PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
Ministry of Higher Education and Scientific Research
Mohamed Khider University – BISKRA
Faculty of Exact Sciences, Natural Sciences and Life
Department of Computer Science

Order n°: Startup RTIC 8/M2/2023

Thesis

Presented for the Academic Master's degree in

Computer Science

Option: Information and Communication Networks and Technologies (RTIC)

A deep learning-based approach for IOT software vulnerability location

---

By:

OMRANI ABIR

Defended on 03 /07 / 2023, in front of the jury composed of:

Naidji Ilyes	MCB	President
Boukhlouf Djemaa	MCB	Supervisor
Mouaki Bennani Nawel	MAA	Examiner

Academic year 2022-2023


بسم الله الرحمن الرحيم

ACKNOWLEDGEMENTS

I am grateful to everyone who has supported and assisted me during this journey. First and foremost, I would like to thank Allah for guiding me and enabling me to complete this work. I am thankful for His blessings and guidance throughout the process.

I want to express my sincere thanks to all the individuals who have provided their support and assistance. I am especially grateful to my parents for their unwavering support and encouragement. Their belief in me has been a driving force behind my success. I would also like to express my deep appreciation to my supervisor, **Dr. Boukhlouf Djemaa**, for her support and valuable guidance throughout this study. You have provided me with important academic support and guidance.

I also want to extend my gratitude to my Uncle **Dr. Khenfri Fouad** for his exceptional guidance and invaluable assistance. He has dedicated his time and expertise to help me, and his guidance and motivation have been a source of strength for me. May Allah reward him abundantly and grant him good health and well-being.

I would like to express my sincere gratitude to the Teachers of Entrepreneurship for their guidance, support, and mentorship, enabling this project to become part of a startup enterprise. I am truly grateful to all of you for your assistance and expertise in this domain.

Lastly, I would like to thank all those who have offered their assistance and support in bringing this study to fruition. I appreciate their collaboration and valuable contributions to the successful completion of this work.

I extend my heartfelt thanks to everyone involved, and I wish them all the best in their endeavors.

Abir

DEDICATION

I dedicate my graduation with love and gratitude to my dear father, who have been a strong support and a source of inspiration throughout my educational journey. I thank you from the bottom of my heart for your valuable guidance and boundless love. Additionally, I dedicate my graduation to my beloved mother, who has been the shining star in my life and the secret to my success. I sincerely thank her for her continuous support and prayers, which have contributed to my inspiration and achievements.

I would also like to express my gratitude to my uncle Fouad, who played a significant role in encouraging and supporting me. Through you, I have learned perseverance and dedication. Thank you for all the valuable advice and guidance you provided.

I dedicate this accomplishment to my sister and brothers, who have been true support and family to me in every stage of my life. With your presence and continuous support, I have been able to reach this special day. You are my strength and a source of inspiration.

Finally, I dedicate my graduation to my supervisor, **Dr. Boukhlouf Djemaa**, who provided me with assistance and support throughout this academic journey. Thank you for your guidance.

Abstract

In a connected world, security is one of the most significant challenges faced by individuals and organizations. Internet of Things (IoT) software, particularly its operating systems, device management, advanced computing, and communication protocols.., are vulnerable to security loopholes, exposing them to the risks of hacking and exploitation. To address this issue, our work proposes a deep learning-based approach to effectively detect vulnerabilities and accurately identify their locations in IoT software, specifically focusing on operating systems using C/C++ language. By harnessing the power of deep learning algorithms, we aim to enhance the accuracy and efficiency of vulnerability detection.

Our work contributes to the field of IoT security by providing a specialized software tool for detecting security vulnerabilities and improving overall security and reliability of IoT operating systems. We strive to protect individuals and organizations from security threats and enhance trust in the usage of IoT technology.

Keywords : IOT security, Vulnerability, Deep learning, Common Weakness Enumeration (CWE), Convolutional Neural Networks (CNN).

ملخص

في عالم متصل بالإنترنت، يُعد الأمان واحدًا من أكثر التحديات التي تواجه الأفراد والمؤسسات. تتعرض برامج إنترنت الأشياء، وخاصة أنظمة تشغيلها وإدارة الأجهزة والحوسبة المتطورة وبروتوكولات الاتصال..، لثغرات أمنية تعرضها للخطر من الاختراق والاستغلال.

لمعالجة هذا الأمر، يقدم مشروعنا نهجًا عميقًا قائمًا على التعلم العميق لاكتشاف نقاط الضعف وتحديد مواقعها بشكل فعال في برامج إنترنت الأشياء، وبالاخص أنظمة تشغيلها باستخدام لغة C/C++. يستفيد المشروع من قوة خوارزميات التعلم العميق لتعزيز دقة وكفاءة اكتشاف الثغرات الأمنية.

يساهم مشروعنا في مجال أمان إنترنت الأشياء من خلال توفير أداة برمجية متخصصة لاكتشاف الثغرات الأمنية وتعزيز الأمان العام وموثوقية أنظمة تشغيل إنترنت الأشياء. نهدف إلى حماية الأفراد والمؤسسات من التهديدات الأمنية وتعزيز الثقة في استخدام تقنية إنترنت الأشياء.

الكلمات الرئيسية: حماية انترنت الاشياء ، الضعف ، التعلم العميق ، تعداد نقاط الضعف الشائعة (CWE) ، الشبكات العصبية التلافيفية (CNN).

Résumé

Dans un monde connecté, la sécurité est l'un des défis les plus importants auxquels sont confrontées les individus et les organisations. Les logiciels de l'Internet des objets (IoT), en particulier leurs systèmes d'exploitation, la gestion des appareils, l'informatique avancée et les protocoles de communication, sont exposés à des failles de sécurité, les rendant vulnérables aux piratages et aux exploitations.
Pour remédier à cette situation, notre projet propose une approche basée sur l'apprentissage profond pour détecter efficacement les points faibles et les localiser avec précision dans les logiciels IoT, en mettant l'accent sur les systèmes d'exploitation utilisant le langage C/C++. En exploitant la puissance des algorithmes d'apprentissage profond, nous visons à améliorer la précision et l'efficacité de la détection des vulnérabilités.
Notre projet contribue au domaine de la sécurité de l'IoT en fournissant un outil logiciel spécialisé pour détecter les vulnérabilités de sécurité et améliorer la sécurité globale et la fiabilité des systèmes d'exploitation de l'IoT. Nous nous efforçons de protéger les individus et les organisations contre les menaces de sécurité et de renforcer la confiance dans l'utilisation de la technologie de l'IoT.

Mots clés : IOT sécurité, Vulnérabilité, Deep Learning, Énumération des faiblesses communes (CWE), Réseaux neuronaux convolutifs (CNN).

Table of Contents

List of Figures

List of Tables

General Introduction

The rapid growth of the Internet of Things (IoT) in recent years has resulted in the integration of various smart devices. These devices rely on specialized operating systems designed specifically for the IoT environment. However, the widespread adoption of IoT devices has raised concerns about their security and vulnerability to cyber-attacks. Software vulnerabilities in IoT can expose critical information, compromise system integrity, and potentially lead to catastrophic consequences. [101][95].
The identification and location of software vulnerabilities in different Internet of Things (IoT) software present notable hurdles, primarily stemming from the intricate nature and wide array of IoT devices. Conventional security methods often prove inadequate when applied to IoT systems, given the resource limitations, constrained processing power, and the dynamic and heterogeneous environments in which they function. As a result, innovative and effective techniques must be devised to tackle the distinctive attributes associated with IoT software vulnerabilities.[42].

This work proposes a deep learning-based approach to tackle the problem of IoT software vulnerability location, with a specific focus on IoT operating systems. Deep learning has demonstrated remarkable success in various domains, such as computer vision, natural language processing, speech recognition, text analytics, and cybersecurity [89].
By leveraging the power of deep learning algorithms, our aim is to enhance the accuracy and efficiency of vulnerability identification and localization in IoT operating systems. Our proposed approach consists of several key steps. Firstly, we collect a diverse and comprehensive dataset of IoT OS vulnerabilities. This dataset serves as the foundation for training our deep learning models. We will employ technique preprocessing to enhance

the dataset's quality and ensure its representativeness.

Next, we design and implement a deep learning architecture suitable for vulnerability location in IoT operating systems. This architecture is convolutional neural networks (CNNs).The deep learning model is trained using the collected dataset. To evaluate the effectiveness of our approach, we conduct extensive experiments using a variety of IoT operating systems. The performance of our deep learning model is assessed through measuring accuracy, precision, recall, and F1-score.

Additionally, we will develop software to facilitate the discovery of vulnerabilities, utilizing the trained model.
The manuscript is structured into four chapters:

- Chapter one introduces the basic concepts of IoT, its attacks, and vulnerabilities.
- Chapter two presents the top 10 important vulnerabilities in IoT based on organizations such as the World Security Organization (OWASP), along with different vulnerability detection methods. A review of related work is also included.
- Chapter three focuses on the general system design, detailed system components, model testing, and provides UML diagrams.
- Lastly, chapter four provides implementation details and presents the obtained results.

Chapter 1

Internet of Things Vulnerabilities

1.1 Introduction

IoT (Internet of Things) security refers to the measures and technologies used to protect IoT devices, networks, and data from unauthorized access, theft, or damage. With the increasing number of IoT devices being connected to the internet and the increasing volume of sensitive data being collected and stored, IoT security is becoming a critical concern. In this chapter, we describe at first what Internet of Things (IoT) is, which structure it usually has and, we continue by IOT technologies, then we talk about vulnerabilities types in IOT and the last thing about IOT security goals.

1.2 Definition of IOT

The lack of a standardized definition of IoT is a natural consequence of the diversity of IoT and the quick advancement brought about by the extensive research being conducted. Consequently, various definitions have been put forth by researchers and standardization organisations, including:

- **Definition by ITU (International Telecommunication Union)[44]:** "The Internet of Things is a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies."

- **Definition by IERC (European Research Cluster)[93]:** "A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual 'things' have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network."

- **Definition by ISOC (Internet Society)[86]:** "The term Internet of Things generally refers to scenarios where network connectivity and computing capability extends to objects, sensors and everyday items not normally considered computers, allowing these devices to generate, exchange and consume."

1.3 Architecture of IOT

A generic high-level architecture composed of three layers has been introduced in the literature:[27]

- **Perception layer:** representing the physical layer of objects and combining all attributes.

- **Network layer:** The network layer focuses on transmitting and processing the data collected by the perception layer.

- **Application layer:** which refers to the level of an application that effectively implements software that offers a specific service.

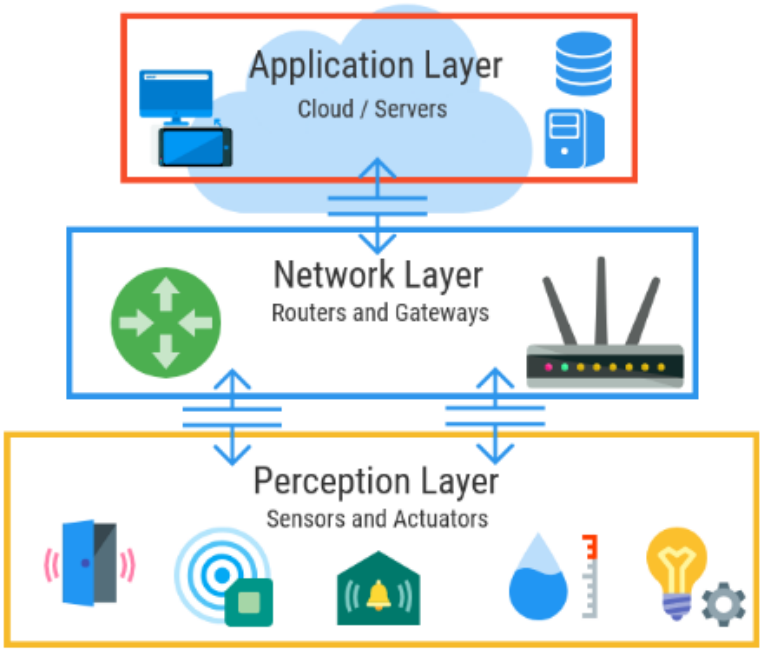


Figure 1.1: Three-tier IOT Architecture .[5]

1.3.1 Perception Layer

The Perception Layer is responsible for gathering and processing data through physical sensors in the IoT system. This layer comprises various sensors and actuators that enable functionalities like location tracking, temperature measurement, weight sensing, motion detection, vibration monitoring, acceleration measurement, humidity sensing, and more. To accommodate the diverse range of objects, standardized plug-and-play mechanisms are necessary within the perception layer for seamless configuration of heterogeneous devices[11].The three-layer IoT architecture encompasses several key features and capabilities of smart objects:[58]

- **Communication:** smart objects can establish connections with each other and access Internet resources to exchange data, update their status, and collaborate to achieve common objectives.

- **Identification:** each object must have a unique identifier to distinguish it from others within the IoT system.

- **Addressability:** objects can be directly accessed and interacted with remotely, allowing for remote configuration and interrogation.

- **Sensing and actuation:** smart objects are equipped with sensors to gather information from the surrounding environment and actuators to manipulate physical or digital entities.

- **Embedded information processing:** smart objects possess computational capabilities to process sensor data and control actuators based on the obtained results.

- **Localization:** objects are either aware of their physical location or can be precisely located within the IoT system.

- **User interface:** objects can communicate appropriately with users via displays or other interfaces.

Table 1.1 presents a range of technologies used to implement these various features in smart objects [24].Several hardware platforms available in the market, such as Raspberry Pi, Arduino, Beaglebone Black, etc., offer these features and serve as popular choices for IoT development.

Table 1	Used Technologies
Communication	Zigbee, Bluetooth, Wifi, Near Field Communication (NFC), Radio-Frequency IDentification (RFID), etc.
Identification	Electronic Product Code (EPC), Ubiquitous Code (uCode), Quick Re-sponse (QR), etc.
addressability Sensing and Actuation	IPv4, IPv6 Micro Electro-Mechanical Systems (MEMS) e Micro-Opto-Electro-Mechanical Systems (MOEMS), embedded sensors, etc.
Embedded information processing	Field Programmable Gate Array (FPGA), Programmable Logic Controller (PLC), microcontrollers, Single-board computer, System-on-Chip (SoC)
Localization User interface	Global Position System (GPS), Galileo, etc. Displays, remote control, etc.

Table 1.1: technologies in IOT field.[91]

1.3.2 Network Layer

The Network Layer plays a crucial role in processing and transmitting data received from the Perception Layer. It utilizes various network technologies, including wireless and wired networks, as well as Local Area Networks (LAN), to establish communication channels between devices. Common transmission media employed in this layer include FTTx, 3G/4G, WiFi, Bluetooth, Zigbee, UMB, infrared technology, and others. Given the substantial volume of data involved, it becomes imperative to have a robust middleware for efficient storage and processing. Cloud computing emerges as the primary technology for this purpose, offering a reliable and dynamic interface for data storage and processing. Research and development efforts focused on the processing aspect are pivotal for the future advancement of IoT. There are a large number of protocols that can be used in IoT. Table 1.2 shows some of the most used protocols, grouped according to the ISO/OSI model.[6]

Application Layer
COAP, MQTT, AMQP, XMPP, DSS Service Discovery: mDMS, DNS-SD, SSDP Security: TLS, DTLS
Transport Layer
TCP, UDP
Network Layer
Addressing: IPv4/IPv6 Routing: RPL, CORPL, CARP, etc.
Adaption Layer
6LOWPAN, 6TISCH, 6LO, etc.
Data Link Layer
IEEE 802.15.4, IEEE 802.15.1 (Bluetooth), LPWAN (LoRaWAN, etc.), RFID, NFC (ZigBee, etc.), IEEE 802.11, IEEE 802.3, IEEE 1901 (WiFi), (Ethernet), (PLC)
Physical Layer

Table 1.2: Protocols used in the IoT field. [91]

The widely used communication technologies include ZigBee, Bluetooth low energy (BLE), IPv6 over low power wireless personal area networks (6LoWPAN) and long-range wide area network (LoRaWAN). The 6LoWPAN protocol was created to meet wireless sensor networks. WSN is composed of devices characterized by low computational power that often have to minimize energy consumption. In specific applications, such Gateway may be necessary.[40]

1.3.3 Application Layer

The application layer is the third layer of IoT systems that provides services to users through mobile and web software. Based on the latest trends and usage of smart things, IoT has many applications in this technologically advanced world Residential/Home/Building, Transportation, Healthcare, Education, Agriculture, Business/Commercial, Power Distribution System, etc.They have become smart due to IoT system and many

services.[94] Some of the software technologies currently widely used to manage the enormous amount of data provided by devices are:

- Cloud computing, when services such as data storage or processing are provided by a set of pre-existing, configurable and remotely accessible resources in the form of a distributed architecture.

- Edge computing, when data processing is partially distributed to peripheral network nodes to improve the performance of IoT systems This level also includes the management of the format of the processed data.

Among the many commercial platforms used to deploy IoT applications, some examples include Amazon AWS, Microsoft Azure, Xively, Firefox WebThings Gateway, etc. [58]

1.4 IOT Applications

IoT offers a wide range of applications to improve people's daily lives and activities. Figure 1.2. Shows possible examples of IoT applications:[40]

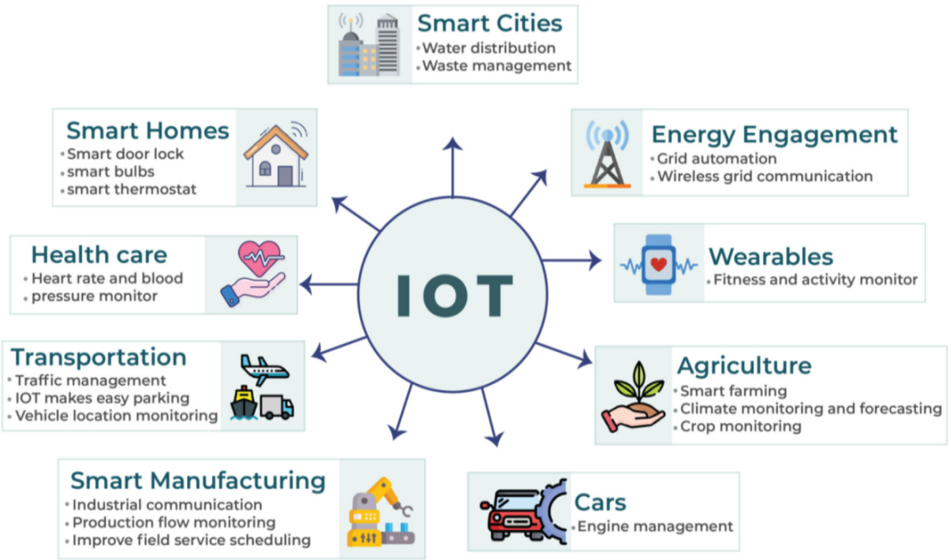


Figure 1.2: IOT Applications.[15]

- **Smart home:** is a home that is equipped with IoT (Internet of Things) devices and technology that allows homeowners to remotely control and automate various

systems and appliances(Show Figure 1.3). These devices are connected to the internet and can be accessed and controlled via a smartphone, tablet, or computer, from anywhere in the world[40]. These functions include lighting, Heating Ventilation Air Conditioning (HVAC), security, and home appliances like washers, dryers, ovens, and refrigerators. With the use of Wi-Fi, users can remotely monitor and manage these systems, which are typically connected to the Internet of Things.[84]

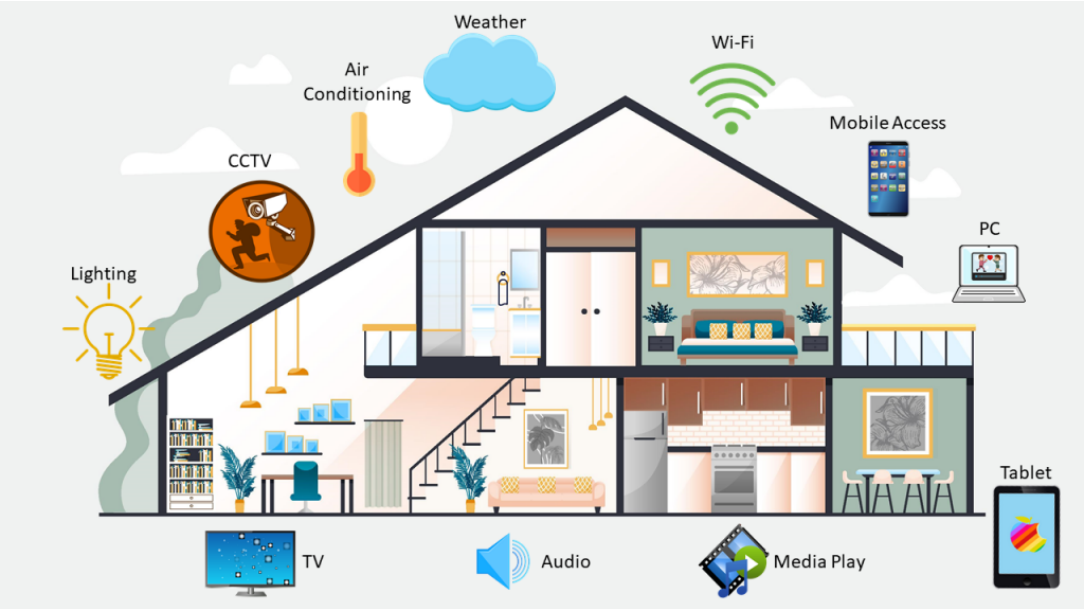


Figure 1.3: Smart home.[45]

- **Smart healthcare:** refers to the use of IoT (Internet of Things) technology in healthcare settings, including hospitals, clinics, and individual homes. This technology includes devices and systems that can collect and transmit healthcare data, such as patient vitals, medication adherence, and activity levels. enables the collection, transmission and storage of patients' physiological information. such as insulin pumps and heart monitors, which can transmit real time data to healthcare providers and alert them to potential issues.[40]

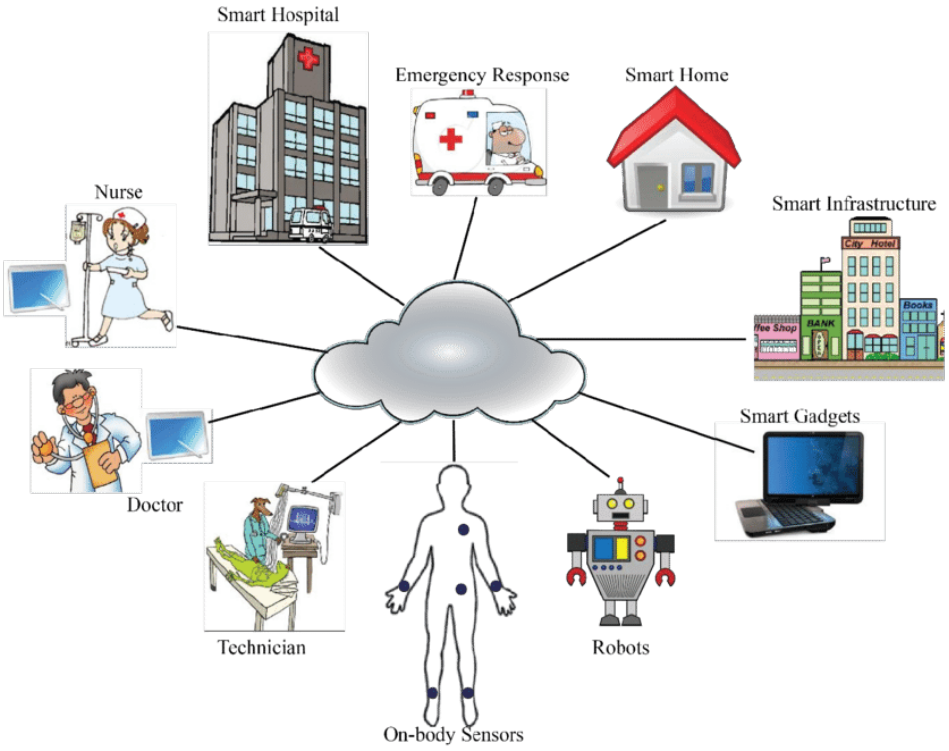


Figure 1.4: Smart healthcare.[64]

- **Smart transportation:** IoT technology has the potential to revolutionize communication, control, and information processing across diverse transportation systems. This technology has far-reaching applications across various aspects of transportation, including vehicles, infrastructure, and drivers or users. By facilitating dynamic and efficient interaction between these components, IoT enables inter- and intra-vehicle communication, smart traffic control, intelligent parking, logistics management, safety, and road assistance, among other benefits.[46]

Figure 1.5: Smart transportation.[62]

- **Smart industry:** known as Industrial IoT (IIoT) uses machine-to-machine technology to automate the process of manufacturing with insignicant human intervention. The IIoT aims to optimize industrial processes, increase efficiency, and reduce costs by collecting and transmitting real-time data on various aspects of the manufacturing process, from production output and energy consumption to machine performance and supply chain management. It includes the use of predictive maintenance systems, asset tracking, energy management, robotics and automation, and logistics optimization to improve productivity and sustainability.[40]

- **Smart Agriculture:** involves the use of IoT sensors and machine metrics to provide farmers with data and recommendations to enhance their farming practices, ranging from livestock management to harvesting. A prime illustration of this is the smart greenhouse, where farmers previously relied on manual intervention to regulate the greenhouse environment. With the integration of IoT technology, the greenhouse's humidity, temperature, sunlight, air quality, air flow, and soil condition can be automatically recorded, analyzed, and adjusted, thereby improving efficiency and yield.[88]

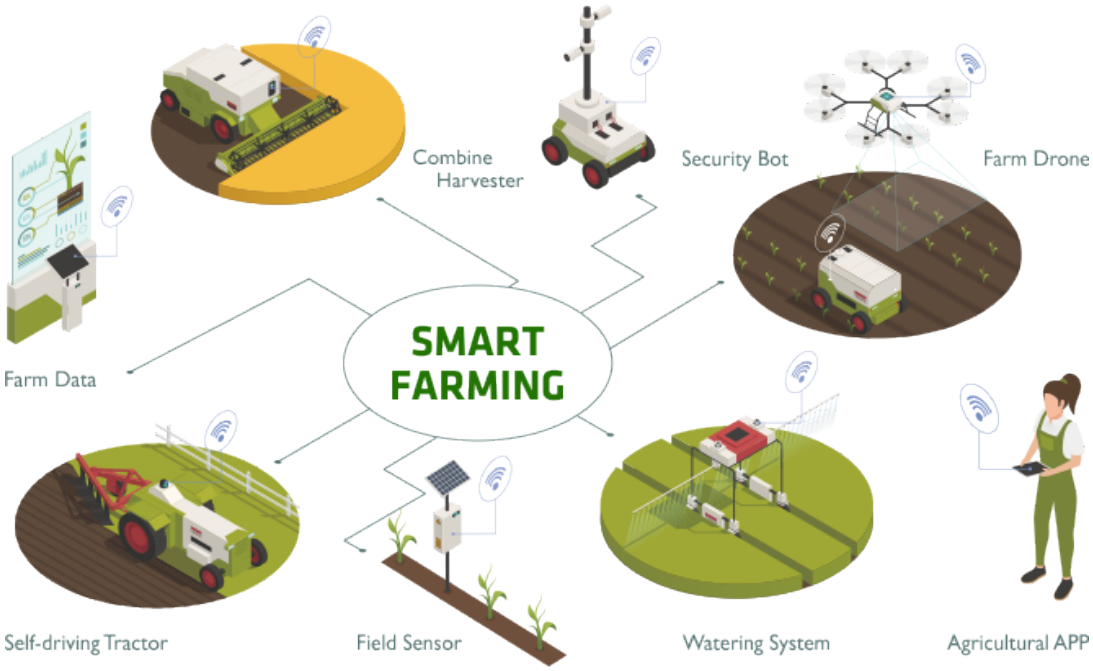


Figure 1.6: Smart Agriculture.[88]

- **Smart retail:** enables the tracking of goods while they are being transported or stored in warehouses. To track the status of a retail item, sensors can be fastened to it. Several smart shopping systems were created to offer customers intelligent services and so attract more customers.[40]

- **Smart grids:** is a typical example of an IoT application that tracks, manages, and measures electricity use. It makes it possible to control electricity efficiently and dependably, offers energy savings, and lowers problems with/failures with power grids.[40]

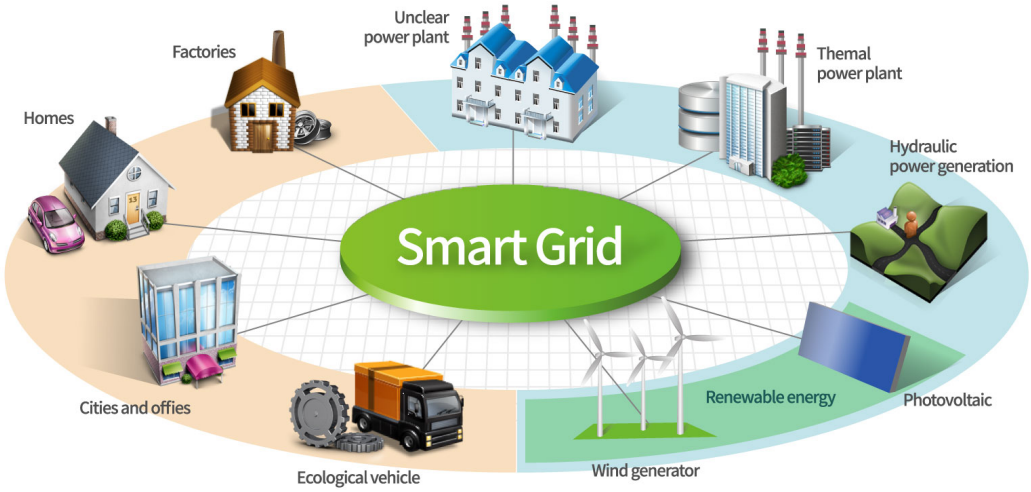


Figure 1.7: Smart grids.[54]

1.5 IOT Technologies

There are many technologies involved in IoT implementation: [61]

- **Radio Frequency Identification (RFID):** is an important technology for making objects clearly identifiable. A sticker-like transceiver microchip that can be active or passive depending on the type of application [32]. An active tag is connected to the battery. That is, active tags are always active and therefore continuously transmit data signals, whereas passive tags only become active when triggered [92]. An RFID system consists of her RFID tag associated with a reader that, when triggered by the generation of an appropriate signal, emits the identity, location, or other details of an object [100]. Emitted object-related data signals are transmitted using radio frequencies to the reader, which transfers the data to a processor for analysis.[32]

- **Near Filed Communication (NFC):** is a relatively new technology for short-range wireless communication. It functions in the unlicensed Radio Frequency band of 13.56 MHz and has an operating distance of up to 20 cm, although this distance may be longer in certain circumstances. The communication distance, however, primarily depends on the size of the antenna, which is usually quite small when

integrated into a mobile phone. Consequently, the communication distance for an NFC-enabled mobile phone is generally limited to approximately 3 to 5 cm.[93]

- **Wireless Sensing Network (WSN):** is a network of distributed autonomous devices that use sensors to keep track of environmental or physical factors at various locations, such as temperature, sound, vibration, pressure, motion, or pollution. can be utilized for a wide range of purposes, including military use, detecting forest fires, and monitoring metrics on human bodies.[61]

- **IoT Wireless Technologies:** The IoT ecosystem relies heavily on wireless technologies to link and aggregate IoT devices with the Wireless Sensor Gateway.[61]

- **ZigBee:** ZigBee is one of the newest and most sophisticated wireless technologies that is being widely incorporated into home automation and smart devices around the world. It has been established expressly as an open global standard to address the special requirements of low-cost, low-power wireless networks for device communication (also known as machine-to-machine or M2M networks). The 2.4 GHz, 900 MHz, and 868 MHz unlicensed bands are used by the ZigBee standard.[31]

Figure 1.8: ZigBee.[31]

- **Bluetooth:** is a wireless technology that connects various devices, including mobile phones, laptops, and other network devices, over short distances. We use it to

move files around or move a little bit of data. At a data rate of 1Mbps, we use this technology over a short range of 50 to 150 meters. It makes use of 2.4–2.485GHz Ultra High Frequency radio waves. Smartphones, smartwatches, laptops, wireless speakers, and wireless headsets all use Bluetooth technology.[55]

- **Z-Wave:** is a wireless technology that allows point-to-point communication over a distance of up to 30 meters. It is designed for low-data-rate applications, such as controlling household appliances, lights, HVAC systems, wearable health care devices, access control, and fire detection. Z-Wave operates in the unlicensed industrial, scientific, and medical (ISM) frequency band of around 900 MHz and provides a transmission rate of up to 40 kbps.[97] Wireless Fidelity (Wi-Fi): Wi-Fi, also known as the IEEE 802.11x standard, is the most common way to connect devices wirelessly to the Internet. Your laptop, smartphone and Tablet PC are equipped with Wi-Fi interfaces and talk to your wireless router and provide you this way access to the Internet.[31]

- **6LoWPAN:** Low-power wireless personal area networks, often known as 6LoWPAN, are a well-liked wireless communication standard. Using IPv6, 6LoWPAN facilitates communication using the IEEE 802.15.4 protocol. Between the communication and transport layers of 802.15.4, this standard creates an adaptability layer. Any other IP-based device can communicate with 6LoWPAN devices over the Internet. The choice of IPv6 is made possible by the size of its addressing space. 6LoWPAN networks use a gateway (WIFI or Ethernet) to access the Internet [55]. It is especially made for Internet of Things (IoT) devices with minimal computing power and power consumption.[61]

- **LoRa:** Long-range (LoRa) network is a low power communication technique that Semtech, a corporation, owns the patent for. It was originally established in 2008, and there isn't much information available about this protocol. This protocol's advantages include its high degree of dependability, moderate cost, and low power consumption over a long distance. Nevertheless, the data rate is only 50Kbps.[96]

- **Sigfox:** the SIGFOX technology, developed by the company of the same name, is designed to establish wireless networks in an unlicensed frequency band. This tech-

nology uses a proprietary ultra-narrowband (UNB) modulation and has a limited uplink connection. The low bit rate used in this technology allows communication over long distances with very low transmission power.[63]

1.6 IoT Characteristics

The Internet of Things (IoT) is a complex system with various characteristics that differ across different domains. While there are many characteristics, some of the key ones are described by (Chandrashekhar, 2016) as follows:

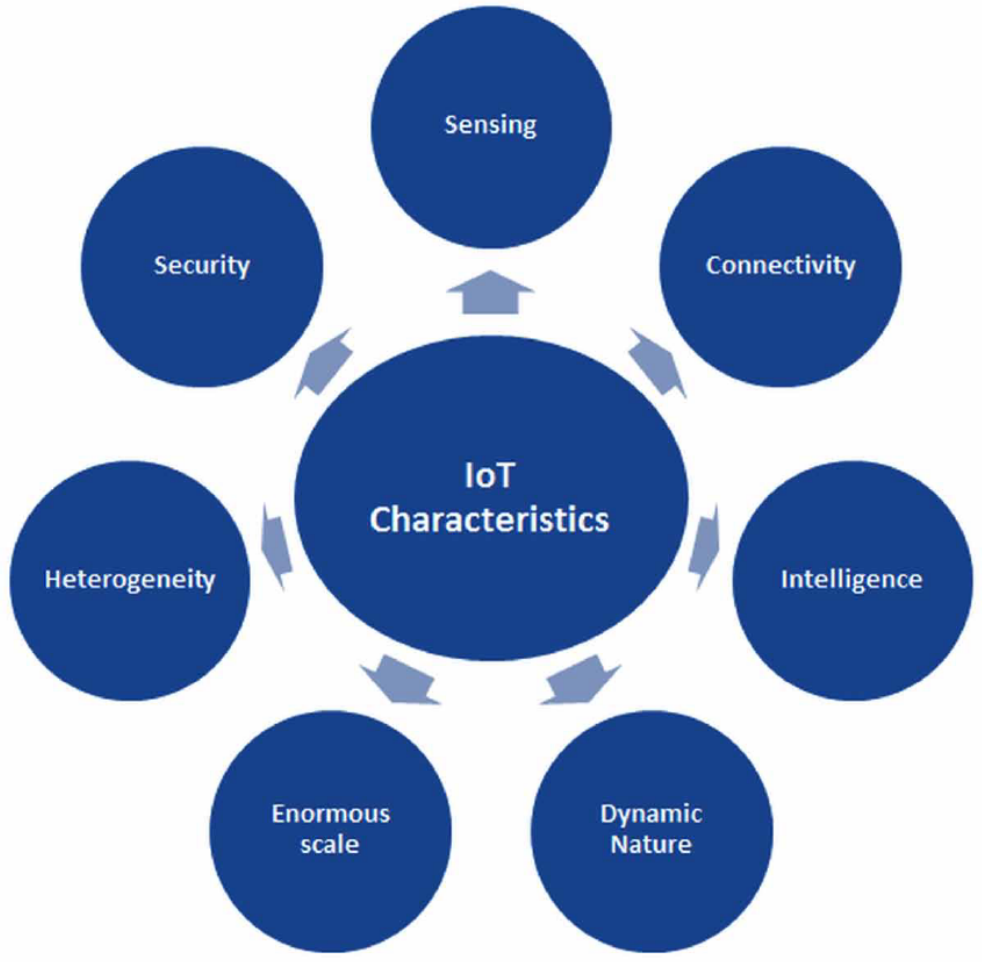


Figure 1.9: IoT Characteristics.[47]

1. **Sensing**

   which is essential for IoT to function. Sensors detect changes in the environment and provide data on the status of objects or interactions with the environment.

