future internet

MDPI

Article

Automated Penetration Testing Framework for Smart-Home-Based IoT Devices

Rohit Akhilesh [1], Oliver Bills [1], Naveen Chilamkurti [2] and Mohammad Jabed Morshed Chowdhury [2,*]

[1] Department of Cybersecurity, School of Engineering and Computer Sciences, University of Southampton, University Road, Southampton SO17 1BJ, UK
[2] Department of Computer Science and Engineering, La Trobe University, Bundoora VIC 3086, Australia
* Correspondence: m.chowdhury@latrobe.edu.au

Abstract: Security testing is fundamental to identifying security vulnerabilities on smart home-based IoT devices. For this, penetration testing is the most prominent and effective solution. However, testing the IoT manually is cumbersome and time-consuming. In addition, penetration testing requires a deep knowledge of the possible attacks and the available hacking tools. Therefore, this study emphasises building an automated penetration testing framework to discover the most common vulnerabilities in smart home-based IoT devices. This research involves exploring (studying) different IoT devices to select five devices for testing. Then, the common vulnerabilities for the five selected smart home-based IoT devices are examined, and the corresponding penetration testing tools required for the detection of these vulnerabilities are identified. The top five vulnerabilities are identified from the most common vulnerabilities, and accordingly, the corresponding tools for these vulnerabilities are discovered. These tools are combined using a script which is then implemented into a framework written in Python 3.6. The selected IoT devices are tested individually for known vulnerabilities using the proposed framework. For each vulnerability discovered in the device, the Common Vulnerability Scoring System (CVSS) Base score is calculated and the summation of these scores is taken to calculate the total score (for each device). In our experiment, we found that the Tp-Link Smart Bulb and the Tp-Link Smart Camera had the highest score and were the most vulnerable and the Google Home Mini had the least score and was the most secure device of all the devices. Finally, we conclude that our framework does not require technical expertise and thus can be used by common people. This will help improve the field of IoT security and ensure the security of smart homes to build a safe and secure future.

Keywords: internet of things (IoT); penetration testing (PT); smart home; IoT devices; Common Vulnerability Scoring System (CVSS)

check for updates

Citation: Akhilesh, R.; Bills, O.; Chilamkurti, N.; Chowdhury, M.J.M. Automated Penetration Testing Framework for Smart-Home-Based IoT Devices. *Future Internet* **2022**, *14*, 276. https://doi.org/10.3390/fi14100276

Academic Editor: Ishaani Priyadarshini

Received: 31 July 2022
Accepted: 19 September 2022
Published: 27 September 2022

1. Introduction

IoT devices are physical objects which offer services through online connectivity. They are equipped with sensors, actuators, and controllers. Any device that can connect, interact, and exchange data with the internet is an IoT device [1]. It is a new way of data exchange, where devices interact with each other for the delivery of services [2]. In recent times, there has been a rapid expansion of the IoT and because of being in a highly technological society, people are surrounded by networked smart devices designed to enhance productivity, efficiency, and efficacy. These devices operate interactively and anonymously and encompass most of the Internet of Things. Therefore, it can be established that the IoT is far more ubiquitous than many realize and based on a recent statistic, the total number of IoT devices (worldwide) is expected to be around 22 billion (2018) and is expected to reach 50 billion by the year 2030 [3]. This expansion is expected to cause an evolution in the government and majorly in the fields of transportation, education, and finance, along with a few other fields [4]. This expansion, with the help of network virtualization, communication capabilities, and cloud computing, has enabled a new connection between the virtual and the physical world.

IoT devices have different applications across various fields through which information can be shared and communicated via the Internet [5]. These devices fall into different categories, such as information technology and enterprise devices; operation technology like SCADA (supervisory control and data acquisition) [6], process control, medical devices; consumer devices like smartphones, smart bulbs, smart kettles; and various other single-purpose devices [4]. Another good example is wearable devices; they are a transformation of ordinary wearable products into smart devices such as clothes, shoes, or watches, which can be worn on the user's body and can interact with their surroundings [7]. Additionally, the concept of smart homes which comes under consumer devices is of great interest in the field of IoT.

Smart homes are an environment built using smart IoT devices. In other words, a smart home "is a house or living environment that contains the technology to allow home devices and systems to be controlled automatically" [8]. There is no formal definition for a smart home but the most suitable was one found in one research paper: "A residence equipped with a communications network, linking sensors and domestic devices and appliances that can be remotely monitored, accessed or controlled, and which provide services that respond to the needs of its inhabitants" [9]. All the IoT devices present in such an environment are smart home-based IoT devices. Deployed at home, these devices generally communicate with the servers on the cloud and access the network (directly or indirectly) through protocols like ZigBee, Wi-Fi, and Bluetooth [10].

There are six important components surrounding the architecture of smart homes [11]:

1. **Hardware:** It includes all the physical components of the device (including the firmware).
2. **User interface:** The devices used by the user to access the IoT devices (i.e. The user's PC or Mobile Phone).
3. **Web Application:** It helps in configuring, accessing, or controlling a device. In most cases, it is hosted on a web server residing in the device itself.
4. **Mobile Application:** It is like a web application, but with more features and is more practical for the user's use.
5. **Network and Cloud:** All the communications between the devices and the applications happen to go through the network and the data is stored in the cloud.
6. **API (Application Protocol Interface):** This plays a crucial role in making interoperability and interaction between the device, application, and the cloud possible.

An example of the architecture described above is shown in Figure 1.

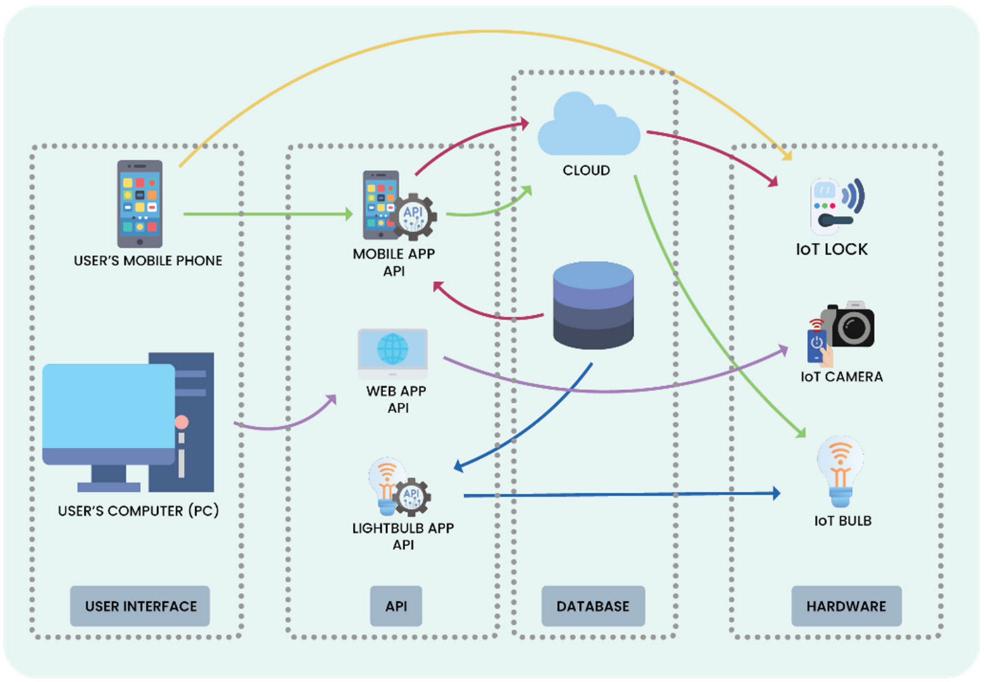


Figure 1. The architecture of a Smart home.

Above Figure 1 represents the connections between the components surrounding the architecture of smart homes. The figure briefly explains how the user connects and interacts with the smart home-based IoT devices in the hardware layer (the IoT lock, the IoT camera and the IoT bulb) using his smartphone or computer (shown in the User Interface layer of the figure). Each device has an application associated with it, which can be used by the user through his smart device to control and interact with the IoT device. This application establishes a connection with the device through its corresponding API that is present between the app and the IoT device. Each application has a corresponding API (shown in the API layer of the figure) that establishes a connection with the specific IoT device. Apart from this, there is also a database layer which contains the cloud and local database. All the data corresponding to the interactions between the user and the IoT device get stored in this layer. Each uniquely coloured line in the figure represents the connections corresponding to one set of interactions between the user interface and the hardware.

