IoTective: Automated Penetration Testing for Smart Home Environments

Kevin Nordnes[1], Jia-Chun Lin[2][a], Ming-Chang Lee[2][b], and Victor Chang[3][c]

[1] *Mnemonic AS, Oslo, Norway*

[2] *Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU), Gjøvik, Norway*

[3] *Department of Operations and Information Management, Aston Business School, Aston University, Birmingham, UK*

[1] *kevinnordnes@proton.me,* [2] *jia-chun.lin@ntnu.no,* [2] *mingchang1109@gmail.com,* [3] *v.chang1@aston.ac.uk*

Keywords: IoT, Smart Homes, Penetration Testing, Security Testing, Vulnerability Identification, Network Scanning

Abstract: As the prevalence of Internet of things (IoT) continues to increase, there is a corresponding escalation in security concerns. Given that many IoT devices lack robust security features, the need for specialized security testing tools has become evident. In this paper, we introduce an open-source automated penetration testing tool named IoTective for smart home environments in response to the increasing security concerns surrounding IoT devices. IoTective aims to discover devices in Wi-Fi, Bluetooth, and Zigbee networks, identify vulnerabilities, and gather valuable information for further analysis. IoTective streamlines the initial stages of reconnaissance, planning, and scanning, which provides a good support for a variety of devices and protocols common used in smart home environments. With a focus on ease of use and flexibility, the tool provides an intuitive user interface and customizable scanning capabilities. We evaluated the effectiveness of IoTective and explored the impact on overall security posture. Ethical considerations for automated penetration testing are also discussed.

1 INTRODUCTION

The Internet of Things (IoT) has evolved and expanded over the years, driven by technological advances such as the development of wireless technologies like Wi-Fi, Bluetooth, and Zigbee. In recent years, IoT applications such as smart home automation, Industrial Internet, and connected cars have entered the market. The growing accessibility of cloud computing and big data analytics has further driven the expansion of IoT, making it applicable across diverse domains, including agriculture, manufacturing, healthcare, and transportation (Khanna and Kaur, 2019)(Tekeste Habte et al., 2019)(Zantalis et al., 2019).

With the escalating prevalence of IoT devices, security and privacy concerns have grown (Alrawais et al., 2017)(Lee et al., 2019). The surge in IoT devices, often lacking security considerations, raises substantial security issues, especially when these devices access sensitive information on home networks. Penetration testing is a cybersecurity method to evaluate the security of a computer system. It involves identifying vulnerabilities within the system and attempting to exploit these vulnerabilities to gain unauthorized access (Shah and Mehtre, 2015). While several cybersecurity-related tools exist such as Nmap[1], Wireshark[2], and Metasploit[3], there is still a growing demand for more specialized tools capable of automated security testing in home environments. This demand arises from the diverse range of devices and heterogeneous network protocols used in smart homes. Existing penetration testing tools do not adequately address the unique challenges posed by IoT devices, necessitating the development of specialized tools.

The paper aims to contribute to this space by creating an open-source automated penetration testing tool called IoTective, which is tailored for testing the security of IoT devices in smart home environments. The paper addresses several research questions, including the identification of common vulnerabilities in smart home environments and the effectiveness of automated penetration testing compared to manual testing. Furthermore, it discusses the impact on overall security posture and ethical considerations.

[a] https://orcid.org/0000-0003-3374-8536

[b] https://orcid.org/0000-0003-2484-4366

[c] https://orcid.org/0000-0002-8012-5852

[1] Nmap https://nmap.org/

[2] Wireshark https://www.wireshark.org/

[3] Metasploit https://www.metasploit.com/

Our primary objective is to develop an automated tool to assist penetration testers in the initial stages of reconnaissance, planning, and scanning. IoTective simplifies the process of collecting information about a target network, making it more accessible and more efficient. Additionally, it provides support for a variety of devices and protocols commonly found in IoT environments, which thereby enhances its applicability.

IoTective is designed for ease of use and flexibility by prioritizing an intuitive user interface and providing customizable scanning capabilities. Ultimately, we aim to provide users with a useful and efficient tool for automatically identifying potential security issues in smart home environments. Furthermore, to emphasize responsible usage, our tool employs passive techniques to minimize risks by collecting information and discovering vulnerabilities without logging private data. Users must have authorized access to the network, with the tool restricted to private IP addresses and refraining from any probing over the Internet.

We demonstrate the effectiveness of IoTective through a case study in a real-world smart home environment. The generated report presents a comprehensive range of information gathered from various sources, protocols, and devices, effectively showcasing the capabilities of IoTective.

The rest of the paper is organized as follows: Sections 2 and 3 introduce smart homes and their security issues, respectively. Section 4 presents related work. In Section 5, we introduce the design of IoTective. Section 6 presents the implementation details and the effectiveness of IoTective through a case study. In Section 7, we discuss the impact of automated penetration testing on security posture and ethical considerations. Finally, we conclude this paper and outline future work in Section 8.

2 SMART HOMES

A smart home is a specific application of IoT technology focused on the residential environment. It involves the use of internet-connected devices to enable remote control and automation of appliances and systems, including lighting, heating, security, and entertainment. The goal of smart homes is to enhance residents' quality of life by providing convenience, comfort, energy efficiency, and entertainment (Li et al., 2021).

The popularity of smart home technology has surged, especially among younger generations, driven by the rapid development of IoT and innovations in artificial intelligence. A 2020 report by Strategy Analytics projected a significant increase in spending on smart home technologies, from \$120 billion in 2021 to \$175 billion by 2025 (Ablondi and Narcotta, 2020). It is anticipated that by 2025, nearly 390 million homes worldwide will have at least one type of smart system installed, accounting for 19% of all households.

Smart home automation systems typically utilize a central hub or controller that connects to various devices and systems in the home. This hub enables remote control and monitoring of these systems through a smartphone application, tablet, or computer. Some smart home automation systems also feature voice control capabilities, allowing homeowners to use voice commands to manage their devices. Moreover, smart home automation can involve the use of sensors, such as motion detectors, to automate specific functions, like turning on lights when someone enters a room. Additionally, these systems can be integrated with other connected devices, such as smart thermostats and locks, providing a more comprehensive and seamless automation of home functions and enhancing user convenience.

3 SECURITY ISSUES IN SMART HOMES

H. Touqeer et al. (Touqeer et al., 2021) provided a comprehensive overview of security challenges in IoT environments and described potential attacks at the application layer, perception layer, network layer, and physical layer of these environments. Considering that smart homes are one of the IoT applications, those attacks are applicable to smart home environments.

Furthermore, many vulnerabilities associated with the network protocols used in smart home environments have been reported. For instances, Zigbee is exposed to a suite of vulnerabilities. One such instance was discovered by Wara et al. (Wara and Yu, 2020) which included a replay attack on the Zigbee protocol in IoT applications. The authors were able to re-transmit captured packets to victim devices and demonstrated this on Phillips Hue bulbs and Xbee S1 and S2C modules. Zigbee is based on the IEEE 802.15.4 protocol specification, which lacks robust replay attack protection. This vulnerability has been exploited multiple times, often using the KillerBee tool (Pan, 2021). In addition, Zigbee maintains an association table that records all child nodes associated with the parent. A problem with the implementation is that the records are not deleted when a child node leaves the network after a power failure. Attackers can exploit this by causing a frequent replacement of child nodes, which consequently fills up the association table, making it impossible for new child nodes to join the network. Other attacks targeting Zigbee can be found in (Hussein and Nhlabatsi, 2022) (Okada et al., 2021) (Morgner et al., 2017).