Sensing technologies enable capabilities that replicate a real-world awareness of the physical world and its inhabitants. Although sensing information is just analog input from the physical world, it can offer a comprehensive understanding of our complex world.[47]

2. **Connectivity**

   The internet of things is becoming more prevalent, and as a result, devices are now more connected than ever. This connectedness, which is the ability of gadgets to communicate and exchange information, offers many potential for companies to build new goods and services. Anything from refrigerators and cars to pets have been connected by the internet of things, allowing for remote control and data sharing between them. According to predictions, the internet of things will continue to grow quickly in the years to come, giving businesses a significant way to interact with customers and increase sales. This connectivity can be facilitated by wired LAN and wireless technologies including Wi-Fi, LPWAN, LoRa, and ZigBee.[88]

3. **Security**

   As the number of devices connected to IoT continues to increase, there is a corresponding rise in vulnerabilities such as data breaches and security issues. A security breach in a single IoT device can have a significant impact on the security of an entire network. Due to their resource limitations, many IoT devices are unable to support advanced security features and may lack the computational power needed to implement robust security protocols. For instance, sensors used for temperature detection may not be capable of handling advanced encryption or other security measures. Currently, there is no widely accepted industry standard for IoT security, and organizations typically have their own specific security standards and frameworks.[97]

4. **Intelligence**

   IoT is intelligent because it combines algorithms, computation, software, and hardware. IoT ambient intelligence improves the characteristics that allow IoT objects to respond intelligently to a particular scenario and assist them in carrying out particular activities. Despite the widespread adoption of smart technologies, intelligence in the Internet of Things (IoT) only refers to a way of communication

between devices; user and device interaction is accomplished through graphical user interfaces and common input methods.[47]

5. **Heterogeneity**

   the devices utilized in IoT are diverse, with varying hardware platforms and network types. IoT can incorporate devices with different processing capabilities, storage capacities, and overall architecture. This heterogeneity is a fundamental characteristic of IoT, enabling its widespread adoption across various platforms and domains. It allows for numerous devices based on different technologies to interact with each other without compatibility issues. Cloud Radio Access Network (Cloud-RAN) and Software Defined Radio (SDR) framework can facilitate heterogeneity in IoT by adapting to the devices' communication technology.[97]

6. **Dynamic Nature**

   Objects within an IoT network are capable of changing their state from rest to motion and vice versa, meaning the system cannot be considered static. The dynamic nature of the data produced by changes in states such as sleeping and waking up or connected and disconnected is also an important consideration. The air conditioner manufacturing company example featured in the "How does it work?" section highlights the changing number of devices connected to a network. Consequently, the ability to adapt to a dynamic environment is a crucial characteristic of an IoT network.[56]

7. **Enormous Scale**

   The quantity of devices that require to be managed which communicate with one another are abundant larger than the devices connected to the present internet. The management of information generated from these devices and their interpretation for application functions becomes a lot of essential. Gartner (2015) confirms the big scale of IoT within the estimated report wherever it explicit that 5.5 million new things can get connected each day and 6.4 billion connected things are in use worldwide in 2016, that is up by 30% from 2015.The report conjointly forecasts that the quantity of connected devices can reach 20.8 billion by 2020. [47]

1.7 Vulnerabilities

1.7.1 What is a Vulnerability?

Vulnerabilities are weaknesses in a system or its design that allow an intruder to execute commands, access unauthorized data, and/or conduct denial-of-service attacks. Vulnerabilities can be found in variety of areas in the IoT systems. In particular, they can be weaknesses in system hardware or software, weaknesses in policies and procedures used in the systems and weaknesses of the system users themselves.[7] IoT systems are made up of two basic components such as system hardware and system software in which, both of which are prone to design faults. Hardware attacks are hard to find and repair, even when they are found, due to hardware compatibility and interoperability, as well as the time and effort required to fix them. Operating systems, application software, and control software, such as communication systems and device drivers, all have software attacks. Program design faults are caused by a variety of variables, particularly human factors and software complexity. Human flaws are frequently the source of technical attacks. Creating the project without a plan, inadequate communication between developers and users, an insufficient resource, abilities, expertise, and failing to manage and control the system are all examples of not knowing the needs.[14]

1.7.2 Vulnerabilities in the IoT

In the area of big data, the Internet of Things (IoT), and artificial intelligence, security is a critical concern. The increasing number of smart devices creates a pressing need to comprehend new security threats on IoT and sensor networks and develop effective countermeasures against such attacks.[83]

- **Software vulnerabilities :** pose a significant threat to the IoT ecosystem, with potential devastating consequences. Adversaries exploit these vulnerabilities to gain control over IoT devices, compromising their security and functionality. Lack of security protection mechanisms is a major contributor to IoT device security issues. Several studies have demonstrated how vulnerable settings, poor authentication,

and insecure default configurations can be exploited in IoT devices.

For example, Chapman [22] and Rodrigues [85] showcased the exploitation of vulnerabilities in IoT devices. Max [52] conducted a security analysis of a smart lock, uncovering unsafe authentication and default configuration. Fernandes et al[33] revealed compromised security in Smart Home IoT devices when using third-party apps. Costin's [25] investigation of firmware upgrades uncovered various vulnerabilities.

Cyberattacks leveraging software vulnerabilities aim to leak information or disrupt the normal operation of targeted systems.A cyber-attack is typically carried out by a malicious program to either leak information or disrupt the normal operation flow of a targeted system. Software vulnerabilities can enable adversaries to gain unauthorized access and control over IoT devices. Given that IoT devices are low-energy devices, software optimization plays a critical role in reducing power consumption, especially in remote areas. Firmware vulnerabilities also pose significant security concerns, as they can disrupt the regular operation of IoT devices.[66] Addressing software vulnerabilities in IoT requires implementing robust security protection mechanisms, secure authentication protocols, and secure default configurations. Regular security audits, firmware updates, and adherence to secure software development practices are essential to mitigate software-related risks in the IoT ecosystem.

- **Hardware Vulnerability :** Sensors play a critical role in collecting real-time data and transmitting it to IoT devices remotely. These low-power sensors are often used in power plants to regulate turbine generators and monitor load demands. To ensure data security, the data transmitted by these sensors is encrypted before being sent to the monitoring system. However, attackers may conduct man-in-the-middle attacks to steal important data or transmit a modified signal to the control center. Attacks on the Advanced Smart Metering Infrastructure (AMI) provide attackers with a new entry point to compromise the power network's data security. These attacks can result in data theft, power theft, localized or global denial-of-service attacks, and power grid disruptions. Smart meters and data collectors usually interact through radio frequency methods in the 900MHz ISM band. As many unlicensed devices compete for spectrum use in the ISM band (industrial, scientific medical

band), a successful spoofing attack on the ISM frequency is relatively easy to execute. The mesh network for data transmission between smart meters, substation data collectors, and utility analysis centers is particularly vulnerable to masquerade attacks that can inject erroneous monitoring data and disrupt regular power distribution. In addition, replay attacks can be used to resend old measurements and tamper with the smart meter without detection in the substation or meter data management system.[83]

- **Network Vulnerability :** in the IoT are a major concern due to the increasing number of low-end edge nodes and the prevalence of big data, IoT, machine learning, and artificial intelligence. Adversaries exploit these vulnerabilities to compromise sensitive information.

  Hackers have successfully compromised Amazon IoT devices like Ring doorbells and Home security cameras by intercepting unencrypted usernames and passwords transmitted over the local user network using Hypertext Transfer Protocol (HTTP). This emphasizes the need for secure communication protocols. Security cameras and voice assistants, such as Alexa and Google Home, are also at risk of breaches.[57]

  The use of UPnP protocols for easy configuration and control of IoT devices further amplifies network vulnerabilities. The lack of authentication, validation, and logging in UPnP protocols allows for various attack methods. IoT devices in smart home environments are particularly vulnerable if proper security measures are not implemented.[66]

  According to Morgner [67], connecting the Zigbee Light Link base to the host network can lead to leaks of sensitive information due to unsecured key management from sharing pre-defined keys. Existing key management system classes may not be suitable for certain scenarios, highlighting the necessity for customized security measures. To enhance network security in the IoT, it is crucial to implement strong encryption and authentication protocols, prioritize secure communication channels, and regularly update IoT devices' security. Robust key management practices and secure configurations are essential to mitigate network vulnerabilities and safeguard sensitive information in the IoT ecosystem.

- **Chip Level Vulnerability :** Chip level vulnerabilities specifically refer to vul-

nerabilities that exist at the silicon or integrated circuit level. These vulnerabilities can be inherent in the design or manufacturing process of the chips used in IoT devices.Hardware Trojans (HTs) pose a significant threat to IoT devices, introducing malicious modifications that can compromise their functionality and security. These alterations can lead to information leaks, manipulation, and security bypassing. Integrated chips, such as application-specific system-on-chip and field-programmable gate array devices, are susceptible to digital and analog HTs. Of particular concern are the cryptography units within IoT devices, which are vulnerable due to their low power features.

Unlike software bugs that can be addressed through firmware updates, hardware vulnerabilities like HTs are more challenging and costly to eliminate. They may even result in permanent damage and service degradation to the affected devices.[66] To enhance chip-level security, rigorous security measures must be implemented across the supply chain, including trusted IP vendors and design houses. Hardware testing and verification processes should be strengthened to detect and prevent the insertion of HTs. Moreover, countermeasures against side-channel attacks, such as the use of secure cryptographic algorithms and protection against electromagnetic emissions, should be adopted to bolster the overall security of IoT devices at the chip level.

1.8 IOT Attacks

1.8.1 Definition of IoT attacks

Attacks are actions taken to harm a system or disrupt normal operations by exploiting vulnerabilities using various techniques and tools.[19] Attackers launch attacks to accomplish objectives, either for their own gratification or to receive retribution. Attack cost is a measurement of the amount of effort that will be made by an attacker, represented in terms of their knowledge, resources, and motivation. Attack actors are individuals who pose a risk to the online environment. They might be lawbreakers, hackers, or even governments. An attack itself may come in many forms, including active network attacks to monitor unencrypted traffic in search of sensitive information; passive attacks such

as monitoring unprotected network communications to decrypt weakly encrypted traffic and getting authentication information; close-in attacks; exploitation by insiders, and so on.[7]

1.8.2 IoT attack surface areas

The Open Web Application Security Project (OWASP) has published a detailed draft list of IoT attack surface areas, or areas in IoT systems and applications where threats and vulnerabilities may exist. Below is a summarization of the IoT attack surface areas:[2]

- **Devices:** IoT devices themselves can be susceptible to attacks. Components such as device memory, firmware, network interface, physical interface, and web service may have security vulnerabilities that can be exploited by attackers. For example, outdated or unpatched components can serve as entry points for malicious activities.

- **Communication Channels:** The communication channels used by IoT devices must be adequately protected to prevent unauthorized access and interception of sensitive data. Weaknesses in encryption, authentication, or insecure protocols can allow attackers to eavesdrop on or manipulate communication between devices, compromising the integrity and confidentiality of the data being transmitted.

- **Software and Applications :** The security of IoT systems can be compromised through weaknesses in the software and applications that control and manage the devices. Inadequate security measures, improper authentication mechanisms, or poorly implemented access controls can provide opportunities for attackers to gain unauthorized access to IoT devices, manipulate their functionality, or extract sensitive information.

1.8.3 Different types of IoT attacks

Below is a list of some of the most frequent IoT attacks:

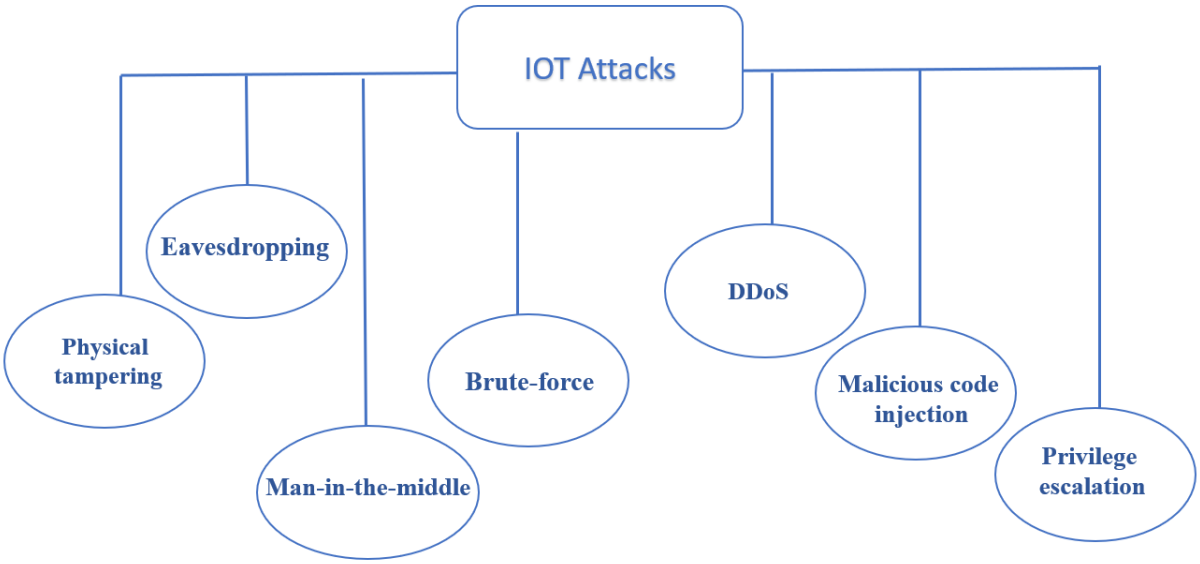


Figure 1.10: Different IOT Attacks.[17]

- **Physical tampering :** hackers can readily reach the devices' physical locations and grab their data. By gaining access to the device's ports and internal circuits, they can also put malware on it or breach into the network.[17]

- **Eavesdropping:** occurs when an adversary intercepts and blocks transmitted packets in the communication channel, preventing them from reaching the intended receiver device. RFID devices are particularly susceptible to eavesdropping attacks. The confidentiality of IoT systems is compromised when eavesdropping takes place on IoT devices.[69]

- **Man-in-the-middle:** attackers who insert traffic between devices and cloud-based services can take advantage of unencrypted connections or inadequately secured IoT networks. Communications between two systems are breached, interrupted, or spoof by an attacker.[17]

- **Brute-force password attacks:** cybercriminals can access your system by attempting various word combinations to guess the password. IoT devices have the easiest passwords to guess since they are designed without security considerations.[66]

- **Malicious code injection:** is a severe form of attack where an attacker gains access to a node and injects malicious code into the system. This type of attack can have devastating consequences, potentially leading to a complete network shutdown or, in the worst-case scenario, granting the attacker full control over the network.[17]

- **Privilege escalation :** attackers can access an IoT device by taking advantage of flaws, such as operating system errors, bugs in the device, or vulnerabilities that haven't been fixed. By further exploiting flaws, they can get into the system, climb the administrative ladder, and get information that will be useful to them.[66]

- **Distributed Denial of Service (DDoS):** a denial-of-service attack, sometimes known as a "DoS attack," aims to prevent users from accessing a computer or network resource by temporarily or permanently interrupting the services of a host that is connected to the Internet. Because the incoming traffic flooding a target during a distributed denial-of-service assault (DDoS) comes from numerous sources, it is challenging to halt the cyberattack by merely blocking one source.[17]

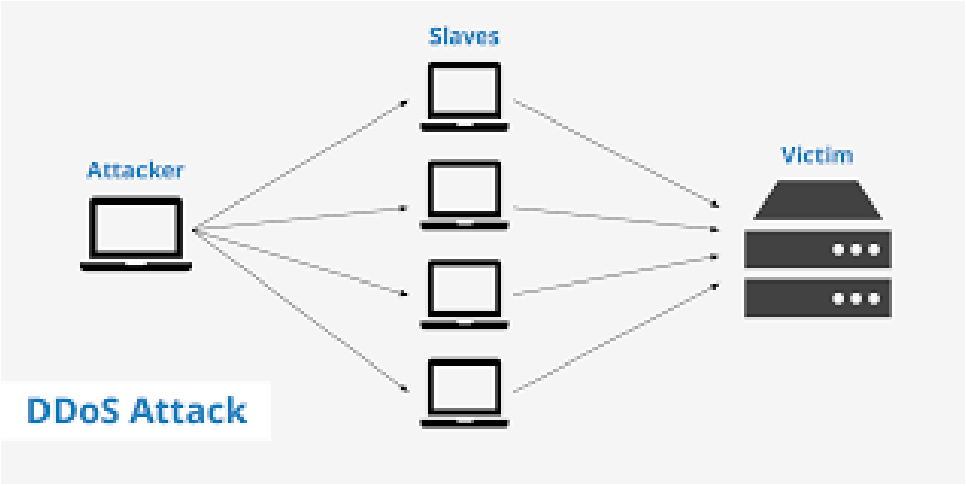


Figure 1.11: DDoS attack.[4]

1.9 IoT security goals

In order to protect the IoT environment, all IoT components should achieve the following security goals:[47]

1. **Confidentiality:** Confidentiality is a critical security feature of IoT systems, as these devices may contain and transmit sensitive information that must not be disclosed to unauthorized parties. Ensuring that only authorized individuals have access to this information is crucial for the protection of user privacy and the prevention of data breaches. Therefore, IoT systems must implement robust encryption and access control mechanisms to maintain the confidentiality of sensitive information.[12]

2. **Integrity :** The IoT relies on the exchange of data and information between a diverse range of devices, making it crucial to ensure the accuracy and integrity of this data. To achieve this, it is essential to verify that the data is being received from the correct sender and has not been tampered with during transmission, whether through intentional or unintentional interference. Maintaining end-to-end security in IoT communications is key to enforcing data integrity. While firewalls and protocols are commonly used to manage data traffic, they may not guarantee integrity at endpoints in IoT due to the limited computational power of IoT nodes, which may not support these mechanisms effectively. Therefore, IoT systems should implement robust end-to-end security measures that are optimized for low-power devices to ensure the integrity of data at all stages of transmission.[99]

3. **Availability :** The vision of IoT is to connect as many smart devices as possible. The users of the IoT should have all the data available whenever they need it. However, data alone is not sufficient for successful IoT implementation. Devices and services must also be readily available and accessible when needed to meet the high demands of IoT. Therefore, timely access to devices and services is essential to realizing the full potential of IoT.[60]

4. **Privacy :** It is the process by which an IoT system follows privacy rules or policies and empowers users to manage their sensitive data.[47]

5. **Non-repudiation :** The non-repudiation property refers to the ability to provide access logs that can serve as evidence in situations where users or objects cannot deny an action. While non-repudiation is not typically considered a crucial security property for most IoT systems, it can be essential in certain contexts, such as payment systems, where both parties must be unable to deny a payment transaction.

In these scenarios, non-repudiation plays a vital role in ensuring the integrity of the payment process and protecting against fraud. Therefore, IoT systems that handle sensitive transactions should implement strong non-repudiation mechanisms to ensure the accuracy and validity of all actions performed.[12]

6. **Audibility :** Ensures that the IoT system can closely monitor its actions.[47]

7. **Accountability :** is a crucial aspect of developing secure network security techniques. It adds redundancy and responsibility for specific actions, duties, and the planning of the implementation of network security policies. While accountability alone may not prevent attacks, it is essential for ensuring the effectiveness of other security techniques. Fundamental security issues, such as integrity and confidentiality, may be ineffective without proper accountability measures in place.[7]

8. **Trustworthiness :** Ensures the IoT system's ability to prove identity and confirm trust with third parties.[47]

Figure 1.12: IoT security goalss.[47]

1.10 Conclusion

In recent years, the Internet of Things (IoT) has been viewed as an essential research area where physical items will connect via different network protocols. The development of IoT services has greatly increased the need for a reliable security system.
In this chapter, we analyzed the IOT vulnerabilities and attacks to protect IoT devices, networks, and data from unauthorized access, theft, or damage. In next chapter, we present how to detect vulnerabilities in IOT.

Chapter 2

Detection and location of vulnerabilities in IOT

2.1 Introduction

Vulnerabilities in Internet of Things (IoT) devices are weaknesses or flaws in the devices' software or hardware that can be exploited by attackers to compromise the security and privacy of the device and the data it collects or processes. These vulnerabilities can arise from various sources and can have severe consequences for individuals, businesses, and critical infrastructure. Therefore, it is crucial to identify and address IoT vulnerabilities to ensure the security and reliability of IoT systems and protect them from potential cyber attacks.

In this chapter, we explain methods for detection vulnerabilities, highlighting relevant work on vulnerability detection in IOT Software.For this work, we aim to apply deep learning to detect vulnerabilities in the IOT operating system.

2.2 The Open Web Application Security Project (OWASP)

As defined by the official OWASP website:[73]

The Open Worldwide Application Security Project (OWASP): is a nonprofit foundation

that works to improve the security of software. Through community-led open-source software projects, hundreds of local chapters worldwide, tens of thousands of members, and leading educational and training conferences.

Figure 2.1: The Open Web Application Security Project (OWASP) Logo

The OWASP Internet of Things Project: is designed to help manufacturers, developers, and consumers better understand the security issues associated with the Internet of Things, and to enable users in any context to make better security decisions when building, deploying, or assessing IoT technologies.
The project looks to define a structure for various IoT sub-projects separated into the following categories-Seek& Understand, Validate& Test, and Governance.[72]

2.3 Top 10 vulnerabilities in IOT

The Open Web Application Security Project (OWASP) published the "OWASP Top 10 Internet of Things" list, which identifies the most critical security vulnerabilities for IoT systems. The varied nature of these vulnerabilities presents a significant obstacle to developing a comprehensive solution for detecting and addressing them effectively. The latest version of the OWASP Top 10 IOT list is published (2018) are the following:

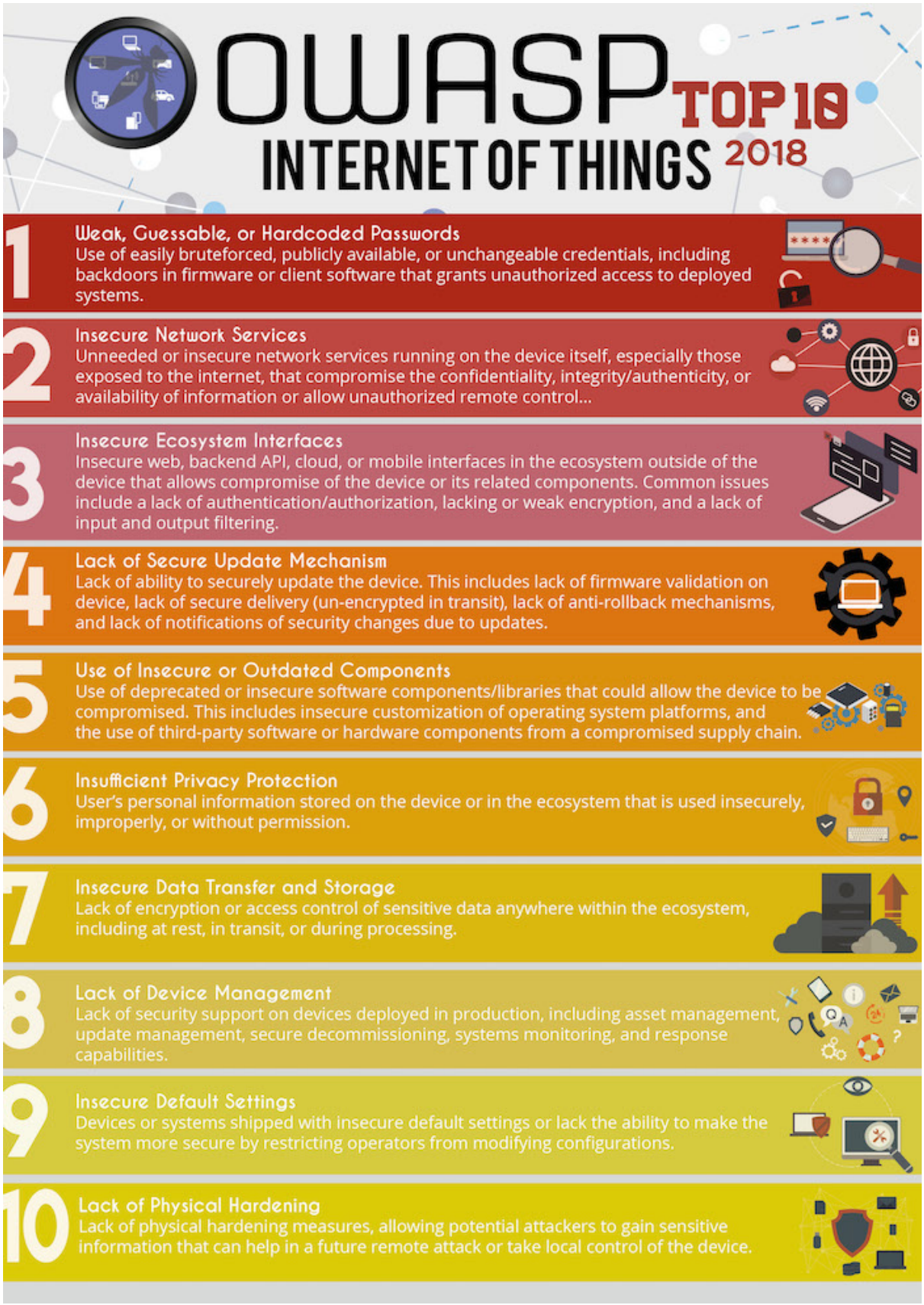


Figure 2.2: OWASP Top 10 Internet of Things.[72]

2.3.1 Weak, guessable, or hard coded passwords

Numerous IoT devices are shipped with default passwords that are weak and easy to guess or crack. Frequently, users either do not change these passwords or are not provided with the means to modify them.[23]

2.3.2 Insecure Network Services

The presence of unnecessary or insecure network services on the IoT device, particularly those exposed to the internet, can jeopardize the confidentiality, integrity, authenticity, or availability of information, and may also enable unauthorized remote control.[72]

2.3.3 Insecure Ecosystem Interfaces

The presence of insecure ecosystem interfaces, such as web, backend API, cloud, or mobile interfaces outside of the device, can compromise the security of the device and its associated components. Common vulnerabilities include a lack of authentication and authorization, insufficient or weak encryption, and inadequate input and output filtering.[34]

2.3.4 Lack of Secure Update Mechanism

The inability to securely update the device is a significant issue that arises due to several factors, including the absence of firmware validation on the device, insecure delivery (unencrypted during transit), the absence of anti-rollback mechanisms, and a lack of notifications regarding security changes resulting from updates.[72]

2.3.5 Use of insecure or outdated components

Over time, software and components become outdated, and vendors may discontinue manufacturing or software upgrades. In such cases, future patches may not be available, and if security vulnerabilities are discovered in such products, components, or software versions, the entire IoT solution may be exposed to a compromised environment.

The use of deprecated or insecure software or components can also provide opportunities for attackers to compromise the system, particularly when utilizing third-party software/libraries or features obtained from a compromised supply chain.[43]

2.3.6 Insufficient privacy protection

An IoT device and its ecosystem may contain a lot of personal data. It can be used insecurely, improperly, or without permission.[23]

2.3.7 Insecure data transfer and storage

Is the failure to encrypt or restrict access to sensitive data. For most IoT devices it is a major problem. Due to performance leak, they will not be able to use secured protocols or have custom security systems.[26]

2.3.8 Lack of device management

The maintenance of IoT devices after deployment is often inadequate. This includes the management of updates, assets, system monitoring, and other related tasks.[23]

2.3.9 Insecure Default Settings

Devices or systems that are shipped with insecure default settings or lack the ability to enhance security by restricting operators from modifying configurations pose a significant risk.[72]

2.3.10 Lack of physical hardening

Can be seen as one of the most obvious problems when it comes to IoT devices. Poor configu ration and weaknesses can allow attackers to gain sensitive information or could leave an IoT device exposed for a future remote attack and device takeover.[34]

2.4 Vulnerability Detection methods in IOT

In this section, we will introduce some of the tools used to detect vulnerabilities in software. These tools and techniques are instrumental in identifying potential vulnerabilities in a system that malicious actors could exploit to compromise the security of the system or the platform on which it operates.