In recent times, the Internet of Things (IoT) has been a very fast-growing and trending technology. People are surrounded by Internet of Things (IoT) devices in their daily life in one way or another. There would not be a day in which people are not using at least one of these devices in their day-to-day life. However, the emphasis on the functionality of these devices is so high that the manufacturers, and the people, are ignoring or forgetting another very important aspect to take into consideration, i.e., 'Security'. As a result, these devices are manufactured with very basic security issues, such as outdated software, default passwords, improper configurations, and poor logging. These vulnerabilities make the devices primary targets for potential hackers, as the hackers can easily brute force the passwords, use a known software vulnerability or exploit the improper configuration to gain access to the device. These devices also contain a lot of sensitive, personal data about the consumer, which upon getting into the wrong hands can have a severe impact on the consumer's life. Along with the concern for the safety of civilians, and their devices, these IoT devices can also be used as potential botnets to attack major digital infrastructures. Several such successful attacks have been reported and there has been an exponential increase in the number of attacks on the IoT every year [12]. Therefore, keeping these repercussions in mind, security testing of IoT devices is what is relevant and very much necessary in today's day and age.

Currently, the most prominent and viable solution to evaluate the overall security of an IoT device, from the attacker's perspective, is Penetration Testing (PT). Although there is no approach yet that completely evaluates the security of an IoT device (yet), PT is the most favoured approach [13]. However, PT of the IoT manually can be cumbersome and time-consuming. Additionally, it might be difficult for a regular user as PT requires a deep knowledge of the possible attacks and the available hacking tools. Therefore, it is evident that automation is required to make PT more efficient, and this also makes it easy for a normal user to evaluate the security of his IoT device as there's no need to have a deep understanding of the topic, unlike in the case of manual PT. This is also what the current IoT security research aims to do, i.e., automation.

In this paper, we aim to build an automated PT framework and test it on five smart home-based IoT devices to discover the most common vulnerabilities present in them. The Common Vulnerability Scoring System (CVSS) base score is evaluated based on the vulnerabilities discovered for each device, and they are compared based on the scores to identify the most secure device. The contributions of our paper are as follows:

1. Design and implement an automated penetration framework, which can be used by both technical and non-technical persons with ease.
2. Demonstrate how to use an automated penetration testing framework to test the security of different available smart home solutions.

The rest of the paper is as follows: The second section focuses on reviewing current IoT PT techniques and selecting a suitable technique for the paper. In the third section, we propose an automated PT framework. In the fourth section, we provide the results along with a Common Vulnerability Scoring System (CVSS) analysis of the results. In the fifth section, we discuss the Scope, Reflection, Risks, Constraints and Contingency planning for the research work, along with the limitations and future works. In the last section, based on the results achieved by the framework, we provide a brief conclusion to the paper.

2. Background and Related Work

PT means performing attacks on a system, for example, an IoT device, to discover its vulnerabilities. It is a well-known technique that makes use of different kinds of tools employed by malicious hackers to simulate attacks and identify the security vulnerabilities in a device. There are numerous methods and techniques to do this, some of which are reviewed and compared in this section, and later the most suitable method is selected.

Ref. [1] describes one such technique, where there was a security analysis performed in an IoT test environment, with the provided requirements using a few PT methodologies, such as port scanning, fingerprinting, process enumeration, and a vulnerability scan. They performed this on devices like Amazon Echo, Nest Cam, Philips Hue, and a few others, and found that these devices are vulnerable. However, there are various loopholes in this research, for example, the vulnerability scanner only scans for known vulnerabilities (CVE), but what if there are some vulnerabilities that are not listed in CVE? The paper talks about scanning for vulnerabilities, which means that it is only detecting them, but not testing them, i.e., exploiting the vulnerabilities and checking them; therefore, there is a possibility of false positives. It also lacks security testing for IoT protocols. Additionally, one major drawback is that it is using a testbed and not a real scenario; what if the results change when you change the conditions of the testbed? Ultimately, all these constraints denote that this is not a proper penetration technique to ensure security.

Another way to conduct PT is described in [2]. This is done using an online tool called PENTOS or penetration testing tool for IoT devices (used on Kali Linux). It is a compilation of PT tools that are used for the security testing of IoT devices. This tool is also user-friendly, i.e., it does not require any major security experience as is it quite straightforward. It also helps novice users increase their security awareness by conducting PT.

This technique overcomes a few of the drawbacks of the first technique, for instance, it does not just scan for known vulnerabilities, it tries to exploit them and check for false positives. The scenario here is a real scenario, unlike a testbed. It also performs attacks like password attacks, Wi-Fi attacks, and Web scanning, among others, unlike the first technique, which just does scanning for vulnerabilities. Although PENTOS is better than the first technique in most aspects, its scope is limited to PT over the network. It does not test for security issues on the software or the OS. This is one major drawback of PENTOS as software vulnerabilities are equally important. Additionally, IoT protocols like MQTT and CoAP cannot be tested using PENTOS, and the ZigBee PT module is still being developed.

To overcome these limitations or drawbacks, a more enhanced version of these manual PT techniques is needed, i.e., automated tools. The usage of automated techniques for the security assessment of IoT devices would lead to a prominent increase in the level of security of IoT devices, while also making sure that the costs are contained [14].

This is done by the following method shown in [15], where PT is divided into three types of specialized testing methods called:

1. **Interface Testing**: Performed on the interfaces that interact with IoT devices, which handles input validation testing.
2. **Transport Testing**: Performed on the network of the IoT devices, the associated cryptographic schemes and communication protocols for messages (protected).
3. **System Testing**: Performed on the software, firmware, Oss, and system services to test implementation flaws, insecure system settings, and other known vulnerabilities. It also involves intelligent grey-box testing.

To deal with the heterogeneity, large number, and resource constraints of IoT devices, the above technique involves a combination of principles of modularization and intelligence along with the manual PT tools and techniques. Modularization helps in providing the flexibility to test various targets and intelligence is used for enlarging test coverage and improving accuracy. Therefore, this technique is better than the second technique as it not only does everything that the PENTOS tool does, i.e., information gathering, password attacks, and many more through interface testing and transport testing, but it also tests the software and the OS of the device for vulnerabilities through system testing. Apart from that, using intelligence ensures that a lot of time and effort can be saved when compared to manual testing.

Another similar technique with the same idea, automation, is mentioned in [13]. This paper follows a layer-based PT approach, like the technique mentioned above, but the differentiation of the layers is the difference between the two techniques. In this technique, the layers are the perception layer, the hardware part of the IoT, the network layer, and the application layer, handled by the user. The network layer connects the application layer and the perception layer. The PT is done for each layer individually, hence, the layer-based PT approach. This paper has a lot of similarities with our paper in establishing the context and the security problem, but the techniques are completely different from each other. Their technique follows a Belief-Desire-Intention (BDI) based automation approach, whereas our technique follows a framework-based approach. It was not possible to comment on the accuracy or the effectiveness of their technique as their results lacked detail, and their research was probably in its initial stages.

After a thorough review and analysis, and keeping the scope of the research in mind, the most suitable method for the research work has been selected, as shown in Figure 2, as it helps in identifying vulnerabilities specifically in smart home-based IoT devices. This method is based on the PT Execution Standard (PTES). The method has various stages and procedures respective to each stage. Given below is a flow chart representing the stages and the procedure in each stage [11].

