_addon** function is invoked when adding a new tool to the IoT Testbed. Its primary objective is to automate the tool's installation process, leveraging the system's package manager. This automation significantly simplifies the setup process for new tools, ensuring that they are ready for use without manual intervention. Listing 6 gives a closer look at how the function might be implemented.

```
1 def install_addon(package_manager: str) -> Tuple[int, str]:
2 ...
3 cmd = [package_manager, "install", "-y", ADDON_NAME]
4 subprocess.run(cmd, check=True)
5 ...
```

Listing 6: Shows the installation process

It is necessary for the *install_addon* function to be defined within the tool's addon file, regardless of whether the tool requires additional software to be installed or not. This ensures consistency in the addon structure and facilitates potential future installations.

The *install_addon* function is expected to return a tuple containing an integer and a string. The integer represents the installation process's success status (usually 0

for success), and the string provides a message or feedback about the installation outcome: (0, ‘<some_message>‘).

- **run(device_tool_info_key, device_tool_info, command, stdout_file, stderr_file, extra_file)**

  The *run* function is central to tool operation within the IoT Testbed framework. It facilitates the execution of diverse tools and the collection of their outputs. It is invoked with several critical parameters, each serving a specific purpose in the tool execution process:

  – **device_tool_info_key**
  The tool uses a unique identifier to store new findings or results related to the device under test. For instance, a tool like Hydra, which specializes in password attacks, might use this key to record vulnerable credentials.

  – **device_tool_info**
  This dictionary aggregates the outcomes from various tools, indexed by their respective keys. It serves as a centralized repository for data gathered during the Testbed's analysis, facilitating cross-tool data utilization and enhancing the depth of security assessments. This is especially useful for pipelines.

  – **command**
  The pre-constructed command that triggers the tool's execution is dynamically generated, often reflecting user-defined parameters or configurations specified through the Testbed's interface.

  – **extra_file**, **stderr_file**, **stdout_file**
  These parameters denote the file paths where the standard output, standard error, and any additional output from the tool's execution are to be stored. This structured approach to output capture ensures comprehensive logging and subsequent analysis.

  The implementation of the ‘run‘ function typically involves executing the specified command using system-level functions (e.g., ‘subprocess.Popen‘ in Python) and redirecting the outputs to the designated files. This process captures the tool's direct outputs and allows for the monitoring of execution status and potential errors.

  Tools that augment the *device_tool_info* dictionary with new data must ensure that this information is accurately returned at the function's conclusion. This return mechanism enables the persistent storage of tool outcomes, making the data accessible in a pipeline for further analysis, reporting, or use by subsequent tools. The run function might look like Listing 7.

- **get_stdout(stdout)** This function processes standard output (stdout) data from tool execution. It formats the raw output into a structured form suitable for display or further analysis. The function takes the stdout data as input and returns a data structure (e.g., a dictionary or a list) representing the formatted output.

```python
def run(device_tool_info_key: str, device_tool_info: Dict[str, Any],
command: list, stdout_file: str, stderr_file: str,
extra_file: str = None) -> Tuple[int, Dict[str, Any]]:
  with open(stdout_file, 'w') as stdout_f, open(stderr_file, 'w') as
  ↪  stderr_f:
    process = subprocess.Popen(command, stdout=stdout_f, stderr=stderr_f,
    ↪  text=True)
    process.communicate()
    returncode = process.returncode

  with open(stdout_file, "r") as f:
    device_tool_info[device_tool_info_key.replace("
    ↪  ","_")]=get_ports_from_stdout(f.read())
    return returncode, device_tool_info
```

Listing 7: Run an addon/tool

- **get_stderr(stderr)** Similar to *get_stdout*, this function processes standard error (stderr) output from tool execution. It formats error messages or logs for easy interpretation and display, ensuring that potential issues or important warnings are not overlooked.
- **get_stdextra(extra)** This function provides a mechanism to process and format additional types of output beyond standard output and error for tools that produce them. This could include binary data, network captures, or any other form of output that doesn't fit the standard stdout or stderr streams.
- **get_failure()** This function is specifically designed to handle and format the output in cases where the tool execution fails or when specific vulnerabilities are not found. It ensures that failures are reported in a clear, actionable manner, aiding in debugging and further analysis.
- **get_html()**
  This function uses the data provided by the functions mentioned in the previous four list points and renders an HTML page with the help of a Jinja template.

The development of Jinja templates is a critical step for visualization and analysis of tool outputs within the IoT Testbed. These templates serve as the bridge between the raw data produced by security tools and the structured, readable reports accessible to users. Understanding the necessity and structure of these templates is essential for effectively integrating new tools into the Testbed.

The creation of Jinja templates follows a specific structure, ensuring that the tool's outputs are appropriately captured and presented:

- <tool_name>_stdout.html
  This template is needed for incorporating the standard output (stdout) of a tool's execution into the Testbed's user interface and downloadable reports. Its presence is mandatory because stdout often contains the primary results or data produced by the tool.

- <tool_name>_stderr.html
  This template is designed to render the standard error (stderr) output, which may contain error messages, warnings, or additional information not captured in stdout.

- <tool_name>_failure.html
  This optional template is used to display tool outputs when the execution fails or does not achieve its expected objectives. If the tool is set up with *SHOW_OUTPUT_ON_FAILURE* set to True and this template exists, it will be utilized to present failure-related outputs. Otherwise, the Testbed defaults to using the stderr content.

- <tool_name>_dashboard.html
  This template is required when a tool is marked with the *Has Dashboard* flag. It enables the creation of a dedicated dashboard page for the tool within the IoT Testbed.

The customization potential offered by Jinja templates allows both tool developers and Testbed administrators to tailor the presentation of tool outputs to meet specific analytical needs or user preferences. This flexibility enhances the user experience, making the analysis of complex tool outputs more accessible.

5.2.2 Incorporating a Tool into the IoT Testbed

After writing a new tool wrapper or downloading one, you can add it to the IoT Testbed by performing one additional step. Before proceeding to the remaining steps on the website, ensure that the tool wrapper and jinja files are named correctly and in their corresponding places.

To add a new tool, go to the IoT Testbed web interface and navigate to *Tools -> Add new Tool*. You will see a form similar to the one shown in Figure 5.2.

- Select the addon name from the list. If the tool is not listed, it indicates a problem with the file or directory structure. Meaning that the names of the folder and/or files does not correspond to the guidelines.
- Specify the command.
- Verify the options, which should be accurately predefined in the addon file.
- Input an optional commentary.

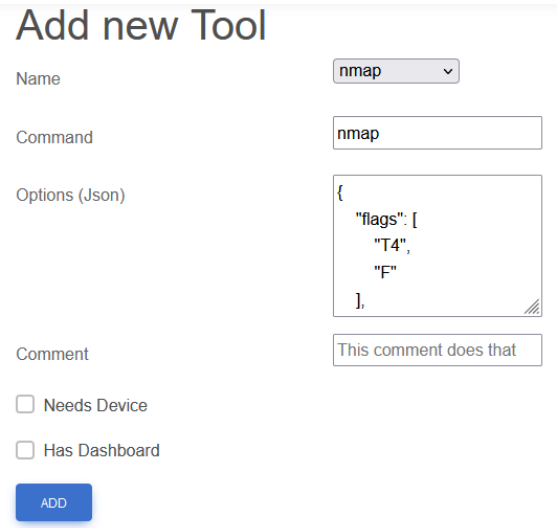

Figure 5.2: Shows the form to add a new tool

- Check the "Needs Device" box if the tool requires a device for operation. Without an associated device, the tool will not execute.
- Check "Has Dashboard" if the tool includes a *<tool_name>_dashboard.html* file, which can be added to the IoT Testbed's navigation bar.