2.4.1 Dynamic analysis

Refers to a technique used in vulnerability detection that focuses on identifying vulnerabilities that occur during the execution or run-time of an application. Unlike static analysis, which examines the source code or binaries without executing them, dynamic analysis involves installing the application and simulating user inputs such as inputs, touches, and clicks.[80]

2.4.2 Automated static analysis

The automated static analysis vocabulary comprises false positives, true positives, and false negatives. A false positive happens when a tool reports the existence of a fault that does not actually exist. On the other hand, a false negative occurs when a fault is present but remains undetected due to the imperfection of static analysis tools, which may fail to identify all errors. A true positive, on the other hand, is when a tool correctly identifies a real defect in the product under analysis.
Examples of automated static analysis tools include FindBugs, Programming Mistake Detector (PMD), and CheckStyle. However, these tools have a significant disadvantage, which is their tendency to produce a high rate of false positives.[16]
Numerous static analysis approaches have been developed in various research areas aimed at detecting Buffer Overflow (BOF) vulnerabilities. These approaches can be classified into six main categories: inference technique, analysis sensitivity, analysis granularity, soundness, completeness, and language.[29]

2.4.3 Fuzzing

It is a method of detecting security vulnerabilities in software by inputting invalid or random data and observing the resulting behavior for unexpected errors or vulnerabilities. Data generation is key to fuzzing, and there are three types of fuzzing: black box, white box, and gray box. Fuzzing can also be categorized by its data generation methods: random, mutation-based, generation-based, and direction-based. Random fuzzing sends completely random input data to the program under test and is useful for testing how a program reacts to large or invalid input data. Mutation-based fuzzing generates new variants of input data based on existing samples and heuristics, while generation-based fuzzing generates program inputs according to some specifications. Direction-based fuzzing uses program control flow to direct the fuzzing process. There are many research and tools available for fuzzing, including Sulley, SPIKE, and Peach.[16]

2.4.4 Web Application Scanners

A web application scanner is a powerful tool that automatically examines web applications for security vulnerabilities. These scanners are not only capable of detecting web application specific vulnerabilities, but they can also identify evidence of software coding errors like unchecked input strings and buffer overflows.
Nowadays, there are numerous web application scanners available in the market. Some of the widely used commercial web application scanners include AppScan, Web Inspect, Hailstorm, and Acunetix Web Vulnerability Scanner (WVS). Open-source web application scanners like Paros and Pantera are also popular among security professionals.[36]
Web application scanners follow a systematic approach to assess the security of an application. They crawl through the web pages of an application and perform penetration testing, which involves actively attacking the application to identify potential vulnerabilities.

2.4.5 Brick

The Binary Run-time Integer Based Vulnerability Checker (BRICK) is an efficient approach that detects integer-based vulnerabilities in software during runtime. This approach produces highly accurate results with minimal false positives and negatives.
The BRICK process consists of three stages. Firstly, it converts the binary code into an intermediate representation known as Valgrind EXecution environment (VEX) on Valgrind. Secondly, the dynamic binary instrumentation framework Valgrind intercepts integer-related statements at runtime and records the necessary information. Finally, vulnerability detection and localization are performed using a set of checking schemes.[16]

2.4.6 Machine Learning

2.4.6.1 Definition of machine Learning

Machine learning is a vital branch of artificial intelligence (AI) that empowers systems to learn and develop experience automatically without the need for explicit programming. Unlike traditional programming methods, machine learning focuses on computer programs that access and use data to improve their performance over time. The learning process begins with feedback, examples, practical experience, or guidance to search for trends in data, which is then used to make informed decisions in the future.
The primary goal of machine learning is to enable computers to automatically learn and adapt without requiring human intervention or assistance. This technology can be applied to any domain where the relationship between input and output is dependent on data. Machine learning algorithms learn from data, making it crucial to choose the right data and prepare it effectively to solve the problem at hand.
The different machine learning models used for forecasting based on input data are categorized as supervised learning, unsupervised learning, and reinforcement learning. Each model has its unique strengths and weaknesses, and choosing the right one depends on the specific requirements of the problem being solved.[76]

2.4.6.2 Different types of Machine Learning

Figure 2.3 illustrates the three methods by which a machine can learn.[59]

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

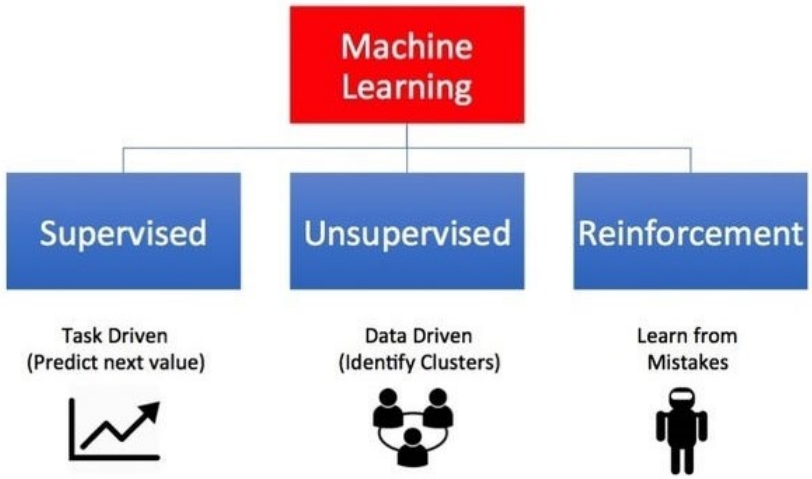


Figure 2.3: Types of Machine Learning.[41]

1. **Supervised Learning :** is a well-defined method of learning from examples. To start with, the learner is presented with two sets of data: training data and testing data. The ultimate goal of the learner is to use the training data as input and accurately identify the unlabeled values in the test data. In supervised learning, each pattern is a combination of an input data set and a target value The primary objective of the supervised learning algorithm is to assess and create a presumed function of the training data.[76]

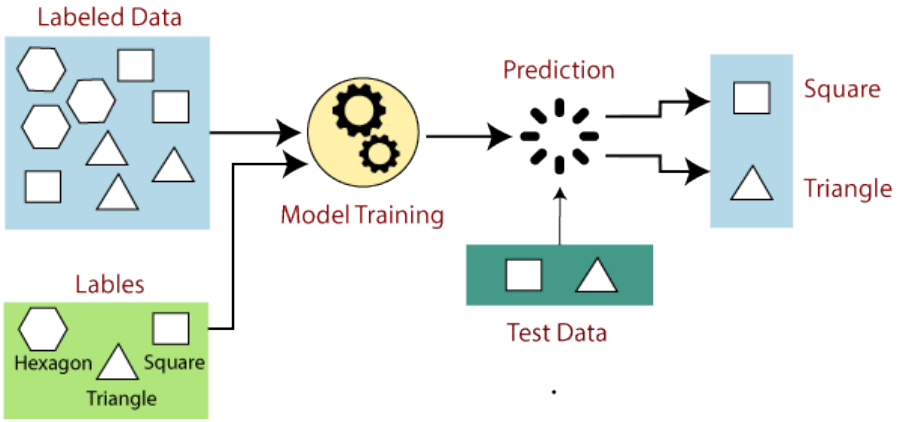


Figure 2.4: Supervised learning .[49]

Supervised learning can be further divided into two types of problems:

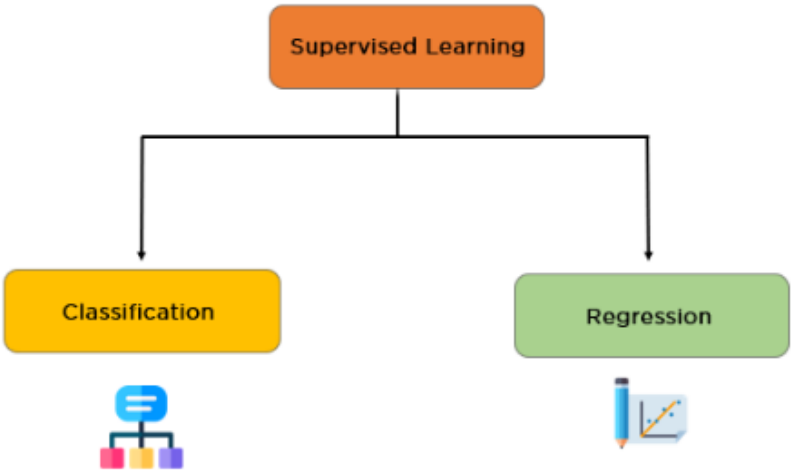


Figure 2.5: Types of supervised learning.[21]

- **Regression :** tasks require a learning machine to analyze one dependent variable and several independent variables, to estimate and comprehend their relationships in a system. This method is useful for forecasting and prediction, especially in determining trends and patterns in the data.[18]
  - Linear Regression
  - Regression Trees

  - Non-Linear Regression
  - Bayesian Linear Regression
  - Polynomial Regression

- **Classification :** the process of Classification involves training a computer program on a dataset and utilizing the training to categorize data into distinct class labels. The algorithm is designed to predict discrete values such as true—false, male—female, spam—not spam, etc.[82]
  - Spam Filtering
  - Random Forest
  - Decision Trees
  - Logistic Regression
  - Support vector Machines Classification

2. **Unsupervised learning :** is based on the principle of learning and improving through trial and error. In contrast to supervised learning, unlabeled data is used in unsupervised learning, and the correct answer is not shown to the machine. Instead, various algorithms are employed to allow the machine to establish relationships by examining and observing the data. Unsupervised learning is a data-driven (clustering) learning algorithm that clusters unlabeled data into different groups. The key aspect of unsupervised learning is the availability of a massive amount of data. The more data available, the easier it is for a machine to identify and examine trends that could lead to meaningful clustering.[59]

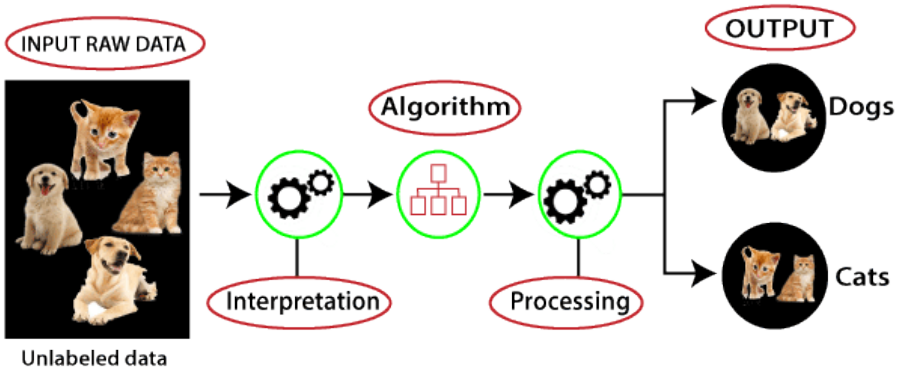


Figure 2.6: Unsupervised learning.[50]

The unsupervised learning algorithm can be further categorized into two types of problems:

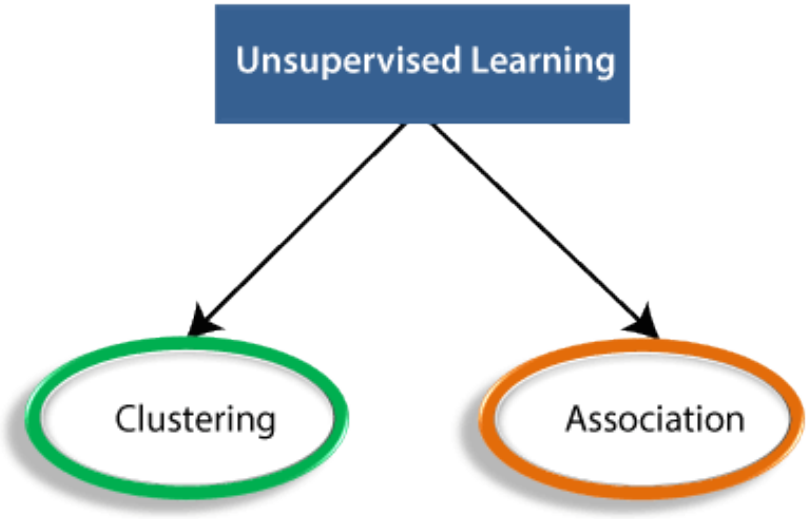


Figure 2.7: Types of unsupervised learning .[50]

- **Clustering :** is a technique for grouping objects based on their similarities. It involves categorizing data objects into clusters, with each cluster comprising objects that share the most similarities and differ significantly from the objects in other clusters. The objective of cluster analysis is to identify commonalities between data objects and group them accordingly based on the presence or absence of those commonalities.[50]

- **Association :** is a type of unsupervised learning that utilizes rules to identify relationships among variables within a dataset. These methods are often applied in market basket analysis and recommendation engines,[30] For instance, shopping stores may use algorithms based on the Association technique to examine the connection between the sale of one product and another based on customer behavior.[65]

3. **Reinforcement learning :** is a distinct field of machine learning focused on taking action in a given situation to maximize rewards. It involves different software

and machines that determine the optimal behavior or path in a particular situation. Reinforcement learning differs from supervised learning, in which the correct answer is provided to the model, but in reinforcement learning, there is no explicit answer. Instead, the reinforcement agent must decide on the best course of action to complete the given task. By learning from its experiences, the agent can improve its performance in the absence of a training dataset. Reinforcement learning has practical applications in various industries, such as robots in industrial automation.[76]

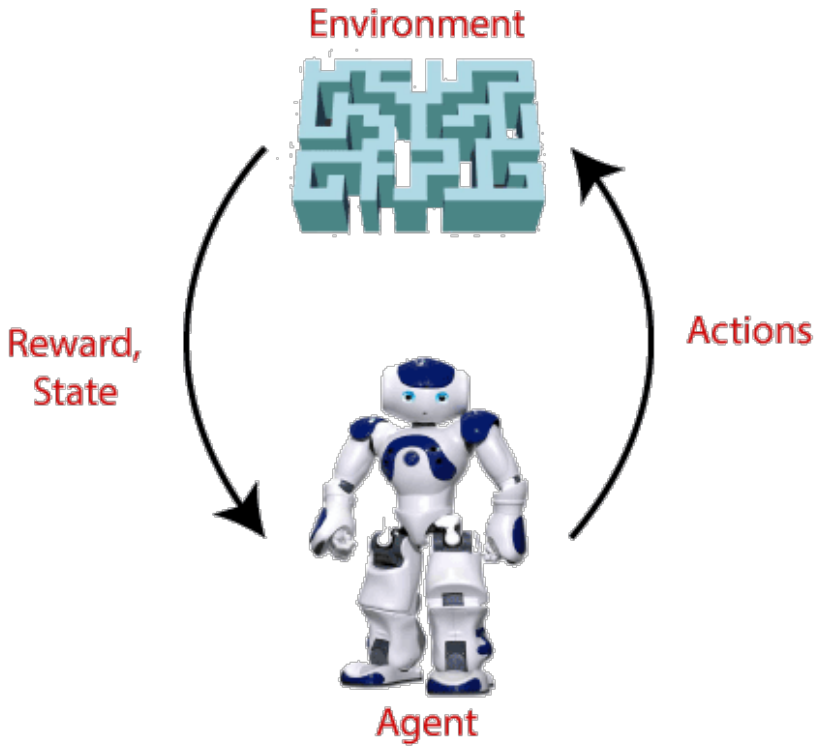


Figure 2.8: Reinforcement learning.[48]

2.4.7 Deep Learning

2.4.7.1 Definition of Deep Learning

Deep Learning is a type of machine learning that uses Artificial Neural Networks (ANNs) as its fundamental building blocks. ANNs are designed to mimic the way the human brain works, comprising of numerous interconnected computational units or 'neurons' that work together to solve a task.

Unlike traditional machine learning algorithms, which rely on manually engineered features, Deep Learning is capable of automatically learning useful features directly from raw data. The term 'Deep' refers to the multiple layers of neurons that are used to process the input data, allowing for more complex and abstract representations to be learned. Deep Learning has become increasingly popular due to the availability of powerful hardware and the ability to train large models efficiently. This approach has shown superior performance in handling complex non-linear processes, thanks to its ability to leverage self-organization and interactions between small units, resulting in better fault tolerance and adaptability to new data.[74]

2.4.7.2 How it works ?

Deep networks are organized into layers of neurons, with each network typically consisting of an Input Layer, one or more Hidden Layers, and an Output Layer. The connections between each pair of neighboring layers are known as weights, which are responsible for determining the strength of the connections. The nodes within each layer, commonly referred to as "neurons," have no direct association with each other. Figure 2.9 provides an illustration of a standard deep neural network architecture.

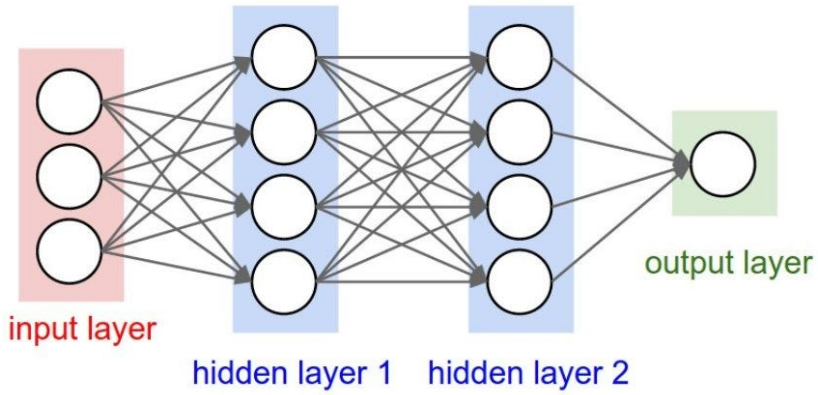


Figure 2.9: The architecture of a Deep Learning model.[90]

Deep learning is an advanced computing system that utilizes various techniques from the field of machine learning. It employs a large number of nonlinear neurons arranged in multiple layers to extract and transform variable values from the input vector, creating multiple levels of abstraction to represent the data. The optimization of weight

parameters and bias between neighboring layers is critical to the learning process, as it helps evaluate the accuracy of the model and allows for better adaptation to the learning data. When the model achieves maximum precision with optimal parameters, it can be generalized for real data. The quantity and quality of the training data determine the degree of learning and, consequently, the accuracy of the models obtained.[39]

2.4.7.3 Some deep learning methods

1. **Deep Neural Networks (DNNs):** A Deep Neural Network (DNN) represents one of the most efficient ways to solve problems and is classified as deep learning. A DNN processes input information and aggregates this data into key learning points. They are a collection of neurons organized in a sequence of multiple layers, known as Multilayer Perceptrons (MLPs). DNNs are distinguished from traditional Artificial Neural Networks (ANNs) by their depth and the number of layers and nodes (neurons) that comprise the network. When an ANN has two or more hidden layers, it is known as a deep neural network. DNNs attempt to model complex data architectures by combining various nonlinear transformations.[13]

2. **Convolutional Neural Networks (CNNs) :** are a type of neural network used in deep learning for image processing and computer vision tasks. They are composed of neurons that self-optimize through learning, similar to traditional ANNs. However, unlike traditional ANNs, CNNs have specialized layers that perform convolutions on the input data, allowing them to extract meaningful features from images. From the input raw image vectors to the final output of the class score, the entire network still expresses a single perceptive score function, with the last layer containing loss functions associated with the classes. All of the regular tips and tricks developed for traditional ANNs still apply to CNNs.[71]

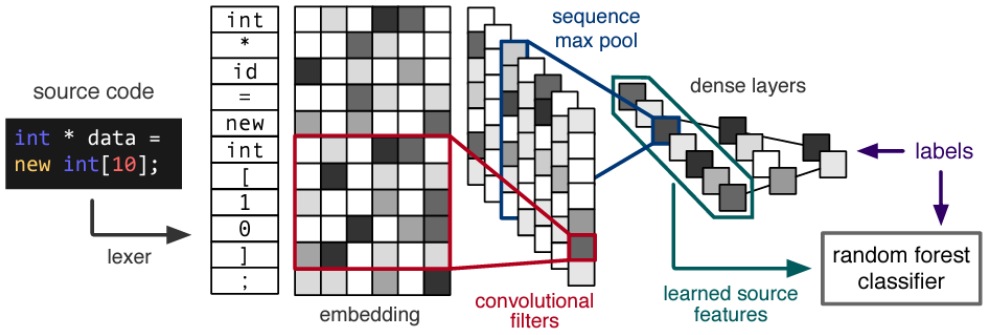


Figure 2.10: Convolutional Neural Networks(CNNs) architecture .[87]

3. **Recurrent neural networks (RNNs) :** are neural networks that draw inspiration from the functioning of biological neurons in the human brain, which are considered the center of reflection and must sometimes memorize certain events to use them later before making a decision. Traditional neural networks do not have this property, which motivates the operation of a recurrent neural network.
   RNNs are feed-forward networks with an internal state (or memory) that takes into account all or part of the data previously seen by the network, in addition to the current input data, to adapt their decision. The key idea behind these networks is the deployment of a recurrent computation using loops in the network architecture. The network's output is a combination of its internal state (memory of inputs) and the last input. At the same time, the internal state changes to integrate this new input data, allowing information to persist in memory.

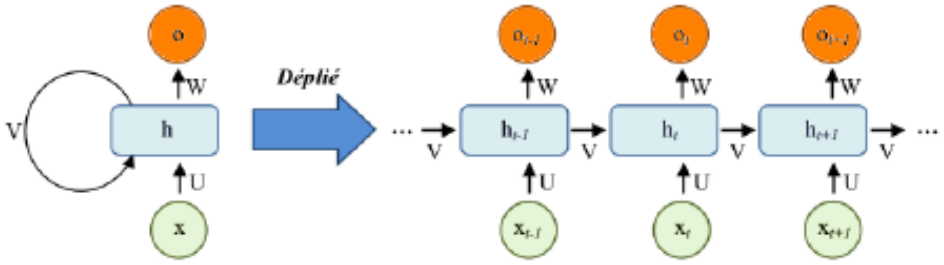


Figure 2.11: Recurrent neural networks(RNNs) architecture.[39]

4. **Long short-term memory networks, or LSTMs :** are a specialized type of recurrent neural network (RNN) that excel at learning long-term dependencies. Unlike traditional RNNs, which often struggle to remember information over long

periods of time, LSTMs are explicitly designed to avoid this issue. They possess a default capacity for retaining information for extended periods, making them particularly useful in natural language processing applications. While LSTMs share the chain-like structure of RNNs, their repeating module is distinct. It consists of four interacting layers, each with its own unique role. Figure 11 provides a visual representation of an LSTM node.

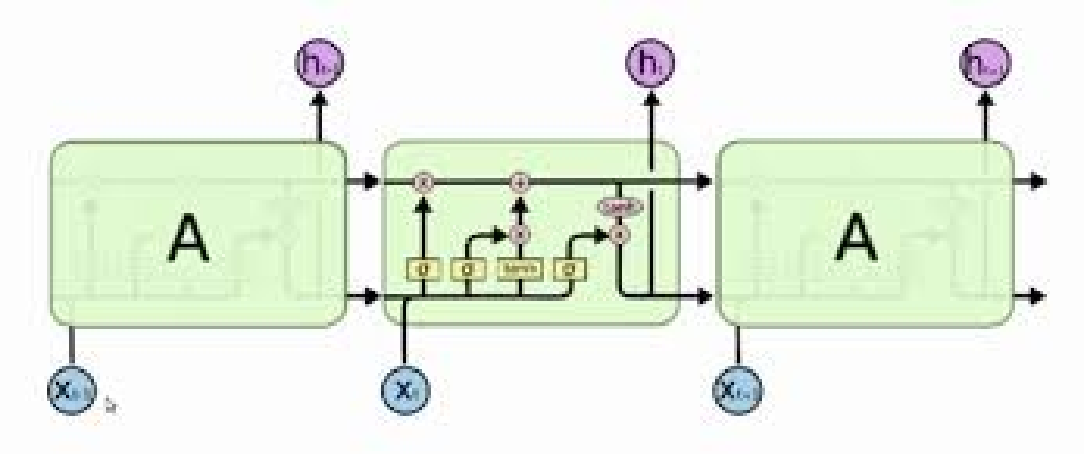


Figure 2.12: Long short-term memory networks node.[75]

2.5 Related Work

In the following some other research works that have explored similar topics.

Ref	Article of	Year	Description	Dataset	Model	Result
[70]	A deep learning-based static taint analysis approach for IoT software vulnerability location	2019	The article presents a proposed approach for detecting and locating vulnerabilities in IoT software. The method combines static taint analysis and deep learning techniques to achieve effective vulnerability identification. The approach emphasizes tracking the flow of tainted data within the program and analyzing how it influences the program's behavior.	Code Gadget Database = (CWE-119), (CWE-399)	Bidirectional Long Short Term Memory (BLSTM)	97,32% 97,21%
[68]	Identifying Vulnerable IoT Applications using Deep Learning	2020	This paper presents an approach for the identification of vulnerable IoT applications using deep learning algorithms.	Corpus1 Corpus2	MLP	92.59% 87.03%
[9]	iDetect for vulnerability detection in Internet of Things operating systems using Machine Learning	2022	This paper aims to use Machine Learning (ML) to create a tool called iDetect for detecting vulnerabilities in C/C++ source code of IoT operating systems.	IoT OS + SARD	RF CNN RNN	96.8% 94% 85.6%

Table 2.1: Table shows Brief overview of related work.

2.5.1 A deep learning based static taint analysis approach for IoT software vulnerability location

This article [70] by Weina Niu a, Xiaosong Zhang b c, Xiaojiang Du d, Lingyuan Zhao b, Rong Cao b, Mohsen Guizani e , proposes automatic detection methods software vulnerabilities, but they often suffer from a high false negative rate.They presented a deep learning-based static taint analysis approach for automatically locating vulnerabilities in Internet of Things (IoT) software. This approach alleviates the need for manual analysis and enhances detection accuracy by considering the program context. They designed selection rules for taint from the difference file between the source program and its patched version, and use static taint analysis to determine taint propagation paths. Furthermore, a two-stage Bidirectional Long Short-Term Memory (BLSTM) model is employed to detect and locate software vulnerabilities. They evaluated their approach using the Code Gadget Database, which includes two types of vulnerabilities in C/C++ programs: buffer error vulnerability (CWE-119) and resource management error vulnerability (CWE-399). Experimental results demonstrate that their proposed approach achieves an accuracy of 0.9732 for CWE-119 and 0.9721 for CWE-399, outperforming three other models (RNN, LSTM, and BLSTM) in terms of accuracy. Moreover, their approach achieves lower false negative and false positive rates compared to other detection methods.

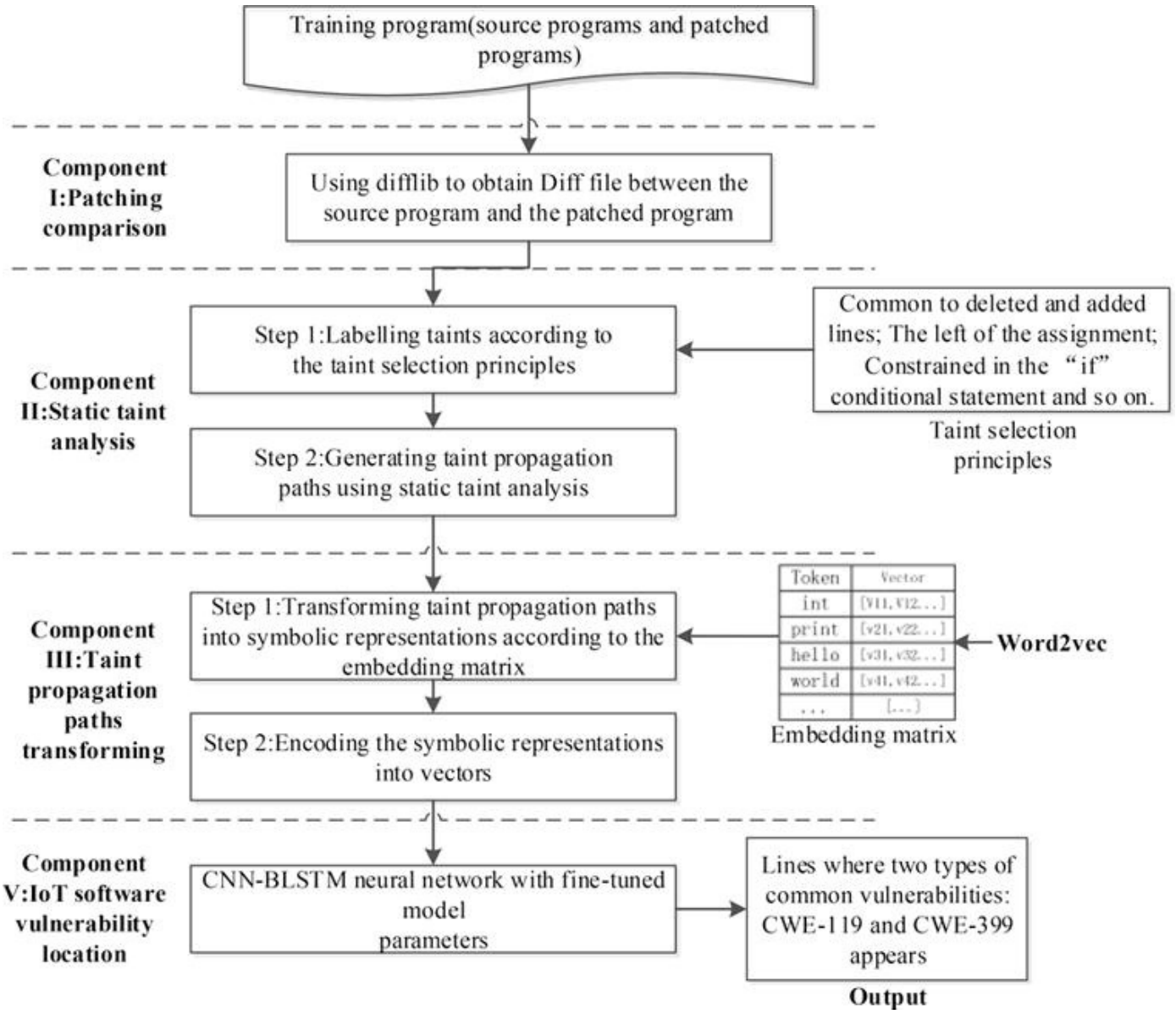


Figure 2.13: Technique flow chart of their proposed approach.[70]

2.5.2 Identifying Vulnerable IoT Applications using Deep Learning

This work [68] by Hajra Naeem, Manar H. Alalf , proposes a method for using deep learning algorithms to identify vulnerable IoT applications. The method focuses on identifying a specific type of vulnerability that can result in the leakage of sensitive information, which is accomplished through taint flow analysis. To achieve this, the source code of IoT applications is analyzed to recover tokens and their frequencies, as well as tainted flows. Two modules, Token2Vec and Flow2Vec, are developed to transform these tokens

and flows into vectors. These vectors are then used to train a deep learning algorithm to create a model for identifying tainted applications. The effectiveness of this approach was evaluated using two datasets, and the experiments showed that by combining tainted flows features with the base benchmark that uses token frequencies only, the accuracy of the prediction models was improved significantly. Specifically, the proposed approach improved the accuracy of the prediction models from 77.78% to 92.59% for Corpus1 and from 61.11% to 87.03% for Corpus2.

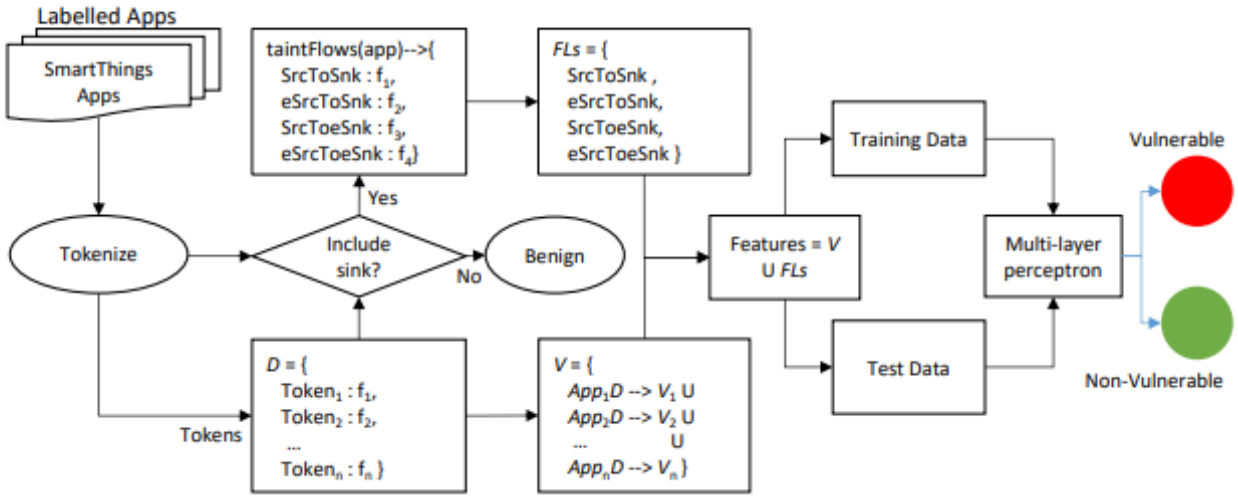


Figure 2.14: Overview of the approach.[68]

2.5.3 iDetect for vulnerability detection in internet of things operating systems using machine learning

The paper [9] by Abdullah Al-Boghdady, Mohammad El-Ramly, Khaled Wassif , proposes using Machine Learning (ML) to create a tool called iDetect, which can detect vulnerabilities in the C/C++ source code of IoT OSs. The authors used the source code for 16 releases of IoT OSs, along with the Software Assurance Reference Dataset (SARD), to create a labeled dataset of vulnerable and benign code using the Common Weakness Enumeration (CWE) vulnerabilities present in IoT OSs as a reference. The study showed that only a subset of CWEs is present in the C/C++ source code of low-end IoT OSs. The authors trained three ML models for vulnerability detection: Random Forest (RF),

Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN). During the testing phase, RF gave the highest accuracy for binary and multiclass classification, so it was chosen as iDetect's ML classifier. The study evaluated iDetect on an unseen dataset of 322 code snippets taken from TinyOS, and iDetect achieved superior results compared to all three Static Analysis Tools (SATs) used to collect the training dataset. iDetect achieved a macro-averaged F1 score (mF1) of 98.5% and a weighted-average F1 score (wF1) of 98% for multiclass classification, and an F1 score (F1) of 97.8% for binary classification.