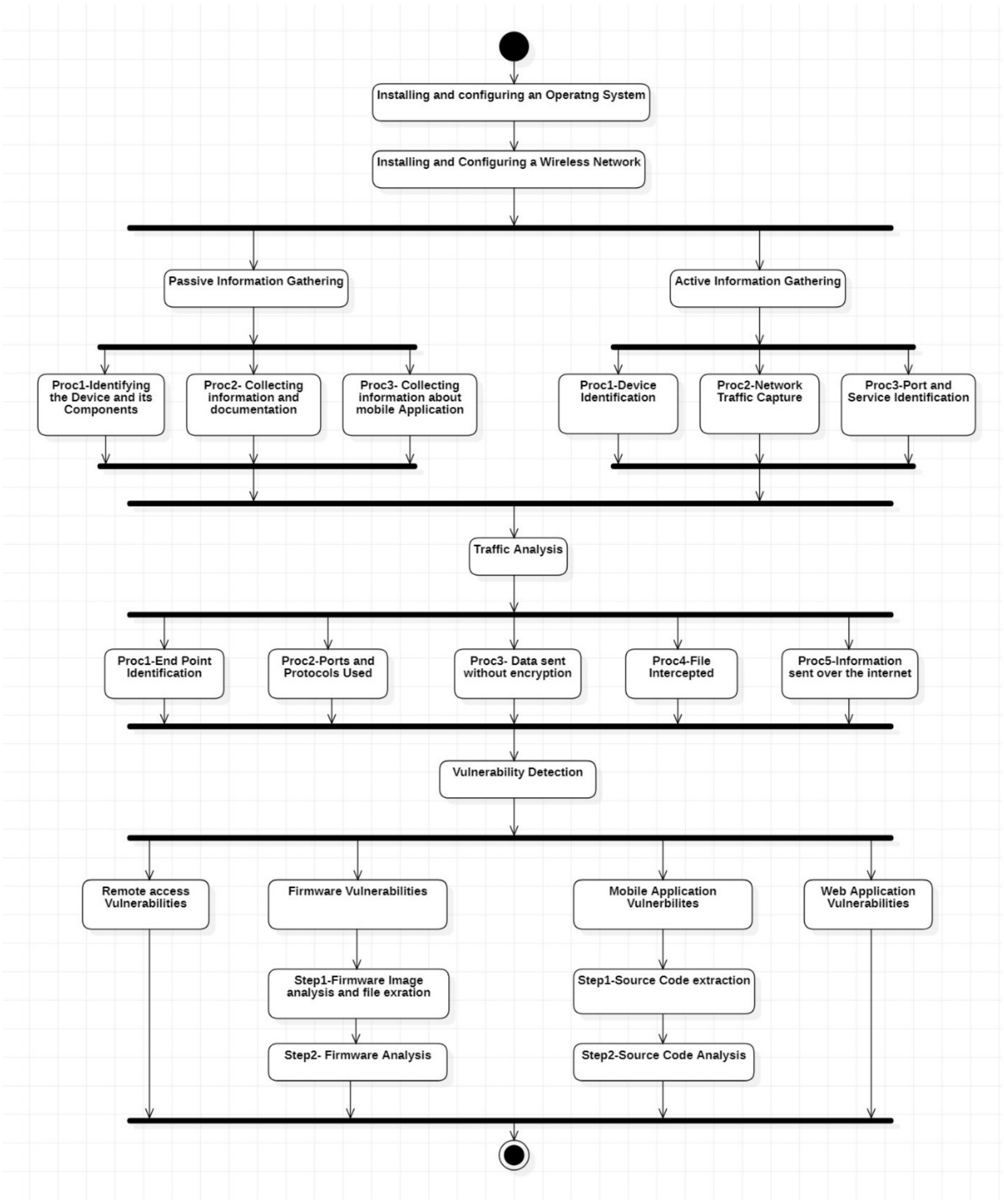

Installing and configuring an Operatng System
Installing and Configuring a Wireless Network
Passive Information Gathering
Active Information Gathering
Proc1-Identifying the Device and its Components
Proc2- Collecting information and documentation
Proc3- Collecting information about mobile Application
Proc1-Device Identification
Proc2-Network Traffic Capture
Proc3-Port and Service Identification
Traffic Analysis
Proc1-End Point Identification
Proc2-Ports and Protocols Used
Proc3- Data sent without encryption
Proc4-File Intercepted
Proc5-Information sent over the internet
Vulnerability Detection
Remote access Vulnerabilities
Firmware Vulnerabilities
Mobile Application Vulnerbilites
Web Application Vulnerabilities
Step1-Firmware Image analysis and file exration
Step2- Firmware Analysis
Step1-Source Code extraction
Step2-Source Code Analysis

Figure 2. UML activity diagram of the manual penetration testing method. Note: The procedures that are at the same level can be executed in parallel, but one can only proceed to the next level if all the procedures in that level are completed.

3. Proposed Framework

3.1. Design of the Framework

The idea behind the research is to make sure the framework developed can detect the most common vulnerabilities present in smart home-based IoT devices. For this, we have decided to go with a list of the top five most common vulnerabilities found in smart home-based IoT devices. The selection of these top five vulnerabilities has been done by taking the OWASP top 10 IoT vulnerabilities [16] along with the research paper [11], and the OWASP top 10 have been filtered to the top five by considering the scope of the research, i.e., only the Network-based, Web Interface based, and Firmware based, vulnerabilities have been selected. Therefore, based on these the top five vulnerabilities selected are given as follows:

1. **Insecure Web Interface**: IoT-based devices come with a corresponding application that handles the operation of the device. In the present era, all these applications are mobile-based, however, some devices still use a web application or have a web interface for handling their operation. For such devices, the framework has been designed to detect the presence of any web interface-based vulnerabilities, i.e., if the device has a web interface. The framework scans for any common web interface-based vulnerabilities, such as SQL injection, cross-site scripting, and many others, and reports its findings. This is done by using a tool called 'Owasp Zap'.
2. **Remote Access Vulnerability (Improper Authentication):** Some of the IoT devices can be accessed remotely from another system by using various remote access protocols. The most known remote access protocols for IoT devices are SSH and Telnet [11]. Any IoT device can be remotely accessed only if the ports corresponding to SSH and Telnet protocols are open, i.e., Port 22 (SSH) and Port 23 (Telnet), respectively. For such devices, the framework has been designed to detect any remote access vulnerabilities present by brute-forcing a password list on a user, mostly root, or a list of users. It scans for both SSH and Telnet-based remote access vulnerabilities. The tool used in the framework for doing this is 'Medusa'.
3. **Insecure Network Services:** Some IoT devices might be having unwanted services, unwanted ports open or services like FTP, Telnet or SSH, on open ports, which might be dangerous as attackers can use these ports to gain access to the device [17]. The framework is designed to detect any such insecure network services vulnerabilities by reading the captured pcap files and through the Nmap frameworks using the Nmap Search Engine (NSE).
4. **Lack of Transport Encryption:** Most IoT devices, even after several updates, still tend to use the HTTP protocol instead of the HTTPS protocol, which can be dangerous as most of the information transported over the internet in such a manner is usually unencrypted. An attacker can easily intercept the traffic and analyse an HTTP packet and retrieve all the sensitive information about the device [18]. The framework is designed to detect any such vulnerabilities by following a similar methodology. It detects if any of the sensitive information is transported in an unencrypted manner, i.e., using the HTTP protocol. This is done using the tool 'tshark'.
5. **Insecure Firmware/Software:** IoT devices need to be updated quite often as manufacturers keep fixing bugs in the software. Some of these devices, while going through an update, use the HTTP protocol to get the firmware files from the cloud. This can be vulnerable if someone intercepts the traffic; the firmware files can be captured easily as they would be unencrypted and according to OWASP top 10 IoT vulnerabilities, if the update file is transferred via HTTP or if it is unencrypted (human-readable), then it is an insecure firmware vulnerability [19]. The framework is designed to detect these types of vulnerabilities using 'tshark'.

The design of the framework, as shown in Figure 3, is based on a modified and automated version of the selected manual penetration technique described in the Literature Review section. The following is a brief UML activity diagram that describes the algorithm followed by the framework. It is based on the use case view of the architectural description mentioned in the papers [20,21].