Once these steps have been completed, click on *ADD* to register the tool on the Tools index page. The tool is then available for penetration testing on devices or for inclusion in a pipeline.

5.3 Extending the IoT Testbed with pipelines

Pipelines are a feature of the IoT Testbed that allows you to automate and streamline the execution of multiple security tools against IoT devices. By chaining together a series of tools, users can conduct security assessments efficiently without needing manual intervention. This feature is particularly helpful for time-consuming tasks, allowing you to initiate a sequence of operations and revisit the results later.

To create a pipeline, the following steps are necessary:

1. On the IoT Testbed website, clicking on *Pipelines* in the navigation bar brings us to the needed form.
2. By clicking on *Create new Pipeline* the process gets started.
3. The appearing form has to be filled out:

- Add a descriptive name for the pipeline.
- Add a brief comment that describes the pipeline.
- Add a Python file containing all the necessary functions for the pipeline.

4. By clicking on *Create Pipeline* a pipeline entry will be created in the Testbed.
5. On the pipeline index view, select the just created pipeline and steps can be added by clicking on *Add new step*.
    - Assign a step number. Step numbers have to be consecutive! If a pipeline contains steps that do not form a consecutive order, then the system will try to create this consecutive order.
    - The tool that is needed for this step has to be selected.
    - Now in the *Function* input field, a function name has to be added, for example *check_port*. This function, however, has to exist in the function file, which was uploaded earlier. This function has to return true, otherwise this step will not run. If Hydra wants to run an attack on a specific port, a preliminary check could already be done here to see if the port is even open. An example of how this could look like is as follows:
    - Finalize the process by clicking on *Add step*

The **pipeline file** dictates the pipeline's flow through conditional checks performed by its functions. These functions receive three key pieces of information:

- device: The target device of the pipeline execution.
- step: The specific step within the pipeline being executed.
- pipeline_log: An object allowing for the reading from and writing to the pipeline's log, facilitating data exchange and condition checks across steps.

Functions within this file must return either *True* or *False*, determining whether subsequent steps should execute. For instance, a function like *check_ssh* as seen in Listing 8, examines the device's tool output for open SSH ports, enabling subsequent SSH-related operations only if the condition is met.

Figure 5.3 depicts the pipeline in question and its first step is to start with Nmap to retrieve open ports. This step is required for mapping the device's network interface and identifying potential entry points for penetration. Once the Nmap scan is completed, the pipeline is designed to execute Hydra SSH and Hydra Postgres in parallel, depending on the results of preliminary checks - *check_ssh* and *check_postgres*, respectively. These functions evaluate the feasibility of the attacks based on the Nmap results, ensuring that the subsequent steps are both relevant and likely to yield actionable insights. The

```python
def check_ssh(device, step, pipeline_log):
  data = json.loads(device.tool_info)

  for result in data.get("nmap", []):
    if result.get("service") == "ssh" and result.get("state") == "open":
      return True, ""
    return False, "Service is not available or not open!"
```

Listing 8: Checks if the SSH service is reachable

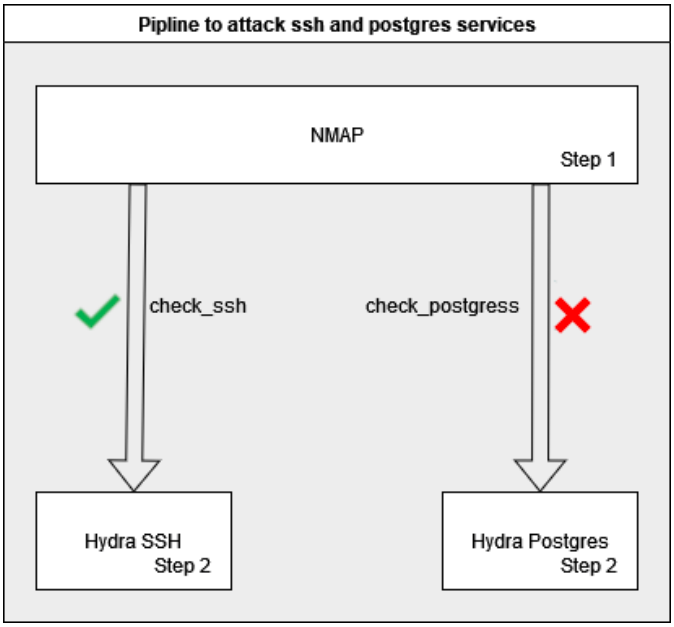

Figure 5.3: Shows an example pipeline with nmap and hydra

Hydra SSH and Hydra Postgres steps are executed independently of each other since they address different attack vectors (SSH and PostgreSQL services).

In this particular case, the *check_ssh* function validates the presence and accessibility of an SSH service, returning True. Thus, the Hydra SSH step is activated, targeting the device's SSH interface for brute-forcing passwords. Conversely, *check_postgres* returns False, indicating the absence of an open PostgreSQL service or the service's inaccessibility. This result prevents the execution of Hydra Postgres, optimizing the pipeline's efficiency by bypassing irrelevant or unfeasible tasks.

5.3.1 Import/Export a pipeline

The IoT Testbed emphasizes the robust execution of security assessments and facilitates collaborative cybersecurity research. The collaborative aspect is supported by the ability to import and export pipelines, enabling the distribution and replication of successful security assessment workflows. This user-friendly feature allows users to share custom-built pipelines and contribute to a collective resource pool that benefits the broader cybersecurity community.

Exporting a pipeline is a process that packages the pipeline's configuration and associated functions into a portable format. To export a pipeline the following steps are necessary:

1. Access the list of available pipelines in the IoT Testbed.
2. Identify and select the pipeline you wish to export. This action will give you a detailed view of the selected pipeline.
3. Look for and select the *export pipeline* option. This will initiate the export process, compiling all relevant pipeline information into a single JSON file. This file includes:
   - The pipeline's configuration details that are stored within the Testbed's database.
   - The Python function file associated with the pipeline, encoded in base64 within the JSON file. This encoding ensures that the Python file is encapsulated in a web-safe format that can be easily decoded upon import.

Importing a pipeline allows users to integrate externally sourced pipelines into their Testbed environment. To add a pipeline the following steps have to be followed:

1. Return to the list of all pipelines.
2. Use the interface to select a JSON file that contains the exported pipeline data. This file should encapsulate all necessary configurations and the base64-encoded Python file.
3. Initiate the import process by confirming the file selection. The Testbed will parse the JSON file, reconstruct the pipeline's database entries, and decode the Python file to restore the original functionality.

It's important to note that the import process checks for the existence of the required tools within the Testbed. If any tools referenced by the pipeline are missing, users need to add these tools to ensure the pipeline functions as intended.

CHAPTER 6

Evaluation

This chapter describes the evaluation process and the results obtained from a series of tests conducted on 14 IoT devices, including a Raspberry Pi with given vulnerabilities. A complete list of the devices that are being used can be found in Table 6.1. All devices in this list were connected via WLAN to the router. Except for the Raspberry, which was connected via a RJ45 cable to the router and was as such, besides the Testbed, the only device that was connect with a cable.
Before delving into the specifics of the test devices, lets revisit our general structure Figure 4.1 and the specifications of our test environment will be shown now. We used a *CH7465MT-AT* model as the router/modem with *CH7465LG-NCIP-6.15.32p2TM-GA-NOSH* as the software version. The Testbed with its container then runs on a mini-pc which is connected to the router (IP:192.168.0.1), its specifications can be seen in the following list.