Bluetooth Low Energy (BLE), also known as Bluetooth Smart, is a wireless communication technology specifically designed for low-power, low-datarate applications (Heydon and Hunn, 2012). BLE is also one of commonly used protocols in smart home environments. The survey conducted by Casar et al. (Cäsar et al., 2022) extensively examines the weaknesses and vulnerabilities present in BLE technology. The authors delve into known vulnerabilities and attacks that have been discovered during the evaluation of BLE development. Some of the prominent attacks against BLE include sniffing, man-in-the-middle (MITM), and jamming. The authors also identify various weaknesses, particularly those related to privacy. These weaknesses encompass issues such as device tracking, the ability to infer user behavior through traffic analysis, and the disclosure of user location through mobile applications. Furthermore, Barua et al. (Barua et al., 2022) provided an overview of the threat model against BLE devices and discussed some of the relevant vulnerabilities found in BLE.

Davis et al. (Davis et al., 2020) conducted a smart home case study, identifying several vulnerabilities in smart bulbs across four aspects: physical, network, software, and encryption by utilizing Common Vulnerabilities and Exposures (CVE) and National Vulnerability Database (NVD). In addition, a vulnerability discovered in Philips Hue published in December of 2020 made the devices vulnerable to a DoS attack [4]. By performing an SYN flood on port tcp/80, the Phillips Hue's hub will not respond until the flooding has stopped. During the attack, users are unable to control the lights or use the vendor's cloud services. Another CVE published in January of the same year showed that the Philips Hue was vulnerable to a heap-based buffer overflow attack, enabling attackers to perform remote code execution [5]. A DoS attack is also possible for the Sengled Zigbee Smart Bulb devices [6].

4 RELATED WORK

PENIOT [7] is an open-source penetration testing tool. It tests commonly used protocols such as Advanced Message Queuing Protocol (AMQP), Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), and BLE. The tool aims for semi- or fully automatic generic security attacks on IoT devices, offering protocol-specific attacks through a graphical interface. Users select an attack, input required information, and receive a downloadable report after completion. The authors planned additional features, including sniffing and attacks against Zigbee and RPL, but due to time constraints and lack of required hardware, they were not implemented. Besides, PENIOT has not undergone extensive development since its launch and, as of the time of writing this paper, it still relies on Python version 2.7.

HomePwn [8] is a security testing tool primarily designed for discovery and device enumeration and offers support for only sniffing and MAC spoofing. It supports MQTT, mDNS, Wi-Fi, and BLE, provides an interactive command line interface, and supports attacks like replay, injection, and eavesdropping. The tool also provides documentation and resources for users to understand the vulnerabilities and their potential impact. HomePwn was written to support Python 3.6; however, it remained unchanged without receiving any updates in the past 3-4 years.

KillerBee [9] is an open-source framework and testing tool designed for exploring and exploiting the security of Zigbee and IEEE 802.15.4 networks. Developed in Python, KillerBee allows researchers and security professionals to sniff, inject, and manipulate Zigbee network traffic in real time. The tool can also be used to conduct replay attacks, capture network keys, and perform other forms of network analysis. KillerBee only provides a classic command line interface, and it was originally written in C and is currently being ported to Python 3.5 or higher.

EXPLIoT [10] is an open-source framework designed for security testing and exploiting IoT products and infrastructure. It was written in Python 3, offering both interactive and non-interactive command line interfaces. EXPLIoT aims to support various IoT protocols, hardware platforms, and IoT products. However, some plugins necessitate hardware connectors to in-

[4] CVE-2018-7580 Detail https://nvd.nist.gov/vuln/detail/CVE-2018-7580

[5] CVE-2020-6007 Detail https://nvd.nist.gov/vuln/detail/CVE-2020-6007

[6] CVE-2022-47100 Detail https://nvd.nist.gov/vuln/detail/CVE-2022-47100

[7] PENIOT: Penetration Testing Tool for IoT https://github.com/yakuza8/peniot

[8] HomePwn - Swiss Army Knife for Pentesting of IoT Devices https://github.com/Telefonica/HomePWN

[9] KillerBee https://github.com/riverloopsec/killerbee

[10] EXPLIoT https://expliot.io/

teract with the relevant protocol, and the functionality may vary for each supported protocol. Despite these considerations, EXPLIoT appears to be a comprehensive framework due to its extensive protocol support and wide range of attacks.

Different from the aforementioned frameworks, IoTective employs protocols such as mDNS, Zigbee, Wi-Fi, and Bluetooth primarily for scanning and sniffing purposes. Furthermore, our tool incorporates a Graphical User Interface (GUI) implemented within the terminal environment, offering users the ability to interact with the tool using cursor-based actions. This design positions it as a hybrid interface, seamlessly combining features of both GUI and interactive command line interfaces. Notably, IoTective is developed to be compatible with the latest iteration of the Python programming language, specifically Python 3.11.

5 IoTective

Figure 1 illustrates the overall design of IoTective, which consists of four phases: Initialization, Scanning, Sniffing, and Reporting.

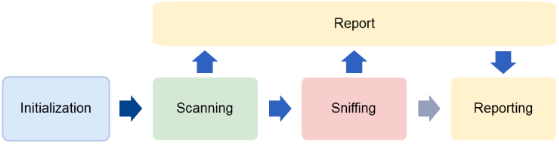


Figure 1: The overall design of IoTective.

In the initialization phase, users have the option to tailor their testing by choosing from four available options: Network scanning, Wi-Fi sniffing, Bluetooth sniffing, and Zigbee sniffing. Network scanning involves conducting an ARP (Address Resolution Protocol) scan to collect information about services, ports, operating systems, and known vulnerabilities associated with the services in the target smart home environment. Wi-Fi sniffing involves using packet capture to identify the hosts connected to the access point through Wi-Fi. Bluetooth scanning conducts a regular Bluetooth scan to gather information about services and their characteristics. Zigbee sniffing utilizes packet capture to provide an overview of Zigbee devices and their communications on each channel.

If users choose the Zigbee sniffing option, they must select a corresponding adapter, which is also a requirement shared with network scanning and Wi-Fi sniffing. In contrast, Bluetooth adapters can be configured automatically by IoTective. After configuration, the tool proceeds based on the user selection and preference.

Moving on to the scanning phase, as illustrated in Figure 2, IoTective initiates an ARP scan to quickly discover all live hosts by obtaining their IP and MAC addresses. Each IP address is then targeted for an extensive nmap enumeration. This enumeration may take varied time based on connection speed. IoTective is configured to probe the top 2000 ports of each host to identify running services. Note that these ports are set by Nmap and is a common way of using the tool for optimizing efficiency. This number strikes a balance between an exhaustive list, which would be time consuming, and allowing the discovery of more obscure ports, such as port 8123 used by Home Assistant. The enumeration also involves guessing the host's operating system and determining the accuracy of that guess.