2.6 Conclusion

This chapter covers methods to prevent vulnerabilities in IoT systems, including the use of machine learning and deep learning. These areas have gained popularity due to their ability to provide solutions to problems through data analysis. Additionally, we highlight relevant work on vulnerability detection in IoT software. In the next chapter, we will design a system that uses deep learning algorithm which has been trained for the IoT operating system

Chapter 3

Conception

3.1 Introduction

In this chapter, we will present a system for detecting vulnerabilities in IoT operating systems using deep learning techniques, specifically a Convolutional Neural Network (CNN) model. Our objective was to design a reliable and efficient system that could accurately identify vulnerabilities. This chapter is divided into two parts: the first part describes the global architecture of our system, while the second part details the design of the proposed model, including each design unit,in addition to the UML diagrams that explain our system.

3.2 System presentation

We will provide a global overview of our system, describing its objectives and architecture.

3.2.1 System objective

The objective of the system designed to detect vulnerabilities in IoT operating systems using deep learning is to improve the accuracy and efficiency of vulnerability detection. This is accomplished by:

- Leveraging deep learning models and datasets of Common Weakness Enumeration (CWE) vulnerabilities specific to IoT operating systems.

- The system can analyze IoT operating system source code in C and C++ languages to identify patterns and anomalies that may indicate the presence of vulnerabilities.

- Determine the type and location of vulnerabilities in the code, providing a more effective way to detect and address potential security threats in IoT operating systems.

3.2.2 Global Architecture of the system

In the following Figure 3.1 the architecture of our system that describes the stages of the system.

1. **Input :** takes as input the source code of an IoT operating system written in C and C++ languages. This source code is the target for vulnerability detection.

2. **Deep Learning Model :** the input code source is passed through a deep learning model. The deep learning model is designed to analyze and process the source code in order to detect vulnerabilities. It utilizes advanced deep learning algorithms and techniques to analyze the code, enabling it to identify distinctive patterns and anomalies that could potentially indicate the presence of vulnerabilities.

3. **Output :** the output of the deep learning model is a prediction indicating whether the input code source is classified as "non-vulnerable" or "vulnerable". This prediction is based on the patterns and anomalies identified by the deep learning model during the analysis of the code.
   We have two possible outputs:

   - **Non-vulnerable :**this output indicates that the code is secure and free from vulnerabilities.

   - **Vulnerable :**in case of vulnerability detection, the system provides additional information regarding the type of vulnerability and its location within the code.

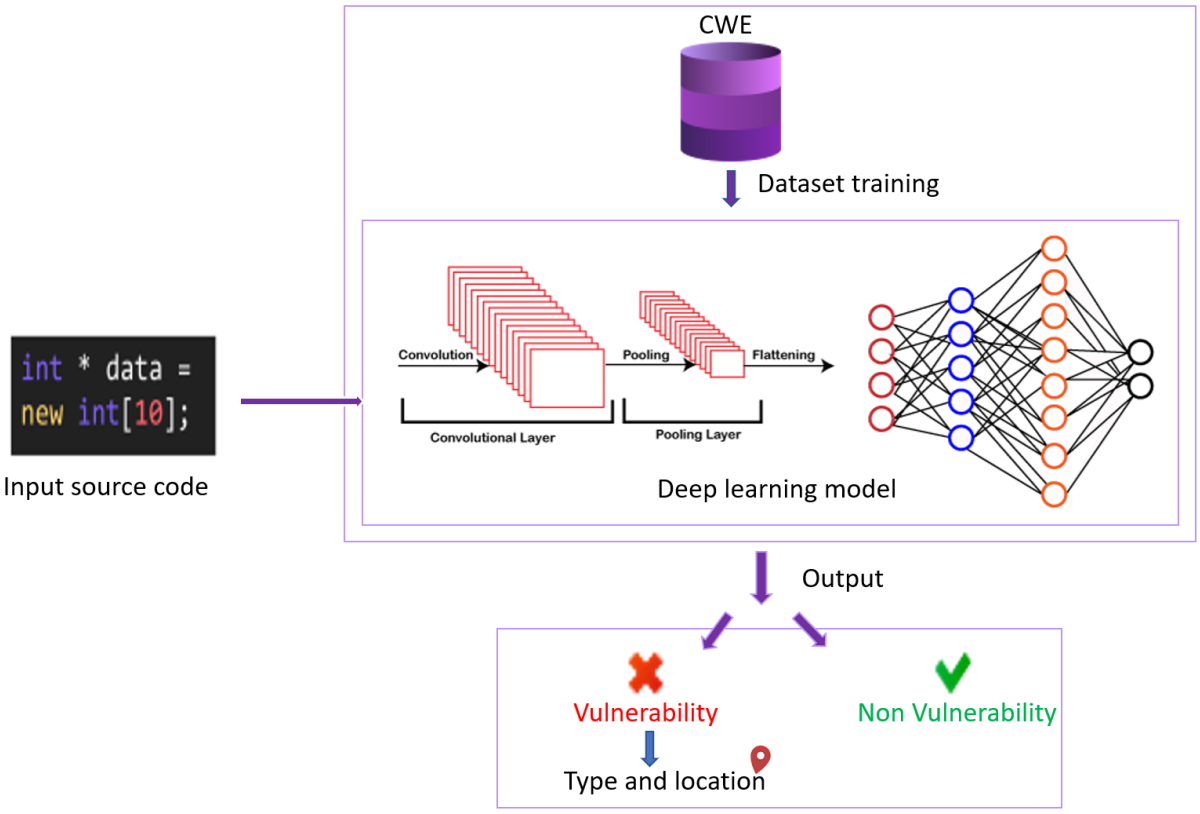


Figure 3.1: Global Architecture of the system.

3.3 Detailed system design

In Figure 3.2 below, we present an overview of the detailed architecture of our system, and we will explain the steps involved.

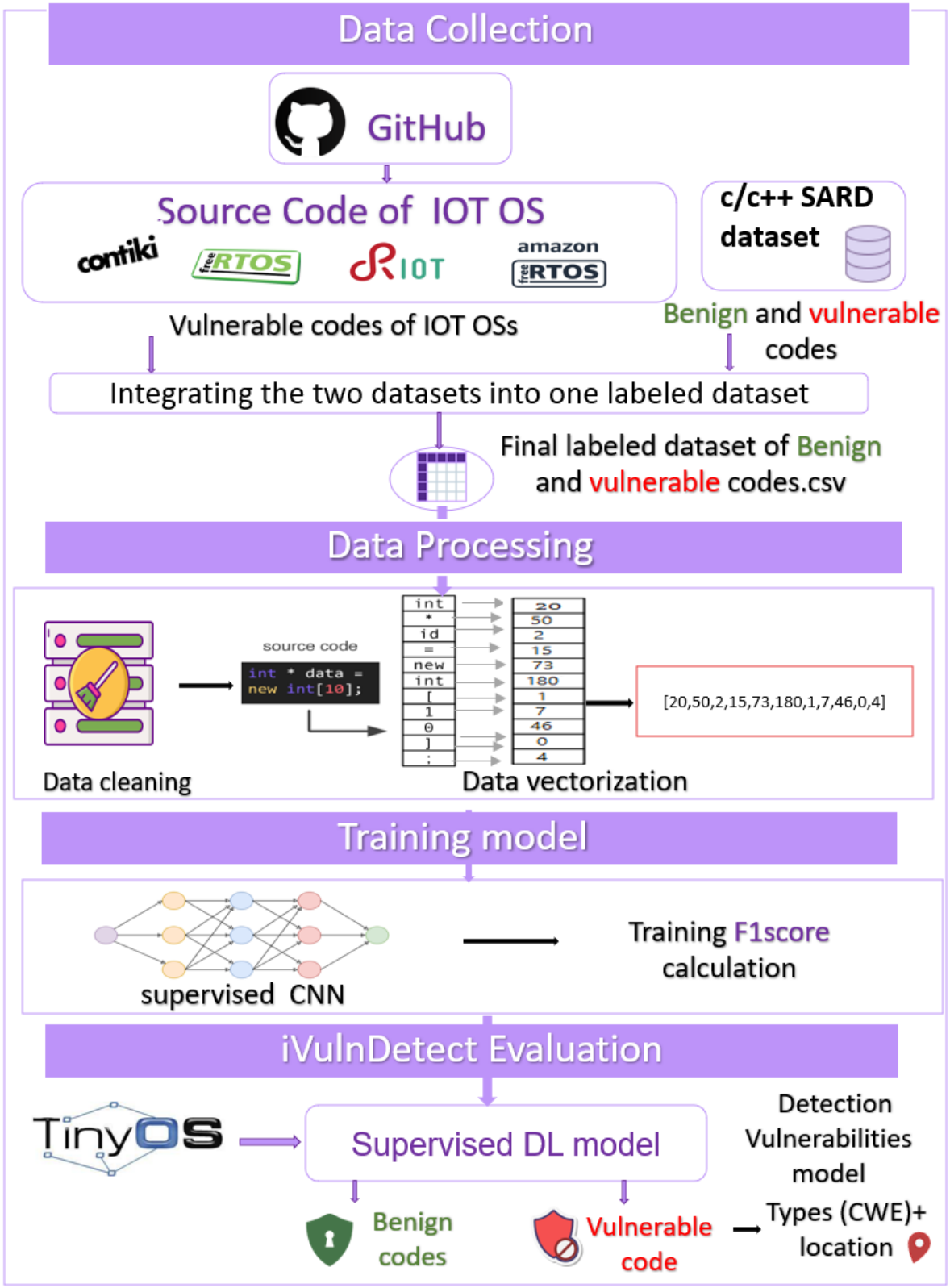


Figure 3.2: The System Architecture with Detailed Components.

3.3.1 Data collection

Data collection is a crucial initial stage of our system that involves the acquisition of a dataset comprising code vulnerable to exploitation by attackers. This is a challenging task that requires a comprehensive search for suitable data sources. After extensive research, we discovered a public dataset on GitHub that was introduced in a 2022 article [9]. The dataset contains vulnerable C/C++ codes for IoT operating systems, and it combines two distinct sources using Common Weakness Enumeration (CWEs) as a benchmark for identifying and labeling the vulnerabilities, covering 54 types of CWEs discovered in the IoT OSs case study [10]. The first source comprises genuine weak code snippets extracted from the source code of sixteen versions of four distinct IoT operating systems (namely, RIOT, Contiki, FreeRTOS, and Amazon FreeRTOS) up to their 2020 releases, as shown in table 3.1. The second source is SARD, a well-documented semi-synthetic C/C++ dataset that enables the creation of a categorized dataset from both benign and weak code.

IoT OS	Release or Version
RIOT	RIOT R. 2015.09 RIOT R. 2017.07 RIOT R. 2019.07 RIOT R. 2020.04
Contiki	Contiki R. 2.4 Contiki R. 2.7 Contiki R. 3.0 Contiki R. 3.1
FreeRTOS	FreeRTOS v. 6.0.3 FreeRTOS v. 9.0.0 FreeRTOS v. 10.0.0 FreeRTOS v. 10.3.1
Amazon FreeRTOS	Amazon FreeRTOS v. 1.0.0 Amazon FreeRTOS v. 1.4.0 Amazon FreeRTOS v. 201908 Amazon FreeRTOS v. 202007

Table 3.1: The releases of IoT operating systems that were used for dataset collection.[9]

3.3.2 Data processing

Data processing is a critical step in any data-driven project because it ensures that the resulting data is accurate, complete, and consistent. After data has been collected, it must be processed to extract useful insights and make it suitable for analysis. This typically involves cleaning and transforming the data, as well as applying machine learning techniques to generate models that can be used for analysis.

1. **Data cleaning :** is an essential step to ensure that the data is accurate, complete, and consistent, which leads to more reliable insights. The goal is to transform the data into a usable format that can be analyzed effectively. In our project, we cleaned the input data. For instance, we removed comments, irrelevant characters, and duplicates using Regular Expression (re) and normalized the text.

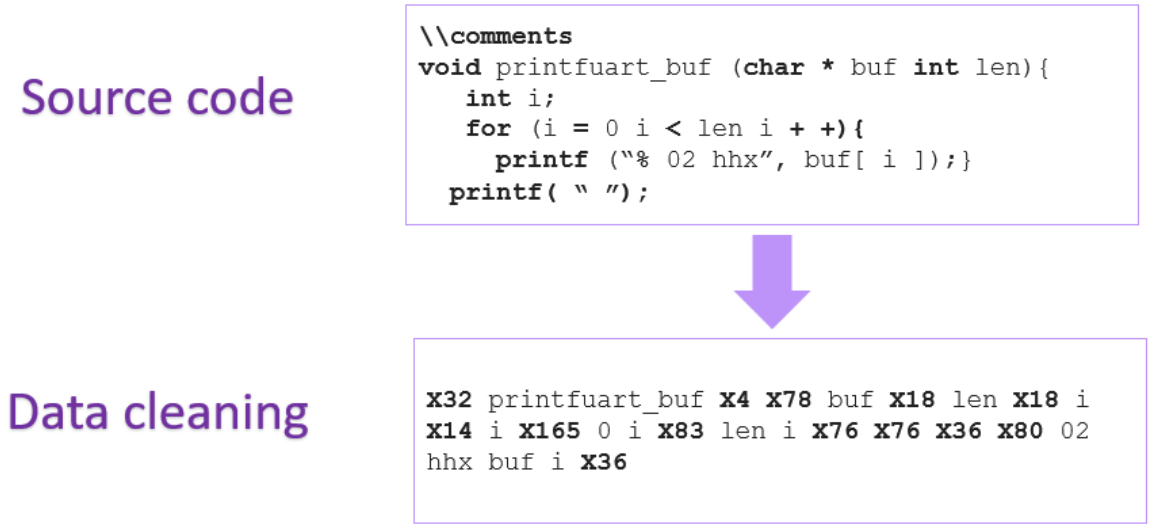


Figure 3.3: Example of data cleaning.

2. **Data Vectorization :** is a process of converting text data into a numerical format that can be easily understood by deep learning algorithms. This involves transforming text data into a set of features that can be represented as vectors in a high-dimensional space. There are several techniques such as bag-of-words, TF-IDF (Term Frequency-Inverse Document Frequency), and word embeddings to convert the cleaned text data into numerical representations.

   In our project, after cleaning the data, we created a vocabulary of C/C++ code. This vocabulary was used in the text vectorization process, which we performed using the TextVectorization layer provided by the Keras API of TensorFlow.

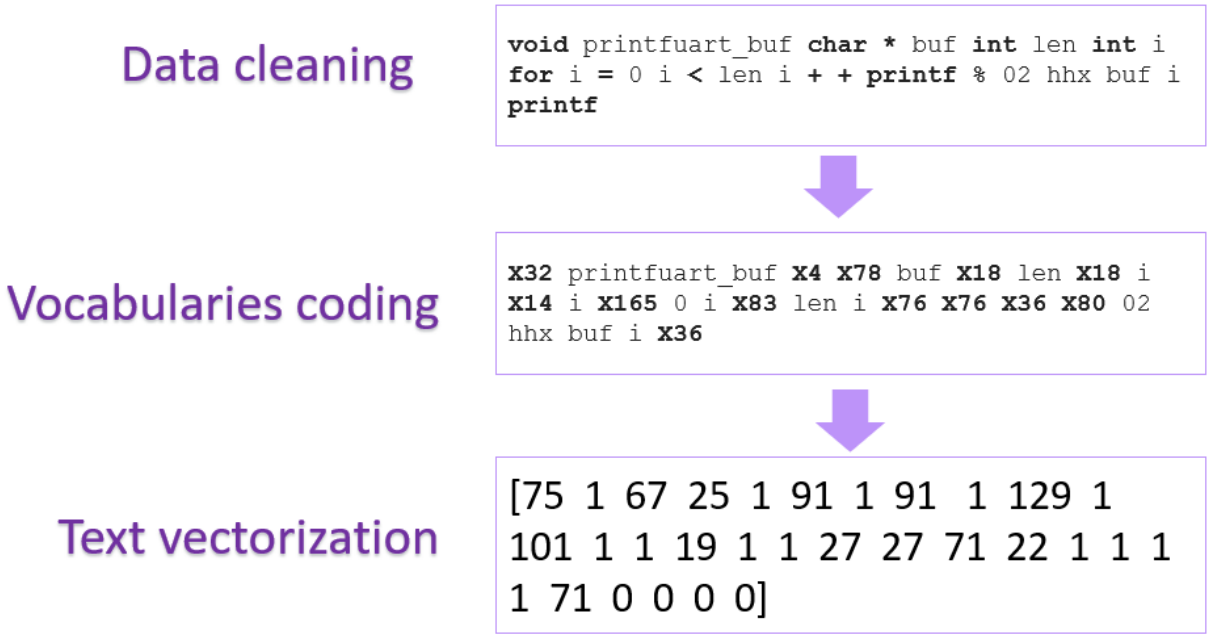


Figure 3.4: Example of data vectorisation.

3.3.3 Training model

Training a model refers to the process of teaching a deep learning model to make predictions or perform a specific task based on a given dataset.

We employed a Convolutional Neural Network (CNN) to train our model, as it outperformed other machine learning algorithms in terms of accuracy. The CNN model allowed us to extract meaningful features from the data and learn complex patterns in the input. We trained the model through multiple steps, the following Figure 5 shows the CNN model steps.

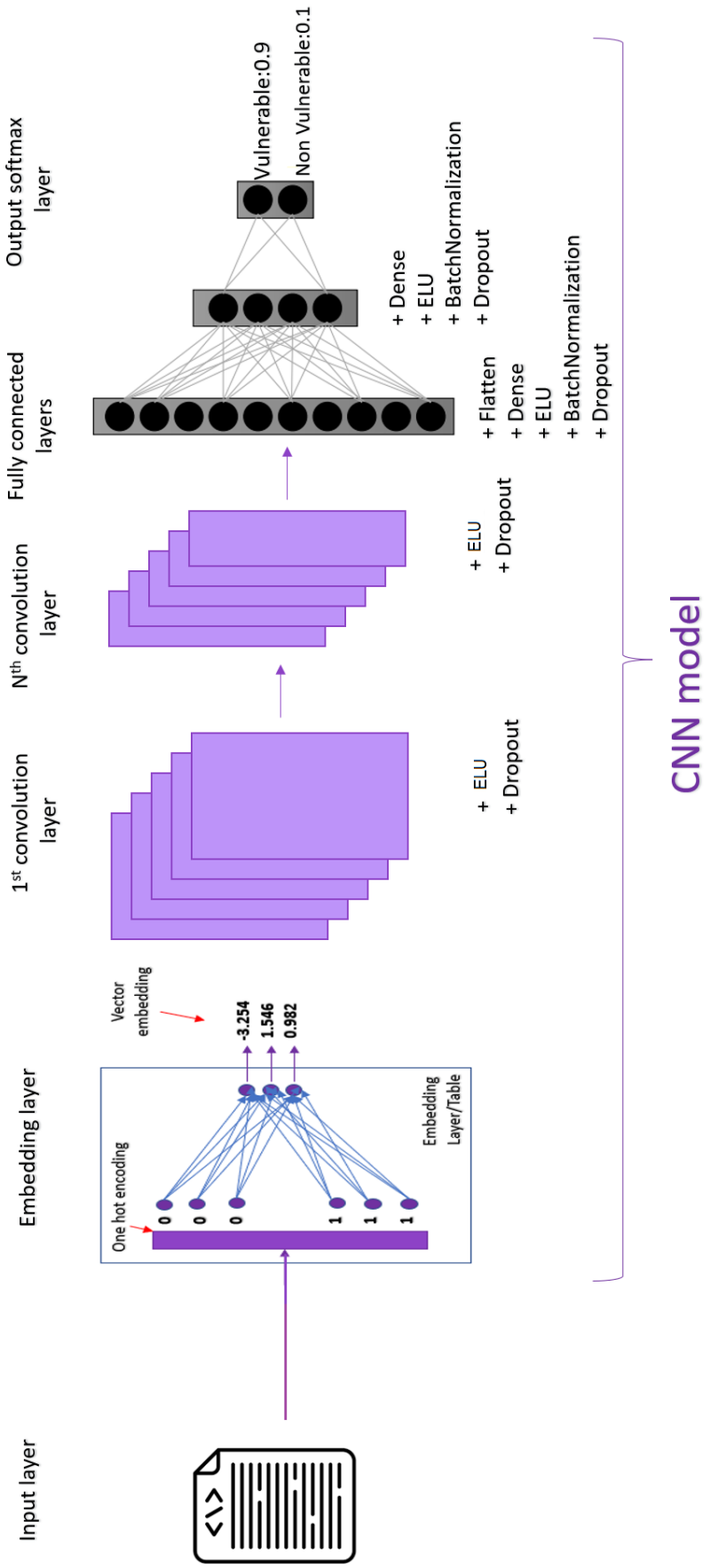


Figure 3.5: Convolutional Neural Network (CNN) model steps.

3.3.4 iVulnDetect evaluation

We evaluated our model with the TinyOS operating system, which is specifically designed for Internet of Things (IoT) applications, to assess the performance and effectiveness of our model

3.4 Design by UML (Unified Modeling Language)

3.4.1 Use Case Diagram

The diagram depicts the structure of the key functionalities required by users of the system. It illustrates the relationships between the users (actors) and the objects within the system. The diagram captures the main interactions and tasks that users can perform to accomplish their goals within the system.

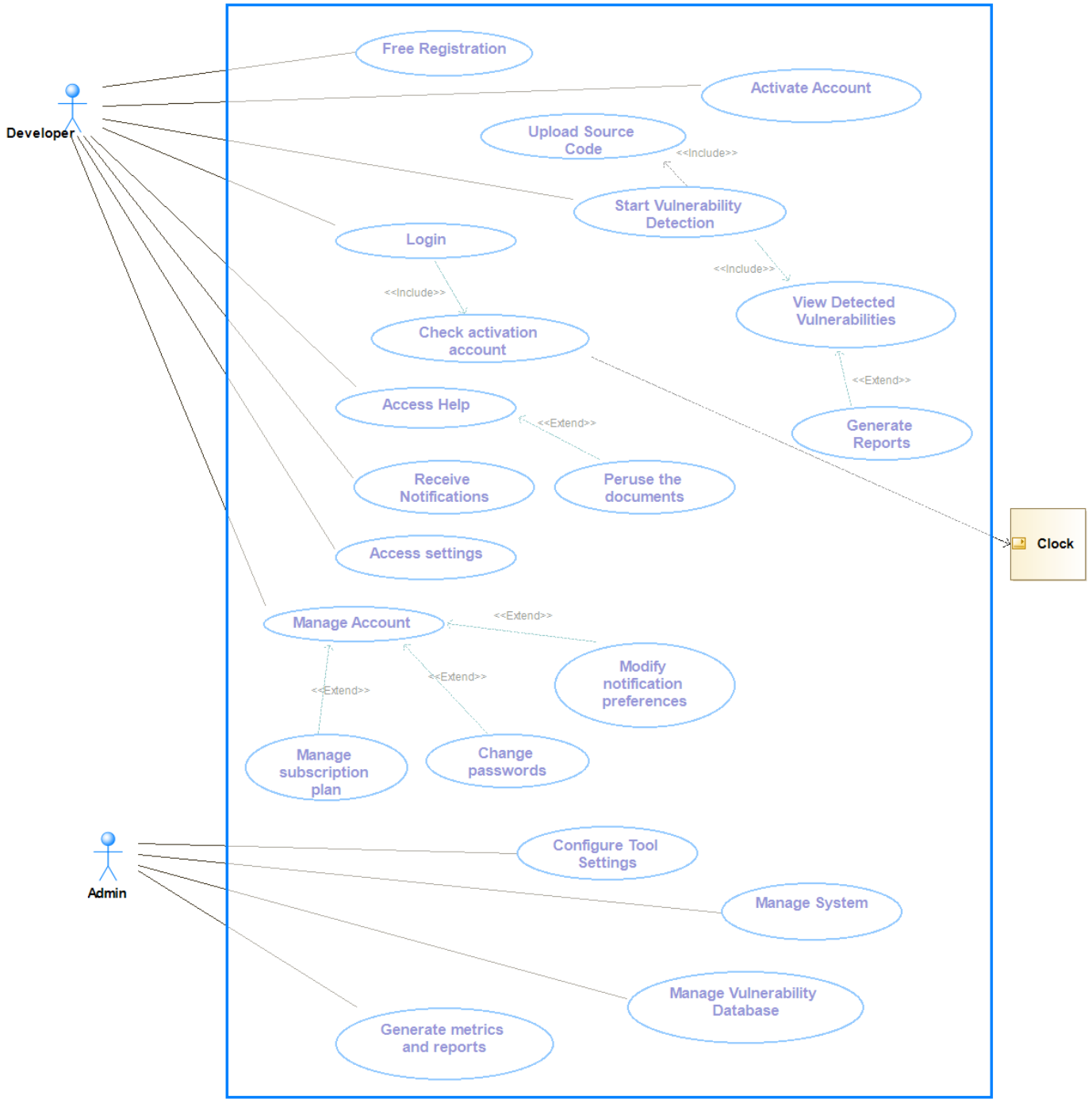


Figure 3.6: Use Case Diagram.

3.4.2 Class Diagram

A class diagram is a powerful tool used to visually represent the static structure of a system. It consists of classes, their attributes, methods, and the associations between them. The diagram provides a clear and concise overview of the system's entities and

their interactions.

In the following diagram, we have illustrated the class relations of our system, showcasing the key classes and their relationships. Each class represents a distinct entity or concept within the system, while the associations between classes depict the relationships and dependencies among them.

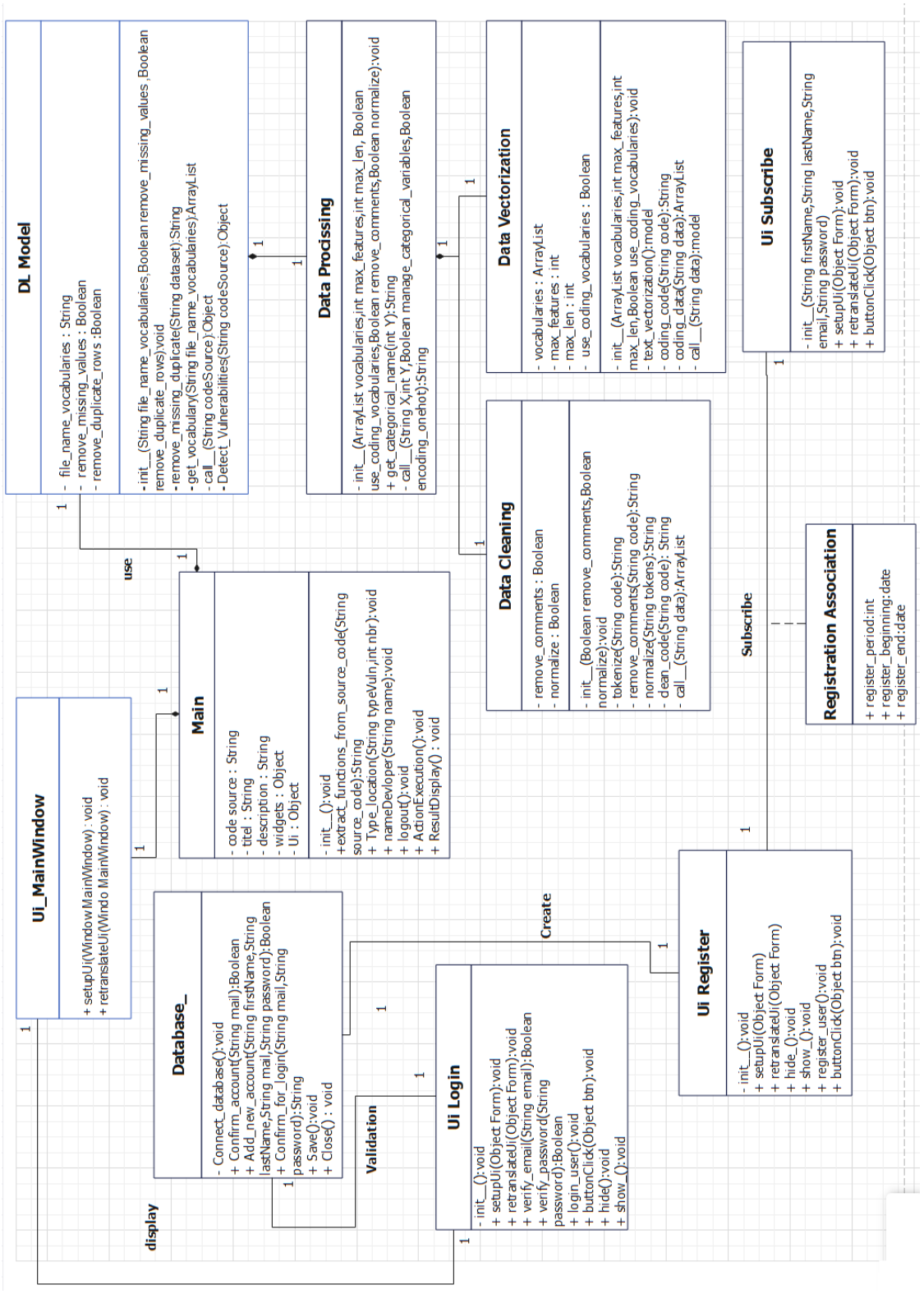


Figure 3.7: class diagram.

3.4.3 Sequence Diagram

3.4.3.1 Sequence Diagram for Login

The following sequence diagram shows the interactions required for developer login.

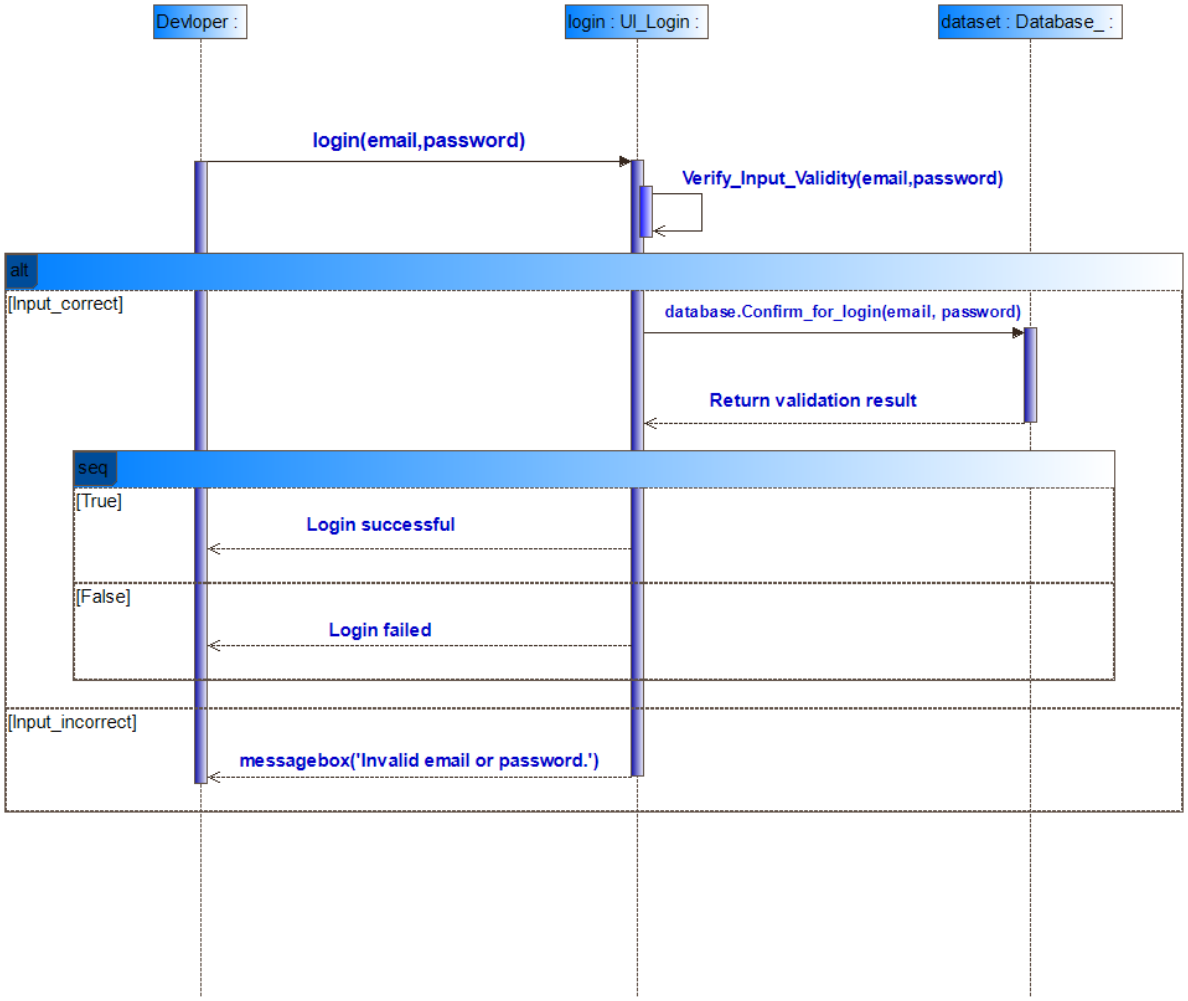


Figure 3.8: Sequence Diagram for Developer Login.