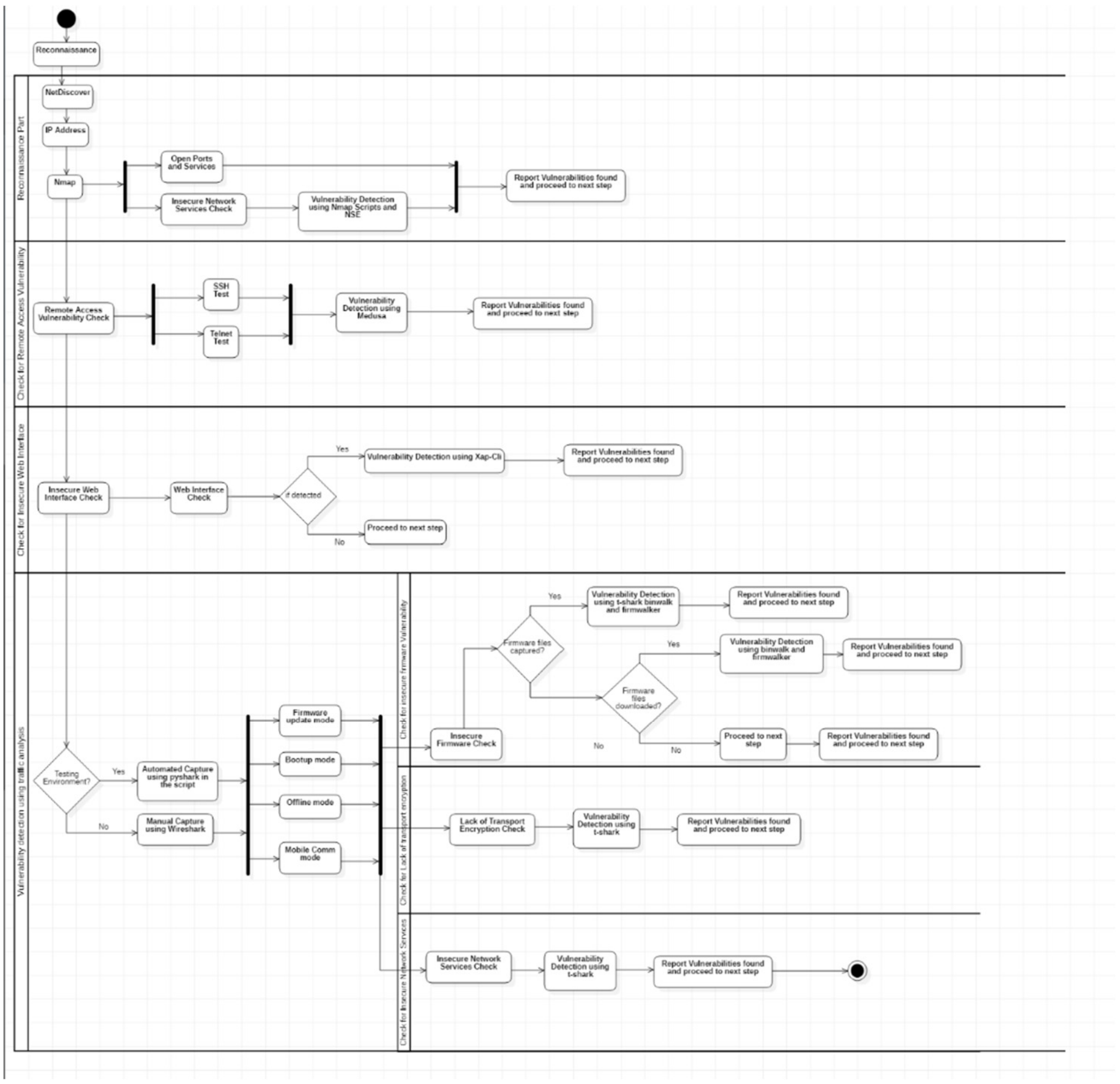

Reconnaissance
Reconnaissance Part
NetDiscover
IP Address
Nmap
Open Ports and Services
Insecure Network Services Check
Vulnerability Detection using Nmap Scripts and NSE
Report Vulnerabilities found and proceed to next step
Check for Remote Access Vulnerability
Remote Access Vulnerability Check
SSH Test
Telnet Test
Vulnerability Detection using Medusa
Report Vulnerabilities found and proceed to next step
Check for Insecure Web Interface
Insecure Web Interface Check
Web Interface Check
if detected
Yes
Vulnerability Detection using Xap-Cli
Report Vulnerabilities found and proceed to next step
No
Proceed to next step
Vulnerability detection using traffic analysis
Testing Environment?
Yes
Automated Capture using pyshark in the script
No
Manual Capture using Wireshark
Firmware update mode
Bootup mode
Offline mode
Mobile Comm mode
Check for insecure firmware Vulnerability
Insecure Firmware Check
Firmware files captured?
Yes
Vulnerability Detection using t-shark binwalk and firmwalker
Report Vulnerabilities found and proceed to next step
No
Firmware files downloaded?
Yes
Vulnerability Detection using binwalk and firmwalker
Report Vulnerabilities found and proceed to next step
No
Proceed to next step
Report Vulnerabilities found and proceed to next step
Check for Lack of transport encryption
Lack of Transport Encryption Check
Vulnerability Detection using t-shark
Report Vulnerabilities found and proceed to next step
Check for Insecure Network Services
Insecure Network Services Check
Vulnerability Detection using t-shark
Report Vulnerabilities found and proceed to next step

Figure 3. UML activity diagram for the algorithm followed by the framework.

Based on the Figure, the design of the framework is such that it can be divided into five parts, which are executed one after the other. This division of the framework into different parts is done to make the explanation of the algorithm easier.

The different parts are as follows:

1. **Reconnaissance part (Net Discover and Nmap Scan):** Initially, the framework does a Net Discover scan over an interface, usually the one to which the IoT device is connected, and retrieves the IP Address and the MAC Vendor Name (device manufacturer name) of the device. After that, a Nmap scan is done by the framework over the IP Address that has been retrieved through the Net Discover scan, and the results of this scan, which contains the list of open ports and the corresponding services running on them, are stored in a corresponding Nmap log file. The framework reports these results and proceeds to the next part.

2. **Check for Remote Access Vulnerability:** In this part, the framework uses Medusa to conduct a brute force password attack over the IP Address of the device, retrieved via the 'Net Discover' part, to check whether any remote access vulnerabilities are present. It takes the user to run the check on, the password list, and the remote access protocol to check as an input, and executes the 'Medusa' tool with these corresponding flags to detect if the device can remotely be accessed using any of these passwords in the given password list. The input can vary between one user and a list of users, and the protocols it checks on are 'SSH' and 'Telnet', which are the two most common protocols used by IoT devices. The framework stores the output of the 'Medusa' tool executed with the provided input flags in a corresponding log file; this log file is scanned to check whether any of the passwords in the list could successfully brute force and provide access. If yes, a remote access vulnerability is recorded and reported and the framework proceeds to the next part.
3. **Check for Insecure Web Interface:** This part can be divided into two steps.
   i. **Step 1:** The framework first checks whether the device has a corresponding web interface. For this, it uses the 'whatweb' tool, which checks whether HTTP is running over the device and reports its status, since HTTP is required for any device to have a web interface. If 'whatweb' detects HTTP running over the device, then it proceeds to the next step, or else the framework checks whether port 80 (HTTP) or port 443 (HTTPS) are open, and asks the user manually if he wishes to check for an insecure web interface. This step has been added for the cases where the 'whatweb' tool cannot detect 'HTTP', or when the devices use 'HTTPS', or when these protocols run over different ports, among others. If the framework detects the presence of a web interface it proceeds to the next step, else it reports that the device does not have a corresponding web interface.
   ii. **Step 2:** If the framework detects the presence of a web interface, this step gets executed, or else the framework moves to part four. In this step, the framework uses the tool Zap-Cli with corresponding flags as input, and actively scans the web interface for all the vulnerabilities like SQL Injection, Cross-Site Scripting, Cross-Site Request Forgery, and many others. The results of this scan are stored in a corresponding log file and this log file is then string processed to check for any vulnerabilities detected by Zap-Cli. If the framework detects issues in the log file, then an insecure web interface vulnerability is recorded and reported and the framework proceeds to the next part.
4. **Automated Traffic Capture:** In this part, initially, the framework asks the user whether he wishes to capture the traffic automatically using the framework or manually capture using Wireshark. This has been added as sometimes automated capture cannot guarantee accurate results if the testing environment is not rightly configured. This provides the user flexibility to capture the files manually or automatically, depending on his will. This part of the framework gets executed only if the user selects 'Automatically'. Otherwise, the framework proceeds to the next part.

   Once the automatic option is selected, the framework asks the user for the state of the device, there are four possible states which are: Booting device, when the device boots up; Mobile Application Interaction, when the device interacts with its corresponding mobile application; Firmware mode, when the device goes through a firmware update; and Offline Mode when the device does not have an internet connection. The framework takes these states as input to capture the traffic in four different states and stores it in four different .pcap files (for analysing later). The capture is done using 'pyshark'. The framework prompts the user to capture in all the states to ensure completeness of the capture. Once the capture is complete the framework proceeds to the next part.

   Note: the user needs to make sure manually that the device is in these states while selecting that corresponding state as input in the framework.