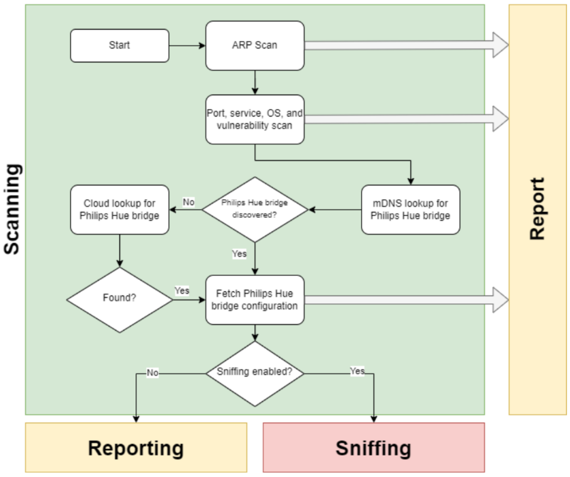

Scanning
Start
ARP Scan
Port, service, OS, and vulnerability scan
Cloud lookup for Philips Hue bridge
No
Philips Hue bridge discovered?
mDNS lookup for Philips Hue bridge
Yes
Found?
Yes
Fetch Philips Hue bridge configuration
No
Sniffing enabled?
Yes
Report
Reporting
Sniffing

Figure 2: The Scanning phase of IoTective.

Following device enumeration, IoTective focuses on finding the Philips Hue Bridge, which is a hub connecting and controlling Philips Hue smart bulbs and lights. This choice is made considering that the Philips Hue Bridge is a widely used and popular central component in the Philips Hue smart lighting system, representing a common device in smart home environments. IoTective initiates the process with an mDNS lookup, a protocol often used by IoT devices for local network service discovery and recommended by Philips Hue[11]. The subsequent steps depend on whether the bridge is discovered. If it is found, IoTective proceeds to retrieve the configuration of the devices. Otherwise, IoTective utilizes a GET request on the broker server URL, which provides the private IP address of any Philips Hue bridge on the network.

[11] How to develop for Hue? https://developers.meethue.com/develop/get-started-2/

The bridge configuration is then acquired by querying https://⟨BRIDGE_PRIVATE_IP_ADDRESS⟩/api/0/config using the acquired IP address.

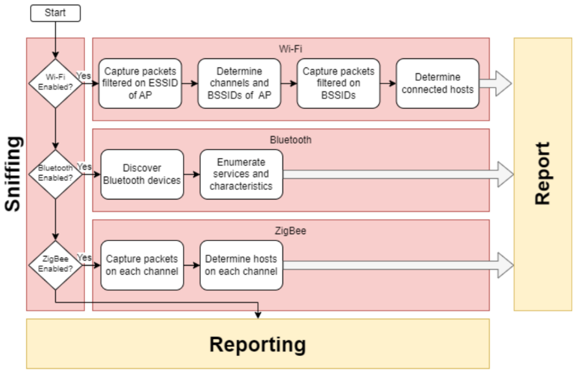

Start
Sniffing
Wi-Fi
Wi-Fi Enabled?
Yes
Capture packets filtered on ESSID of AP
Determine channels and BSSIDs of AP
Capture packets filtered on BSSIDs
Determine connected hosts
Bluetooth
Bluetooth Enabled?
Yes
Discover Bluetooth devices
Enumerate services and characteristics
ZigBee
ZigBee Enabled?
Yes
Capture packets on each channel
Determine hosts on each channel
Report
Reporting

Figure 3: The Sniffing phase of IoTective.

If any of the four scanning/sniffing options is chosen in the Initialization phase, IoTective proceeds to the Sniffing phase, as depicted in Figure 3.

For Wi-Fi monitoring, IoTective collects information by identifying the Wi-Fi network name, or Extended Service Set Identifier (ESSID), of the connected network. This name may either be the default set by the manufacturer or customized by the network administrator. While the network name helps in identifying the network, the actual data transmissions are tagged with a different identifier, the Basic Service Set Identifier (BSSID), which is unique to each access point (AP). This is because multiple APs can share the same ESSID to create a single, seamless network across a larger area, allowing devices to connect to the Internet regardless of which specific access point they are connected to. After identifying all BSSIDs associated with the ESSID, IoTective collects capture data packets related to these BSSIDs. Any unique hosts communicating via these BSSIDs are considered to be connected hosts.

Bluetooth sniffing is a straightforward process. It begins with a standard Bluetooth scan, allowing IoTective to recognize all Bluetooth devices in the surrounding area. Following this, the tool initiates connections with each device, seeking to gather additional information about the services and characteristics associated with each one.

In Zigbee sniffing, IoTective systematically checks every ZigBee channel one by one. This is done to capture traffic across the entire ZigBee spectrum to ensure no communications are missed. Note that ZigBee networks operate across various channels in the radio frequency spectrum, primarily in the 2.4 GHz band. As IoTective scans each channel, it identifies and records information about each host it finds, including their personal area network identifier (PAN ID). All information gathered during the Sniffing phase is added to the report before proceeding to the final phase.

In the last phase, the Reporting phase, IoTective takes the report as input and generates a JSON file. This decision is driven by the capability of Textual[12], a rapid application development framework for Python, to automatically generate a navigation user interface. This feature enhances user experience by facilitating the navigation of information and devices in the report.

6 IMPLEMENTATION AND CASE STUDY

IoTective is developed using Python 3.11 on Kali Linux. Python 3.11 is the latest version at the time of writing. Python is an ideal language for penetration testing tools due to its user-friendly syntax, flexibility, and rich libraries. On the other hand, Kali Linux is a Debian-based Linux distribution designed for security-related tasks. It comes pre-installed with numerous security tools and utilities, making it a popular choice among cybersecurity professionals, ethical hackers, and individuals interested in testing and securing computer systems.

IoTective requires the user to have a wireless adapter, a Zigbee adapter, and a Bluetooth adapter capable of monitoring mode. This enables the tool to conduct wireless, Zigbee, and Bluetooth network analyses, capturing and scrutinizing network traffic to identify potential vulnerabilities. Additionally, the inclusion of Bluetooth support enables the tool to scan and analyze Bluetooth Low Energy (BLE) devices.

To demonstrate the effectiveness of IoTective in identifying vulnerabilities and weaknesses in smart home environments, we deployed it in a real smart home setup. The environment included a Ethernet-connected Philips Hue Bridge, a Philips Hue Bloom light, a Philips Hue motion sensor, a Philips Hue smart plug, as shown in Fig 4. Additionally, the environment comprised other devices not visible in the figure, such as a Samsung TV, smartphones, computers, and a Raspberry Pi running Home Assistant and Pi-hole. Home Assistant [13] controlled the Philips Hue Bridge and connected IoT devices, while Pi-hole[14] served as a network-wide advertisement and malware blocker. The Raspberry Pi operated on the Raspbian

[12] Textual https://textual.textualize.io/

[13] Home Assistant https://www.home-assistant.io/

[14] Pi-hole https://pi-hole.net/

OS, with both Home Assistant and Pi-hole set up as Docker instances. This comprehensive setup represented a diverse range of devices commonly found in smart home environments, enabling IoTective to thoroughly assess and identify potential vulnerabilities across different types of connected devices.