- **Model**: HP EliteDesk 800 G3 DM
- **Processor**: i5-6500
- **RAM**: 16 GB
- **Storage**: 512 GB SSD
- **OS**: Kali Linux 2023.3
- **Python**: v3.10
- **IP**: 192.168.0.2

All test devices, the Testbed and the PC to access the website have to be in the same network and in our case they all belonged to this network: *192.168.0.0/24* where the router has the first address: *192.168.0.1.* Then follows the Testbed: *192.168.0.2* and all

Device ID	Device Type	Vendor
IG301S Smart Garden	Planting station	Tuya Smart Inc.
Pawaboo du7l-w	Smart Cat Feeder	Tuya Smart Inc.
LF-C1t	Smart Cam	Tuya Smart Inc.
Bakibo smart bulb	Smart Bulb	Tuya Smart Inc.
Hutakuze smart bulb	Smart Bulb	Tuya Smart Inc.
Fitop smart bulb	Smart Bulb	Tuya Smart Inc.
FireTV Stick	FireTV	Amazon Technologies Inc.
Echo Show	Amazon Echo	Amazon Technologies Inc.
Echo Dot 2 Gen	Amazon Echo	Amazon Technologies Inc.
9C7613D0FFC4-Ring Bell, 5at1s2	Ring Bell	Ring LLC
Hisense H49MEC3050	TV	CyberTan Technology Inc.
Yeedi vac hybrid	Smart Vacuum	FN-Link Technology Limited
Steamlink-1003	Steamlink	Valve Corporation
Raspberry Pi 2	Raspberry Pi	Raspberry Pi Foundation

Table 6.1: Device test set

new additional devices receive their address by DHCP (Important: It is crucial that no other DHCP-Server is active in this network) which also gives them the Testbed as their standard gateway as well as the DNS server which again is the Testbed.

To summarize the network settings for the Raspberry Pi looked like this:

- **IPv4**: 192.168.0.152
- **Netmask**: 255.255.255.0
- **DNS Server**: 192.168.0.2
- **Default Gateway**: 192.168.0.2

And the network settings for the testbed are set up like this:

- **IPv4**: 192.168.0.2
- **Netmask**: 255.255.255.0
- **DNS Server**: 8.8.8.8
- **Default Gateway**: 192.168.0.1

Before using the Testbed a couple of steps are necessary to ensure that DNS, DHCP and Standard Gateway are set up correctly.

6.1 Initial Setup Process

A preliminary setup process is required to start the DNS and DHCP services, which can be accessed by clicking the "First-time setup for DNS!" and the corresponding DHCP button in the Testbed's settings section. This setup process involves several critical steps:

For DNS:

1. **General Configuration**: The process begins by fetching DNS settings from the database.
2. **DNS Helper File Location**: It identifies the location of the DNS helper file, which is expected at `testbed/helper/dns/<dns_helper_file>.py`.
3. **Technitium Token Retrieval**: It secures a communication token from Technitium.
4. **Log App Installation in Technitium**: It ensures that DNS requests are logged and stored within a dedicated database for subsequent analysis.

For DHCP:

1. **General Configuration Retrieval**: Similar to DNS, it begins by fetching the relevant settings from the database.
2. **Helper File Identification**: It locates the DNS helper file as outlined for the DNS setup.
3. **Token Acquisition**: It obtains a token for communication with Technitium, which is crucial for both DNS and DHCP setups.
4. **Existing DHCP Scope Removal**: It clears any pre-existing DHCP scope in Technitium to ensure a fresh configuration baseline.
5. **New DHCP Scope Setup**: It configures a new DHCP scope based on the specifications detailed in the settings, facilitated by the helper file.

This initial setup ensures that the IoT Testbed is correctly configured to manage network traffic, providing DNS and DHCP services while capturing detailed communication data for analysis. This process not only enhances the Testbed's network management capabilities but also solidifies its role as a crucial tool for security analysis and research within IoT ecosystems.
The important setup for the gateway however is explained in section 4.2 but the short set up process will be listed here:

1. Go to the settings page

2. Change the setup and cleanup line according to your operating system (If you are using Kali Linux no change should be needed)
3. Click on *Save*
4. Click on *Set Standard Gateway*

Now, the Testbed is ready to evaluate on the given test set of devices, they were chosen to represent a household which employs a couple of different IoT devices. First, the devices have been added to the network as seen in Figure 4.1 through either LAN or WLAN. This has been done one device after the other ensuring that the devices enter the Testbed in a controlled way so the automatically assigned data to the devices can be checked. Now the next step in the lifecycle of a device is to check the traffic it produces.

6.2 Traffic analysis

Upon integrating a device into the Testbed, the crucial process of information acquisition commences immediately. The first step involves extracting vital data from the DNS/DHCP server to monitor the timelines for adding devices. This is followed by cross-checking the MAC address list provided by the IEEE website to determine the device vendor and add their details to the database.

This initial reconnaissance phase is important in gathering information about a device even before the first traffic arrives. During the process of adding 13 test devices and verifying the MAC address vendor assignment, we made the following discovery. Six out of the 13 devices had Tuya Smart Inc. as their network card vendor, despite originating from different vendors. This information is presented in Table 6.1, and the mac vendor distribution is visualized in Figure 6.1.

Currently, Technitium DNS actively captures DNS queries and stores them in its database. A summary of these queries can be extracted on the respective device's detailed page. Moreover, you can download a comprehensive list of DNS requests directly from the Technitium DNS web user interface or the device's detail page.

We analyzed the DNS request records to gather more information about the devices in the Testbed. Our analysis revealed that two devices in our test set did not accept the given DNS but instead used their own.

These two smart appliances are the *IG301S Smart Garden* and the *pawaboo du7l-w smart Cat Feeder*. Upon analyzing their DNS logs, we found that both devices do not provide any information. This makes it difficult to analyze if they are leaking any information about the device or the user. However, it is important to note that some devices may have a hardcoded DNS and do not use the offer they receive from the DHCP as their DNS. To address this issue, we developed a tool wrapper for tcpdump.

We used tcpdump to record the traffic that runs through the Testbed. After analyzing the traffic of the two previous devices, we found that all traffic going out of them is

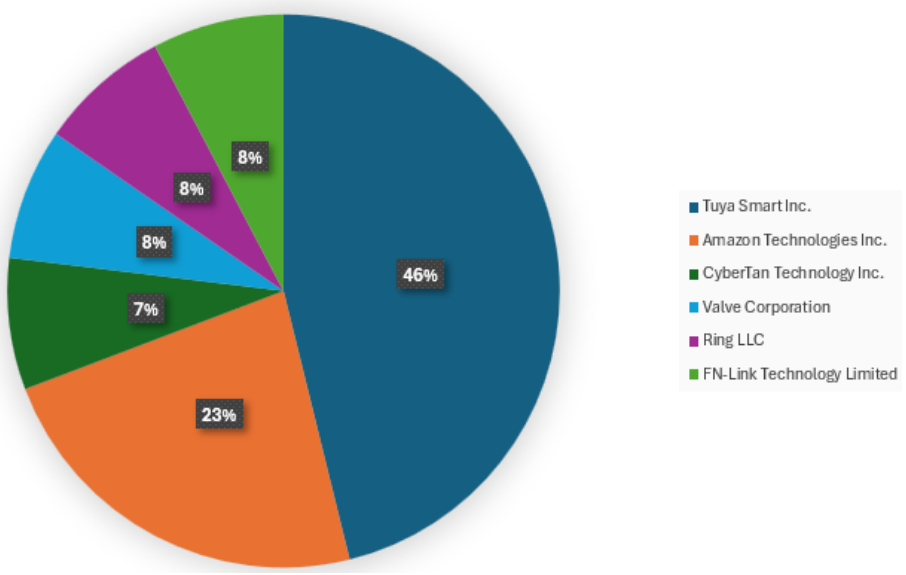

Figure 6.1: Shows the distribution of the vendors

encrypted and goes to two Amazon AWS addresses: *3.124.225.12* and *3.121.131.36*. Further the traffic going out to these addresses did not show up as DNS traffic in the tcpdump.