5. **Vulnerability detection through traffic analysis:** After the capture is complete, either automatically through the framework or manually, the framework then analyses these captured .pcap files and checks for vulnerabilities present in them. To make the explanation easier, this part has further been divided into three parts, which are executed one after the other.

    i. **Check for Insecure firmware:** Initially, the framework asks the user whether the firmware files have been captured during a firmware update or downloaded from the manufacturer's website. If neither is true, then the framework proceeds to the next step.

       If the firmware files have been captured:

       The framework uses 'tshark' to read the .pcap file, captured during the firmware update, to fetch the URL link to the binary image file of the firmware (which is on the cloud interface of the device). Once, it gets access to the URL link, the framework automatically downloads the binary image file (.bin file) and stores it in the same directory on the user's system. The framework then extracts the firmware file system from the .bin file using the 'binwalk' tool. After that, the extracted firmware file system is searched using the 'firmwalker' tool for any firmware-based vulnerabilities. The framework records and reports an 'insecure firmware vulnerability' once it finds the URI link in 'HTTP', or once the download is successful, irrespective of whether the extraction of the firmware file system through binwalk or the firmwalker search was successful or not [19]. Post the 'firmwalker' scan, and once the check is complete, the framework proceeds to the next part.

       If the firmware files have been downloaded from the manufacturer's website:

       The framework extracts the firmware file system from the .bin file using the 'binwalk' tool. After that, the extracted firmware file system is searched by the framework using the 'firmwalker' tool for any firmware-based vulnerabilities. If the firmwalker tool finds any firmware-based vulnerabilities, then the framework records and reports them, or else the framework proceeds to the next part.

    ii. **Check for Lack of transport encryption:** The framework uses 'tshark' to read the pcap file, captured during the firmware update, to fetch the URL link, through which the device communicates with its corresponding cloud interface; generally, this is like the .bin file HTTP link, as above. Once it gets access to the URL link, the framework checks whether the device uses HTTP to communicate with the cloud interface to download the firmware files [18]. If the device uses HTTP, then 'lack of transport encryption vulnerability' is recorded and reported, or else the framework proceeds to the next part.

    iii. **Check for Insecure Network Services:** The framework uses 'tshark' to read all the pcap files, captured in all the different modes, and to check for any insecure network-based vulnerabilities. This is done by checking the following:

       - Any unwanted or unknown services are on open ports.
       - If services like SSH, Telnet, and FTP, are on open ports.
       - If any of the protocols like TCP, SSL, or TLS, have vulnerable older versions.
       - If the services used by the device are not properly authenticated by the server.

       If the framework detects any of the above, then the 'Insecure Network Services' vulnerability is recorded and reported. If not, the framework proceeds to the next part.

Post this, an additional part is added to the framework to ensure a more comprehensive check of the Insecure Network Services. In this part, the framework uses Nmap along with some Nmap search engine (NSE) frameworks to detect any network-based vulnerabilities. The framework stores the output of the network-based nmap scan in a corresponding log file and scans this log file for any vulnerabilities detected by Nmap. If the framework finds any issues or vulnerabilities, it records and reports them, else with this the design of the device is complete.

In addition to the UML activity diagram, the following are the UML component, shown in Figure 4, and UML deployment diagrams of the framework, shown in Figure 5, added to gain a broader perspective and understand the architectural description of the design of the framework, based on [20,21].

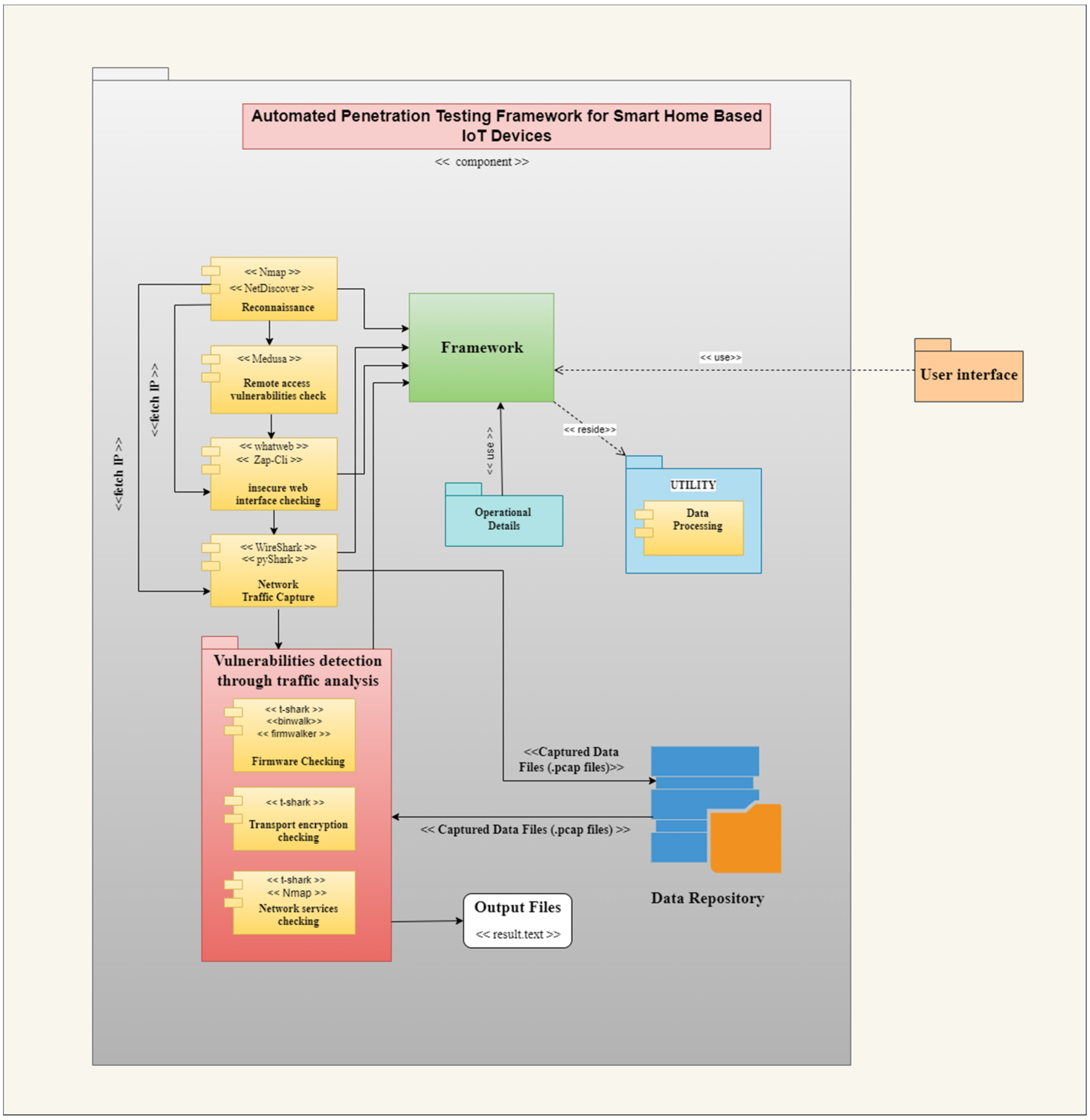


Figure 4. UML component diagram of the framework.