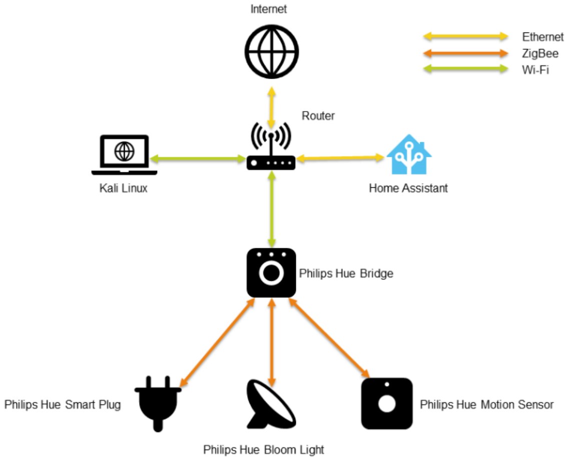
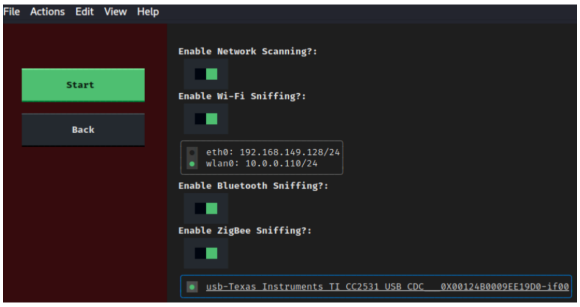


Figure 4: A simplified illustration of the smart home environment used.

6.1 The Initialization phase

To assess the the capability of IoTective, we activated all the options in the Initialization phase, as shown in Figure 5. This allows us to gain a comprehensive view of the tested smart home environment. It is clear that the interface of IoTective is user-friendly as it did not require users to input any commands, thus lowering the barrier for conducting penetration testing.



Enable Network Scanning?:

Enable Wi-Fi Sniffing?:

```
○ eth0: 192.168.149.128/24
● wlan0: 10.0.0.110/24
```

Enable Bluetooth Sniffing?:

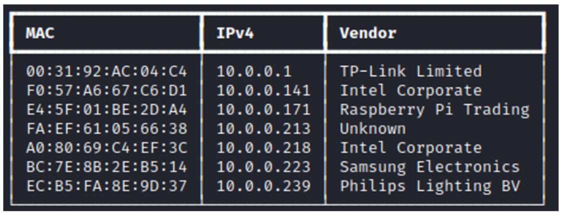
Enable ZigBee Sniffing?:

```
● usb-Texas_Instruments_TI_CC2531_USB_CDC___0X00124B0009EE19D0-if00
```

Figure 5: The initialization interface of IoTective.

6.2 The Scanning phase

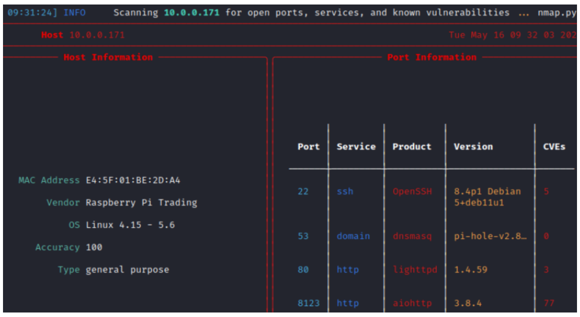
The Scanning phase began with an ARP scan to quickly identify and link all IP addresses to their corresponding MAC addresses in the local network. This scan enables the identification of all connected devices, with MAC addresses providing vendor information. The scanning result was presented in an easy-to-read table on the command line interface of IoTective, as shown in Figure 6. From the scan, we observed seven active hosts in the network, each accompanied by its respective MAC and IP addresses. The last column provides additional information about the vendor, which facilitates the identification of devices based on their manufacturer.



```
 MAC                 | IPv4        | Vendor
 00:31:92:AC:04:C4   | 10.0.0.1    | TP-Link Limited
 F0:57:A6:67:C6:D1   | 10.0.0.141  | Intel Corporate
 E4:5F:01:BE:2D:A4   | 10.0.0.171  | Raspberry Pi Trading
 FA:EF:61:05:66:38   | 10.0.0.213  | Unknown
 A0:80:69:C4:EF:3C   | 10.0.0.218  | Intel Corporate
 BC:7E:8B:2E:B5:14   | 10.0.0.223  | Samsung Electronics
 EC:B5:FA:8E:9D:37   | 10.0.0.239  | Philips Lighting BV
```

Figure 6: Results of the ARP scan shown on the command line interface of IoTective.

After identifying the connected hosts, IoTective proceeded to perform a comprehensive Nmap scan for each device. This scan gathered information about the operating systems, open ports, running services on each port, and any known vulnerabilities associated with these services. Figure 7 shows the scanning results for the Raspberry Pi, a device running the Home Assistant software. The results revealed four open ports, with port 8123 being particularly noteworthy. This port is commonly used by Home Assistant for web services over HTTP, which is known as an insecure protocol due to the lack of encryption for data transmission. Additionally, the scan identified 77 known vulnerabilities associated with the services running on this port.



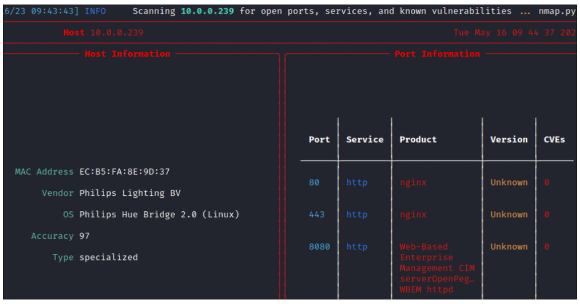
```
09:31:24] INFO     Scanning 10.0.0.171 for open ports, services, and known vulnerabilities ... nmap.py
    Host 10.0.0.171                                                        Tue May 16 09 32 03 202
──── Host Information ────────────┐┌──────────── Port Information ─────────

                                      Port | Service | Product  | Version       | CVEs
                                     ──────┼─────────┼──────────┼───────────────┼─────
  MAC Address E4:5F:01:BE:2D:A4
                                      22   | ssh     | OpenSSH  | 8.4p1 Debian  | 5
       Vendor Raspberry Pi Trading         |         |          | 5+deb11u1     |
           OS Linux 4.15 - 5.6
                                      53   | domain  | dnsmasq  | pi-hole-v2.8… | 0
     Accuracy 100
         Type general purpose         80   | http    | lighttpd | 1.4.59        | 3

                                      8123 | http    | aiohttp  | 3.8.4         | 77
```

Figure 7: Results of the Nmap scan for the Raspberry Pi deployed in the tested smart home.

Figure 8 shows the scanning results for the Philips Lighting BV host in the network. As expected, the host was identified as a Philips Hue bridge, with the operating system recognized as Philips Hue Bridge 2.0. The host had open ports 80 and 443, indicating the presence of a web server.



```
6/23 09:43:43] INFO     Scanning 10.0.0.239 for open ports, services, and known vulnerabilities ... nmap.py
        Host 10.0.0.239                                                     Tue May 16 09 44 37 202
──────── Host Information ────────────  ───────────── Port Information ─────────────

                                         Port │ Service │ Product             │ Version │ CVEs
                                        ──────┼─────────┼─────────────────────┼─────────┼──────
 MAC Address EC:B5:FA:8E:9D:37           80   │ http    │ nginx               │ Unknown │ 0
      Vendor Philips Lighting BV
          OS Philips Hue Bridge 2.0 (Linux)
                                         443  │ http    │ nginx               │ Unknown │ 0
    Accuracy 97
        Type specialized                 8080 │ http    │ Web-Based           │ Unknown │ 0
                                              │         │ Enterprise          │         │
                                              │         │ Management CIM      │         │
                                              │         │ serverOpenPeg…      │         │
                                              │         │ WBEM httpd          │         │
```

Figure 8: The result of the Nmap scan for the Philips Lighting BV host deployed in the tested smart home.