3.4.3.2 Sequence Diagram for Register

This diagram explains how to create a new account in our software.

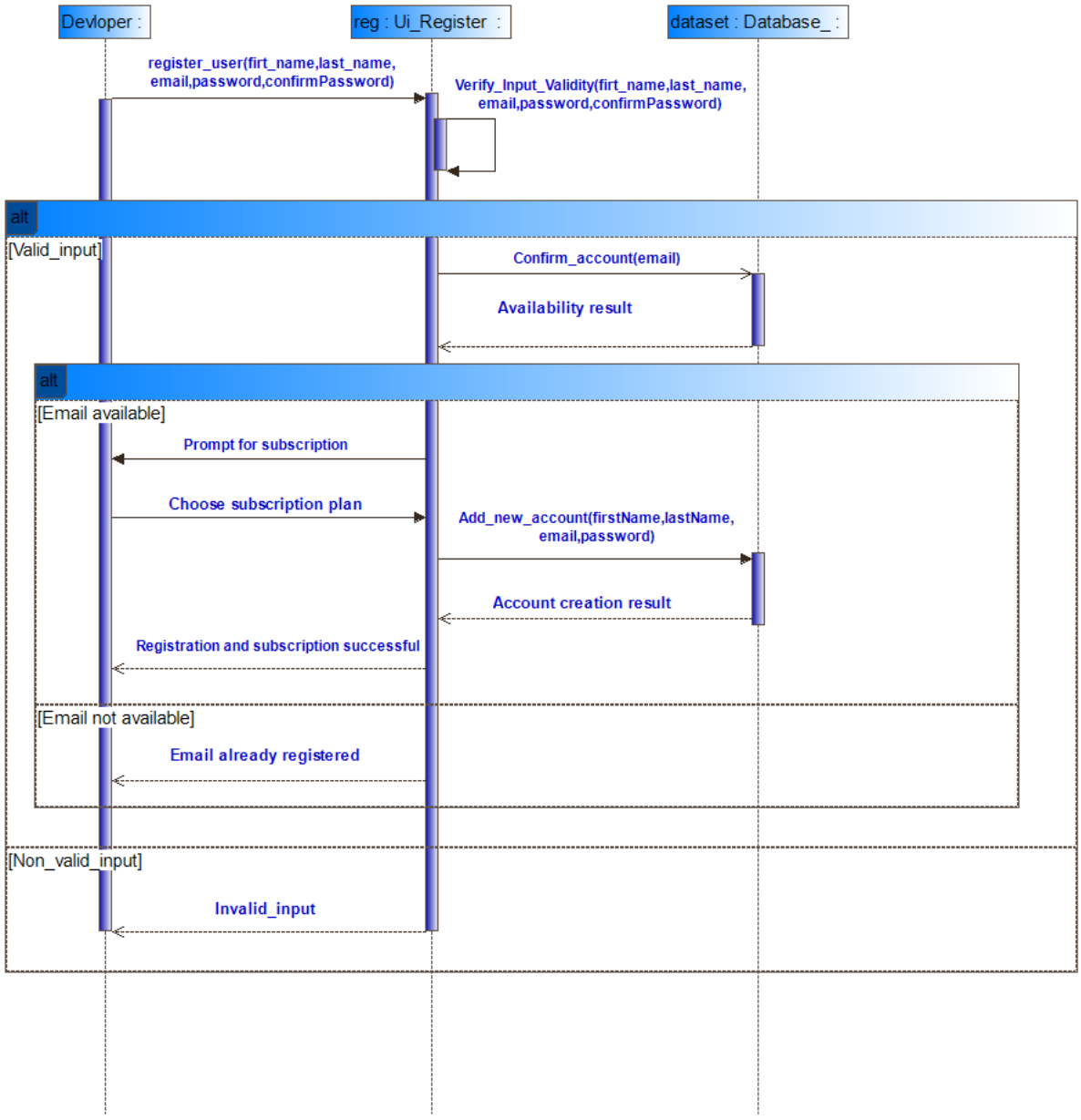


Figure 3.9: Sequence Diagram for Developer Register.

3.4.3.3 Sequence Diagram for Detection Vulnerabilities

This diagram shows how to discover vulnerabilities and all related operations in our program, where the source code is entered and using the deep learning model, vulnerabilities are discovered and their types and location are determined, and the system ensures a view of the detected vulnerabilities with tips to avoid them.

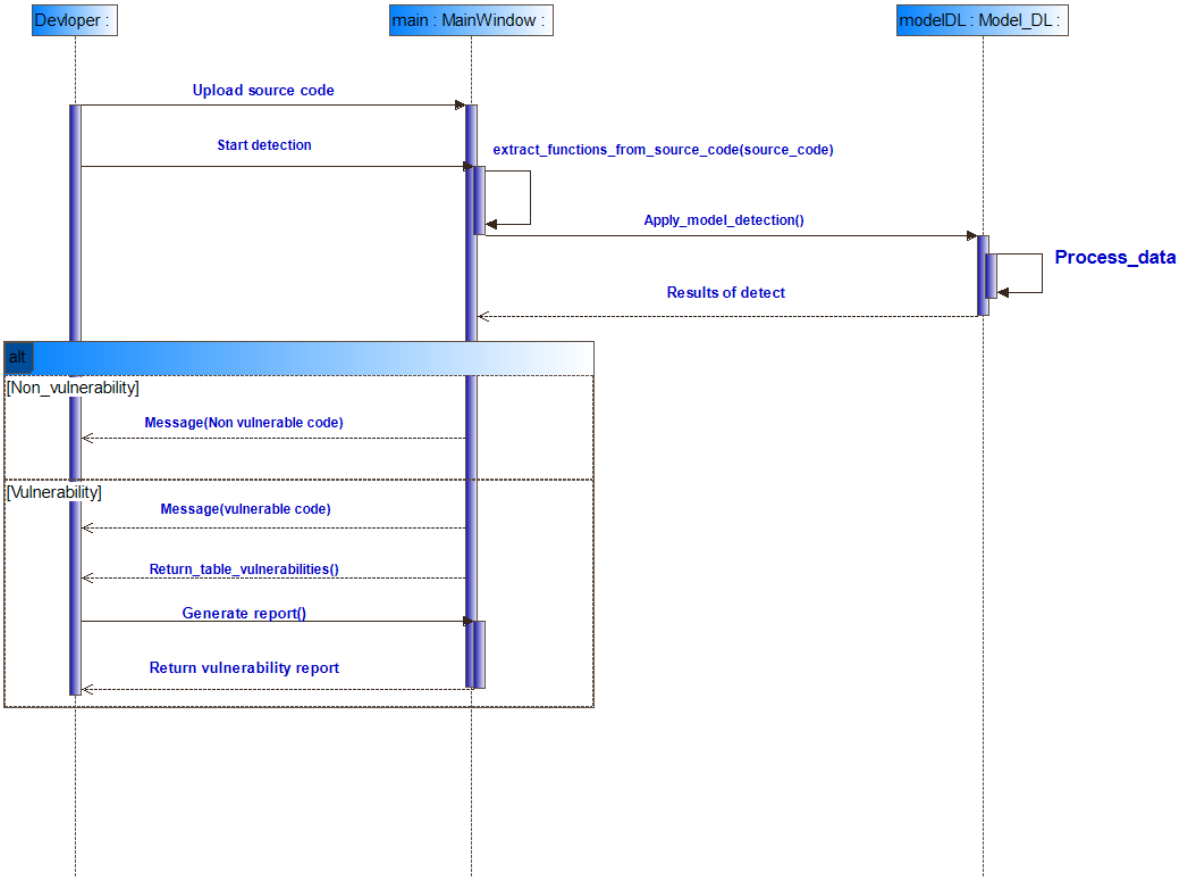


Figure 3.10: Sequence Diagram for Detection Vulnerabilities.

3.5 Conclusion

In this chapter, we presented the design of our system, where we presented the comprehensive(global) and detailed design and UML diagrams that explain our system. We also clarified the purpose of our system and detailed and explained its components (data collection, data preparation and deep learning model).

Chapter 4

Implementation and results

4.1 Introduction

In this chapter, we will introduce the working environment, programming language, tools, and libraries used in building our system. We will also show the components of the dataset used and all the stages involved in creating the CNN deep learning model, and in the end we will show a comparison of our results with other related work, as well as some pictures of our system interface and how it works.

4.2 Development Environment

The subsequent section provides an overview of the working environment, programming languages, and software tools employed in the development process.

4.2.1 Programming Languages

4.2.1.1 Python

Python is an interpreted, object-oriented, high-level programming language known for its dynamic semantics. It offers a range of built-in high-level data structures and features dynamic typing and dynamic binding. These characteristics make Python highly suitable for Rapid Application Development and as a scripting or glue language for connecting

existing components. The language's simplicity and readability contribute to reduced program maintenance costs. Python supports modules and packages, promoting code modularity and reusability. The Python interpreter and extensive standard library are freely available in source or binary form for all major platforms and can be distributed without charge.[79]

Figure 4.1: Python logo.

4.2.2 Software Tools

4.2.2.1 Google Colab

Google Colab, also known as Colaboratory or Colab, is a web-based environment developed by Google Research. It offers a convenient platform for writing and executing Python code directly through a web browser. Primarily designed for machine learning, data analysis, and educational purposes, Colab provides users with a powerful toolset.[37] Colab stands out as an improved version of Jupyter Notebook, offering enhanced features and functionality. It was initially introduced by Google to provide free access to GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units), enabling users to harness accelerated computing resources without any cost. This aspect makes Colab particularly attractive for building machine learning and deep learning models.[78]

Figure 4.2: Google Colab logo.

Advantage of Colab:[78]

- Its ease of use and zero-configuration setup.
- Users can write and execute Python code seamlessly.
- Benefiting from the collaborative nature of Jupyter notebooks.
- Interactive tutorials to learn machine learning and neural networks.
- Import datasets from external sources such as Kaggle.
- Save your Notebooks to Google Drive.
- Import Notebooks from Google Drive.
- Free cloud service, GPUs and TPUs.
- Import or publish directly from/to GitHub.

Additionally, Colab's integration with Google Cloud APIs suggests a long-term perspective of building a customer base for Google Cloud services.

4.2.2.2 PyCharm

PyCharm is an Integrated Development Environment (IDE) specifically designed for Python programming. Developed by JetBrains, PyCharm provides a comprehensive set of tools and features that facilitate efficient and productive Python development. PyCharm supports various frameworks and libraries commonly used in Python development, such as Django, Flask, and NumPy, providing built-in project templates and integration for streamlined development workflows.[51][77]

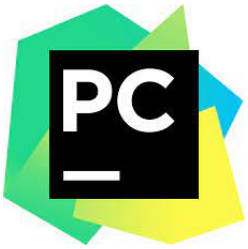


Figure 4.3: PyCharm logo.

4.2.3 Design Tools

4.2.3.1 Qt Design

Qt Designer is a powerful tool provided by Qt for designing and constructing graphical user interfaces (GUIs) using Qt Widgets. It allows users to create and customize windows or dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, providing a visual representation of the interface. Additionally, Qt Designer enables users to test the GUIs with various styles and resolutions.[81]

Figure 4.4: Qt Design logo.

4.2.4 Database Tools

4.2.4.1 XAMPP

XAMPP stands as one of the most widely used PHP development environments. It is a comprehensive Apache distribution that includes MariaDB, PHP, and Perl, providing a seamless and hassle-free setup. XAMPP is an open-source package designed with the goal of simplifying the installation and usage process, making it accessible even to users with limited technical expertise.[20]

Figure 4.5: XAMPP logo.

4.2.5 Conception Tools

4.2.5.1 Modelio

Modelio is a versatile open-source modeling environment tool that offers comprehensive support for the latest industry standards, such as UML 2 and BPMN 2. It provides the flexibility to extend its functionalities by incorporating additional modules. The Modelio offers a wide range of free and open-source modules, enabling features like code management (generation and reverse), adherence to various modeling standards (including TOGAF, SysML, SoaML), and document generation.[35]

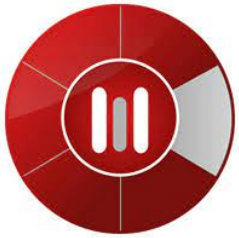


Figure 4.6: Modelio logo.

4.3 Library Tools

4.3.1 TensorFlow

TensorFlow is a widely adopted open-source platform designed for the development of end-to-end Machine Learning applications. As a symbolic math library, TensorFlow utilizes dataflow and differentiable programming to facilitate a range of tasks, particularly in training and inference of deep neural networks. It provides developers with a comprehensive ecosystem comprising tools, libraries, and community resources to create sophisticated machine learning applications. Currently, the most famous deep learning library in the world is Google's TensorFlow.[53]

Figure 4.7: Tensorflow logo.

4.3.2 Keras

Keras[3], a deep learning API, is implemented in Python and operates on the TensorFlow machine learning platform. It was designed with the primary goal of enabling rapid experimentation. The ability to swiftly transition from an idea to obtaining results is crucial for conducting effective research.

Figure 4.8: Keras logo.

Keras is characterized by the following attributes:

- **Simplicity:** Keras minimizes the cognitive load on developers, allowing them to concentrate on the core aspects of the problem at hand. While it simplifies the development process, Keras is not oversimplified, ensuring that essential complexities are not overlooked.

- **Flexibility:** Keras follows the principle of progressive disclosure of complexity. Simple workflows can be executed quickly and effortlessly, while more intricate workflows can be accomplished through a clear progression that builds upon the foundational knowledge you have acquired.

- **Power:** Keras delivers robust performance and scalability, making it a suitable choice for organizations and companies with demanding requirements. Renowned entities such as NASA, YouTube, and Waymo employ Keras for their deep learning needs, attesting to its industry-strength capabilities.

4.3.3 PySide6

PySide6 is a Python binding of the cross-platform GUI toolkit Qt. Applications built with PySide6 will run on any platform supported by Qt & Python including Windows, OS X, Linux, iOS and Android. PySide is the official binding for Qt on Python and is now developed by The Qt Company itself.[98]

Figure 4.9: PySide6 logo.

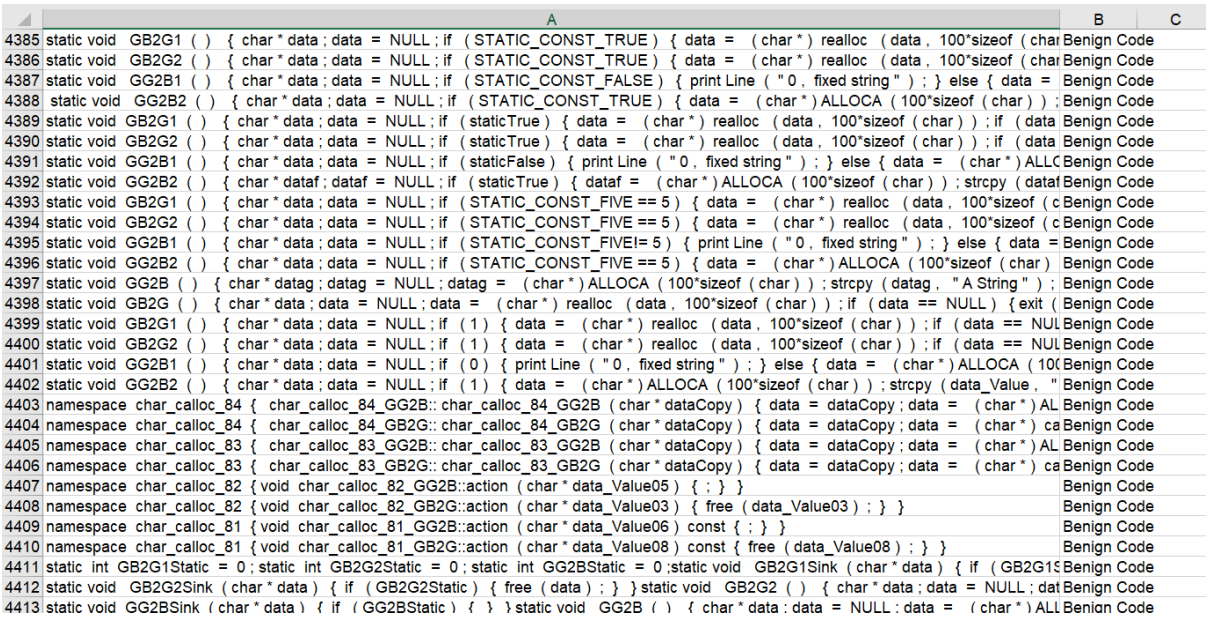
4.4 Structure of Data

4.4.1 Dataset Description

The dataset consists of two main categories: Benign Codes and Vulnerable Codes.

- **Benign Codes**

  Benign codes refer to non-vulnerable code samples. These are code snippets or segments that do not contain any security vulnerabilities, the figure represents a portion of the benign codes.



	A	B	C
4385	static void GB2G1 ( ) { char * data ; data = NULL ; if (STATIC_CONST_TRUE) { data = (char *) realloc (data , 100*sizeof (char	Benign	Code
4386	static void GB2G2 ( ) { char * data ; data = NULL ; if (STATIC_CONST_TRUE) { data = (char *) realloc (data , 100*sizeof (char	Benign	Code
4387	static void GG2B1 ( ) { char * data ; data = NULL ; if (STATIC_CONST_FALSE) { print Line (" 0 , fixed string ") ; } else { data =	Benign	Code
4388	static void GG2B2 ( ) { char * data ; data = NULL ; if (STATIC_CONST_TRUE) { data = (char *) ALLOCA (100*sizeof (char)) ;	Benign	Code
4389	static void GB2G1 ( ) { char * data ; data = NULL ; if (staticTrue) { data = (char *) realloc (data , 100*sizeof (char)) ; if (data	Benign	Code
4390	static void GB2G2 ( ) { char * data ; data = NULL ; if (staticTrue) { data = (char *) realloc (data , 100*sizeof (char)) ; if (data	Benign	Code
4391	static void GG2B1 ( ) { char * data ; data = NULL ; if (staticFalse) { print Line (" 0 , fixed string ") ; } else { data = (char *) ALL	Benign	Code
4392	static void GG2B2 ( ) { char * dataf ; dataf = NULL ; if (staticTrue) { dataf = (char *) ALLOCA (100*sizeof (char)) ; strcpy (dataf	Benign	Code
4393	static void GB2G1 ( ) { char * data ; data = NULL ; if (STATIC_CONST_FIVE == 5) { data = (char *) realloc (data , 100*sizeof (c	Benign	Code
4394	static void GB2G2 ( ) { char * data ; data = NULL ; if (STATIC_CONST_FIVE == 5) { data = (char *) realloc (data , 100*sizeof (c	Benign	Code
4395	static void GG2B1 ( ) { char * data ; data = NULL ; if (STATIC_CONST_FIVE!= 5) { print Line (" 0 , fixed string ") ; } else { data =	Benign	Code
4396	static void GG2B2 ( ) { char * data ; data = NULL ; if (STATIC_CONST_FIVE == 5) { data = (char *) ALLOCA (100*sizeof (char)	Benign	Code
4397	static void GG2B ( ) { char * datag ; datag = NULL ; datag = (char *) ALLOCA (100*sizeof (char)) ; strcpy (datag , " A String ") ;	Benign	Code
4398	static void GB2G ( ) { char * data ; data = NULL ; data = (char *) realloc (data , 100*sizeof (char)) ; if (data == NULL) { exit (	Benign	Code
4399	static void GB2G1 ( ) { char * data ; data = NULL ; if (1) { data = (char *) realloc (data , 100*sizeof (char)) ; if (data == NUL	Benign	Code
4400	static void GB2G2 ( ) { char * data ; data = NULL ; if (1) { data = (char *) realloc (data , 100*sizeof (char)) ; if (data == NUL	Benign	Code
4401	static void GG2B1 ( ) { char * data ; data = NULL ; if (0) { print Line (" 0 , fixed string ") ; } else { data = (char *) ALLOCA (10(	Benign	Code
4402	static void GG2B2 ( ) { char * data ; data = NULL ; if (1) { data = (char *) ALLOCA (100*sizeof (char)) ; strcpy (data_Value , "	Benign	Code
4403	namespace char_calloc_84 { char_calloc_84_GG2B:: char_calloc_84_GG2B (char * dataCopy) { data = dataCopy ; data = (char *) AL	Benign	Code
4404	namespace char_calloc_84 { char_calloc_84_GB2G:: char_calloc_84_GB2G (char * dataCopy) { data = dataCopy ; data = (char *) ca	Benign	Code
4405	namespace char_calloc_83 { char_calloc_83_GG2B:: char_calloc_83_GG2B (char * dataCopy) { data = dataCopy ; data = (char *) AL	Benign	Code
4406	namespace char_calloc_83 { char_calloc_83_GB2G:: char_calloc_83_GB2G (char * dataCopy) { data = dataCopy ; data = (char *) ca	Benign	Code
4407	namespace char_calloc_82 { void char_calloc_82_GG2B::action (char * data_Value05) { ; } }	Benign	Code
4408	namespace char_calloc_82 { void char_calloc_82_GB2G::action (char * data_Value03) { free (data_Value03) ; } }	Benign	Code
4409	namespace char_calloc_81 { void char_calloc_81_GG2B::action (char * data_Value06) const { ; } }	Benign	Code
4410	namespace char_calloc_81 { void char_calloc_81_GB2G::action (char * data_Value08) const { free (data_Value08) ; } }	Benign	Code
4411	static int GB2G1Static = 0 ; static int GB2G2Static = 0 ; static int GG2BStatic = 0 ;static void GB2G1Sink (char * data) { if (GB2G1S	Benign	Code
4412	static void GB2G2Sink (char * data) { if (GB2G2Static) { free (data) ; } } static void GB2G2 ( ) { char * data ; data = NULL ; dat	Benign	Code
4413	static void GG2BSink (char * data) { if (GG2BStatic) { } } static void GG2B ( ) { char * data ; data = NULL ; data = (char *) ALL	Benign	Code

Figure 4.10: Part of Benign Codes.

- **Vulnerable Codes**

  The dataset includes a multi-class classification of vulnerable codes. These codes are categorized into 54 types based on the Common Weakness Enumeration (CWE) standard. CWE provides a standardized taxonomy for identifying and categorizing software weaknesses and vulnerabilities.

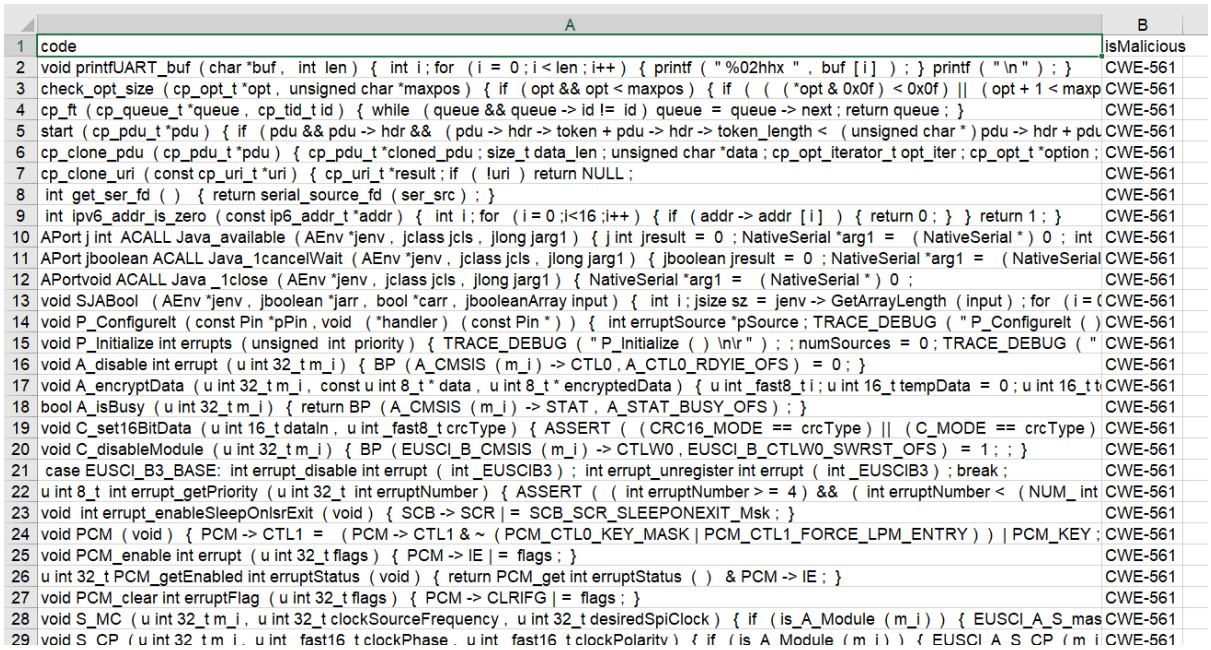
  In the table below provided that lists the 54 CWE types associated with the vulnerable codes.

CWE-ID	**Description**
CWE-561	Dead Code
CWE-398	Seven Pernicious Kingdoms (7PK) vulnerability—Code Quality
CWE-563	Assignment to Variable without Use
CWE-686	Function Call with Incorrect Argument Type
CWE-570	Expression is Always False
CWE-476	NULL Pointer Dereference
CWE-571	Expression is Always True
CWE-758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior
CWE-457	Use of Uninitialized Variable
CWE-664	Improper Control of a Resource Through its Lifetime
CWE-783	Operator Precedence Logic Error
CWE-665	Improper Initialization
CWE-190	Integer Overflow or Wraparound
CWE-467	Use of sizeof() on a Pointer Type
CWE-788	Access of Memory Location After End of Buffer
CWE-682	Incorrect Calculation
CWE-477	Use of Obsolete Function
CWE-685	Function Call with Incorrect Number of Arguments
CWE-775	Missing Release of File Descriptor or Handle after Effective Lifetime
CWE-401	Missing Release of Memory after Effective Lifetime
CWE-683	Function Call with Incorrect Order of Arguments
CWE-369	Divide By Zero
CWE-704	Incorrect Type Conversion or Cast
CWE-562	Return of Stack Variable Address
CWE-475	Undefined Behavior for Input to API
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer
CWE-252	Unchecked Return Value
CWE-628	Function Call with Incorrectly Specified Arguments

CWE-672	Operation on a Resource after Expiration or Release
CWE-687	Function Call with Incorrectly Specified Argument Value
CWE-786	Access of Memory Location Before Start of Buffer
CWE-415	Double Free
CWE-768	Incorrect Short Circuit Evaluation
CWE-762	Mismatched Memory Management Routines
CWE-120	Buffer Copy without Checking Size of Input (Classic Buffer Overflow)
CWE-134	Use of Externally Controlled Format String
CWE-831	Signal Handler Function Associated with Multiple Signals
CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CWE-78	Improper Neutralization of Special Elements used in an OS Command
CWE-20	Improper Input Validation
CWE-807	Reliance on Untrusted Inputs in a Security Decision
CWE-244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')
CWE-350	Reliance on Reverse DNS Resolution for a Security-Critical Action
CWE-367	Time-of-check Time-of-use (TOCTOU) Race Condition
CWE-829	Inclusion of Functionality from Untrusted Control Sphere
CWE-377	Insecure Temporary File
CWE-226	Sensitive Information in Resource Not Removed Before Reuse
CWE-126	Buffer Over-read
CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization
CWE-676	Use of Potentially Dangerous Function
CWE-732	Incorrect Permission Assignment for Critical Resource
CWE-785	Use of Path Manipulation Function without Maximum-sized Buffer
CWE-250	Execution with Unnecessary Privileges
CWE-22	Improper Limitation of a Pathname to a Restricted Directory

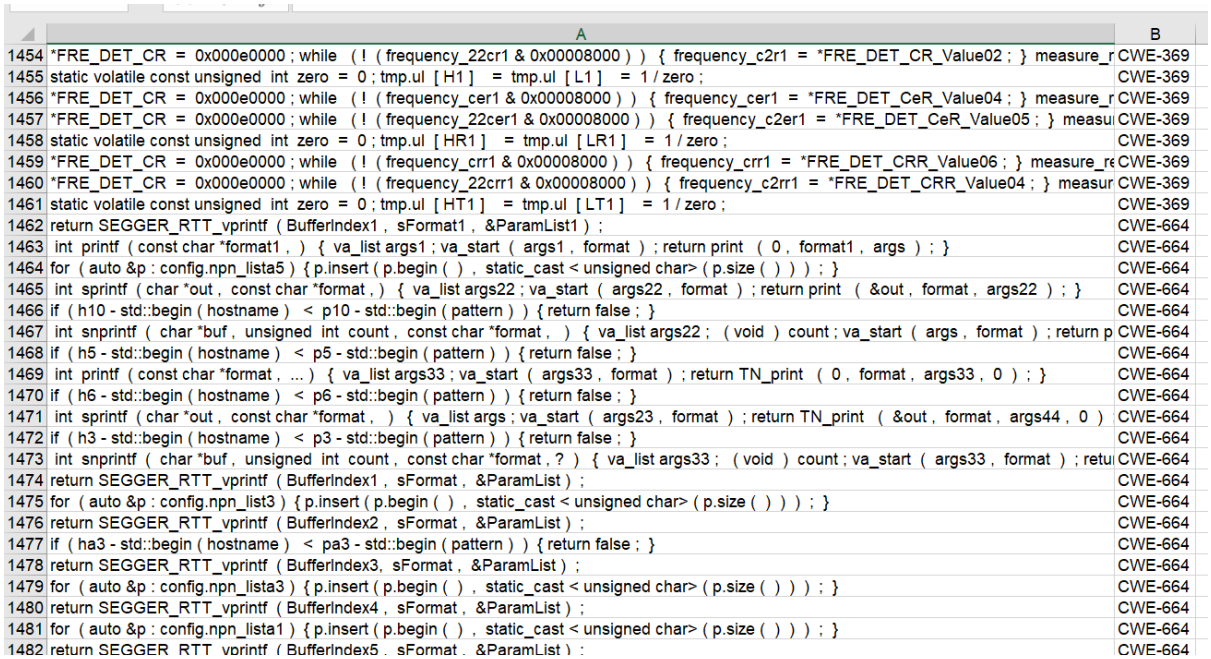
Table 4.1: The 54 types of Common Weakness Enumeration (CWEs).[1][10]

The figure provided in the dataset represents a portion of the vulnerable codes, showcasing examples from different CWE types.