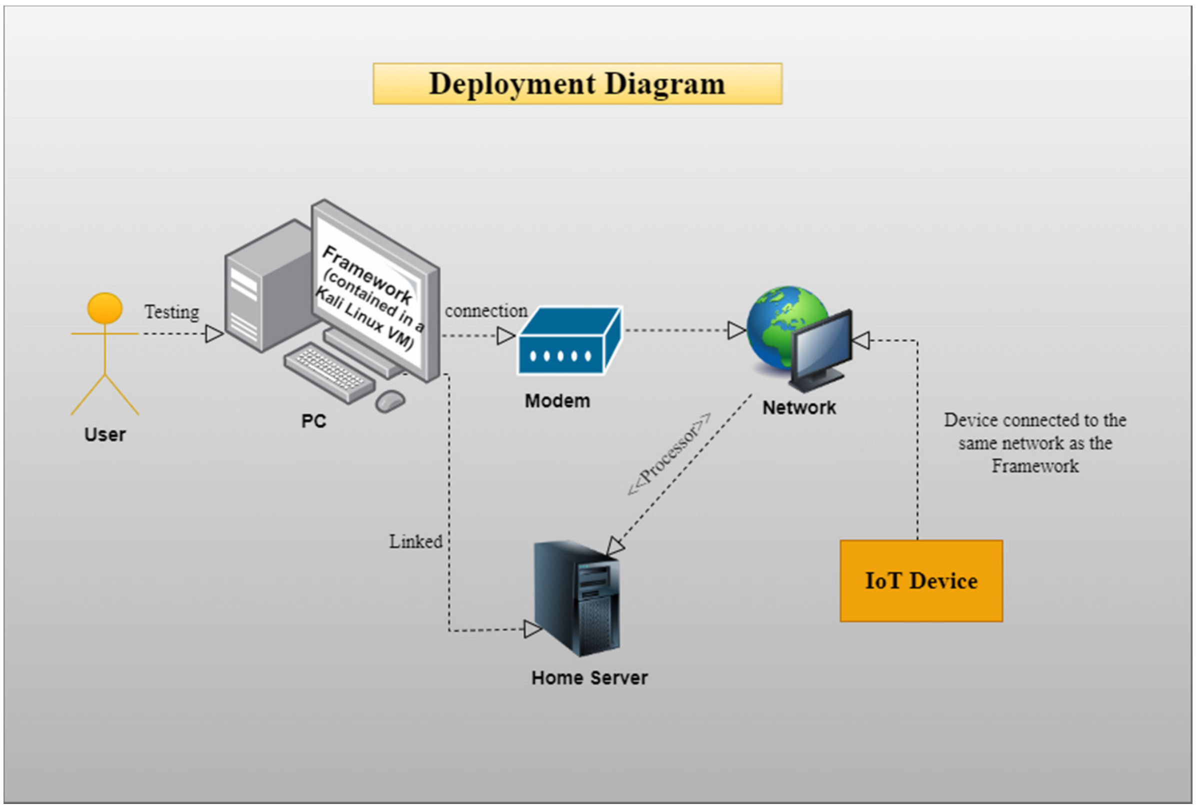

Deployment Diagram
User
Testing
Framework (contained in a Kali Linux VM)
PC
connection
Modem
Network
<<Processor>>
Linked
Home Server
Device connected to the same network as the Framework
IoT Device

Figure 5. UML deployment diagram of the framework.

3.2. Implementation

The implementation of the automated PT framework is carried out based on the algorithm mentioned above in the design section. We used the algorithm to write a framework in Python 3.6. This framework contains a combination of tools executed in a specific order, at the user's discretion. The framework uses these tools to detect the most common vulnerabilities in the IoT device (mentioned in the Design section). To be specific, each vulnerability has a corresponding method, a tool or combination of tools, that is used by the framework to detect that specific vulnerability. Most of the tools used by the framework are available in Kali Linux, by default. The tools that are used by the framework are Net Discover, Nmap, OWASP ZAP (zap-cli), Medusa, WhatWeb, Wireshark (t-shark and pyshark), Binwalk, and Firmwalker.

We used the implemented framework to test the security of five smart home-based IoT devices: Tp-link smart plug; Tp-link smart bulb; Tp-link smart camera; Google home mini; and the LIFX Smart bulb, to find the most common vulnerabilities present in them. In addition, we calculated the Common Vulnerability Scoring System (CVSS) Base Score for each device individually, based on the vulnerabilities detected by the framework, to identify the most secure device (mentioned in the Results section).

We did the testing on the Kali Linux operating system, which is contained in a Virtual Machine (VM) on the Windows Operating system (host). Ideally, the VM should have a base memory of 4096MB and four processors at least, to ensure that the framework does not slow down. Additionally, the network adapter of the VM should be of the Network Address Translation type (NAT), so that the host, machine, and VM, are on the same network. After that, a Wi-Fi-based hotspot needs to be established on the Windows OS with a network band of 2.4 GHz as IoT devices fail to connect to a network band that is higher than this. The device that is being tested should be set up accordingly, based on the setup instructions provided for each device, and connected to this Wi-Fi hotspot; this is done by selecting the network of the Wi-Fi hotspot as the network to connect the device to, on the corresponding mobile application of the device. Once the device connects to this network, the device, the VM, and the host machine are all on the same network as the host machine and VM are connected through a NAT adapter.

4. Results and Analysis

Before the testing, the following 'pip3' based packages and tools need to be installed for the framework to work. For testing environment Kali Linux, here are some prerequisites:

- Pip3-based packages 'pyshark' and 'scapy' need to be installed using the commands "pip3 install pyshark" and "pip3 install scapy", respectively, on the terminal.
- Zap-Cli needs to be installed using the command "pip3 install –upgrade zap-cli" on the terminal, and the $ZAP_PATH and $ZAP_PORT environment variables need to be set [22].
- Firmwalker needs to be installed in the same directory in which the folder containing the script is present. It can be installed using the command "git clone https://github.com/craigz28/firmwalker.git" [23].

After configuring the testing environment, we connected each device to the Wi-Fi hotspot individually and used the PT framework on the devices to detect the vulnerabilities present in them. A summary of the results of the testing for each device, along with the execution time taken by the framework for testing it, is shown in Table 1.

Table 1. Results of testing.

Test Case	Description	Device Tested	Vulnerabilities Detected by the Framework	Execution Time for Testing by the Framework
1	Used the framework to test for the top vulnerabilities in the Smart plug.	Tp-Link Smart Plug	A probable Insecure Network Services vulnerability.	2–3 s (approximately)
2	Used the framework to test for the top vulnerabilities in the Smart bulb.	Tp-Link Smart bulb	Lack of Transport Encryption and Insecure Firmware Vulnerability.	5–8 s (approximately)
3	Used the framework to test for the top vulnerabilities in the Smart Camera.	Tp-Link Smart Camera	Lack of Transport Encryption and Insecure Firmware Vulnerability.	5–8 s (approximately)
4	Used the framework to test for the top vulnerabilities in the Google Home Mini.	Google Home Mini	No Vulnerabilities were detected.	1–2 s (approximately)
5	Used the framework to test for the top vulnerabilities in the LIFX Smart bulb.	LIFX Smart bulb	No Vulnerabilities were detected.	1–2 s (approximately)

Note: The execution time is an approximate range as it was difficult to calculate the actual time, due to several manual inputs to execute the framework. However, we can confirm that the time taken by the framework was minimum and at a generally acceptable range.

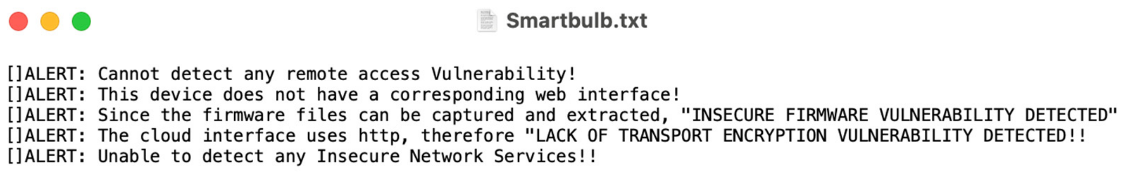
For the smart plug, the framework was able to detect that the DNS server was not authenticated for requests and responses which could be vulnerable. This is a probable Insecure Network Services vulnerability; therefore, it has been noted.

For the Tp-link smart bulb and smart camera, by taking the firmware update pcap file as the input, the framework was able to find the HTTP request URI link that the device uses to fetch the binary image firmware file from its corresponding cloud interface. The link to the bin file was as follows: http://download.tplinkcloud.com/firmware/smartBulb_FCC_1.8.6_Build_180809_Rel.091659_2_1536546605436.bin.

Once, the bin file was found, the framework was able to download the file from its cloud interface and store it in the same directory as the framework. The firmware file system contained in this bin file was not recognised by the 'binwalk' tool, so the framework could not completely extract it from the downloaded bin file. As the extraction of the firmware file system was not properly executed, the framework could not detect any firmware-based vulnerabilities using the 'firmwalker' tool. However, since the update file was unencrypted and was transmitted via the HTTP protocol, this device has an Insecure Firmware Vulnerability. Similarly, by taking the same firmware update pcap file as input, the framework was able to find the HTTP link to the cloud interface (i.e., http://download.tplinkcloud.com/). Therefore, as the device communicates with its corresponding cloud interface through HTTP, the framework detected a Lack of Transport Encryption Vulnerability in the device.

The framework could not detect any vulnerabilities in the Google Home Mini and LIFX Smart bulb devices.

An example of the output results ('results.txt') file when the Tp-link smart bulb was tested by the framework is shown in Figure 6 (given below).