While analyzing the traffic of our Raspberry Pi test device, we discovered that the capacity of tcpdump to document comprehensive network traffic is limited by the Testbed's inability to function as the network's edge router. To document traffic, we employed tcpdump within the Testbed, viewed the traffic in Wireshark, and applied the filter *(ip.src == 192.168.0.155) || (ip.dst == 192.168.0.155)*. This allowed us to see only the traffic going to and coming from the test Raspberry Pi device. During an ICMP ping test, the outgoing request and the incoming reply are both captured, illustrating the bidirectional traffic flow as seen on the test device itself. However, when we inspected the equivalent traffic captured from the Testbed's perspective, we found that the incoming replies were not captured by the Testbed.

This discrepancy highlights the inherent limitation of the Testbed's operational design as a standard gateway rather than a fully-fledged routing entity. Although it adeptly captures outbound traffic from the test device, inbound traffic is usually routed directly to the test device by the network's primary router, bypassing the Testbed.

In order to capture all of a device's network traffic, it is necessary to place the traffic capture mechanism at the edge routing device of the network. This placement ensures that all incoming and outgoing traffic is routed through the capturing device, which helps to overcome any limitations and expand the scope of traffic analysis.

We have found that it may suffice to monitor the requests to identify their destination and the information they may be leaking for our current use case. After analyzing the

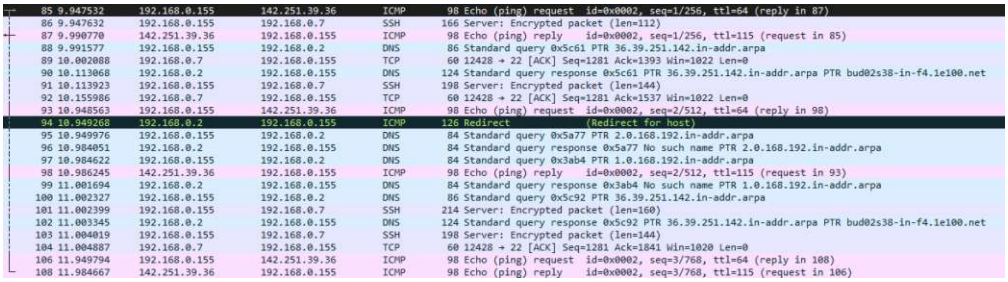

Figure 6.2: Shows the traffic of the test raspberrypi

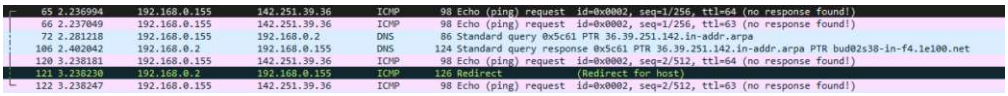

Figure 6.3: Shows the traffic of the test raspberrypi seen from the Testbed

Device	Addresses
IG301S Smart Garden	AmazonAWS(3.124.225.12,3.121.131.36)
Pawaboo du7l-w	AmazonAWS(3.7.242.33,18.195.249.137)
Bakibo smart bulb	AmazonAWS(52.58.249.45,3.121.131.36,18.185.182.159)
HUTAKUZE smart bulb	AmazonAWS(3.121.131.36,18.185.218.106,18.185.182.159)

Table 6.2: Garden and Feeder most contacted addresses

traffic using tcpdump on the Testbed and Technitium DNS logs, we discovered that 11 of our 13 test devices were using the provided DNS server, at least initially. One device, the Fitop Alexa smart bulb, initially used the provided DNS server but later switched to using "h3.iot-dns.com" for further communication.

To give an overview of the addresses that have been contacted by the smart garden and the cat feeding station the following table has been created. Two light bulbs in the test set communicated with the same addresses which can be seen in the following Table 6.2.

Out of the 13 devices, the *Yeedi vac hybrid* had very few requests and only contacted the address *mq-ww.ecouser.net*. It did not leak any information about the device or its user in the traffic. However, using tcpdump on the edge router and gathering the incoming data for this device might provide more information about the communication.

Echo Dot, *Echo Show*, and *FireTV stick* contacted mainly Amazon addresses, but they did not leak any information about the device or its user in the traffic. The smart camera LF-C1t contacted AmazonAWS and also Google under *35.207.11.126*, but it did not leak any information about the device or its user in the traffic.

For our remaining three devices, while searching for keywords in the traffic to find out if they leak any information about what or who the device is and if they leak information about the owner we found out that indeed they do. The following three devices either

leaked information in an endpoint they contacted or had the information right in the package.

The Hisense TV contacted *hisensetracker.fxmconnect.com* and leaked that the device is at least from Hisense. However, since Hisense produces a wide variety of devices, it is not possible to know for sure that it is a Hisense TV just from this address.

Furthermore, we found that the Steamlink contacted the endpoint */steamlink/06_2015/public_builds.txt*. This gave us information that the traffic is coming from a Steamlink device and the build is *06_2015*, which might be vulnerable.

The Ring Bell also revealed its name in the traffic, specifically in the packages. It can be found that it is a *Ring Chime* but it is stated that it is one from the second generation. Further, its traffic also leaked its MAC address in the traffic.

Lastly, the Raspberry Pi made some DNS requests, one of them went for *raspbian.raspberrypi.com*. This however only says that the device making the request runs Raspbian on it, but it is not a give away that it is a Raspberry Pi.

To summarize, some devices had encrypted traffic, making it challenging to analyze for leaked information. However, we found the vendors' names for three devices, and for two devices, we found the name and even a build number in the traffic. We also searched the traffic for leaked credentials, but we did not find any mention of the user's email address.

This means that for our RQ1, we found that by not using machine learning to classify devices and just using the traffic and analyzing it, we were able to identify two out of 13 devices by their model and brand *(Steamlink, Ring)*, one of those remaining 11 devices gave only the brand name *(Hisense)* giving us a success rate of 15.38% for a perfect guess and 23.07% only to guess the brand. These rates, compared to the machine learning rates are bad, and it shows that the machine learning approach gives more promising results since it relies on more factors than just plain name leaked in the traffic. Machine Learning solutions, as shown in the related work chapter 2 specifically in Table 2.2, yields a successful guess rate of around 95.82% by not only checking for brand or model name in the traffic but instead the model gets trained on four-tuples consisting of source and desination IP and port numbers from SYN to FIN. Further each session gets represented as a vector of features of the network, transport and application layers [14].

A quick summary of the devices can be found in Table 6.3. The column labeled *Ping* indicates whether a device is pingable and responds to a ping. The column labeled *DNS* indicates whether the device used the given DNS, which in our case is the Testbed. The column labeled *Leaking* describes whether a device leaked information about itself or its vendor. Lastly, the column labeled *URLs* provides a quick overview of the information that was leaked or mainly contacted by the device.

Device ID	Device Type	IP	MAC	Vendor
Hisense H49MEC3050	TV	192.168.0.168	C8-3D-D4-C5-ED-A8	CyberTan Technology Inc.
Steam link-37B3	Steamlink	192.168.0.165	E0-31-9E-2E-01-22	Valve Corporation
IG301S smart garden	Planting station	192.168.0.161	50-8B-B9-91-65-61	Tuya Smart Inc.
Pawaboo du7l-w	Smart Cat Feeder	192.168.0.160	10-D5-61-84-A0-D4	Tuya Smart Inc.
Echo Dot 2 Gen	Amazon Echo	192.168.0.157	68-37-E9-B5-89-54	Amazon Technologies Inc.
Hutakuze smart bulb	Smart Bulb	192.168.0.156	7C-F6-66-6D-56-CE	Tuya Smart Inc.
Fitop smart bulb	Smart Bulb	192.168.0.155	7C-F6-66-6D-47-CE	Tuya Smart Inc.
Yeedi vac hybrid	Smart Vacuum	192.168.0.154	2C-D2-6B-79-9A-3A	FN-Link Technology Limited
Bakibo smart bulb	Smart Bulb	192.168.0.153	10-D5-61-A5-FB-2A	Tuya Smart Inc.
Ring Bell, 5at1s2	Ring Bell	192.168.0.174	9C-76-13-D0-FF-C4	Ring LLC
LF-C1t	Smart Cam	192.168.0.175	84-E3-42-3B-F9-CD	Tuya Smart Inc.
FireTV Stick	FireTV	192.186.0.179	80-0C-F9-C7-25-10	Amazon Technologies Inc.
Echo Show	Amazon Echo	192.186.0.180	70-70-AA-8C-F2-C0	Amazon Technologies Inc.
Raspberry Pi	RPi Model 2 V1.1	192.186.0.152	B8-27-EB-B1-BA-56	Raspberry Pi Fdn.