	A	B
1	code	isMalicious
2	void printfUART_buf (char *buf , int len) { int i ; for (i = 0 ; i < len ; i++) { printf (" %02hhx " , buf [i]) ; } printf (" \n ") ; }	CWE-561
3	check_opt_size (cp_opt_t *opt , unsigned char *maxpos) { if (opt && opt < maxpos) { if (((*opt & 0x0f) < 0x0f) || (opt + 1 < maxp	CWE-561
4	cp_ft (cp_queue_t *queue , cp_tid_t id) { while (queue && queue -> id != id) queue = queue -> next ; return queue ; }	CWE-561
5	start (cp_pdu_t *pdu) { if (pdu && pdu -> hdr && (pdu -> hdr -> token + pdu -> hdr -> token_length < (unsigned char *) pdu -> hdr + pdu	CWE-561
6	cp_clone_pdu (cp_pdu_t *pdu) { cp_pdu_t *cloned_pdu ; size_t data_len ; unsigned char *data ; cp_opt_iterator_t opt_iter ; cp_opt_t *option ;	CWE-561
7	cp_clone_uri (const cp_uri_t *uri) { cp_uri_t *result ; if (!uri) return NULL ;	CWE-561
8	int get_ser_fd ( ) { return serial_source_fd (ser_src) ; }	CWE-561
9	int ipv6_addr_is_zero (const ip6_addr_t *addr) { int i ; for (i = 0 ;i<16 ;i++) { if (addr -> addr [i]) { return 0 ; } } return 1 ; }	CWE-561
10	APort j int ACALL Java_available (AEnv *jenv , jclass jcls , jlong jarg1) { j int jresult = 0 ; NativeSerial *arg1 = (NativeSerial *) 0 ; int	CWE-561
11	APort jboolean ACALL Java_1cancelWait (AEnv *jenv , jclass jcls , jlong jarg1) { jboolean jresult = 0 ; NativeSerial *arg1 = (NativeSerial	CWE-561
12	APortvoid ACALL Java _1close (AEnv *jenv , jclass jcls , jlong jarg1) { NativeSerial *arg1 = (NativeSerial *) 0 ;	CWE-561
13	void SJABool (AEnv *jenv , jboolean *jarr , bool *carr , jbooleanArray input) { int i ; jsize sz = jenv -> GetArrayLength (input) ; for (i = (	CWE-561
14	void P_ConfigureIt (const Pin *pPin , void (*handler) (const Pin *)) { int erruptSource *pSource ; TRACE_DEBUG (" P_ConfigureIt ( )	CWE-561
15	void P_Initialize int errupts (unsigned int priority) { TRACE_DEBUG (" P_Initialize ( ) \n\r ") ; ; numSources = 0 ; TRACE_DEBUG ("	CWE-561
16	void A_disable int errupt (u int 32_t m_i) { BP (A_CMSIS (m_i) -> CTL0 , A_CTL0_RDYIE_OFS) = 0 ; }	CWE-561
17	void A_encryptData (u int 32_t m_i , const u int 8_t * data , u int 8_t * encryptedData) { u int _fast8_t i ; u int 16_t tempData = 0 ; u int 16_t t	CWE-561
18	bool A_isBusy (u int 32_t m_i) { return BP (A_CMSIS (m_i) -> STAT , A_STAT_BUSY_OFS) ; }	CWE-561
19	void C_set16BitData (u int 16_t dataIn , u int _fast8_t crcType) { ASSERT ((CRC16_MODE == crcType) || (C_MODE == crcType)	CWE-561
20	void C_disableModule (u int 32_t m_i) { BP (EUSCI_B_CMSIS (m_i) -> CTLW0 , EUSCI_B_CTLW0_SWRST_OFS) = 1 ; ; }	CWE-561
21	case EUSCI_B3_BASE: int errupt_disable int errupt (int _EUSCIB3) ; int errupt_unregister int errupt (int _EUSCIB3) ; break ;	CWE-561
22	u int 8_t int errupt_getPriority (u int 32_t int erruptNumber) { ASSERT ((int erruptNumber > = 4) && (int erruptNumber < (NUM_ int	CWE-561
23	void int errupt_enableSleepOnIsrExit (void) { SCB -> SCR	= SCB_SCR_SLEEPONEXIT_Msk ; }	CWE-561
24	void PCM (void) { PCM -> CTL1 = (PCM -> CTL1 & ~ (PCM_CTL0_KEY_MASK | PCM_CTL1_FORCE_LPM_ENTRY))	PCM_KEY ;	CWE-561
25	void PCM_enable int errupt (u int 32_t flags) { PCM -> IE	= flags ; }	CWE-561
26	u int 32_t PCM_getEnabled int erruptStatus (void) { return PCM_get int erruptStatus ( ) & PCM -> IE ; }	CWE-561
27	void PCM_clear int erruptFlag (u int 32_t flags) { PCM -> CLRIFG	= flags ; }	CWE-561
28	void S_MC (u int 32_t m_i , u int 32_t clockSourceFrequency , u int 32_t desiredSpiClock) { if (is_A_Module (m_i)) { EUSCI_A_S_mas	CWE-561
29	void S_CP (u int 32_t m_i , u int _fast16_t clockPhase , u int _fast16_t clockPolarity) { if (is_A_Module (m_i)) { EUSCI_A_S_CP (m_i	CWE-561

Figure 4.11: Part of vulnerable codes.



	A	B
1454	*FRE_DET_CR = 0x000e0000 ; while (! (frequency_22cr1 & 0x00008000)) { frequency_c2r1 = *FRE_DET_CR_Value02 ; } measure_r	CWE-369
1455	static volatile const unsigned int zero = 0 ; tmp.ul [H1] = tmp.ul [L1] = 1 / zero ;	CWE-369
1456	*FRE_DET_CR = 0x000e0000 ; while (! (frequency_cer1 & 0x00008000)) { frequency_cer1 = *FRE_DET_CeR_Value04 ; } measure_r	CWE-369
1457	*FRE_DET_CR = 0x000e0000 ; while (! (frequency_22cer1 & 0x00008000)) { frequency_c2er1 = *FRE_DET_CeR_Value05 ; } measu	CWE-369
1458	static volatile const unsigned int zero = 0 ; tmp.ul [HR1] = tmp.ul [LR1] = 1 / zero ;	CWE-369
1459	*FRE_DET_CR = 0x000e0000 ; while (! (frequency_crr1 & 0x00008000)) { frequency_crr1 = *FRE_DET_CRR_Value06 ; } measure_re	CWE-369
1460	*FRE_DET_CR = 0x000e0000 ; while (! (frequency_22crr1 & 0x00008000)) { frequency_c2rr1 = *FRE_DET_CRR_Value04 ; } measur	CWE-369
1461	static volatile const unsigned int zero = 0 ; tmp.ul [HT1] = tmp.ul [LT1] = 1 / zero ;	CWE-369
1462	return SEGGER_RTT_vprintf (BufferIndex1 , sFormat1 , &ParamList1) ;	CWE-664
1463	int printf (const char *format1 ,) { va_list args1 ; va_start (args1 , format) ; return print (0 , format1 , args) ; }	CWE-664
1464	for (auto &p : config.npn_lista5) { p.insert (p.begin ( ) , static_cast < unsigned char> (p.size ( ))) ; }	CWE-664
1465	int sprintf (char *out , const char *format ,) { va_list args22 ; va_start (args22 , format) ; return print (&out , format , args22) ; }	CWE-664
1466	if (h10 - std::begin (hostname) < p10 - std::begin (pattern)) { return false ; }	CWE-664
1467	int snprintf (char *buf , unsigned int count , const char *format ,) { va_list args22 ; (void) count ; va_start (args , format) ; return p	CWE-664
1468	if (h5 - std::begin (hostname) < p5 - std::begin (pattern)) { return false ; }	CWE-664
1469	int printf (const char *format , ...) { va_list args33 ; va_start (args33 , format) ; return TN_print (0 , format , args33 , 0) ; }	CWE-664
1470	if (h6 - std::begin (hostname) < p6 - std::begin (pattern)) { return false ; }	CWE-664
1471	int sprintf (char *out , const char *format ,) { va_list args ; va_start (args23 , format) ; return TN_print (&out , format , args44 , 0)	CWE-664
1472	if (h3 - std::begin (hostname) < p3 - std::begin (pattern)) { return false ; }	CWE-664
1473	int snprintf (char *buf , unsigned int count , const char *format , ?) { va_list args33 ; (void) count ; va_start (args33 , format) ; retu	CWE-664
1474	return SEGGER_RTT_vprintf (BufferIndex1 , sFormat , &ParamList) ;	CWE-664
1475	for (auto &p : config.npn_list3) { p.insert (p.begin ( ) , static_cast < unsigned char> (p.size ( ))) ; }	CWE-664
1476	return SEGGER_RTT_vprintf (BufferIndex2 , sFormat , &ParamList) ;	CWE-664
1477	if (ha3 - std::begin (hostname) < pa3 - std::begin (pattern)) { return false ; }	CWE-664
1478	return SEGGER_RTT_vprintf (BufferIndex3, sFormat , &ParamList) ;	CWE-664
1479	for (auto &p : config.npn_lista3) { p.insert (p.begin ( ) , static_cast < unsigned char> (p.size ( ))) ; }	CWE-664
1480	return SEGGER_RTT_vprintf (BufferIndex4 , sFormat , &ParamList) ;	CWE-664
1481	for (auto &p : config.npn_lista1) { p.insert (p.begin ( ) , static_cast < unsigned char> (p.size ( ))) ; }	CWE-664
1482	return SEGGER_RTT_vprintf (BufferIndex5 , sFormat , &ParamList) ;	CWE-664

Figure 4.12: Part of vulnerable codes.

4.4.2 Data processing

As we mentioned in Chapter 3, the data processing steps involved in preparing the code for vulnerability detection. This process consisted of two main components: cleaning and vectorization.

During the cleaning phase, we applied various techniques to remove irrelevant information and ensure that the code is in a suitable format for analysis. One specific step involved removing comments from the code, as they do not contribute to vulnerability detection. We also utilized regular expressions to identify and eliminate unnecessary parts of the code, as depicted in the figure below. These cleaning steps helped streamline the code and improve the accuracy of the subsequent analysis.

```
# Fonction pour éliminer les commentaires
def __remove_comments(self, code):
  # Expressions régulières pour identifier les commentaires
  comment_regex = re.compile(r'//.*?$|/\*.*?\*/', re.DOTALL | re.MULTILINE)

  # Supprimer les commentaires du code
  code = comment_regex.sub('', code)

  # Suppression des caractères non pertinents
  code = re.sub(r'[{};,()\[\]\"\'\']+', '', code)
  return code
```

Figure 4.13: Function using regular expressions for data cleaning.

In the vectorization phase, we transformed the cleaned code into a numerical representation that could be used as input for the vulnerability detection model. We created a layer called text vectorization, which involved creating a structured layer to handle the vectorization process. This is after we coding data with convert the vocabulary of c/c++. This layer facilitated the conversion of the code into a format that could be efficiently processed by the deep learning model.
These preprocessing steps were essential in ensuring the accuracy and effectiveness of the subsequent vulnerability detection process.

4.4.3 Model training

4.4.3.1 Dataset Split

The dataset split is an important step in machine learning model development. It involves dividing the dataset into two subsets: training and testing. The training subset

is used to train the model, while the testing subset is used to evaluate its performance on unseen data.

In our approach, we randomly split the dataset using a specific ratio: 70% for training and 30% for testing. By randomly splitting, we can assess how well the trained model generalizes to new instances. This validation process helps us measure the model's performance and determine its ability to handle unseen data effectively. In the following figure 14 the code for split dataset

```
# Split the data set into training and test data
from sklearn import model_selection
X_train, X_test, target_train, target_test = model_selection.train_test_split(Data_X, Data_Y, test_size=0.30, random_state=30)
print(Data_X.shape,Data_Y.shape)
# After splait
print(X_train.shape,target_train.shape)
```

```
(4810,) (4810,)
(3367,) (3367,)
```

Figure 4.14: Code for split dataset.

4.4.3.2 Model Selection/ Creation

1. **Model Selection**

   In our study focused on analyzing C/C++ source code of IOT OS, we employed a CNN-supervised model, specifically designed for processing structured grid-like data like images. Despite the inherent differences between C/C++ code and images, we hypothesized that the CNN's ability to learn hierarchical patterns and features could be effectively utilized to enhance the analysis of code snippets.

   We acknowledged the potential of the CNN-supervised model to capture hierarchical patterns and features within the source code, ultimately aiming to improve the performance of our research tasks.

   The decision to select the CNN-supervised model was based on its well-documented success in computer vision and pattern recognition domains. Our objective was to attain higher accuracy and enhanced performance by capitalizing on the model's capabilities.

2. **Model Creation**

   Our Convolutional Neural Network (CNN) training model specifically tailored for analyzing code snippets. The model architecture consisted of the following components:

   2.1 The main input layer with 150 neurons representing the maximum length of a code snippet.

   2.2 One embedding layer with 150 neurons representing each word with a unique integer.

   2.3 Five convolutional layers, after ethe one we use function activation and Dropout.

   2.4 Fully connected layers for further processing. We used the following structure:

   - Flattening layer to convert the output from the convolutional layers into a 1-dimensional vector.
   - Three hidden layers

   2.5 The output layers.

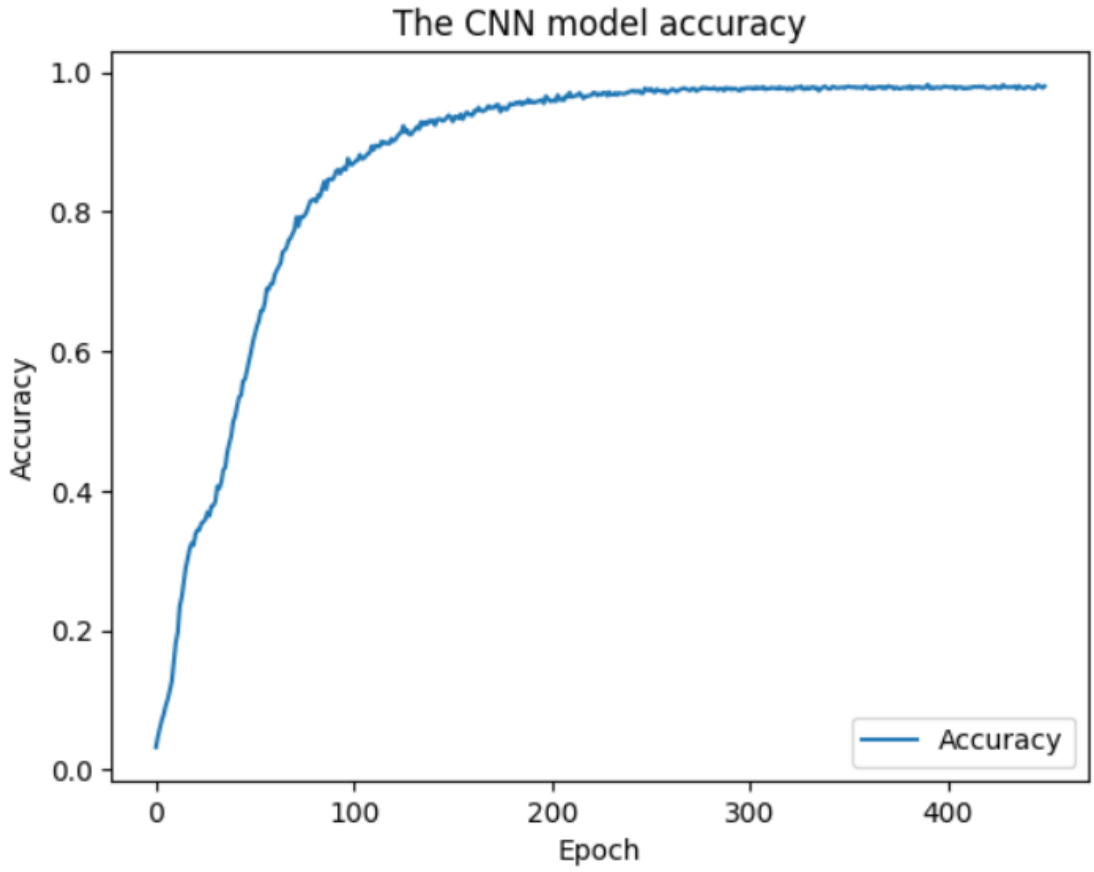
   2.6 The Adam optimizer

   For multiclass classification, the output layer applied the **"Softmax"** activation function, suitable for multiclass classification tasks. The output layer had a shape corresponding to 55 types, including 54 types of CWE (Common Weakness Enumeration) and Benign.
   The model was trained over 400 epochs, using a batch size of 64 batches. During training, the Adam optimizer was employed to optimize model parameters and improve training efficiency. In our experiments, we achieved a final Cross-Validation accuracy of 98% for multiclass classification, indicating the effectiveness of our CNN model in analyzing and classifying code snippets.
   In the following Table 4.3 the summary of model and in Figure 4.15 training model accuracy of our model.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 150)	0
embedding_1 (Embedding)	(None, 150, 10)	2000
elu_6 (ELU)	(None, 150, 10)	0
conv1d_4 (Conv1D)	(None, 150, 64)	1984
elu_7 (ELU)	(None, 150, 64)	0
dropout_7 (Dropout)	(None, 150, 64)	0
conv1d_5 (Conv1D)	(None, 150, 128)	24704
elu_8 (ELU)	(None, 150, 128)	0
dropout_8 (Dropout)	(None, 150, 128)	0
conv1d_6 (Conv1D)	(None, 150, 256)	98560
elu_9 (ELU)	(None, 150, 256)	0
dropout_9 (Dropout)	(None, 150, 256)	0
conv1d_7 (Conv1D)	(None, 150, 512)	393728
elu_10 (ELU)	(None, 150, 512)	0
dropout_10 (Dropout)	(None, 150, 512)	0
conv1d_8 (Conv1D)	(None, 150, 768)	1180416
elu_11 (ELU)	(None, 150, 768)	0
flatten	(None, 115200)	0
dense_2 (Dense)	(None, 1024)	117965824
elu_12 (ELU)	(None, 1024)	0
batch_normalization_2 (BatchNormalization)	(None, 1024)	4096
dropout_11 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 2048)	2099200
elu_13 (ELU)	(None, 2048)	0
batch_normalization_3 (BatchNormalization)	(None, 2048)	8192
dropout_12 (Dropout)	(None, 2048)	0
dense_4 (Dense)	(None, 3072)	6294528
elu_14 (ELU)	(None, 3072)	0

batch_normalization_4 (BatchNormalization)	(None, 3072)	12288
dropout_13 (Dropout)	(None, 3072)	0
dense_5 (Dense)	(None, 55)	169015
Total params: 128,254,535		
Trainable params: 128,242,247		
Non-trainable params: 12,288		

Table 4.2: Summary of model layers and parameters.




```
Final Cross-Validation Accuracy of CNN training model 0.9815859794616699
```

Figure 4.15: Training model accuracy.

4.5 Model Testing

4.5.1 Evaluation Metrics

We used the F1 score as a measure of accuracy and effectiveness in evaluating our model. The F1 score combines both precision and recall into a single value and provides a balanced assessment of the model's performance. It ranges from 0 to 1, with higher values indicating better performance.[38]

- **Precision** is the ratio of true positives to the total number of positively predicted units, including both true positives and false positives. True positives are the elements correctly labeled as positive by the model, while false positives are elements incorrectly labeled as positive.[38]

$$\text{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}}$$

- **Recall** is the ratio of true positives to the total number of actual positive units. False negatives are elements incorrectly labeled as negative by the model, but are actually positive.[38]

$$\text{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}}$$

In our multi-category classification task involving different types of code fragments (CWE and Benign types) , we utilized two metrics: the macro F1 score (mF1) and the weighted average F1 score (wF1). The macro F1 score calculates the F1 score for each class independently and then takes the average. The weighted average F1 score considers the class imbalance by accounting for the number of samples in each class. The equations to calculate mF1, wF1, average precision, and average recall are as follows:

$$\text{AveragePrecision} = \frac{1}{K}\sum_{k=1}^{K}(\text{Precision}_k)$$

$$\text{AverageRecall} = \frac{1}{K}\sum_{k=1}^{K}(\text{Recall}_k)$$

$$mF1 = \frac{1}{K}\sum_{k=1}^{K}\left(\frac{2 * \text{Precision}_k * \text{Recall}_k}{\text{Precision}_k + \text{Recall}_k}\right)$$

$$wF1 = \frac{1}{\sum_{k=1}^{K} X_k}\sum_{k=1}^{K} X_k\left(\frac{2 * \text{Precision}_k * \text{Recall}_k}{\text{Precision}_k + \text{Recall}_k}\right)$$

where :

K is the number of different types of code snippets (CWE types and Benign)

X is the number of samples.

- **Accuracy** is one of the most commonly used metrics in machine learning and statistics, assesses the overall correctness of a classification or prediction model . It is determined by dividing the number of correctly predicted classifications (true positives (TP) and true negatives (TN)) by the total number of predictions made (true positives, true negatives, false positives (FP), and false negatives (FN))[28]. Accuracy is computed directly from the confusion matrix [38], providing a measure of the model's performance in terms of correct predictions.

  $$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

  where:

  The confusion matrix is a fundamental tool in evaluating the performance of machine learning classification models. It is a table that summarizes the predictions made by the model against the actual values, allowing for a comprehensive analysis of the model's performance.[8]

		Actual Class	
		Positive	Negative
Predicted Class	Positive	True Positives (TP)	False Positives (FP)
	Negative	False Negatives (FN)	True Negatives (TN)

Table 4.3: Confusion Matrix

4.5.2 Testing with data test

In order to assess the performance of our model on unseen data, we conducted testing with a dedicated portion of our dataset. The test set, which was previously separated from the training data, was used to evaluate the model's predictions and measure its effectiveness in real-world scenarios.

The test of part dataset code and results are presented in Figure 4.16. The table 4.5 displays the performance metrics obtained from testing on the test set, including accuracy, precision, recall, and F1 score.

```
from tensorflow import keras
X_,Y_ = data_preprocessing_our(X_test, target_test,
                          Y_manage_categorical_variables = True,
                          Y_encoding_onehot = encoding_onehot)

model_file = '/content/CNN_Model_MultiClass_iVulnDetect.h5'
our_model = keras.models.load_model(model_file)

#7- test model
loss, accuracy = our_model.evaluate(X_, Y_, verbose= 1)
print('\nFinal Cross-Validation Accuracy of CNN training model', accuracy, '\n')
evaluation(our_model,X_,Y_)
```

Figure 4.16: Evaluation Code for the Test Data.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Benign Code	0.98	0.97	0.98	511
CWE-119	0.82	0.92	0.87	39
CWE-120	1.00	1.00	1.00	17
CWE-126	0.89	0.67	0.76	12
CWE-134	0.75	0.60	0.67	15
CWE-190	0.86	0.86	0.86	14
CWE-20	0.87	0.62	0.72	21
CWE-22	1.00	1.00	1.00	11
CWE-226	1.00	0.62	0.76	13
CWE-244	1.00	0.91	0.95	11
CWE-250	1.00	1.00	1.00	12
CWE-252	0.87	0.87	0.87	15
CWE-327	1.00	0.76	0.87	17
CWE-350	1.00	1.00	1.00	7
CWE-362	0.94	1.00	0.97	16
CWE-367	0.90	0.90	0.90	20
CWE-369	0.83	1.00	0.91	10
CWE-377	0.90	0.90	0.90	10
CWE-398	0.87	0.93	0.90	56
CWE-401	0.72	0.93	0.81	14
CWE-415	1.00	0.81	0.90	16
CWE-457	1.00	0.74	0.85	23
CWE-467	1.00	1.00	1.00	14
CWE-475	0.86	0.90	0.88	20
CWE-476	0.96	0.96	0.96	24
CWE-477	1.00	0.90	0.95	20
CWE-561	0.94	0.80	0.86	60
CWE-562	0.92	1.00	0.96	11
CWE-563	0.83	0.90	0.86	21
CWE-570	0.53	0.94	0.68	17

CWE-571	1.00	0.73	0.85	15
CWE-628	1.00	1.00	1.00	14
CWE-664	1.00	1.00	1.00	21
CWE-665	0.91	0.83	0.87	12
CWE-672	1.00	0.96	0.98	25
CWE-676	1.00	0.94	0.97	16
CWE-682	0.94	1.00	0.97	16
CWE-683	0.93	1.00	0.97	14
CWE-685	1.00	0.96	0.98	23
CWE-686	0.94	1.00	0.97	16
CWE-687	0.83	1.00	0.91	10
CWE-704	1.00	1.00	1.00	19
CWE-732	0.57	0.92	0.71	13
CWE-758	0.92	1.00	0.96	11
CWE-762	1.00	1.00	1.00	21
CWE-768	1.00	0.80	0.89	15
CWE-775	1.00	1.00	1.00	8
CWE-78	0.90	1.00	0.95	19
CWE-783	0.92	1.00	0.96	11
CWE-785	1.00	1.00	1.00	10
CWE-786	1.00	1.00	1.00	17
CWE-788	0.85	0.85	0.85	13
CWE-807	0.74	1.00	0.85	23
CWE-829	0.50	0.25	0.33	4
CWE-831	0.90	0.90	0.90	10
Accuracy			0.91	1443
Macro Avg	0.91	0.90	0.90	1443
Weighted Avg	0.94	0.93	0.93	1443

Table 4.4: Classification Metrics.

Evaluating a portion of the dataset provides us with valuable insights into how well our

model performs and how effective it is in real-world applications. It helps us assess the model's reliability and robustness, ensuring that it can make accurate predictions and meet our desired goals.

4.6 Results of Model Evaluation for TinyOS

We evaluated the performance of our model using unseen data obtained from TinyOS V. 2.1.2, which was sourced from [9]. This dataset allowed us to assess how well our model generalized to real-world data that it had not been previously exposed to. By testing our model on this external dataset, we aimed to validate its ability to accurately analyze and classify code snippets from the TinyOS version mentioned.
In the following figures, we present the results of evaluating our model using the TinyOS dataset. These figures showcase the performance metrics obtained from testing our model on the TinyOS code snippets, including accuracy, precision, recall, F1 score. Additionally, we provide a representation of the confusion matrix. This matrix visually depicts the distribution of predicted labels compared to the actual labels, allowing us to analyze the model's performance in classifying the different code snippets in the TinyOS dataset.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Benign Code	1.00	0.88	0.93	48
CWE-119	0.70	0.93	0.80	15
CWE-120	1.00	0.97	0.98	29
CWE-126	1.00	1.00	1.00	14
CWE-134	1.00	0.71	0.83	14
CWE-190	0.83	1.00	0.91	5
CWE-20	0.74	0.93	0.82	15
CWE-327	1.00	0.75	0.86	4
CWE-350	1.00	1.00	1.00	4
CWE-362	1.00	0.80	0.89	5
CWE-367	1.00	1.00	1.00	4
CWE-377	0.00	0.00	0.00	0

CWE-398	0.76	0.84	0.79	37
CWE-401	0.00	0.00	0.00	1
CWE-457	0.90	0.90	0.90	10
CWE-467	1.00	1.00	1.00	4
CWE-476	1.00	0.89	0.94	9
CWE-561	0.79	0.94	0.86	16
CWE-563	1.00	0.70	0.82	10
CWE-570	1.00	1.00	1.00	7
CWE-571	1.00	1.00	1.00	9
CWE-664	0.00	0.00	0.00	9
CWE-676	0.00	0.00	0.00	1
CWE-682	0.60	1.00	0.75	3
CWE-686	0.92	0.75	0.83	16
CWE-687	1.00	1.00	1.00	7
CWE-704	1.00	1.00	1.00	15
CWE-758	0.00	0.00	0.00	0
CWE-768	1.00	1.00	1.00	2
CWE-78	1.00	1.00	1.00	5
CWE-783	0.50	1.00	0.67	1
CWE-807	0.38	1.00	0.55	3
CWE-829	0.00	0.00	0.00	0
Accuracy			0.87	322
Macro Avg	0.73	0.76	0.73	322
Weighted Avg	0.88	0.87	0.87	322

Table 4.5: Results of TinyOS classification Metrics.

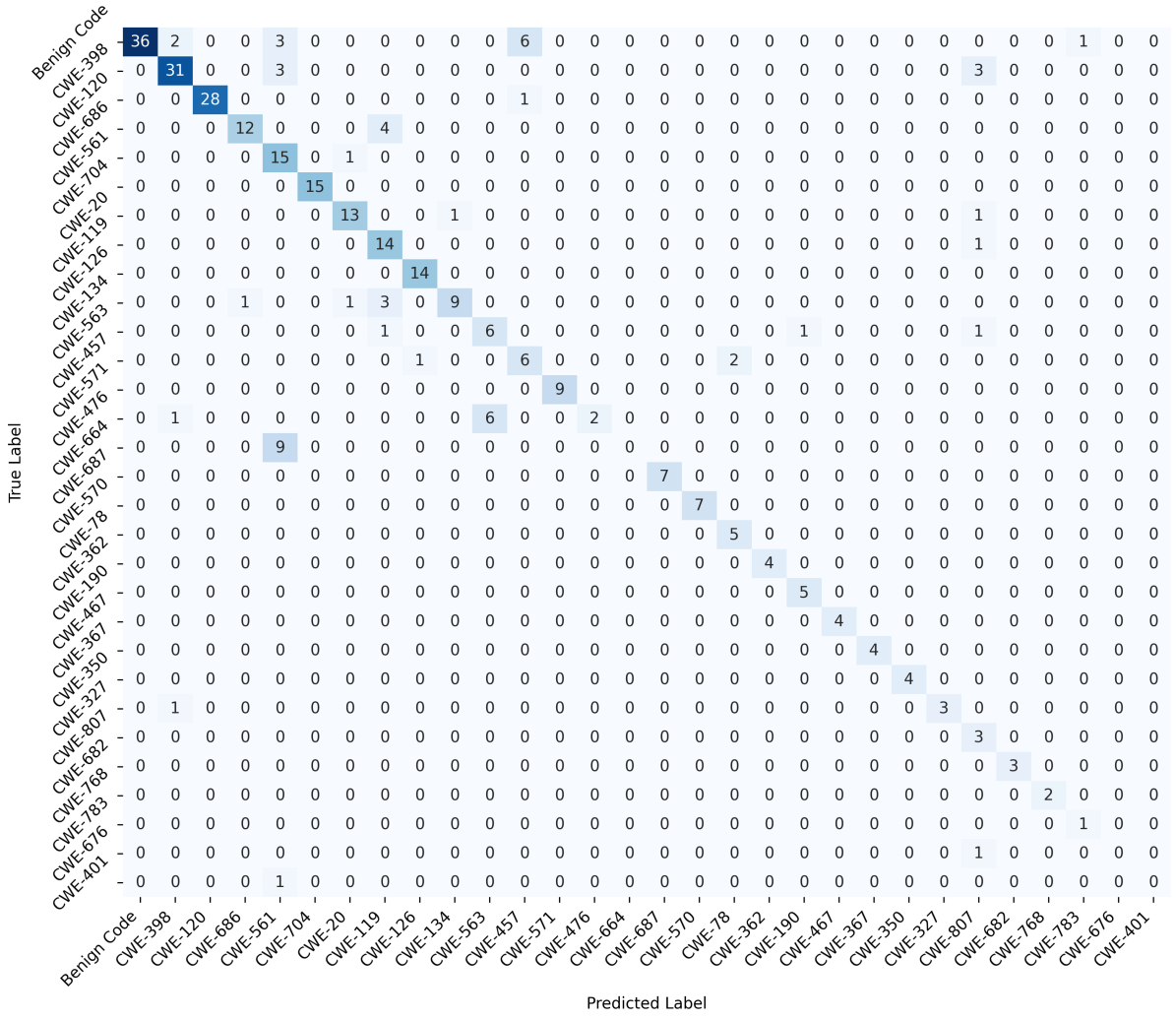


Figure 4.17: Confusion matrix of TinyOS evaluation.

4.6.1 Comparison of Results

In this part, we show a comparison between our model, iVulnDetect, and the iDetect model discussed in the article [9]. The comparison aims to assess the performance and effectiveness of our model in detecting vulnerabilities in IOT OS code snippets of c/c++. This tables presents the evaluation results of two models, "iDetect" and "iVulnDetect," using a Convolutional Neural Network (CNN) model on testing data and TinyOS dataset. The two tables includes metrics such as accuracy, weighted F1-score, and macro F1-score.

1. **For testing data model**

		iDetect	iVulnDetect
CNN Model	Accuracy	94%	93%
	Weighted F1-score	93%	93%
	Macro F1-score	93%	90%

Table 4.6: Comparison testing data of CNN model

For the "iDetect" model, it achieved an accuracy of approximately 94%, a weighted F1-score of around 93%, and a macro F1-score of about 93%. On the other hand, our model "iVulnDetect" had an accuracy of approximately 93%, a weighted F1-score of around 93%, and a slightly lower macro F1-score of about 90%.

2. **For evaluation model with TinyOS**

		iDetect	iVulnDetect
CNN Model	Accuracy	85%	87%
	Weighted F1-score	86%	87%
	Macro F1-score	73%	73%

Table 4.7: Comparison TinyOS evaluation

For the "iDetect" model, it achieved an accuracy of approximately 85%, a weighted F1-score of around 86%, and a macro F1-score of about 73%. On the other hand, our model "iVulnDetect" had an accuracy of approximately 87%, a weighted F1-score of around 87%, and a similar macro F1-score of about 73%. The "iVulnDetect" model slightly outperforming the "iDetect" model.

4.7 Presentation system

4.7.1 Database

The following figure illustrates the database structure in phpMyAdmin, specifically the **"ivulndetect"** database. The "developers" table in this database is utilized to store the registration information of developers.

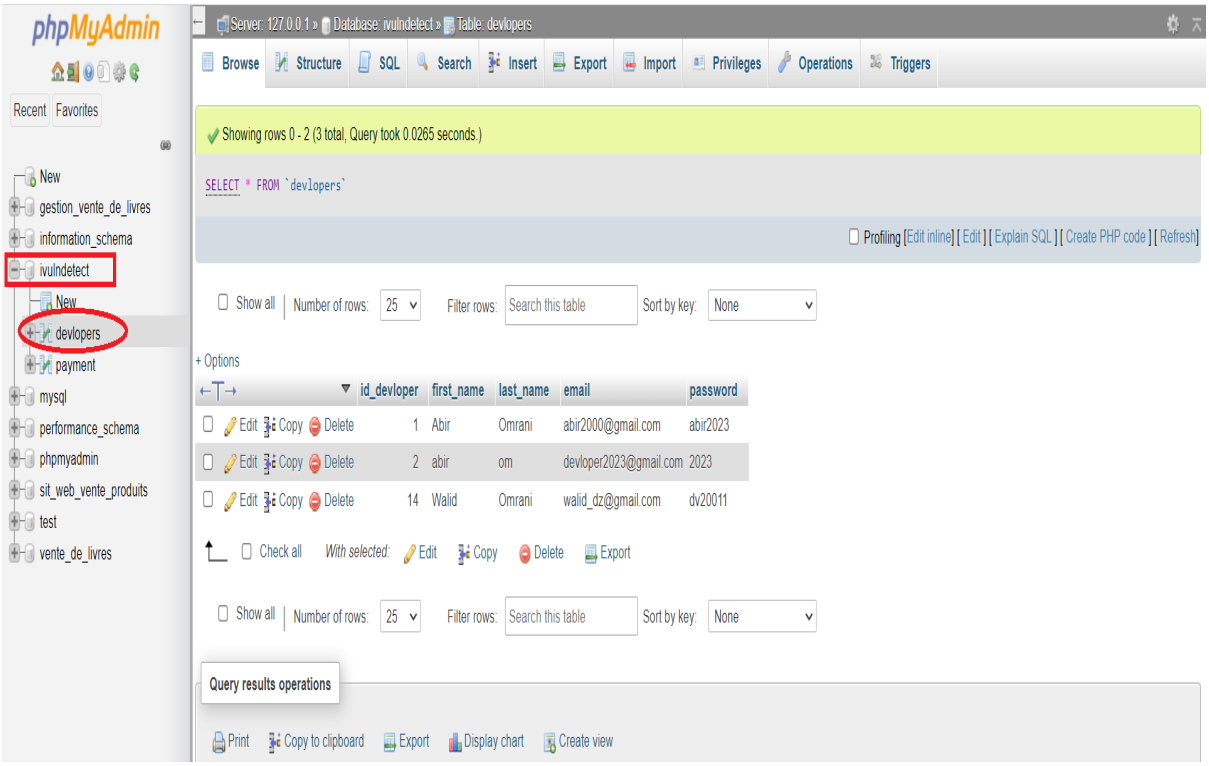


Figure 4.18: Database phpMyAdmin with XAMP.