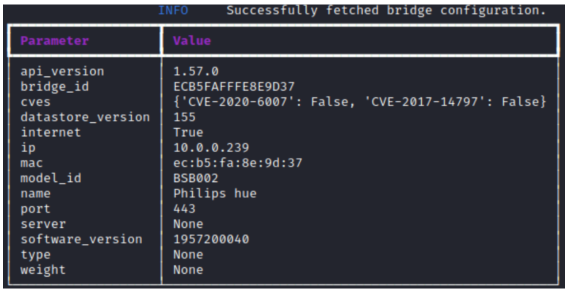
Once all the devices were enumerated, IoTective searched for a Philips Hue bridge on the network. Without prior knowledge of whether a Philips Hue bridge exists, the tool initiated an mDNS (Multicast DNS) lookup using the address "_hue._tcp.local", commonly associated with Philips Hue devices. However, since the mDNS query did not yield a response, IoTective then resorted to a cloud lookup by sending a GET request to https://discovery.meethue.com. As the bridge periodically queries the cloud to announce its private network address, if the bridge exists, the cloud lookup returns the device's private network address.

Having confirmed the presence of a Philips Hue bridge on the network and obtained its associated private network address, IoTective gained crucial information. Each Philips Hue bridge features an API that allows other applications, such as Home Assistant or custom-developed software, to control connected smart devices. Most API commands require authentication and authorization by the application. However, the "config" query serves as an exception, providing significant information about the bridge. This information was acquired by querying https://⟨BRIDGE_PRIVATE_IP_ADDRESS⟩/api/0/config, and the outcome is presented in Figure 9.

The configuration information includes properties such as model ID, bridge ID, and name. Of particular significance for both the users and IoTective are the API version and software version, as certain CVEs are associated with specific versions of the API and software used by the Philips Hue bridge. Ensuring that the API and software versions are current and patched is crucial for safeguarding against CVEs like CVE-2020-6007 (a buffer overflow vulnerability), and CVE-2017-14797 (deficiency in transport layer encryption).

6.3 The Sniffing phase

After the scanning phase, IoTective progressed to the Sniffing phase, starting with the capture of Wi-Fi packets. It determined the BSSIDs of the access point to which the wireless adapter was connected. This was achieved by capturing IEEE 802.11 broadcast packets on each channel and filtering them based on the access point's ESSID. This approach is used to avoid relying solely on the BSSID provided during Wi-Fi connection, as many routers support both 2.4 GHz and 5 GHz bands under the same ESSID. Each band has its distinct BSSID, and filtering based on only one of them could potentially result in missing relevant packets.



```
                 INFO     Successfully fetched bridge configuration.
┌──────────────────┬─────────────────────────────────────────────────────────┐
│ Parameter        │ Value                                                   │
├──────────────────┼─────────────────────────────────────────────────────────┤
│ api_version      │ 1.57.0                                                  │
│ bridge_id        │ ECB5FAFFFE8E9D37                                        │
│ cves             │ {'CVE-2020-6007': False, 'CVE-2017-14797': False}       │
│ datastore_version│ 155                                                     │
│ internet         │ True                                                    │
│ ip               │ 10.0.0.239                                              │
│ mac              │ ec:b5:fa:8e:9d:37                                       │
│ model_id         │ BSB002                                                  │
│ name             │ Philips hue                                             │
│ port             │ 443                                                     │
│ server           │ None                                                    │
│ software_version │ 1957200040                                              │
│ type             │ None                                                    │
│ weight           │ None                                                    │
└──────────────────┴─────────────────────────────────────────────────────────┘
```

Figure 9: Configuration information gathered from Philips Hue bridge API.

After identifying all the BSSIDs, the tool proceeded to capture packets on each channel, applying filters based on the BSSIDs to determine the band to which each connected device was connected. Despite the encrypted nature of the packets limiting the acquisition of new information, IoTective successfully established a mapping between MAC addresses and the addresses discovered during the Scanning phase. However, IoTective could only discover two out of four connected hosts, which is likely attributed to the 10-second time frame for capturing packets on each channel, as not all devices may transmit packets within that short period.

After identifying Wi-Fi hosts, IoTective proceeded to perform Bluetooth sniffing. It captured information such as device name, company name, device local name (i.e., human-readable name), signal strength, and service UUIDs (Universally Unique Identifiers). IoTective then attempted to establish a connection with each device to acquire additional information, including services and characteristics. The scan detected a total of 44 Bluetooth devices, with only three of them permitting access to information regarding their services and characteristics.

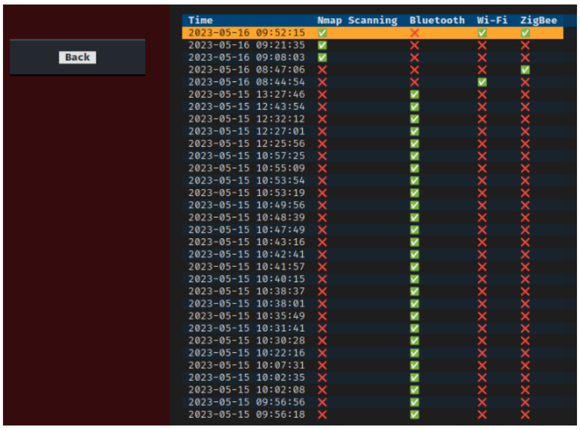
Finally, IoTective proceeded with Zigbee packet capture on each Zigbee channel to identify devices associated with each channel. This phase requires a Zigbee USB dongle with packet capture capabilities, such as the Texas Instruments CC2531[15]. The dongle

[15] Texas Instruments CC2531. https://www.ti.com/product/CC2531

had been flashed with a custom firmware designed for this specific purpose. Although the encrypted nature of the packets limits the available information, IoTective successfully identified crucial details including the channel, protocol version, PAN ID, etc. The scan discovered hosts distributed across channels 15, 20, and 25, with a total of five devices.

6.4 The Reporting phase

Once the scanning and sniffing phases are completed, IoTective offers users a convenient way to access the scan results through the ”View Reports” option. The reports are listed chronologically, with the most recent report displayed at the top, as shown in Figure 10. Each report includes the date, time, and enabled scan types for reference.