Device ID	Ping	DNS	Leaking	Urls
Hisense H49MEC3050	Yes	Yes	No	hisensetracker.fxmconnect.com
Steam link-37B3	Yes	Yes	Yes	/steamlink/06_2015/public_builds.txt
IG301S smart garden	Yes	No	No	Encrypted, AmazonAWS 6.2
Pawaboo du7l-w	Yes	No	No	Encrypted, AmazonAWS 6.2
Echo Dot 2 Gen	No	Yes	No	Multiple
Hutakuze smart bulb	Yes	Yes	No	Encrypted, AmazonAWS 6.2
Fitop smart bulb	Yes	Yes	No	After some time only used h3.iot-dns.com
Yeedi vac hybrid	Yes	Yes	No	2x: mq-ww.ecouser.net
Bakibo smart bulb	Yes	Yes	No	Encrypted, AmazonAWS 6.2
Ring Bell, 5at1s2	Yes	Yes	yes/no	MAC Address in package, exposes ring chime but not second gen
LF-C1t	Yes	Yes	No	Google 35.207.11.126 and AmazonAWS
FireTV Stick	Yes	Yes	No	Amazon traffic
Echo Show	Yes	Yes	No	Amazon traffic
Raspberry Pi	Yes	Yes	Yes	raspbian.raspberrypi.com

Table 6.3: Full test device list

Device ID	Device Type	Vendor	Ports
Hisense H49MEC3050	TV	CyberTan Technology Inc.	-
Steam link-37B3	Steamlink	Valve Corporation	-
IG301S smart garden	Planting station	Tuya Smart Inc.	-
Pawaboo du7l-w	Smart Cat Feeder	Tuya Smart Inc.	-
Echo Dot 2 Gen	Amazon Echo	Amazon Technologies Inc.	8888
Hutakuze smart bulb	Smart Bulb	Tuya Smart Inc.	-
Fitop smart bulb	Smart Bulb	Tuya Smart Inc.	-
Yeedi vac hybrid	Smart Vacuum	FN-Link Technology Limited	8888
Bakibo smart bulb	Smart Bulb	Tuya Smart Inc.	-
Ring Bell, 5at1s2	Ring Bell	Ring LLC	-
LF-C1t	Smart Cam	Tuya Smart Inc.	-
FireTV Stick	FireTV	Amazon Technologies Inc.	-
Echo Show	Amazon Echo	Amazon Technologies Inc.	8009,9
Raspberry pi 3	Raspberry pi	Raspberry Pi Fdn.	22,5432

Table 6.4: Ports for each device

6.3 Tool execution

To start with the tool execution, we created the first pipeline to reveal any obstacles. We started by writing a wrapper for the NMap tool. NMap allows us to quickly scan a device for open ports, which might reveal an attack vector. However, after using this tool on our test device set, we found that none of our test devices offered a port that could be attacked with a simple password brute force attack. The open ports of each device can be seen in Table 6.4.

Unfortunately, our test devices, as seen in Table 6.4 offer only three ports in total and we assumed that services are being used with their standard ports:

- **8888**: This port is often used to access web services. In the context of IoT, devices might use it to receive commands or send data to a controlling app or service.
- **8009**: This port is commonly associated with the Apache JServ Protocol (AJP). AJP is used for server-to-server communications, like between an application server and an Apache web server.
- **9**: Traditionally known as the Discard Protocol, it's used for debugging and network performance testing, essentially acting as a sinkhole for sending data that will be discarded.

Our Raspberry, however, gives us better control for this test since we can set up the services and use password/username credentials as we like for each one of those services.

When running NMap on the raspberry pi, we get the output as seen in Listing 9.

```
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-03-08 15:59 CET
Nmap scan report for 192.168.0.155
Host is up (0.0021s latency).
Not shown: 98 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
5432/tcp open  postgresql
MAC Address: B8:27:EB:B1:BA:56 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 0.52 seconds
```

Listing 9: Output of Nmap

We have identified that our Raspberry Pi has two open services: SSH on port 22 and PostgreSQL on port 5432. The next step would be to use a tool called Hydra to attack these open services. However, if a tool does not have an open port/service, there is no need to run the following attack after the Nmap probe, as the port would not be open anyway.

We also noticed that it is important for a tool to have the ability to save data in any form to the device's information, so that the next tool or a tool further down the pipeline could look back at previous steps or tools to retrieve their information and use them for the next step/attack.

In our case, Hydra is the tool in the pipeline that checks if a device has an open port and gets attacked in the pipeline. This is why devices have a field in the database called "tool_info", and after the Nmap run, this dictionary will look like the Listing 10.

A possible pipeline setup is illustrated in Figure 5.3. Here, Nmap provides information on open services, and subsequently, the Testbed utilizes a brute force attack to identify login credentials for these services.

However, before running any step, such as the hydra step after the Nmap probe, a checking procedure was required. We decided that each pipeline should have its own Python file where the pipeline developer can add methods that will return either True or False to determine if a step should be executed or not.

In the case of the Raspberry Pi, as it has two open ports, an attack against Postgresql will look like this: *hydra -L addons/hydra/wordlists/iot_wordlist.txt -P addons/hydra/wordlists/iot_wordlist.txt -t 4 postgres://192.168.0.155.*

As a result of Hydra's execution, credentials for the Postgresql service were successfully identified, which can be seen in the output of hydra in Listing 11.

Analogously, the penetration test conducted on the SSH service yielded comparable outcomes, effectively uncovering valid access credentials.

```
{
"nmap": [
  {
    "port": "22",
    "state": "open",
    "service": "ssh"
  },
  {
    "port": "5432",
    "state": "open",
    "service": "postgresql"
  }
],
}
```

Listing 10: NMap tool info

```
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak -
Please do not use in military or secret service organizations, or for
↪  illegal
purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-03-08
↪  16:05:11
[DATA] max 4 tasks per 1 server, overall 4 tasks, 81 login tries (l:9/p:9),
~21 tries per task
[DATA] attacking postgres://192.168.0.155:5432/
[5432][postgres] host: 192.168.0.155   login: pi   password: raspberry
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra)
finished at 2024-03-08 16:05:15
```

Listing 11: Hydra Postgres output

The hydra ssh process (*hydra -L addons/hydra/wordlists/iot_wordlist.txt -P addons/hydra/wordlists/iot_wordlist.txt -t 4 postgres://192.168.0.155*) returned with the the output seen in Listing 12.

This data is systematically collated into a dictionary structure, which is subsequently persisted within the database for potential utilization across additional add-ons, an example of this can be seen in Listing 13.

```
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak -
Please do not use in military or secret service organizations, or for
↪  illegal
purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-03-08
↪  16:05:11
[DATA] max 4 tasks per 1 server, overall 4 tasks, 81 login tries (l:9/p:9),
~21 tries per task
[DATA] attacking ssh://192.168.0.155:22/
[22][ssh] host: 192.168.0.155   login: pi   password: raspberrz
[STATUS] 79.00 tries/min, 79 tries in 00:01h, 2 to do in 00:01h, 4 active
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra)
finished at 2024-03-08 16:06:14
```

Listing 12: Hydra SSH output

Currently, in the Testbed, pipelines from tools are available but require a specific Python file to run checks between tasks and determine whether specific steps are necessary or not. Additionally, a dictionary is necessary to store and retain data found by tools, which can then be utilized by other tools.