4.7.2 Interfaces of the system

4.7.2.1 Login Interface

The login interface provides a secure and convenient way for registered users to access the system. Users are required to enter their email and password to gain entry. If the provided email and password match the records, the user is granted access to the system.

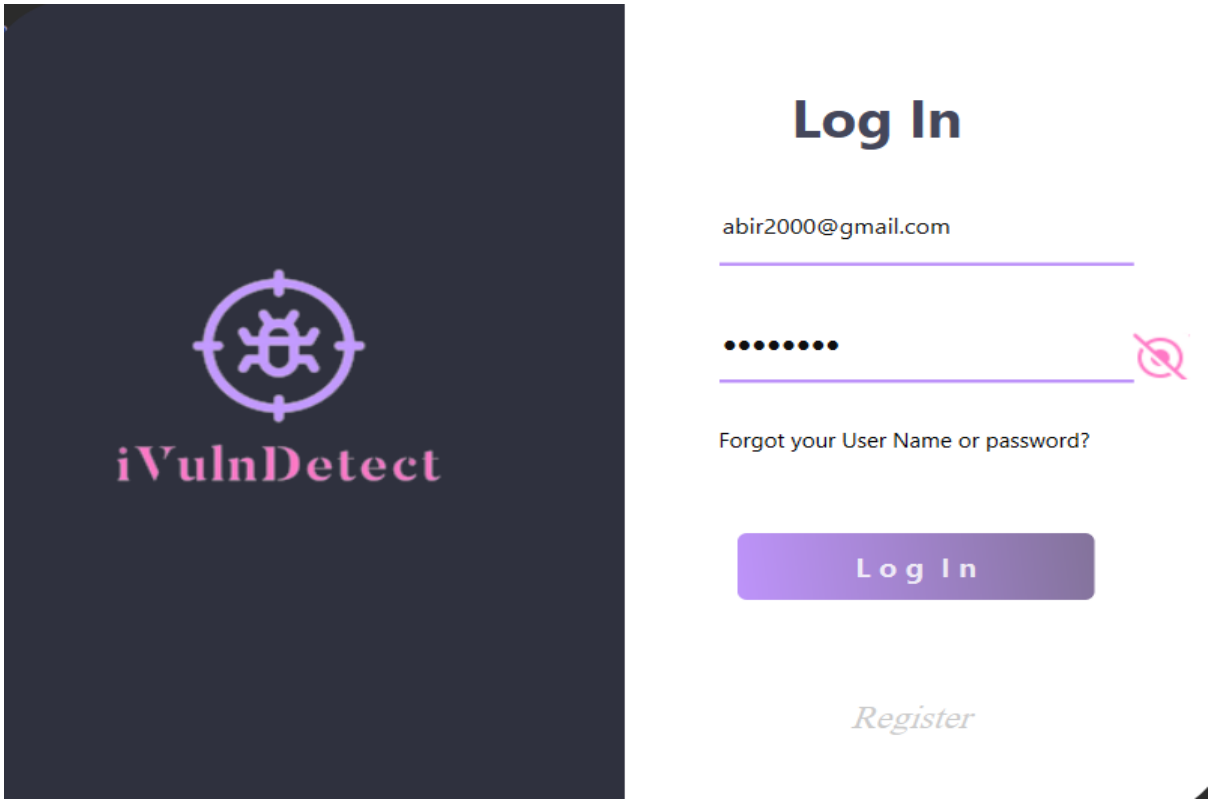


Figure 4.19: Interface login.

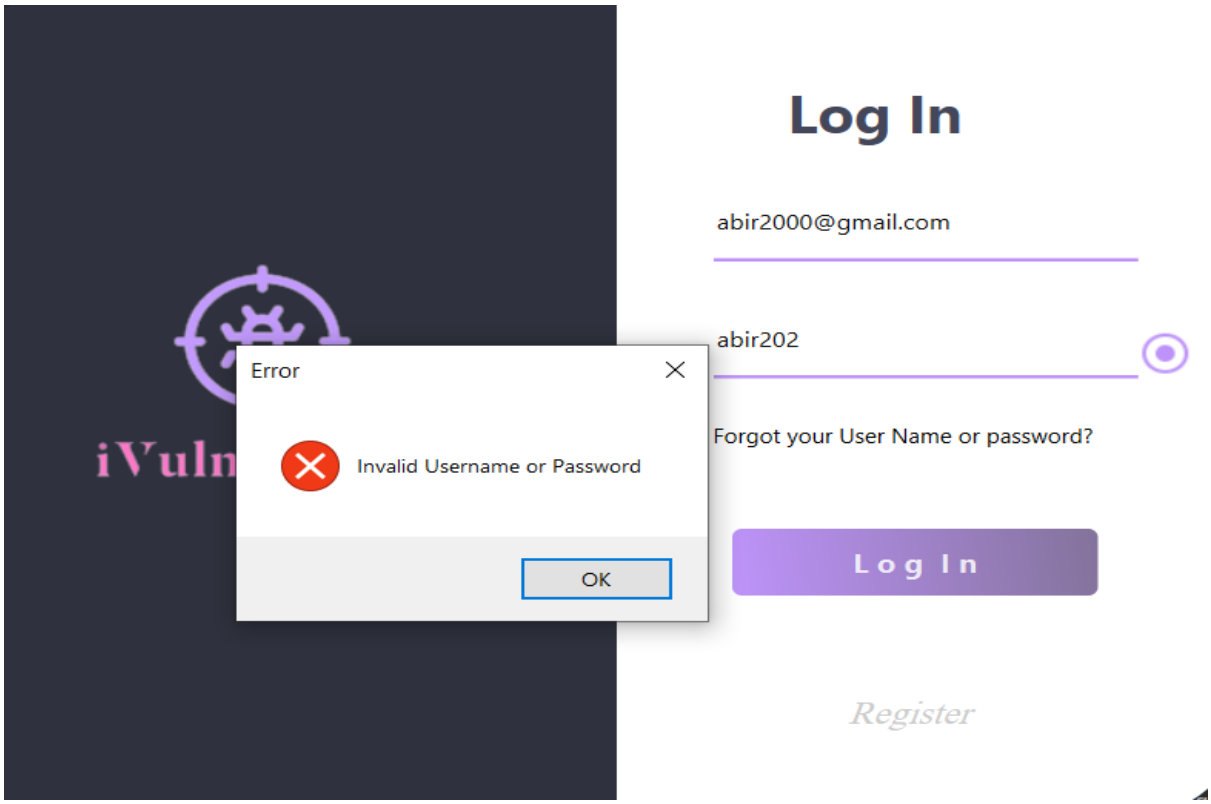


Figure 4.20: Interface login error.

4.7.2.2 Registration Interface

The registration interface serves as the gateway for new users to create an account and become a part of the system. It presents a user-friendly form that allows developers to input their personal information and complete the registration process. The interface typically includes fields for entering the first name, last name, email address, and password.

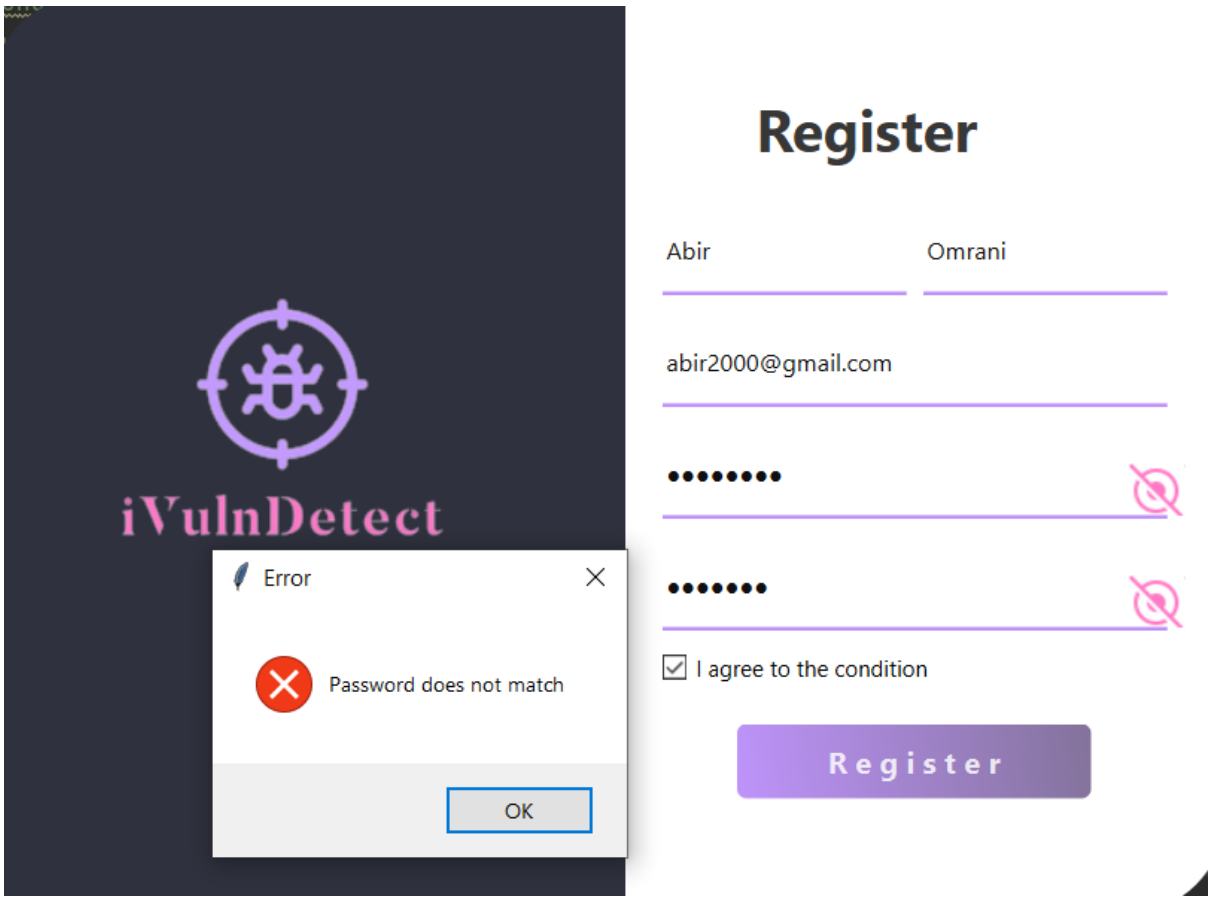


Figure 4.21: Error in registration interface.

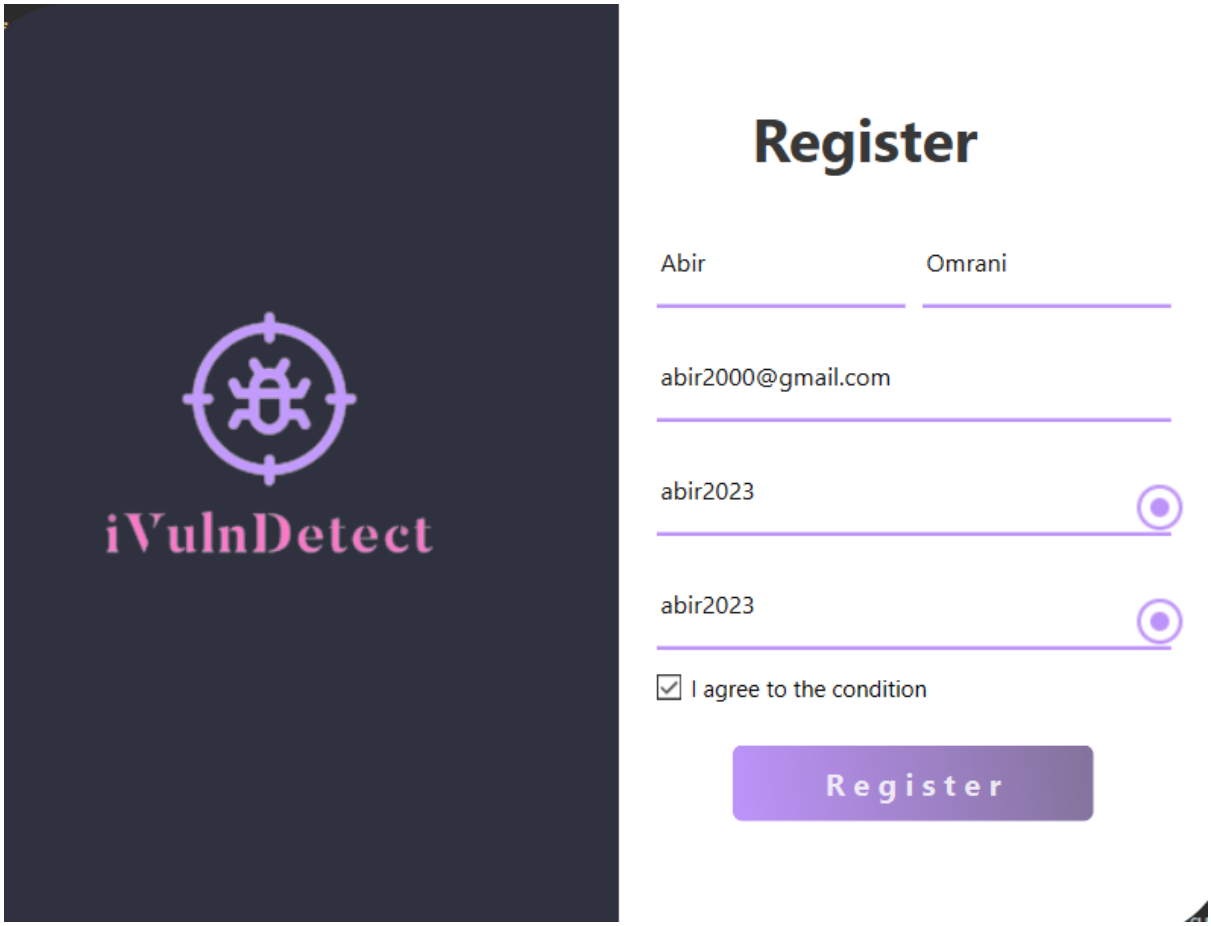


Figure 4.22: Register interface.

Additionally, the registration interface may include a checkbox or a statement that requires users to accept the system's terms and conditions. By checking the box or acknowledging the statement, users indicate their agreement to comply with the rules and guidelines set by the system.

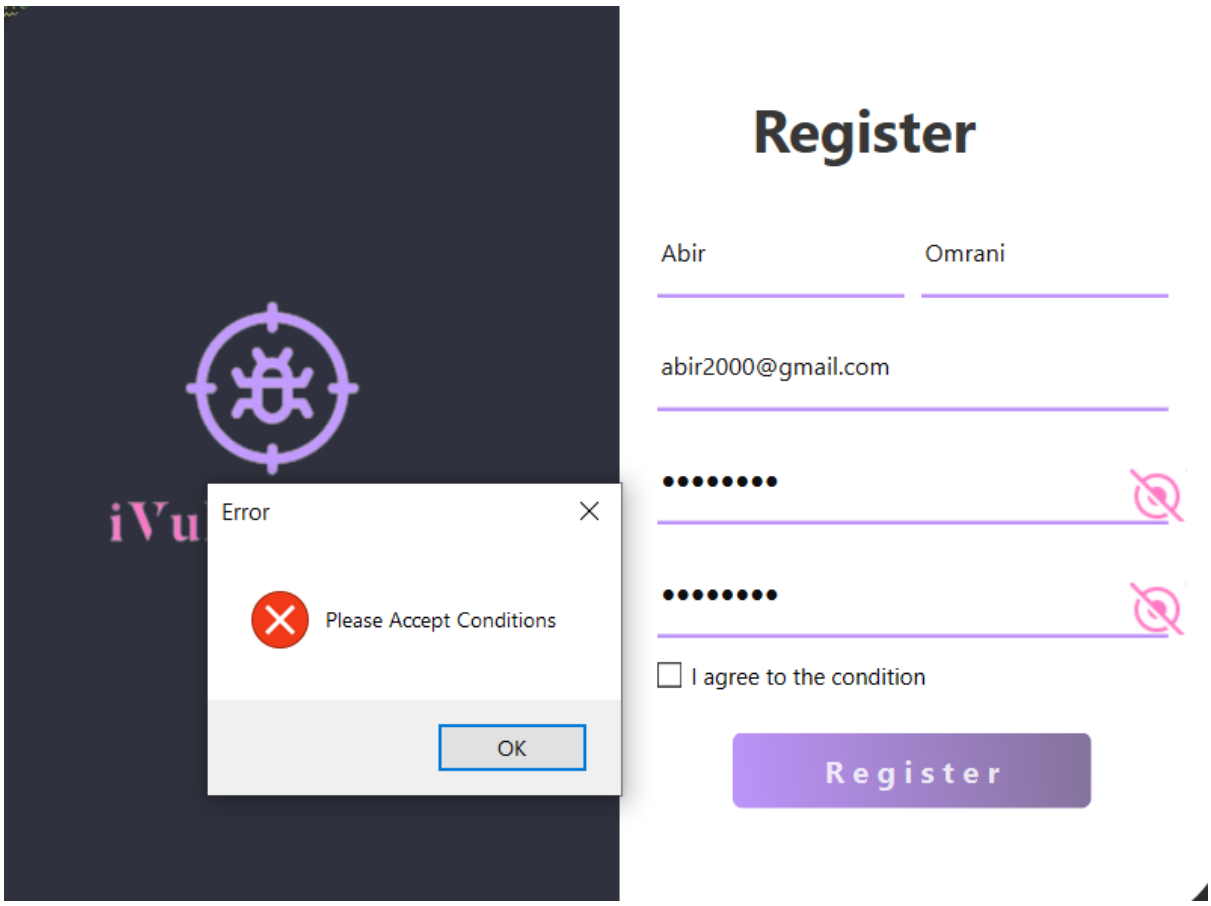


Figure 4.23: Error accept condition in registration interface.

4.7.2.3 Subscription Interface

The subscription interface plays a crucial role in the registration process, as it enables registrants to finalize their account creation and gain access to our system. Once all the necessary information has been submitted and the terms of the system have been accepted, users are prompted to proceed with the subscription step.

In this interface, users are presented with options to choose their preferred subscription plan. As an incentive, we offer a free seven-day subscription plan that allows users to experience the full functionality of our system without any charges during this initial period.

Figure 4.24: Suscription Interface.

4.7.2.4 System Interfaces

The system interfaces serve as the primary means for users to interact with our system and utilize its functionalities. In the following figures, we present a look of some key system interfaces:

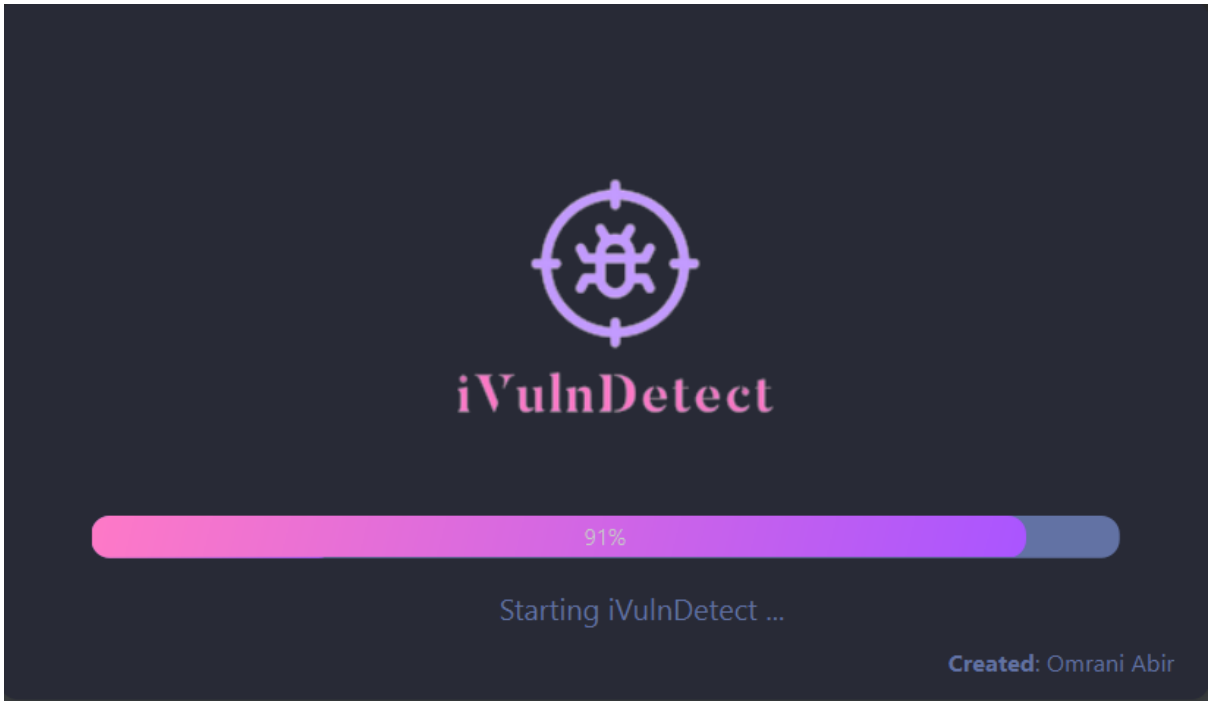


Figure 4.25: Starting iVulnDetect.

1. **Dashboard Interface:** The dashboard interface provides an overview of the system's main features and provides users with quick access to essential tools and functions.

Figure 4.26: Dashboard Interface.

2. **Test vulnerabilities Interface:** This interface allows users to detect vulnerabilities in the code. Users can upload their code or input code directly into the interface for detection.

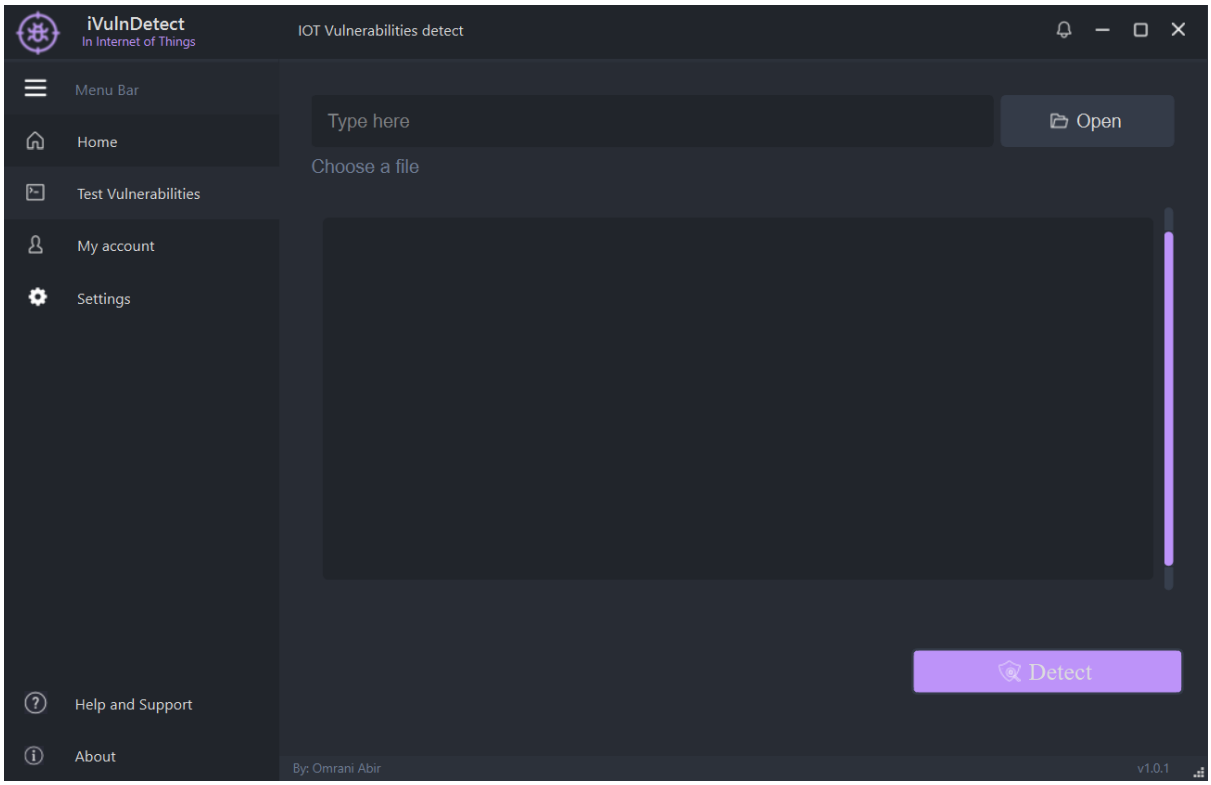


Figure 4.27: Test vulnerabilities Interface.

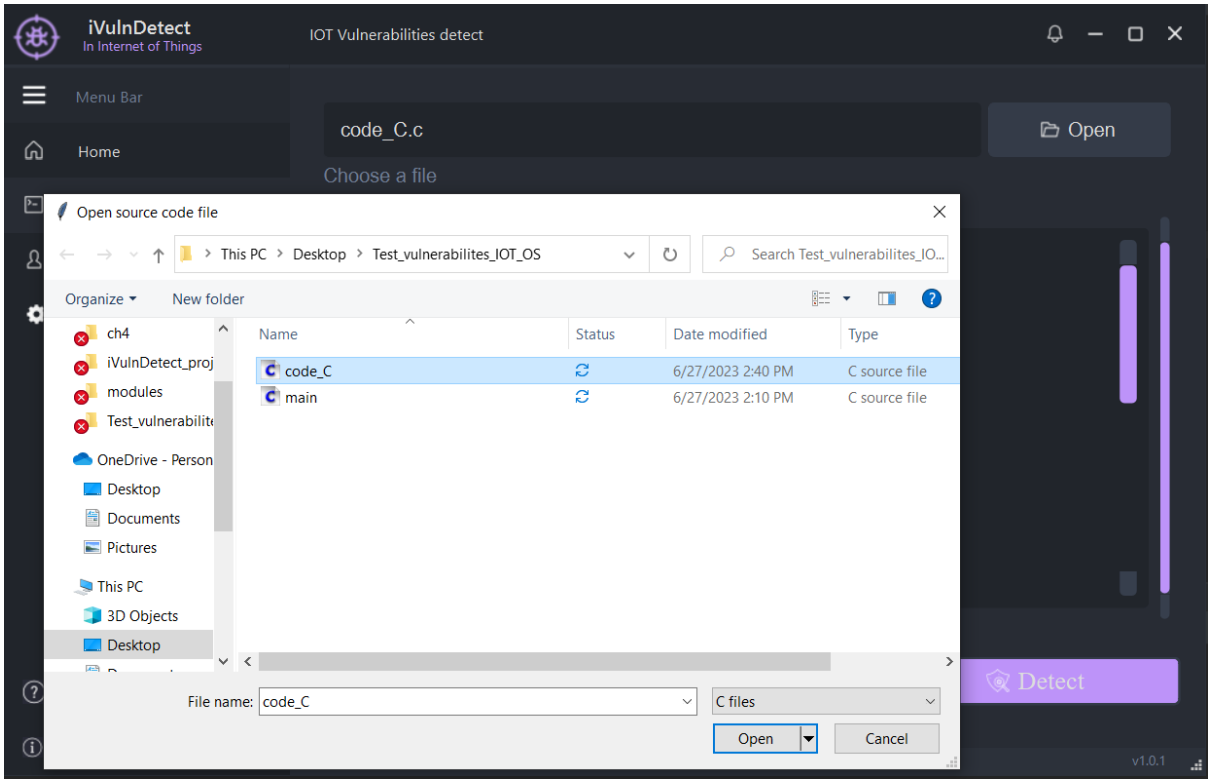


Figure 4.28: Upload source code.

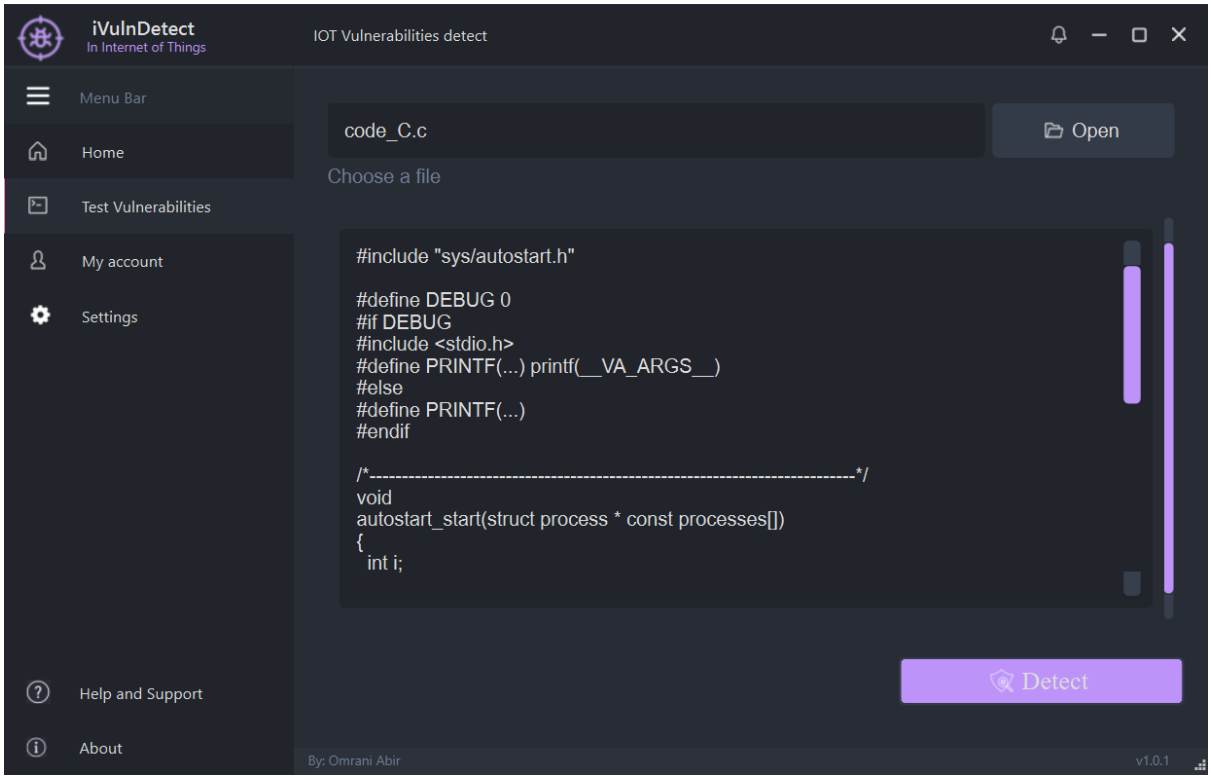


Figure 4.29: Click bottom Detect.

3. **Results Display Interface:** this interface allows displaying the detected vulnerabilities with details about each type, in addition to the approximate location of the vulnerability, and the graphics place the proportions of each vulnerability in the code.