Smartbulb.txt

```
[]ALERT: Cannot detect any remote access Vulnerability!
[]ALERT: This device does not have a corresponding web interface!
[]ALERT: Since the firmware files can be captured and extracted, "INSECURE FIRMWARE VULNERABILITY DETECTED"
[]ALERT: The cloud interface uses http, therefore "LACK OF TRANSPORT ENCRYPTION VULNERABILITY DETECTED!!
[]ALERT: Unable to detect any Insecure Network Services!!
```

Figure 6. Security testing results for the Tp-link Smart bulb.

Based on the vulnerabilities detected, we calculated the Common Vulnerability Scoring System (CVSS) score for each device. The scope of this research work is mainly focused on the base metrics and impact metrics, i.e., the base score for determining the severity of the vulnerability [24]. A summation of the base scores for each vulnerability in the platform (IoT device) is taken and the total score is calculated for each platform. Based on the total scores, the platform with the highest score would be the least secure. The results of the Total Common Vulnerability Scoring System (CVSS) base scores evaluated for each device are shown in Table 2.

Table 2. Total CVSS scores for each device.

Device	Total Score
Tp-Link Smart Plug	N/A
Tp-Link Smart Bulb	14.4
Tp-Link Smart Camera	14.4
Google Home Mini	0
LIFX Smart Bulb	0

For the Tp-link smart plug, even though the vulnerability detected for this device was a probable Network-based vulnerability, it was not the kind of vulnerability listed in [16] and, therefore, the base metrics and impact metrics for such a vulnerability would be uncertain. So, for this device, the Common Vulnerability Scoring System (CVSS) Base score could not be calculated.

For the Tp-link Smart bulb and the smart camera, the base scores for the Insecure Firmware and lack of transport encryption vulnerabilities were 6.5 and 7.9, respectively. Therefore, the total score for both devices was the same, i.e., 14.4, as the vulnerabilities detected for these devices were also the same. The scores of the Google Home Mini and the LIFX Smart bulb were '0' as there were no vulnerabilities found in these devices. However, after further research, we found that although no vulnerabilities were found in both these devices, the Google Home Mini has an additional security feature in which the user is prompted to connect both the device and the corresponding mobile handling the device to the same network. Whereas, in the case of the LIFX Smart Bulb, an external user can connect to the device through a different Wi-Fi network, which could be vulnerable. Additionally, the Google Home Mini uses the latest versions of its protocol, e.g., TLS v1.3, in comparison with the LIFX Smart Bulb, e.g., TLS v1.2.

5. Discussion

When compared with testing the devices manually for each vulnerability, this method can save us a lot of time and effort. Any user having no, or limited PT experience, can also use the framework to test the devices at their home; all they need to do is execute the framework and follow the instructions provided. For someone with thorough PT experience, it provides a scope to improvise as the user can use different inputs to get better results. It makes the life of a user easy by giving an insight into the security loopholes of the device so that the user can protect himself from any cyber-criminal attacks. It makes the life of a user easy by giving an insight into the security loopholes of the device so, that the user can protect himself from any cyber-criminal attacks. It also has minimal processing and execution time, in comparison to using the same techniques manually.

There is also a major advantage in using software patterns for the security problems of IoT devices. These software security patterns can solve IoT security problems [25]. It is also important to note that integrating software patterns like blockchain technology in the environment of IoT can help in ensuring trust amongst IoT devices, and thus increase their level of security [26]. This aspect can be extended to be used for our framework, i.e., considered for future work [25,26]. In addition, a simulated cyber-attack could also be a good way to test the security of the system [27].

5.1. Scope

Given below is a list of points that cover the scope of this research work:

- The framework can run on other Unix platforms (apart from Kali Linux) provided that Python 3.6, the tools mentioned in the Design chapter, and the python-pip packages like ‘scapy’, and ‘pyshark’ are installed.
- This research only deals with the top five list of vulnerabilities mentioned in the Design subsection of the Proposed Framework section. It does not involve detecting cloud, mobile application, hardware, and physical device vulnerabilities.
- Although the Common Vulnerability Scoring System (CVSS) score depends on several metrics, this research mainly focuses on the base metrics and impact metrics, i.e., the base score for determining the severity of the vulnerability.
- Currently, the framework can only test the devices which are accessible over the network and, preferably, on the same network.
- Currently, this research does not involve providing a fix for the vulnerabilities found, it only involves discovering them.

5.2. Risks and Constraints

The following are possible risks and constraints that could arise during the research work:

- Each device has its configuration process, so there is a risk of not being able to configure them properly for testing. Therefore, there is a possibility that devices may not be successfully configured, or misconfigured.
- It might be challenging to divert the network traffic from the IoT device to Kali Linux as its gateway. i.e., configuring a hotspot on Kali Linux.
- The automated network traffic capture of the framework might not work properly in case the testing environment is not properly configured, i.e., the traffic from the device does not have the testing OS Kali Linux as its gateway.

5.3. Contingency Planning

In case there is a risk of misconfiguration of the device, initially, we would try to reset the device and configure it once again taking necessary precautions. If that still does not work, then as a backup plan five devices are selected for the research and the idea is to test at least three of them successfully. So, even if one or two devices do not work properly, then there is a high probability that the others would.

In case configuring a hotspot on the Kali Linux does not work, our other idea was to use ARP spoofing where we can do a man-in-the-middle attack between the IoT device and the gateway to capture the traffic. However, after a bit of research, we realised that it would not be ideal to do this as some devices, such as the 'Google Home Mini', are secure and do not support ARP spoofing. Another idea we had was to misconfigure a wireless router (by changing its configuration settings), make it an access point, and have the Kali VM as the gateway so that all the traffic from the router goes to Kali. However, it was not practically feasible to do this.

So, finally, we decided to add the option of manual or automated capture in the framework and manually capture the traffic by hosting a Wi-Fi hotspot on windows and connecting the device to it. The captured traffic was stored into .pcap files by using 'Wireshark' on windows. These 'pcap' files would be provided as input to the framework to detect vulnerabilities in them. In this way, although the automated capture part of the framework works, the user needs to make sure that the device and the testing environment (Kali Linux VM) are on the same network.

5.4. Limitations

The framework is designed to detect only the top five vulnerabilities in IoT devices. This aspect can be extended, and the framework can be made to detect all the OWASP top IoT vulnerabilities with more tools, as required. Secondly, the framework can be configured in such a way that the automated capture part works even if the framework runs on a different network (w.r.t the device). This can be done by adding an extensive ARP Spoofing part to the framework. Thirdly, the Common Vulnerability Scoring System (CVSS) Base score calculation part can also be automated by adding it to the end of the framework. Fourthly, the framework can be extended to test non-Wi-Fi-based IoT devices, i.e., the IoT devices that are not accessible over the network. Additionally, the scope of the project can be broadened, and the research can be extended to provide a possible fix for the detected vulnerabilities. Moreover, the Insecure Network Services part of the framework can be improved by ensuring a more automated and extensive check of all the aspects of a captured '.pcap' file. In addition, a more extensive user input error handling can be added to some parts of the framework to ensure that the framework is free from any buffer overflow, or command-line injection errors.

In terms of future work, in this project, the framework is designed to detect only the top five vulnerabilities in IoT devices. This aspect can be extended, and the framework can be made to detect all the OWASP top IoT vulnerabilities with more tools, as required. Secondly, the framework can be configured in such a way that the automated capture part works even if the framework runs on a different network (w.r.t the device). This can be done by adding an extensive ARP Spoofing part to the framework. Thirdly, the CVSS Base score calculation part can also be automated by adding it to the end of the framework. Additionally, the scope of the project can be broadened, and the research can be extended to provide a possible fix for the detected vulnerabilities.

6. Conclusions

In this paper, we initially introduced the concept of Internet of Things, Smart homes, and Smart home-based IoT devices. We then discussed the importance of IoT security, why it is necessary, and how PT is the best possible solution to evaluate the security of IoT devices. After that, we discussed the current IoT security research, the current research on PT of IoT devices, and how the current PT research is shifting towards automation. We reviewed different PT techniques, spoke about their limitations, and selected a manual penetration technique that is most suitable, according to the scope of the research work. Then, we proposed an automated PT framework for smart home-based IoT devices, which is an automated version of the selected technique. We tested five different smart home-based IoT devices using this framework to discover the most common vulnerabilities present in them. Then, we calculated the Total Common Vulnerability Scoring System (CVSS) Base scores for the vulnerabilities found in each device and found that the Tp-Link Smart bulb and the Tp-Link Indoor Camera both had a high score of 14.4, as they had two vulnerabilities, which makes them comparatively less secure than the rest of the devices. The Tp-Link Smart bulb is next in line, i.e., comparatively more secure, as there was only one probable Insecure Network Services Vulnerability found in it. So, from the results obtained, we initially deducted that the Google Home Mini and the LIFX Mini Smart Bulb, with a CVSS total score of '0', were the most secure devices in comparison with the others. However, after further research (mentioned in the Results section), we found that the Google Home Mini had some additional security features in comparison to the LIFX Smart Bulb, therefore making it the most secure device of the five devices.