By doing so, we answered our RQ2 by successfully creating a modular framework that allows us to easily extend the framework capabilities by adding new tools; each tool then brings its functions and contributes to the device results dictionary, which information is then shared between other tools. Further chaining tools together by using the result dictionary and the pipeline functions makes an attack workflow possible and allows for an automated vulnerability check of a device. In our test environment, we used the Raspberry Pi that exposed two services with weak credentials, and both credentials sets for the SSH and PostgreSQL services were found.

Before a test device finishes its lifecycle in the Testbed, a report is now generated to collect the information about the tools that have been running against a test device and, if possible, classify if a vulnerability has been found by a tool or if a device is secure against the attack of a specific tool.

6.4 Automatic classification and dynamic reporting

To address the issue of automatically interpreting the results, we initially used a simple approach of parsing the return code for each tool. However, we soon realized that relying solely on the return code was not enough to determine the success of a tool. For instance, a hydra run that does not find any passwords or a nmap run that does not find any

```
{
  "nmap": [
  {
    "port": "22",
    "state": "open",
    "service": "ssh"
  },
  {
    "port": "5432",
    "state": "open",
    "service": "postgresql"
  }
  ],
  "hydra_postgres": [
  {
    "port": "5432",
    "service": "postgres",
    "host": "192.168.0.155",
    "login": "pi",
    "password": "raspberry"
  }
  ],
  "hydra_shh": [
  {
    "port": "22",
    "service": "ssh",
    "host": "192.168.0.155",
    "login": "pi",
    "password": "raspberrz"
  }
  ]
}
```

Listing 13: Tool_info json with NMap and Hydra information

open ports can return a 0, indicating that the task ran successfully but providing no information about the status of the task.

Therefore, we added a function to each addon that interprets the results, utilizing regex checking for the presence of a string in most cases, while allowing for more complex checks where necessary. If a tool does not provide data that can be easily categorized as success or failure, the user can manually set the status of the tool run. To streamline this process and present the information in a clear and user-friendly manner, we developed

a jinja-driven solution. This approach enables the customization of data presentation, allowing the user to view the information in a format that is both accessible and visually appealing. For instance, in the case of the test Raspberry Pi device, the output from the Nmap scan and the retrieved credentials are organized in a table, as shown in Figure 6.4. This method enhances the readability of the data while retaining the option for users to download the raw output for further analysis. The use of a tabular presentation in this particular instance is effective in delivering a concise and organized view of the data. Additionally, the implementation of the project to this thesis allows for the results to be compiled into a downloadable PDF document using the same HTML Jinja templates that are used to show the results on the web page. This flexibility ensures that the Testbed can accommodate divergent presentation styles with simple modifications to the Jinja template if needed.

The results of each individual test are thoroughly documented in the logs, and the Jinja templates elegantly display these results. However, this segment focuses on an aggregate examination of results, which has been explained in the implementation section of this thesis. Let us briefly examine the logs index, illustrated in Figure 6.5. This interface catalogs each execution of a tool or pipeline, presenting essential information at a glance. Entries highlighted in orange indicate logs that have not yet been reviewed or have undergone recent changes.

When you select a pipeline log, you'll be able to see the status of each step involved in the process. Additionally, if you access a tool's log from either the pipeline log or the index view, you'll be able to view detailed information related to that specific execution.

Furthermore, the detailed view of each device - such as the Raspberry Pi test device shown here - consolidates all relevant logs, making it easier for you to access and review them from a centralized interface.

Once you engage the "Download Report" feature, a comprehensive report will be generated. This report will encapsulate all the findings and analyses in a structured format. For instance, in the case of Raspberry Pi, the report will look like this example attached here.

Device Overview

Device Name	testpi
Vendor	Raspberry Pi Foundation(B827EB)
ID	B8-27-EB-B1-BA-56
Discovered	Nov. 26, 2023, 5:29 p.m.
Last Seen	March 8, 2024, 3:18 p.m.
Updated	Feb. 9, 2024, 2 p.m.
Hardware Address	B8-27-EB-B1-BA-56
IP Address	192.168.0.155
Host Name	testerpi
Comments	raspberry pi

Selected Logs

ID	Name	Result
277	hydra shh	Vulnerable
276	hydra postgres	Vulnerable
275	nmap	Vulnerable

Log Entry(277): hydra shh

Extracted Credentials

Port	Service	Host	Login	Password
22	ssh	192.168.0.155	pi	raspberrz

Log Entry(276): hydra postgres

Extracted Credentials

Port	Service	Host	Login	Password
5432	postgres	192.168.0.155	pi	raspberry

Log Entry(275): nmap

Nmap Scan Results

Port	State	Service
22	open	ssh
5432	open	postgresql

Figure 6.4: Shows nmap results of the Raspberry Pi

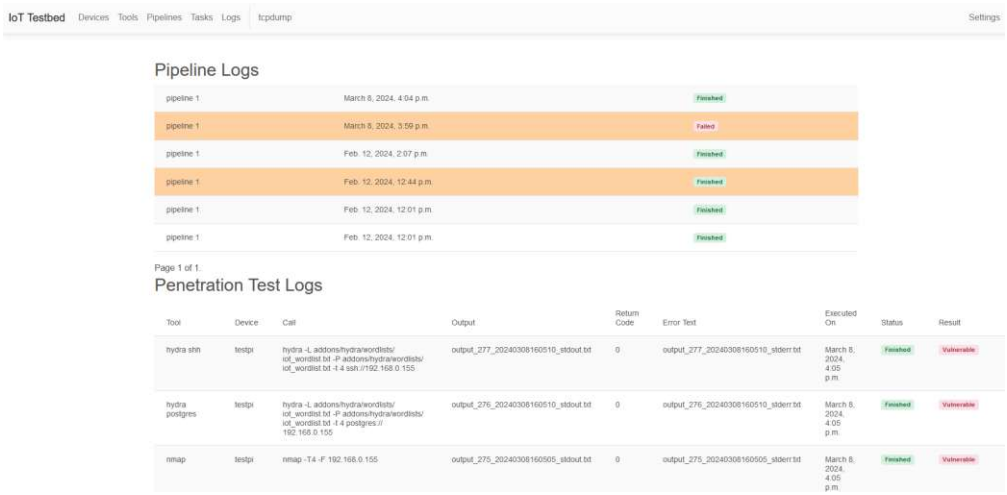

Figure 6.5: Index view of the logs.

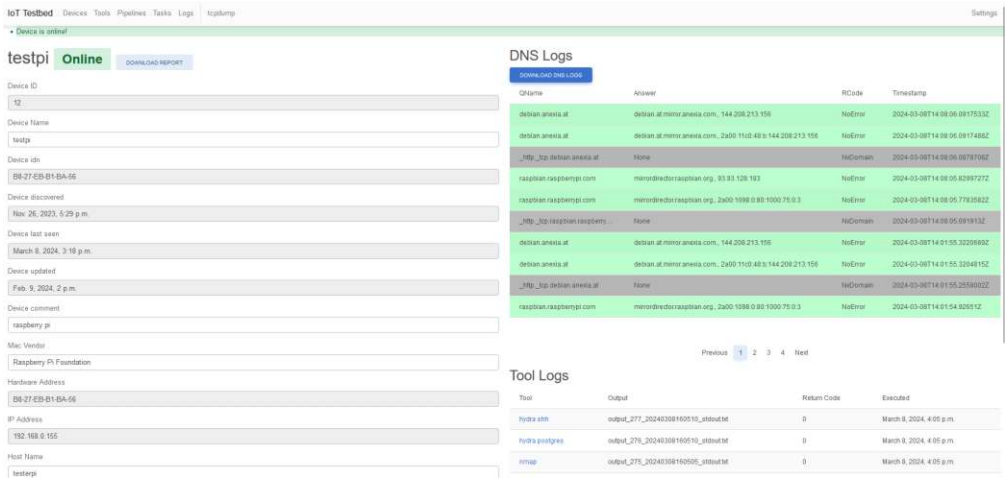

Figure 6.6: Detail page of the test Raspberry Pi.