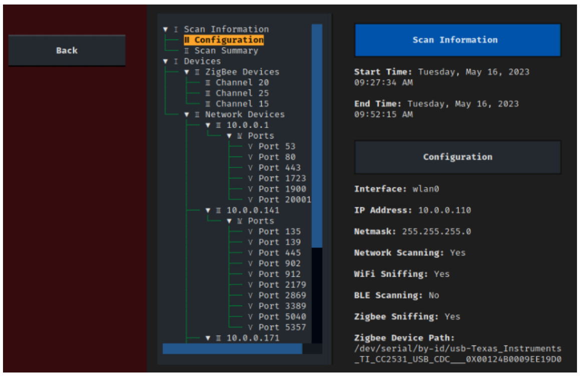


Time	Nmap Scanning	Bluetooth	Wi-Fi	ZigBee
2023-05-16 09:52:15	✓	✗	✓	✓
2023-05-16 09:21:35	✓	✗	✗	✗
2023-05-16 09:08:03	✓	✗	✗	✗
2023-05-16 08:47:06	✗	✗	✗	✓
2023-05-16 08:44:54	✗	✗	✓	✗
2023-05-15 13:27:46	✗	✓	✗	✗
2023-05-15 12:43:54	✗	✓	✗	✗
2023-05-15 12:32:12	✗	✓	✗	✗
2023-05-15 12:27:01	✗	✓	✗	✗
2023-05-15 12:25:56	✗	✓	✗	✗
2023-05-15 10:57:25	✗	✓	✗	✗
2023-05-15 10:55:09	✗	✓	✗	✗
2023-05-15 10:53:54	✗	✓	✗	✗
2023-05-15 10:53:19	✗	✓	✗	✗
2023-05-15 10:49:56	✗	✓	✗	✗
2023-05-15 10:48:39	✗	✓	✗	✗
2023-05-15 10:47:49	✗	✓	✗	✗
2023-05-15 10:43:16	✗	✓	✗	✗
2023-05-15 10:42:41	✗	✓	✗	✗
2023-05-15 10:41:57	✗	✓	✗	✗
2023-05-15 10:40:15	✗	✓	✗	✗
2023-05-15 10:38:37	✗	✓	✗	✗
2023-05-15 10:38:01	✗	✓	✗	✗
2023-05-15 10:35:49	✗	✓	✗	✗
2023-05-15 10:31:41	✗	✓	✗	✗
2023-05-15 10:30:28	✗	✓	✗	✗
2023-05-15 10:22:16	✗	✓	✗	✗
2023-05-15 10:07:31	✗	✓	✗	✗
2023-05-15 10:02:35	✗	✓	✗	✗
2023-05-15 10:02:08	✗	✓	✗	✗
2023-05-15 09:56:56	✗	✓	✗	✗
2023-05-15 09:56:18	✗	✓	✗	✗

Figure 10: List of reports generated by IoTective.

By opening a specific report, users can view the information obtained during the scan (for example, see Figure 11). The report is generated using Markdown language and presents the data in a structured format, allowing for easy navigation between devices. In this particular run, Bluetooth enumeration was performed separately, so the report states that Bluetooth scanning was not conducted. The scanning and sniffing processes took a total of 29 minutes and 55 seconds, with Wi-Fi and Zigbee accounting for 24 minutes and 41 seconds, while Bluetooth scanning took 4 minutes and 14 seconds.

Examining the Zigbee device section of the report reveals detailed information about each device, as presented in the table shown in Figure 12. This table includes relevant details such as the PAN ID, router capacity, device capacity, Link Quality Indicator (LQI), and protocol version, etc. By analyzing the channel number and PAN ID, users gain insights into nearby Zigbee networks. If they have a Zigbee device capable of packet injection, further active tests can be conducted using tools like KillerBee.



```
▼ I Scan Information
├── II Configuration
└── II Scan Summary
▼ I Devices
├── ▼ II ZigBee Devices
│   ├── III Channel 20
│   ├── III Channel 25
│   └── III Channel 15
└── ▼ II Network Devices
    ├── ▼ III 10.0.0.1
    │   └── ▼ IV Ports
    │       ├── V Port 53
    │       ├── V Port 80
    │       ├── V Port 443
    │       ├── V Port 1723
    │       ├── V Port 1900
    │       └── V Port 20001
    ├── ▼ III 10.0.0.141
    │   └── ▼ IV Ports
    │       ├── V Port 135
    │       ├── V Port 139
    │       ├── V Port 445
    │       ├── V Port 902
    │       ├── V Port 912
    │       ├── V Port 2179
    │       ├── V Port 2869
    │       ├── V Port 3389
    │       ├── V Port 5040
    │       └── V Port 5357
    ├── ▼ III 10.0.0.171
```

Scan Information

Start Time: Tuesday, May 16, 2023 09:27:34 AM

End Time: Tuesday, May 16, 2023 09:52:15 AM

Configuration

Interface: wlan0

IP Address: 10.0.0.110

Netmask: 255.255.255.0

Network Scanning: Yes

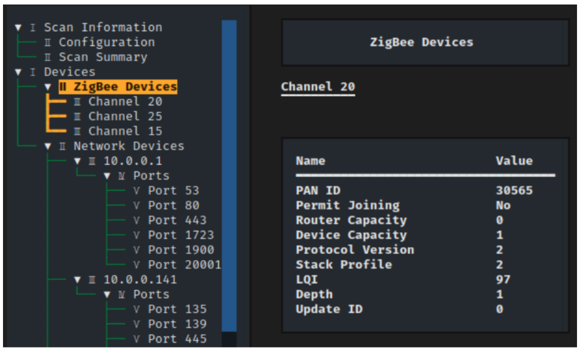
WiFi Sniffing: Yes

BLE Scanning: No

Zigbee Sniffing: Yes

Zigbee Device Path: /dev/serial/by-id/usb-Texas_Instruments_TI_CC2531_USB_CDC___0X00124B0009EE19D0

Figure 11: Scan information displayed in the report.



```
▼ I Scan Information
├── II Configuration
└── II Scan Summary
▼ I Devices
├── ▼ II ZigBee Devices
│   ├── III Channel 20
│   ├── III Channel 25
│   └── III Channel 15
└── ▼ II Network Devices
    ├── ▼ III 10.0.0.1
    │   └── ▼ IV Ports
    │       ├── V Port 53
    │       ├── V Port 80
    │       ├── V Port 443
    │       ├── V Port 1723
    │       ├── V Port 1900
    │       └── V Port 20001
    ├── ▼ III 10.0.0.141
    │   └── ▼ IV Ports
    │       ├── V Port 135
    │       ├── V Port 139
    │       ├── V Port 445
```

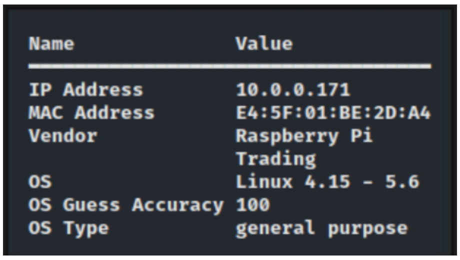
ZigBee Devices

Channel 20

Name	Value
PAN ID	30565
Permit Joining	No
Router Capacity	0
Device Capacity	1
Protocol Version	2
Stack Profile	2
LQI	97
Depth	1
Update ID	0

Figure 12: Zigbee device information displayed in the report.

If we move to the network device section, we can see all the network devices discovered by IoTective, including the Raspberry Pi device and its information, as shown in Figure 13.

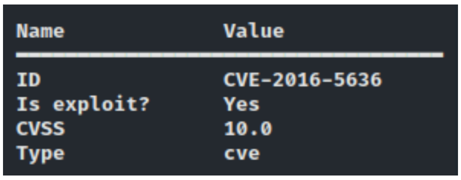


Name	Value
IP Address	10.0.0.171
MAC Address	E4:5F:01:BE:2D:A4
Vendor	Raspberry Pi Trading
OS	Linux 4.15 - 5.6
OS Guess Accuracy	100
OS Type	general purpose

Figure 13: Information about Raspberry Pi device gathered by IoTective.