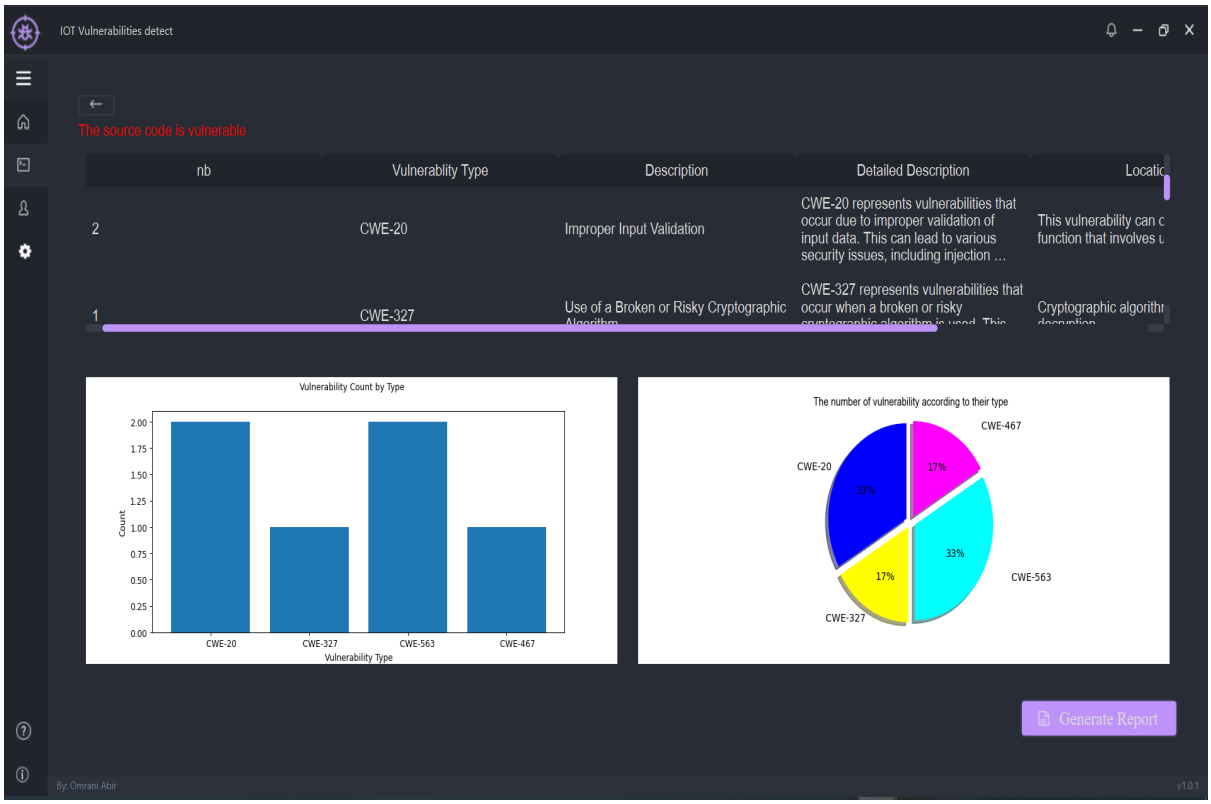


Figure 4.30: Detection Results Interface.

The following figure shows the report generated by our system.

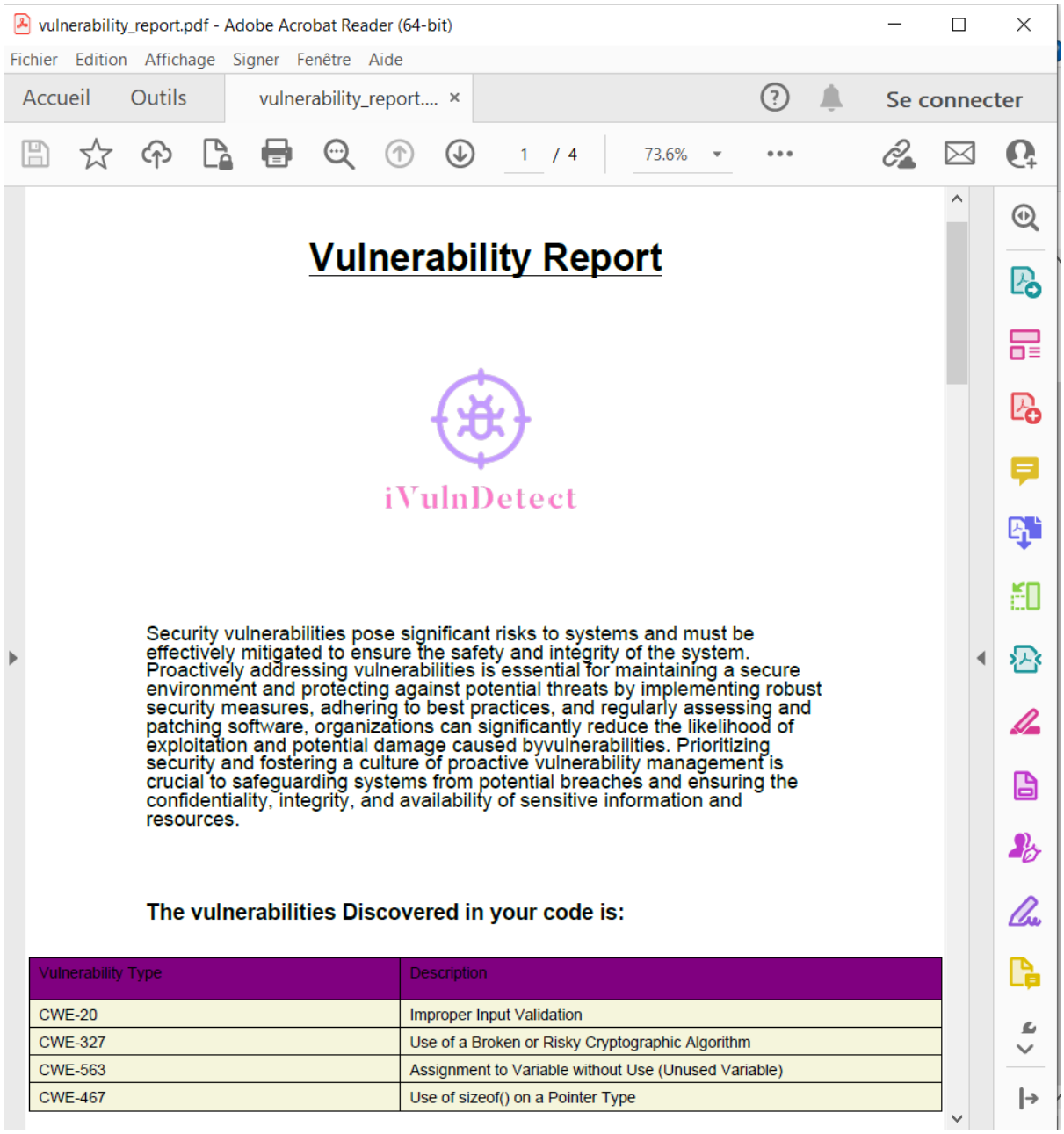


Figure 4.31: Report generated by our system.

4. **Account Interface:** This interface allows managing the account and viewing the subscription status and other features.

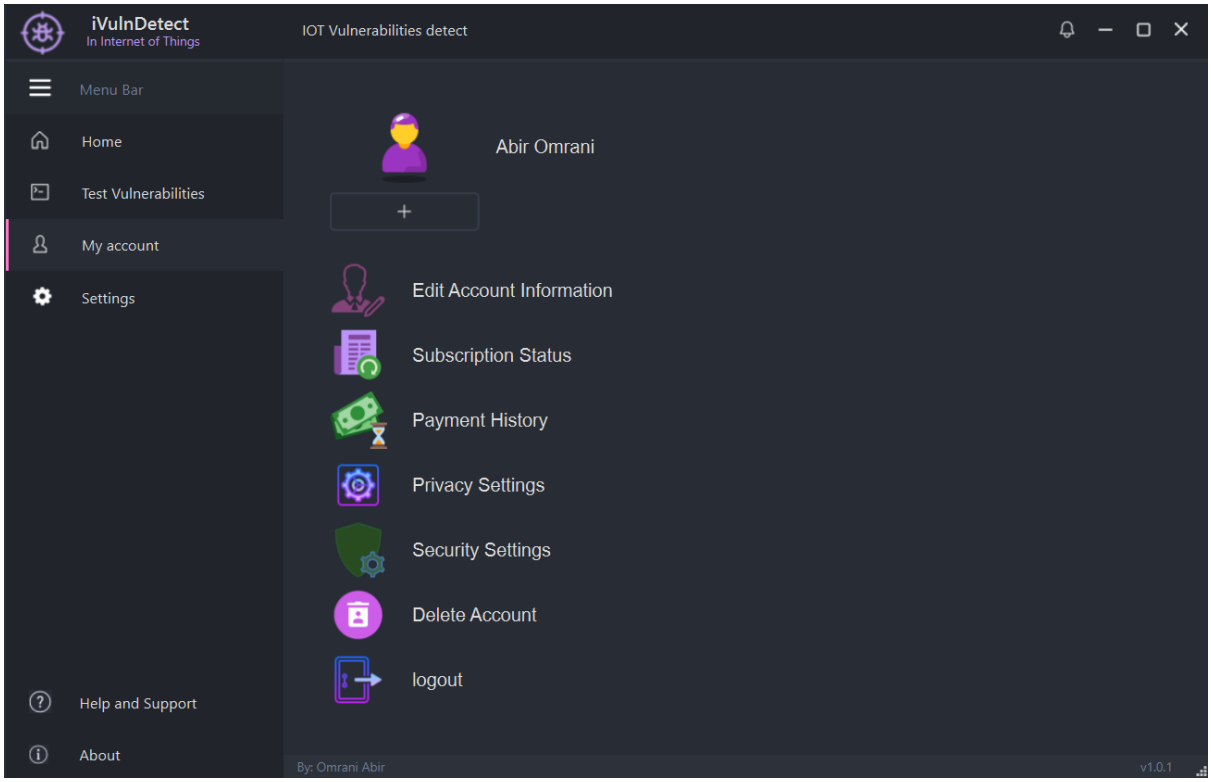


Figure 4.32: Account interface.

5. **Help and support Interface:** is designed to assist users by providing a range of helpful resources, documentation, and communication channels. This interface aims to address common questions and provide support to users.

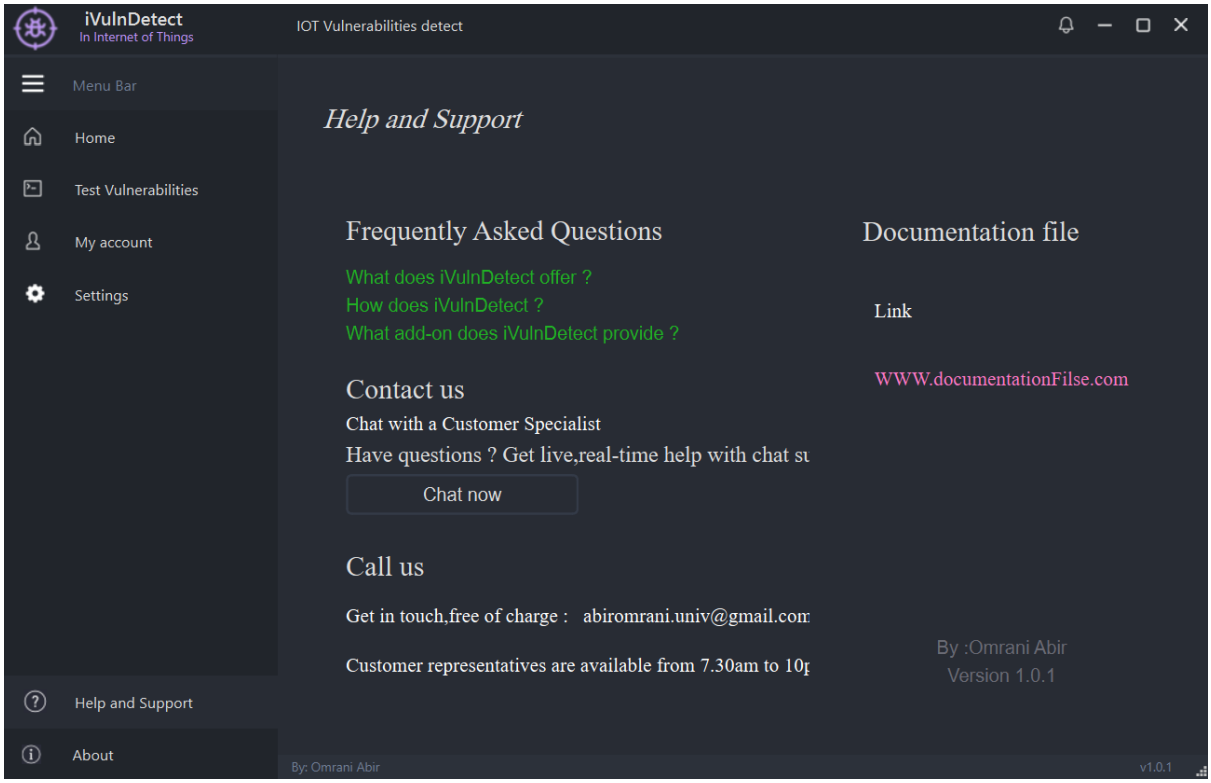


Figure 4.33: Help and support of our system.

4.8 Conclusion

In this chapter, we have covered the various implementation stages of our system. We began by discussing the environment and development tools that were utilized, along with the database setup. The trained model was then tested on the database to evaluate its performance. Additionally, we introduced graphical interfaces to enhance the usability of our system.

To conclude, we presented the results obtained from the implementation process. These insights provide valuable information on the effectiveness and functionality of our system.

General conclusion

The task of avoiding or eliminating security vulnerabilities in IoT operating systems (OS) is undeniably challenging due to the constant innovation and adaptation of hackers and attackers. As technology evolves, new attack vectors and vulnerabilities emerge, necessitating proactive measures to protect IoT systems.

Our project has made a valuable contribution to the field of IoT security by leveraging deep learning techniques to effectively vulnerabilities in IoT operating systems. By harnessing the power of deep learning algorithms, we have achieved remarkable results in enhancing the security of IoT devices and mitigating potential risks, surpassing the capabilities of traditional safety approaches.

One of the key outcomes of our project is the development of a specialized software tool which contributes to:

- Facilitate the discovery and identification of vulnerabilities in IoT OS.
- Provide an important resource for the security of the Internet of Things, empowering developers and system administrators to identify and address vulnerabilities in a more efficient and proactive method.
- Give an effectiveness of the proposed approach, that conducting extensive experiments using a variety of IoT operating systems.

As the Internet of Things continues to grow and evolve, further research and development in the field of IoT security is crucial. In order to advance the state of IoT security, future work could focus on the following areas:

- Expanding the dataset: Incorporating additional vulnerabilities into the dataset would provide a more comprehensive understanding of the security landscape.
- Improved deep learning data processing and models: Enhancements in data processing techniques and the development of more advanced deep learning models can improve the accuracy and effectiveness of vulnerability detection.
- Considering other aspects of IoT security: While this project focuses on operating systems, future research could explore other aspects of IoT security, such as IOT devices and communication protocols, to provide a holistic approach.
- Focusing on precise vulnerability location: Further efforts can be directed towards accurately locating vulnerabilities within IoT systems, enabling targeted and effective remediation.

By continuously refining our approaches and techniques, we can proactively address emerging threats and better protect the integrity and security of IoT systems.
In conclusion, this project represents a significant step towards the development of IoT software vulnerability location using a deep learning-based approach. By contributing to the overall security and reliability of IoT, we pave the way for safer and more trustworthy IoT deployments.

Bibliography

[1] CWE List Version 4.11, October 2021. Accessed: March 2023 .Available at `https://cwe.mitre.org/data/index.html`.

[2] IoT a New Cyber-attack Target. Sennovate, September 2022. Accessed: May 13, 2023 .Available at `https://sennovate.com/iot-a-new-cyber-attack-target/`.

[3] About keras. Website, May 2023. Accessed on May 23, 2023 .Available at `https://keras.io/about/`.

[4] DDoS Attack. KeyCDN, March 2023. Accessed on May 13, 2023 .Available at `https://www.keycdn.com/support/ddos-attack`.

[5] Younes Abbassi and Habib Benlahmer. Un aperçu sur la sécurité de l'internet des objets (an overview of internet of things (iot) security). In *Colloque sur les Objets et systèmes Connectés-COC'2021*, 2021.

[6] Mohammed Riyadh Abdmeziem, Djamel Tandjaoui, and Imed Romdhani. Architecting the internet of things: state of the art. *Robots and Sensor Clouds*, pages 55–75, 2016.

[7] Mohamed Abomhara and Geir M Køien. Cyber security and the internet of things: vulnerabilities, threats, intruders and attacks. *Journal of Cyber Security and Mobility*, pages 65–88, 2015.

[8] Sumeet Kumar Agrawal. Metrics to evaluate your classification model to take the right decisions. *Analytics Vidhya*, May 2023. Accessed: June

8, 2023 .Available at `https://www.analyticsvidhya.com/blog/2021/07/metrics-to-evaluate-your-classification-model-to-take-the-right-decisions/`.

[9] Abdullah Al-Boghdady, Mohammad El-Ramly, and Khaled Wassif. idetect for vulnerability detection in internet of things operating systems using machine learning. *Scientific Reports*, 12(1):17086, 2022.

[10] Abdullah Al-Boghdady, Khaled Wassif, and Mohammad El-Ramly. The presence, trends, and causes of security vulnerabilities in operating systems of iot's low-end devices. *Sensors*, 21(7):2329, 2021.

[11] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, 2015.

[12] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Khalid Al-Ali, Xiaojiang Du, Ihsan Ali, and Mohsen Guizani. A survey of machine and deep learning methods for internet of things (iot) security. *IEEE Communications Surveys & Tutorials*, 22(3):1646–1685, 2020.

[13] Hassan Hadi Al-Maksousy, Michele C Weigle, and Cong Wang. Nids: neural network based intrusion detection system. In *2018 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–6. IEEE, 2018.

[14] Mohammed Ateeq Alanezi. Vulnerabilities, threats and challenges on cyber security and the artificial intelligence based internet of things: A comprehensive study. *IJCSNS*, 22(2):153, 2022.

[15] Yara Alghofaili and Murad A Rassam. A trust management model for iot devices and services based on the multi-criteria decision-making approach and deep long short-term memory technique. *Sensors*, 22(2):634, 2022.

[16] Richard Amankwah, Patrick Kwaku Kudjo, and Samuel Yeboah Antwi. Evaluation of software vulnerability detection methods and tools: a review. *International Journal of Computer Applications*, 169(8):22–27, 2017.

[17] Aqsa Amir. What are iot attacks? Educative, February 2023. Accessed: May 2, 2023 .Available at `https://www.educative.io/answers/what-are-iot-attacks`.

[18] Anastasia Anisimova. Types of machine learning out there. Online, March 2023. Accessed: March 28, 2023 .Available at `https://idapgroup.com/blog/types-of-machine-learning-out-there/`.

[19] Siham Aouad, Abderrahim Maizate, and Abdelouahed Zakari. Cyber security and the internet of things: vulnerabilities and security requirements. *Revue Méditerranéenne des Télécommunications*, 9(2), 2019.

[20] Apache Friends. Xampp. Website. Accessed on May 23, 2023 .Available at `https://www.apachefriends.org/`.

[21] Mayank Banoula. Supervised machine learning: All you need to know. Simplilearn, February 2023. Accessed: March 28, 2023 .Available at `https://www.simplilearn.com/tutorials/machine-learning-tutorial/supervised-machine-learning`.

[22] A. Chapman. Hacking into internet connected light bulbs. Context Information Security, 2014. Available at `https://www.contextis.com/en/blog/hacking-into-internet-connected-light-bulbs`.

[23] Yash Choudhary, B Umamaheswari, and Vijeta Kumawat. A study of threats, vulnerabilities and countermeasures: An iot perspective. *Shanlax Int. J. Arts Sci. Humanit*, 8:39–45, 2021.

[24] Alem Čolaković and Mesud Hadžialić. Internet of things (iot): A review of enabling technologies, challenges, and open research issues. *Computer networks*, 144:17–39, 2018.

[25] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A large-scale analysis of the security of embedded firmwares. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 95–110, 2014.

[26] Mikhail Danyeko, Jonathan Dolan, Kevin Albino, and Huguens Lops. Iot vulnerabilities. 2020.

[27] Dina Darwish. Improved layered architecture for internet of things. *Int. J. Comput. Acad. Res.(IJCAR)*, 4(4):214–223, 2015.

[28] DataRobot. The value of model accuracy. *DataRobot Blog*, July 2020. Accessed: June 8, 2023 .Available at `https://www.datarobot.com/blog/the-value-of-model-accuracy/#:~:text=Model%20accuracy%20is%20defined%20as,certainly%20not%20the%20only%20way`.

[29] Vinicius Rafael Lobo de Mendonca, Cassio Leonardo Rodrigues, Fabrízzio Alphonsus A de MN Soares, and Auri Marcelo Rizzo Vincenzi. Static analysis techniques and tools: A systematic mapping study. *ICSEA*, 2013.

[30] Julianna Delua. Supervised vs. unsupervised learning: What's the difference? IBM Analytics, Data Science/Machine Learning, March 2021. Accessed: March 29, 2023 .Available at `https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning`.

[31] Charalampos Doukas. *Introduction to the Internet of Things*. December 2012.

[32] M Umar Farooq, Muhammad Waseem, Sadia Mazhar, Anjum Khairi, and Talha Kamal. A review on internet of things (iot). *International journal of computer applications*, 113(1):1–7, 2015.

[33] Earlence Fernandes, Amir Rahmati, Jaeyeon Jung, and Atul Prakash. Security implications of permission models in smart-home application frameworks. *IEEE Security & Privacy*, 15(2):24–30, 2017.

[34] Pietro Ferrara, Amit Kr Mandal, Agostino Cortesi, and Fausto Spoto. Static analysis for discovering iot vulnerabilities. *International Journal on Software Tools for Technology Transfer*, 23:71–88, 2021.

[35] Fileeagle. Modelio. Website. Accessed on May 23, 2023 .Available at `https://www.fileeagle.com/software/fr/507/Modelio`.

[36] Elizabeth Fong, Romain Gaucher, Vadim Okun, Paul E Black, and Eric Dalci. Building a test suite for web application scanners. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, pages 478–478. IEEE, 2008.

[37] Google. Colaboratory. Website. Accessed on May 23, 2023 .Available at `https://research.google.com/colaboratory/faq.html`.

[38] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020.

[39] DJALLEL HAMOUDA. Un système de détection d'intrusion pour la cybersécurité. 2020.

[40] Yasmine Harbi. *Security in internet of things*. PhD thesis, 2021.

[41] Hunter Heidenreich. What are the types of machine learning? Towards Data Science, March 2023. Accessed: March 28, 2023 .Available at `https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f`.

[42] Fatima Hussain, Rasheed Hussain, Syed Ali Hassan, and Ekram Hossain. Machine learning in iot security: Current solutions and future challenges. *IEEE Communications Surveys & Tutorials*, 22(3):1686–1721, 2020.

[43] iosentrix. Internet of things and owasp top 10. iosentrix, March 2023. Accessed: March 12, 2023 .Available at `https://iosentrix.com/blog/internet-of-things-and-owasp-top-10/`.

[44] ITU-T. ITU-T Y.4050/Y.2069- (07/2012). Terms and definitions for the Internet of things. ITU-T Recommendation Y.4050/Y.2069, International Telecommunication Union, July 2012. Available at `https://www.itu.int/rec/T-REC-Y.2069-201207-I/en`.

[45] Kate Jackson. How to transform your home into a smart home. Voltora Industries, March 2021. Accessed: May 23, 2023 .Available at `https://voltoraindustries.com.au/how-to-transform-your-home-into-a-smart-home/`.

[46] Mohammad Ali Jabraeil Jamali, Bahareh Bahrami, Arash Heidari, Parisa Allahverdizadeh, and Farhad Norouzi. *Towards the Internet of Things: Architectures, Security, and Applications*. google books, June 2019.

[47] Keyurbhai Arvindbhai Jani and Nirbhay Chaubey. Iot and cyber security: Introduction, attacks, and preventive steps. In *Quantum Cryptography and the Future of Cyber Security*, pages 203–235. IGI Global, 2020.

[48] JavaTpoint. Reinforcement machine learning. Online, March 2023. Accessed: March 29, 2023 .Available at `https://www.javatpoint.com/reinforcement-learning`.

[49] javatpoint. Supervised machine learning. javatpoint, March 2023. Accessed: March 28, 2023 .Available at `https://www.javatpoint.com/supervised-machine-learning`.

[50] JavaTpoint. Unsupervised machine learning. Website, March 2023. Accessed: March 28, 2023 .Available at `https://www.javatpoint.com/unsupervised-machine-learning`.

[51] JetBrains. Quick start guide. Website. Accessed on May 23, 2023 .Available at `https://www.jetbrains.com/help/pycharm/quick-start-guide.html`.

[52] Jmaxxz. Backdooring the frontdoor. DEF CON, July 2016. Last accessed: April 10, 2023 .Available at `https://doi.org/10.5446/36251`.

[53] Daniel Johnson. What is tensorflow? how it works? introduction & architecture. Website, March 2023. Accessed on May 23, 2023 .Available at `https://www.guru99.com/what-is-tensorflow.html#12`.

[54] Julien Jormot. Quelles économies vont offrir les smart grids à la france? les-smartgrids, August 2017. Accessed: 22nd August 2017 .Available at `https://les-smartgrids.fr/economies-smart-grids-france/`.

[55] Kamlesh Lakhwani, Hemant Kumar Gianey, Joseph Kofi Wireko, and Kamal Kant Hiran. *Internet of Things (IoT): Principles, paradigms and applications of IoT*. Bpb Publications, 2020.

[56] Kamlesh Lakhwani, Hemant Kumar Gianey, Joseph Kofi Wireko, and Kamal Kant Hiran. *Internet of Things (IoT): Principles, Paradigms and Applications of IoT*. google books, 2021.

[57] Xinyu Lei, Guan-Hua Tu, Alex X Liu, Chi-Yu Li, and Tian Xie. The insecurity of home digital voice assistants-vulnerabilities, attacks and countermeasures. In *2018 IEEE conference on communications and network security (CNS)*, pages 1–9. IEEE, 2018.

[58] Marco Lombardi, Francesco Pascale, and Domenico Santaniello. Internet of things: A general overview between architectures, protocols and applications. *Information*, 12(2):87, 2021.

[59] Mihir Rajendra Mahajan. *Color classification using machine learning*. PhD thesis, California State University, Sacramento, 2020.

[60] Rwan Mahmoud, Tasneem Yousuf, Fadi Aloul, and Imran Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *2015 10th international conference for internet technology and secured transactions (ICITST)*, pages 336–341. IEEE, 2015.

[61] Georgios Mavridis. Security mechanisms for internet of things (IoT). Master's thesis, Piraeus, 2017.

[62] Steve Mazur. An introduction to smart transportation: Benefits and examples. Digi Blog, December 2020. Accessed: May 23, 2023 .Available at `https://www.digi.com/blog/post/introduction-to-smart-transportation-benefits`.

[63] Jozef Mocnej, Adrian Pekar, Winston KG Seah, and Iveta Zolotova. Network traffic characteristics of the iot application use cases. Technical report, School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand, 2018.

[64] Saraju P Mohanty, Uma Choppali, and Elias Kougianos. Everything you wanted to know about smart cities: The internet of things is the backbone. *IEEE Consumer Electronics Magazine*, 5(3):60–70, 2016.

[65] Mohit,Gupta$_O MG$. $ML|Typesof Learning–Part 2, February 2023$. *Accessed* : $March 28, 2023. URL$: *https* : $//www.geeksforgeeks.org/mltypes - learning - part2/$.

[66] Mohammad Mezanur Rahman Monjur. *Internet-of-Things (IoT) Security Threats: Attacks on Communication Interface*. PhD thesis, University of New Hampshire, 2020.

[67] Philipp Morgner, Stephan Mattejat, and Zinaida Benenson. All your bulbs are belong to us: Investigating the current state of security in connected lighting systems. *arXiv preprint arXiv:1608.03732*, 2016.

[68] Hajra Naeem and Manar H Alalfi. Identifying vulnerable iot applications using deep learning. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 582–586. IEEE, 2020.

[69] Mukrimah Nawir, Amiza Amir, Naimah Yaakob, and Ong Bi Lynn. Internet of things (iot): Taxonomy of security attacks. In *2016 3rd international conference on electronic design (ICED)*, pages 321–326. IEEE, 2016.

[70] Weina Niu, Xiaosong Zhang, Xiaojiang Du, Lingyuan Zhao, Rong Cao, and Mohsen Guizani. A deep learning based static taint analysis approach for iot software vulnerability location. *Measurement*, 152:107139, 10 2019.

[71] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[72] OWASP. OWASP. Online, March 2023. Accessed: October 3, 2023 .Available at `https://owasp.org/www-project-internet-of-things/`.

[73] OWASP Foundation. Who is the owasp® foundation? OWASP, March 2023. Accessed: October 3, 2023 .Available at `https://owasp.org/`.

[74] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1 1*, pages 128–144. Springer, 2020.

[75] Subarno Pal, Soumadip Ghosh, and Amitava Nag. Sentiment analysis in the light of lstm recurrent neural networks. *International Journal of Synthetic Emotions (IJSE)*, 9(1):33–39, 2018.

[76] Punyasloka Pattnaik, Ankush Sharma, Mahavir Choudhary, Vijander Singh, Pankaj Agarwal, and Vikas Kukshal. Role of machine learning in the field of fiber reinforced polymer composites: A preliminary discussion. *Materials Today: Proceedings*, 44:4703–4708, 2021.

[77] pcmag. Pycharm. Website. Accessed on May 23, 2023 .Available at `https://www.pcmag.com/encyclopedia/term/pycharm`.

[78] Prabanjan Raja. What is google colab? Website, Feb 2022. Accessed on May 23, 2023 .Available at `https://www.scaler.com/topics/what-is-google-colab/`.

[79] Python Software Foundation. What is python? executive summary. Website. Accessed on May 23, 2023 .Available at `https://www.python.org/doc/essays/blurb/`.

[80] Abdullah Qasem, Paria Shirani, Mourad Debbabi, Lingyu Wang, Bernard Lebel, and Basile L Agba. Automatic vulnerability detection in embedded devices and firmware: Survey and layered taxonomies. *ACM Computing Surveys (CSUR)*, 54(2):1–42, 2021.

[81] Qt. *Qt Designer Manual*. Qt. Accessed on May 23, 2023 .Available at `https://doc.qt.io/qt-6/qtdesigner-manual.html`.

[82] Sabita Rajbanshi. Everything you need to know about machine learning. Analytics Vidhya, March 2021. Published on March 30, 2021. Accessed: March 28, 2023 .Available at `https://www.analyticsvidhya.com/blog/2021/03/everything-you-need-to-know-about-machine-learning/`.

[83] Rabie Ramadan. Internet of things (iot) security vulnerabilities: A review. *PLOMS AI*, 2(1), 2022.

[84] M Reddy, Engineering Student, Assoc Professor, and Revu Krishnamohan. Applications of iot: A study, 04 2017.

[85] B. Rodrigues. Arris cable modem has a backdoor in the backdoor. Blog, November 2015. Available at `https://w00tsec.blogspot.com/2015/11/arris-cable-modem-has-backdoor-in.html`.

[86] Karen Rose, Scott Eldridge, and Lyman Chapin. The internet of things: An overview. *The internet society (ISOC)*, 80:1–50, 2015.

[87] Rebecca Russell, Louis Kim, Lei Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul Ellingwood, and Marc McConley. Automated vulnerability detection in source code using deep representation learning. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pages 757–762. IEEE, 2018.

[88] Rayan Al Sarih. Internet of things: Fundamentals and applications. Joun Technologies, 2020.

[89] Iqbal H Sarker. Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2(6):420, 2021.

[90] Laura Schröder, Nikolay Krasimirov Dimitrov, David Robert Verelst, and John Aasted Sørensen. Wind turbine site-specific load estimation using artificial neural networks calibrated by means of high-fidelity load simulations. In *Journal of Physics: Conference Series*, volume 1037, page 062027. IOP Publishing, 2018.

[91] Cheena Sharma and Naveen Kumar Gondhi. Communication protocol stack for constrained iot systems. In *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*, pages 1–6. IEEE, 2018.

[92] Guicheng Shen and Bingwu Liu. The visions, technologies, applications and security issues of internet of things. In *2011 International conference on E-Business and E-Government (ICEE)*, pages 1–4. IEEE, 2011.

[93] Harald Sundmaeker, Patrick Guillemin, Peter Friess, Sylvie Woelfflé, et al. Vision and challenges for realising the internet of things. *Cluster of European research projects on the internet of things, European Commision*, 3(3):34–36, 2010.

[94] Syeda Manjia Tahsien, Hadis Karimipour, and Petros Spachos. Machine learning based solutions for security of internet of things (iot): A survey. *Journal of Network and Computer Applications*, 161:102630, 2020.

[95] Usman Tariq, Irfan Ahmed, Ali Kashif Bashir, and Kamran Shaukat. A critical cybersecurity analysis and future research directions for the internet of things: A comprehensive review. *Sensors*, 23(8):4117, 2023.

[96] Monisha Macharla Vasu. *INTRODUCTION TO INTERNET OF THINGS: A DEFINITE GUIDE TO LEARN IOT - ENABLING TECHNOLOGIES, CONNECTIVITY, PROTOCOLS AND CLOUD.* (Publisher name, if available), 2021. Kindle Edition.

[97] Umer Wani. An introduction to iot , its architecture and various protocols. chapter outline. 08 2019.

[98] Website. Pyside6, May 2023. Accessed on May 23, 2023 .Available at `https://www.pythonguis.com/topics/pyside6/`.

[99] Tasneem Yousuf, Rwan Mahmoud, Fadi Aloul, and Imran Zualkernan. Internet of things (iot) security: current status, challenges and countermeasures. *International Journal for Information Security Research (IJISR)*, 5(4):608–616, 2015.

[100] Handong Zhang and Lin Zhu. Internet of things: Key technology, architecture and challenging problems. In *2011 IEEE International Conference on Computer Science and Automation Engineering*, volume 4, pages 507–512. IEEE, 2011.

[101] Yousaf Bin Zikria, Sung Won Kim, Oliver Hahm, Muhammad Khalil Afzal, and Mohammed Y Aalsalem. Internet of things (iot) operating systems management: Opportunities, challenges, and solution. *Sensors*, 19(8):1793, 2019.