This integrated approach to logging and reporting not only ensures meticulous documentation of each test and its outcomes but also enhances the interpretability of results through structured presentation formats. For RQ3, the results from different tools can be automatically interpreted to a certain extent. This means a way has been found to parse all kinds of output and display them in a nice way with Jinja, and by doing so, allows for an easy way to make a fitting report template for a tool.

CHAPTER 7

Conclusion and Furture Work

The subsequent chapters will delve into the conclusion and delineate prospective avenues for future work.

7.1 Conclusion

This thesis provides a detailed examination of the functionality of a Testbed, from the initial integration and information acquisition regarding devices, through the automation of penetration testing, to the sophisticated presentation of findings and comprehensive reporting. The following are the key points of the thesis:

- The first research question focuses on the Testbed's ability to acquire device-specific information through its traffic. The goal was to identify each device by just looking at its traffic and search through it with keywords. While the system captures a range of data and conducts preliminary reconnaissance, there are some inherent limitations. Specifically, the potential for devices to utilize alternative DNS settings diminishes the effectiveness of this approach, compromising the comprehensiveness of data acquisition. Additionally, the failure to capture inbound traffic to the test device results in the exclusion of potentially critical data, underscoring a significant constraint in the current operational framework of the Testbed.
  This manual approach however, checking the traffic for keywords such as model or brand type as well as the username to which account the are linked to, yielded a success rate of 23.07% (two out of 13 devices) which allowed for an identification of their brand and model showing us that the AI approach here yields more promising results and is a lot faster.
- The second research question examines the utility of automated penetration testing facilitated by the Testbed's integration with tools such as Hydra. The deployment

of pipelines significantly enhances the Testbed's usability, enabling systematic and extensive security assessments. The modular architecture of the system further amplifies its capabilities, allowing for the incorporation of a diverse array of auxiliary tools to conduct thorough penetration tests. The successful identification and brute force extraction of credentials across services underscore the potency and effectiveness of the existing toolkit within the Testbed's ecosystem and how easily it can be extended.

- The third research question benefits significantly from the Testbed's modular design, particularly through the utilization of HTML Jinja templates for result presentation. While attempting to automatically interpret and classify the first results as succeeded or not worked for some tools, it was rather difficult for others. On the other hand, the tailored rendering of complex raw outputs into accessible and aesthetically appealing formats is working well. By selectively emphasizing pertinent information and omitting extraneous data, the Testbed ensures that outputs are not only readable but also contextually enriched, catering to a wide range of analytical and reporting requirements. The system not only facilitates real-time display of findings within the Testbed's interface but also supports the generation of detailed reports. These reports serve as an exhaustive summary of the device's security posture and the outcomes of the conducted tests, offering a pivotal resource for both immediate analysis and future reference.

7.2 Future Work

This thesis has established a robust framework for a modular IoT Testbed that has great potential in addressing the current cybersecurity challenges faced by IoT devices. While the Testbed has shown considerable strengths, particularly in automated penetration testing and data presentation, the exploration of the research questions has illuminated areas that warrant further development. These insights provide a clear path for future work, aiming to enhance the Testbed's capabilities and extend its applicability. Below are the key areas identified for future enhancements:

- **Enhanced Information Acquisition**:
  The current architecture of the Testbed is effective in capturing certain aspects of data, but it has limitations in comprehensively acquiring incoming traffic data. This lack of coverage highlights the need for a more robust information acquisition system that can successfully monitor and analyze both outbound and inbound network traffic. To achieve this, future work should focus on integrating advanced traffic capture mechanisms that can operate at a granular level. This could potentially involve leveraging deep packet inspection (DPI) to provide detailed insights into the data transmitted to and from IoT devices. Enhancing the Testbed to function as a central node or deploying network taps could significantly improve visibility into network communications, thereby overcoming the current limitations. Further,

a tool wrapper will be written for Ettercap, which allows the Testbed to play Man in the Middle and capture not only outgoing traffic but also the responses that come back to the IoT device that is being tested. This should work since Ettercap can use ARP poising to change the MAC address of the router/gateway; by doing so, the router and the victim device will send their traffic through the Testbed. This solution effectively addresses the issue of capturing traffic. However, we have observed that utilizing machine learning approaches yields higher success rates compared to our manual analysis. This means that for device brand/type detection, it would be more reasonable to incorporate an established tool like ProfilIoT as an addon to the Testbed.

- **Expansion of Toolset**:
  The Testbed's modular design provides opportunities for incorporating additional tools, expanding its versatility, and enhancing its capabilities. In future expansions, priority should be given to integrating tools that cover a wider range of cybersecurity assessments. Diversifying the Testbed's toolset will enable it to provide a more comprehensive security assessment, addressing the evolving threat landscape and the growing complexity of IoT devices.

- **Root permissions**:
  At the moment, certain tools like tcpdump require root privileges to access system resources. Although this approach is effective, it is not considered the best solution for managing access permissions and resource utilization. A better strategy would be to create a dedicated user account with the necessary permissions tailored to the operational needs of these tools. However, creating such a user and managing its permissions in alignment with the introduction of new tools can be complex. In this project, this aspect was not addressed, assuming that the Testbed would operate only in a local network environment on a dedicated host. Looking ahead, there is a need to develop comprehensive rights management functionality to simplify the allocation and administration of permissions, thereby improving the security and usability of the Testbed in future implementations.

- **Integration of Machine Learning Techniques**:
  The utilization of machine learning techniques offers a promising opportunity to improve the analytical capabilities of the Testbed. By integrating machine learning algorithms, the Testbed would be able to automatically recognize unusual behaviors and possibly malicious activities within network traffic. This methodology could greatly simplify the process of detecting vulnerabilities and identifying potential threats, making the Testbed more efficient and effective for real-time security monitoring.

List of Figures

List of Tables

List of Listings

Bibliography

[1] Number of IoT devices. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/. (Accessed on 2023-03-28).

[2] Hospitals attacked by ransomware. https://phoenixnap.com/blog/ransomware-healthcare. (Accessed on 2023-03-28).

[3] Bertino, Elisa and Islam, Nayeem Botnets and Internet of Things Security 2017 10.1109/MC.2017.62

[4] Top 10 Vulnerabilities that Make IoT Devices Insecure https://venafi.com/blog/top-10-vulnerabilities-make-iot-devices-insecure/ (Accessed on 2023-03-28).

[5] Internet-of-Things Security and Vulnerabilities: Taxonomy, Challenges, and Practice Chen, K., Zhang, S., Li, Z. et al. https://doi.org/10.1007/s41635-017-0029-7 (Published on 10 May 2018).

[6] IoT Devices Recognition Through Network Traffic Analysis Shahid, Mustafizur R. and Blanc, Gregory and Zhang, Zonghua and Debar, Hervé 10.1109/BigData.2018.8622243 (Published 2018).

[7] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1093–1110, Vancouver, BC, August 2017. USENIX Association.

[8] Bertino, Elisa and Islam, Nayeem Botnets and Internet of Things Security 2017 10.1109/MC.2017.62

[9] fit iot lab. The very large scale iot testbed. https://www.iot-lab.info/. (Accessed on 2022-05-10).