During runtime, IoTective gains visibility into open ports on the host and the corresponding services running on those ports. For each open port, the report presents a list of potential vulnerabilities. Additionally, the report includes information on the Common Platform Enumeration (CPE), which standardizes the naming of software applications, operating systems, and hardware platforms. In this context, the CPE refers to the service operating on the respective port.

Users can utilize this information for further analysis, such as examining the source code of the software. In the case of the Home Assistant service running on the Raspberry Pi, IoTective detected 77 CVEs. In the report, each CVE entry provide information such as the CVE ID, the Common Vulnerability Scoring System (CVSS) score (offering a numerical representation of the vulnerability's severity), vulnerability type, and an indication of whether an exploit for the known vulnerability exists.



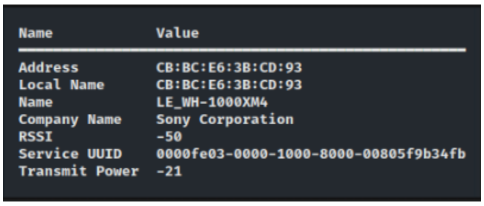
```
Name                Value
─────────────────────────────────────
ID                  CVE-2016-5636
Is exploit?         Yes
CVSS                10.0
Type                cve
```

Figure 14: The identified CVE-2016-5636 on the Home Assistant service.

One example is CVE-2016-5636, as shown in Figure 14, which relates to a vulnerability found in certain earlier versions of Python. This vulnerability involves an integer overflow issue that could potentially trigger a heap-based buffer overflow, enabling a remote attacker to exploit the vulnerability. It is important to note that although this CVE is associated with Python 3.10 in the identified CPE, it does not exclusively apply to the Home Assistant software implementation. Instead, it is relevant to any software utilizing Python as a programming language. Further investigation of the CVE ID reveals that this vulnerability only affects Python versions below 3.5.2, indicating that it does not pose a direct risk to the current Python installation used by Home Assistant.

Figure 15 illustrates the information gathered from a Sony WH-1000XM4 wireless Bluetooth headset. This data provides valuable insights for the user to understand the device's capabilities and functionalities. The gathered information reveals that the device is a product made by the Sony Corporation with the name "LE_WH-1000XM4," which represents the simplified name of the product. The local name, which is usually a shorter or alternative name that can be changed by the user, remains unchanged in this case and is identical to the Bluetooth MAC address of the device.

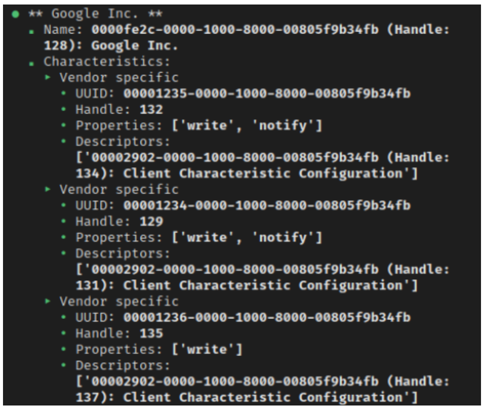
In Bluetooth technology, Received Signal Strength Indicator (RSSI) indicates the signal strength on the receiving unit's antenna, providing an estimation of the signal's quality and proximity of the Bluetooth device. On the other hand, TX power refers to the amount of energy transmitted by the sending unit's antenna. Service UUIDs, represented as 128-bit values, are used to uniquely identify different Bluetooth services or profiles offered by a device. Each Bluetooth device may support one or more services, and each service is identified by a unique UUID. When scanning a Bluetooth device, the list of service UUIDs provides information about the available services that the device supports. This information is useful for determining the capabilities and functionalities of the Bluetooth device.



```
Name              Value
──────────────────────────────────────────────────────────
Address           CB:BC:E6:3B:CD:93
Local Name        CB:BC:E6:3B:CD:93
Name              LE_WH-1000XM4
Company Name      Sony Corporation
RSSI              -50
Service UUID      0000fe03-0000-1000-8000-00805f9b34fb
Transmit Power    -21
```

Figure 15: Information gathered from Sony WH-1000XM4 headset.

Figure 16 illustrates an example of one of the services identified on the Sony headset. Each service is accompanied by a description, which in this case is "Google Inc." This could indicate that the headset has some sort of integration with Google, which aligns with the presence of Google Assistant as a feature service on the Sony WH-1000XM4 headset.



```
● ** Google Inc. **
  ▪ Name: 0000fe2c-0000-1000-8000-00805f9b34fb (Handle:
    128): Google Inc.
  ▪ Characteristics:
    ▸ Vendor specific
      • UUID: 00001235-0000-1000-8000-00805f9b34fb
      • Handle: 132
      • Properties: ['write', 'notify']
      • Descriptors:
        ['00002902-0000-1000-8000-00805f9b34fb (Handle:
        134): Client Characteristic Configuration']
    ▸ Vendor specific
      • UUID: 00001234-0000-1000-8000-00805f9b34fb
      • Handle: 129
      • Properties: ['write', 'notify']
      • Descriptors:
        ['00002902-0000-1000-8000-00805f9b34fb (Handle:
        131): Client Characteristic Configuration']
    ▸ Vendor specific
      • UUID: 00001236-0000-1000-8000-00805f9b34fb
      • Handle: 135
      • Properties: ['write']
      • Descriptors:
        ['00002902-0000-1000-8000-00805f9b34fb (Handle:
        137): Client Characteristic Configuration']
```

Figure 16: Service information gathered from Sony WH-1000XM4 headset.

Services further consist of "characteristics," which are individual data elements within a Bluetooth service that contain specific information or attributes. Both the service itself and each characteristic have unique UUIDs. An interesting attribute for the user to consider is the properties associated with each characteristic. These properties indicate the capabilities of each characteristic, specifying what actions can be performed, such as write, read, or notify. This information can be valuable for further testing, enabling the user to explore possible device modifications or gather additional information.

By examining the gathered Bluetooth device information, users gain insights into the manufacturer, names, signal strength, transmission power, supported services, and their associated characteristics. This knowledge allows for a better understanding of the device's capabilities, potential integration, and possible actions that can be performed with the Bluetooth device.

The combination of network scanning, Wi-Fi sniffing, Zigbee sniffing, and Bluetooth sniffing enables IoTective to provide a comprehensive view of connected devices and their services. The report generated presents a comprehensive range of information gathered from various sources, protocols, and devices, showcasing the capabilities of IoTective.

7 DISCUSSION

Although automated penetration testing is effective in identifying a broad range of vulnerabilities in smart home environments, its effectiveness is contingent upon the quality of the test cases and the skill of users. Manual testing is indispensable for uncovering complex vulnerabilities that cannot be identified through automated means alone. While IoTective addresses the initial phases of information gathering and scanning, further analysis is needed for the assessment to be comprehensive. Therefore, a combination of automated and manual testing is recommended for conducting thorough security assessments of smart home environments.