Finally, there is no denying the fact that we need IoT devices to make our lives easier, but ensuring their security is very much equally important and for this purpose having an automated testing framework of this kind will certainly help in improving the current state of the field of IoT security. It is a progressive step into a safe, secure, and brighter future.

Author Contributions: Conceptualization, R.A. and O.B.; methodology, R.A. and O.B.; software, R.A.; validation, R.A. and O.B.; writing—original draft preparation, R.A. and M.J.M.C.; writing—review and editing, R.A. and M.J.M.C.; supervision, M.J.M.C. and N.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Acknowledgments: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Source Code: Ref. [28] contains the link to the source code repository of the framework.

References

1. Sachidananda, V.; Toh, J.; Siboni, S.; Bhairav, S.; Shabtai, A.; Elovici, Y. 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, co-located with ASIA CCS 2017, New York, NY, USA, 2 April 2017; pp. 3–10. [CrossRef]
2. Visoottiviseth, V.; Akarasiriwong, P.; Chaiyasart, S.; Chotivatunyu, S. PENTOS: Penetration Testing Tool for Internet of Thing Devices. In Proceedings of the IEEE Region 10 Annual International Conference, Proceedings/TENCON, Penang, Malaysia, 5–8 November 2017; pp. 2279–2284. [CrossRef]
3. Papatsimouli, M.; Lazaridis, L.; Ziouzios, D.; Dasygenis, M.; Fragulis, G. Internet Of Things (IoT) Awareness in Greece. *SHS Web Conf.* **2022**, *139*, 3013. [CrossRef]
4. Patton, M.; Gross, E.; Chinn, R.; Forbis, S.; Walker, L.; Chen, H. Uninvited Connections: A Study of Vulnerable Devices on the Internet of Things (IoT). In Proceedings of the 2014 IEEE Joint Intelligence and Security Informatics Conference, JISIC 2014, The Hague, The Netherlands, 24–26 September 2014; pp. 232–235. [CrossRef]
5. Zhang, Z.K.; Cho, M.C.Y.; Wang, C.W.; Hsu, C.W.; Chen, C.K.; Shieh, S. IoT Security: Ongoing Challenges and Research Opportunities. In Proceedings of the IEEE 7th International Conference on Service-Oriented Computing and Applications, SOCA 2014, Matsue, Japan, 17–19 November 2014; pp. 230–234. [CrossRef]
6. Duggan, D.P. *Penetration Testing of Industrial Control Systems*; Sandia National Laboratories: Albuquerque, New Mexico; Livermore, CA, USA, 2005.
7. Lee, J.; Elovici, Y.; Shabtai, A.; Tippenhauer, N.O.; Siboni, S. Advanced Security Testbed Framework for Wearable IoT Devices. *ACM Trans. Internet Technol.* **2016**, *16*, 1–25. [CrossRef]
8. Bing, K.; Fu, L.; Zhuo, Y.; Yanlei, L. Design of an Internet of Things-Based Smart Home System. In Proceedings of the 2nd International Conference on Intelligent Control and Information Processing, ICICIP 2011, Harbin, China, 25–28 July 2011; Volume 2, pp. 921–924. [CrossRef]

9. Ghaffarianhoseini, A.; Ghaffarianhoseini, A.; Tookey, J.; Omrany, H.; Fleury, A.; Naismith, N.; Ghaffarianhoseini, M. The Essence of Smart Homes: Application of Intelligent Technologies towards Smarter Urban Future. *Artif. Intell. Concepts Methodol. Tools Appl.* **2016**, *1*, 79–121. [CrossRef]
10. Yu, M.; Zhuge, J.; Cao, M.; Shi, Z.; Jiang, L. A Survey of Security Vulnerability Analysis, Discovery, Detection, and Mitigation on IoT Devices. *Future Internet* **2020**, *12*, 27. [CrossRef]
11. Costa, L.; Barros, J.P.; Tavares, M. Vulnerabilities in IoT Devices for Smart Home Environment. In Proceedings of the 5th International Conference on Information Systems Security and Privacy–ICISSP 2019, Prague, Czech Republic, 23–25 February 2019; pp. 615–622. [CrossRef]
12. Myridakis, D.; Spathoulas, G.; Kakarountas, A.; Schinianakis, D. Smart Devices Security Enhancement via Power Supply Monitoring. *Future Internet* **2020**, *12*, 48. [CrossRef]
13. Chu, G.; Lisitsa, A. Penetration Testing for Internet of Things and Its Automation. In Proceedings of the 20th International Conference on High Performance Computing and Communications, 16th International Conference on Smart City and 4th International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2018, Exeter, UK, 28–30 June 2018; pp. 1479–1484. [CrossRef]
14. Rak, M.; Salzillo, G.; Granata, D. ESSecA: An Automated Expert System for Threat Modelling and Penetration Testing for IoT Ecosystems. *Comput. Electr. Eng.* **2022**, *99*, 107721. [CrossRef]
15. Chen, C.K.; Zhang, Z.K.; Lee, S.H.; Shieh, S. Penetration Testing in the IoT Age. *Computer (Long Beach Calif)* **2018**, *51*, 82–85. [CrossRef]
16. OWASP Internet of Things Project–OWASP. Available online: https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Top_10 (accessed on 29 July 2022).
17. Smith, C. Top 10 2014-I3 Insecure Network Services–OWASP. Available online: https://wiki.owasp.org/index.php/Top_10_2014-I3_Insecure_Network_Services (accessed on 29 July 2022).
18. Smith, C. Top 10 2014-I4 Lack of Transport Encryption–OWASP. Available online: https://wiki.owasp.org/index.php/Top_10_2014-I4_Lack_of_Transport_Encryption (accessed on 29 July 2022).
19. Smith, C. Top 10 2014-I9 Insecure Software/Firmware–OWASP. Available online: https://wiki.owasp.org/index.php/Top_10_2014-I9_Insecure_Software/Firmware (accessed on 29 July 2022).
20. Kruchten, P.B. The 4+1 View Model of Architecture. *IEEE Softw* **1995**, *12*, 42–50. [CrossRef]
21. Górski, T. The 1+5 Architectural Views Model in Designing Blockchain and IT System Integration Solutions. *Symmetry* **2021**, *13*, 2000. [CrossRef]
22. Grunwell, D. GitHub–Grunny/Zap-Cli: A Simple Tool for Interacting with OWASP ZAP from the Commandline. Available online: https://github.com/Grunny/zap-cli (accessed on 20 September 2019).
23. Smith, C. GitHub–Craigz28/Firmwalker: Script for Searching the Extracted Firmware File System for Goodies! Available online: https://github.com/craigz28/firmwalker (accessed on 20 September 2019).
24. CVSS v3.1 Specification Document. Available online: https://www.first.org/cvss/v3.1/specification-document (accessed on 29 July 2022).
25. Fernandez, E.B.; Washizaki, H.; Yoshioka, N.; Okubo, T. The Design of Secure IoT Applications Using Patterns: State of the Art and Directions for Research. *Internet Things* **2021**, *15*, 100408. [CrossRef]
26. Kumar, R.; Sharma, R. Leveraging Blockchain for Ensuring Trust in IoT: A Survey. *J. King Saud Univ.–Comput. Inf. Sci.* **2021**. [CrossRef]
27. Adhikari, U.; Morris, T.; Pan, S. WAMS Cyber-Physical Test Bed for Power System, Cybersecurity Study, and Data Mining. *IEEE Trans. Smart Grid.* **2017**, *8*, 2744–2753. [CrossRef]
28. GitHub - Rocky9624/Automated-Penetration-Testing-Framework. Available online: https://github.com/rocky9624/Automated-Penetration-testing-Framework (accessed on 18 September 2022).