[10] Risk Prediction of IoT Devices Based on Vulnerability Analysis https://dl.acm.org/doi/10.1145/3510360. (Accessed on 2023-03-28).

[11] TUI Model for data privacy assessment in IoT networks https://www.sciencedirect.com/science/article/pii/S2542660521001062?via%3Dihub. (Accessed on 2023-03-28).

[12] Peek-a-boo: i see your smart home activities, even encrypted! Acar, Abbas and Fereidooni, Hossein and Abera, Tigist and Sikder, Amit Kumar and Miettinen, Markus and Aksu, Hidayet and Conti, Mauro and Sadeghi, Ahmad-Reza and Uluagac, Selcuk 2020 10.1145/3395351.3399421

[13] Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics Arunan Sivanathan and Hassan Habibi Gharakheili and Franco Loi and Adam Radford and Chamith Wijenayake and Arun Vishwanath and Vijay Sivaraman 2019 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8440758

[14] ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis Yair Meidan and Michael Bohadana and Asaf Shabtai and Juan David Guarnizo and Martín Ochoa and Nils Ole Tippenhauer and Yuval Elovici 2017 https://dl.acm.org/doi/10.1145/3019612.30198780

[15] Andy Greenberg. Hackers remotely kill a jeep on the highway—with me in it. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. (Accessed on 2022-05-09).

[16] Matthew L. Hale, Dalton Ellis, Rose Gamble, Charles Waler, and Jessica Lin. Secu wear: An open source, multi-component hardware/software platform for exploring wearable security. In *2015 IEEE International Conference on Mobile Services*, pages 97–104, 2015.

[17] Zhen Ling, Kaizheng Liu, Yiling Xu, Yier Jin, and Xinwen Fu. An end-to-end view of iot security and privacy. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–7, 2017.

[18] Zhen Ling, Junzhou Luo, Yiling Xu, Chao Gao, Kui Wu, and Xinwen Fu. Security vulnerabilities of internet of things: A case study of the smart plug system. *IEEE Internet of Things Journal*, 4(6):1899–1909, 2017.

[19] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. Smartphones attacking smart-homes. In *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '16, page 195–200, New York, NY, USA, 2016. Association for Computing Machinery.

[20] Vinay Sachidananda, Shachar Siboni, Asaf Shabtai, Jinghui Toh, Suhas Bhairav, and Yuval Elovici. Let the cat out of the bag: A holistic approach towards security analysis of the internet of things. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security*, IoTPTS '17, page 3–10, New York, NY, USA, 2017. Association for Computing Machinery.

[21] Aakanksha Tewari and B.B. Gupta. Security, privacy and trust of different layers in internet-of-things (iots) framework. *Future Generation Computer Systems*, 108:909–920, 2020.

[22] Jacob Wurm, Khoa Hoang, Orlando Arias, Ahmad-Reza Sadeghi, and Yier Jin. Security analysis on consumer and industrial iot devices. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 519–524, 2016.

[23] Mengmei Ye, Nan Jiang, Hao Yang, and Qiben Yan. Security analysis of internet-of-things: A case study of august smart lock. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 499–504, 2017.

[24] Wei Zhou and Selwyn Piramuthu. Security/privacy of wearable fitness tracking iot devices. In *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–5, 2014.

[25] Celery is a task queue with batteries included. `https://docs.celeryq.dev/`. (Accessed on 2024-02-05).

[26] Nmap is a utility for network exploration or security auditing. `https://www.kali.org/tools/nmap/`. (Accessed on 2024-04-19).

[27] Hydra is a parallelized login cracker which supports numerous protocols to attack. `https://www.kali.org/tools/hydra/`. (Accessed on 2024-04-19).

[28] This program allows you to dump the traffic on a network. `https://www.kali.org/tools/tcpdump/`. (Accessed on 2024-04-19).

[29] This package contains a Network Forensic Analysis Tool (NFAT) that performs deep processing and inspection of network traffic (mainly PCAP files, but it also capable of directly live capturing from a network interface). `https://www.kali.org/tools/bruteshark/`. (Accessed on 2024-04-19).

[30] Self host a DNS server for privacy & security `https://technitium.com/`. (Accessed on 2024-02-05).

[31] The most advanced Penetration Testing Distribution. https://www.kali.org/. (Accessed on 2024-02-05).

[32] The ultimate framework for your Cyber Security operations https://www.parrotsec.org/. (Accessed on 2024-02-05).

[33] BlackArch Linux is an Arch Linux-based penetration testing distribution for penetration testers and security researchers. https://blackarch.org/. (Accessed on 2024-02-05).

[34] The most comprehensive Zero Trust solution for IoT devices. https://www.paloaltonetworks.com/network-security/enterprise-iot-security/. (Accessed on 2024-02-26).

[35] Django makes it easier to build better web apps more quickly and with less code. https://www.djangoproject.com/. (Accessed on 2024-02-27).

[36] The World's Most Advanced Open Source Relational Database https://www.postgresql.org/. (Accessed on 2024-02-27).

[37] Visoottiviseth, Vasaka and Akarasiriwong, Phuripat and Chaiyasart, Siravitch and Chotivatunyu, Siravit PENTOS: Penetration testing tool for Internet of Thing devices In *TENCON 2017 - 2017 IEEE Region 10 Conference* doi: 10.1109/TENCON.2017.8228241

[38] Mohd Bakry, Batrisyia B and Bt Adenan, Alisa Rafiqah and Mohd Yussoff, Yusnani Bt Security Attack on IoT Related Devices Using Raspberry Pi and Kali Linux 2022 doi: 10.1109/ICONDA56696.2022.10000370

[39] Ronen, Eyal and Shamir, Adi Extended Functionality Attacks on IoT Devices: The Case of Smart Lights 2016 doi: 10.1109/EuroSP.2016.13

[40] Seralathan, Yogeesh and Oh, Tae Tom and Jadhav, Suyash and Myers, Jonathan and Jeong, Jaehoon Paul and Kim, Young Ho and Kim, Jeong Neyo IoT security vulnerability: A case study of a Web camera 2018 doi: 10.23919/ICACT.2018.8323686

[41] Tekeoglu, Ali and Tosun, Ali Şaman A Testbed for Security and Privacy Analysis of IoT Devices 2016 doi: 10.1109/MASS.2016.051

[42] Omnia (Abu Waraga) and Meriem Bettayeb and Qassim Nasir and Manar (Abu Talib) Design and implementation of automated IoT security testbed 2020 doi: https://doi.org/10.1016/j.cose.2019.101648

[43] Anagnostopoulos, Marios and Spathoulas, Georgios and Viaño, Brais and Augusto-Gonzalez, Javier Tracing Your Smart-Home Devices Conversations: A Real World IoT Traffic Data-Set 2020 doi: 10.3390/s20226600

[44] Hafeez, Ibbad and Antikainen, Markku and Ding, Aaron Yi and Tarkoma, Sasu IoT-KEEPER: Detecting Malicious IoT Network Activity Using Online Traffic Analysis at the Edge 2020 doi: 10.1109/TNSM.2020.2966951

[45] Subahi, Alanoud and Theodorakopoulos, George Detecting IoT User Behavior and Sensitive Information in Encrypted IoT-App Traffic 2019 doi: 10.3390/s19214777

[46] Bachy, Yann and Basse, Frédéric and Nicomette, Vincent and Alata, Eric and Kaâniche, Mohamed and Courrège, Jean-Christophe and Lukjanenko, Pierre Smart-TV Security Analysis: Practical Experiments 2015 doi: 10.1109/DSN.2015.41

[47] Wood, Daniel and Apthorpe, Noah and Feamster, Nick Cleartext Data Transmissions in Consumer IoT Medical Devices 2017 doi: 10.1145/3139937.3139939