Automated penetration testing has a positive impact on the security posture of a smart home environment. It identifies vulnerabilities, provides remediation recommendations, reduces the attack surface, reveals security gaps, and offers ongoing insights into the environment's security. However, it is important to acknowledge that automated penetration testing should not be the sole security measure in a smart home environment. Implementing other measures such as regular software updates, strong passwords, and network segmentation is essential to ensure comprehensive security.

When it comes to ethical considerations, the development and use of automated penetration testing tools hold the potential to benefit both legitimate users and malicious actors. Therefore, ethical considerations are paramount to ensure that these tools are employed exclusively for ethical purposes. This involves obtaining user consent, preventing potential damage, and restricting access to the environment used for testing.

One crucial ethical consideration involves the risk of collateral damage. Automated penetration testing tools can inadvertently disrupt network services or disable critical functions. Testing must be conducted within a controlled environment to minimize the potential for unintended damage. This challenge was encountered during the development of IoTective, particularly in scoping the tool to specifically scan for relevant Bluetooth and Zigbee devices. While striving for automation and minimal user input, filtering out devices outside the scope proved challenging due to limited information. As a compromise, IoTective collects information from all devices in the vicinity but does so with minimal disruption, focusing on authorized information from publicly available sources.

Protecting users' privacy is another critical ethical consideration. Automated penetration testing tools may collect sensitive information, such as passwords or personal data, during the testing process. It is imperative to ensure that any data collected remains confidential and is not used for unauthorized purposes. In the case of IoTective, no sensitive data is collected as it does not attempt to exploit target devices. Only authorized information is gathered and analyzed using publicly available sources.

Furthermore, automated penetration testing should only be conducted with explicit consent from users. Providing clear information about the purpose of the testing, potential risks involved, and the steps taken to minimize those risks is vital. Users should have the option to opt-out of testing if they are uncomfortable with it. Additionally, testing should comply with relevant laws and regulations, such as data protection laws and regulations governing the use of automated tools for security testing. Lastly, it is essential to ensure that automated penetration testing is conducted transparently and accountably. This includes documenting the testing process, the obtained results, and the steps taken to address identified vulnerabilities in a clear and accessible manner.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced IoTective, which is an open-source automated penetration testing tool for identifying vulnerabilities in smart home environments. While many existing tools focus on providing a wide range of features, relying on analysts to select and perform specific tasks, IoTective takes a different approach by emphasizing automation and user experience. It does not aim to match the power and thoroughness of tools. Instead, it focuses on automating aspects of the penetration testing process that demand minimal expertise and involve substantial manual effort. As a result, IoTective can be seen as a valuable supplement to these other tools, which excel in more specialized and aggressive security testing. By automating repetitive and time-consuming tasks, IoTective streamlines the initial stages of IoT device assessment, enabling users to focus on more advanced security analysis and exploitation techniques. The use case demonstration illustrates IoTective's ability to automate the discovery of network interfaces and devices, enhancing usability through flexible scan options and user-friendly reports.

In our future work, we would like to further enhance and extend IoTective by investigating more efficient solutions for capturing Bluetooth and Zigbee network information, providing exploitation capability, and incorporating support for additional communication protocols commonly used in smart home environments, such as Z-wave or other proprietary protocols. Additionally, we plan to establish a process for regular updates and maintenance to ensure that IoTective stays current with the latest security vulnerabilities, attack techniques, and changes in device firmware and communication protocols.

ACKNOWLEDGEMENT

The authors want to thank the anonymous reviewers for their reviews and valuable suggestions to this paper. This work was partially conducted within the SFI-NORCICS (https://www.ntnu.edu/norcics). This project has received funding from the Research Council of Norway under grant no. 310105 "Norwegian Centre for Cybersecurity in Critical Sectors."

REFERENCES

Ablondi, W. and Narcotta, J. (2020). 2020 Global Smart Home Forecast - June 2020.

Alrawais, A., Alhothaily, A., Hu, C., and Cheng, X. (2017). Fog computing for the internet of things: Security and privacy issues. *IEEE Internet Computing*, 21(2):34–42.

Barua, A., Al Alamin, M. A., Hossain, M. S., and Hossain, E. (2022). Security and privacy threats for bluetooth low energy in iot and wearable devices: A comprehensive survey. *IEEE Open Journal of the Communications Society*, 3:251–281.

Cäsar, M., Pawelke, T., Steffan, J., and Terhorst, G. (2022). A survey on Bluetooth Low Energy security and privacy. *Computer Networks*, 205:108712.

Davis, B. D., Mason, J. C., and Anwar, M. (2020). Vulnerability studies and security postures of IoT devices: A smart home case study. *IEEE Internet of Things Journal*, 7(10):10102–10110.

Heydon, R. and Hunn, N. (2012). Bluetooth low energy. *CSR Presentation, Bluetooth SIG https://www. bluetooth. org/DocMan/handlers/DownloadDoc. ashx.*

Hussein, N. and Nhlabatsi, A. (2022). Living in the Dark: MQTT-Based Exploitation of IoT Security Vulnerabilities in ZigBee Networks for Smart Lighting Control. *IoT*, 3(4):450–472.

Khanna, A. and Kaur, S. (2019). Evolution of Internet of Things (IoT) and its significant impact in the field of Precision Agriculture. *Computers and electronics in agriculture*, 157:218–231.

Lee, M.-C., Lin, J.-C., and Owe, O. (2019). PDS: Deduce elder privacy from smart homes. *Internet of Things*, 7:100072.

Li, W., Yigitcanlar, T., Erol, I., and Liu, A. (2021). Motivations, barriers and risks of smart home adoption: From systematic literature review to conceptual framework. *Energy Research & Social Science*, 80:102211.

Morgner, P., Mattejat, S., Benenson, Z., Müller, C., and Armknecht, F. (2017). Insecure to the touch: attacking ZigBee 3.0 via touchlink commissioning. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 230–240.

Okada, S., Miyamoto, D., Sekiya, Y., and Nakamura, H. (2021). New LDoS attack in zigbee network and its possible countermeasures. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 246–251. IEEE.

Pan, T. (2021). ZigBee Wireless Network Attack and Detection. In *Advances in Artificial Intelligence and Security: 7th International Conference, ICAIS 2021, Dublin, Ireland, July 19-23, 2021, Proceedings, Part III 7*, pages 391–403. Springer.

Shah, S. and Mehtre, B. M. (2015). An overview of vulnerability assessment and penetration testing techniques. *Journal of Computer Virology and Hacking Techniques*, 11:27–49.

Tekeste Habte, T., Saleh, H., Mohammad, B., Ismail, M., Tekeste Habte, T., Saleh, H., Mohammad, B., and Ismail, M. (2019). IoT for healthcare. *Ultra Low Power ECG Processing System for IoT Devices*, pages 7–12.

Touqeer, H., Zaman, S., Amin, R., Hussain, M., Al-Turjman, F., and Bilal, M. (2021). Smart home security: challenges, issues and solutions at different IoT layers. *The Journal of Supercomputing*, 77(12):14053–14089.

Wara, M. S. and Yu, Q. (2020). New replay attacks on zigbee devices for internet-of-things (iot) applications. In *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–6. IEEE.

Zantalis, F., Koulouras, G., Karabetsos, S., and Kandris, D. (2019). A review of machine learning and IoT in smart transportation. *Future Internet*, 11(4):94.