HELWAN UNIVERSITY
Faculty of Computers and Artificial Intelligence
Information Systems Department

[Secure To Conquer]

A graduation project dissertation by:

[Ahmed Mohammed Sayed (20210100)]

[Abdelrahman Sayed Zenhom (20210508)]

[Ahmed Abdelhamid Zaki (20210069)]

[Saad Hazem Saad (20210406)]

[Ahmed Mohammed Mohammed (20210109)]

[Ahmed Mohammed Ramadan (20210099)]

Submitted in partial fulfilment of the requirements for the degree of Bachelor of Science in Computers & Artificial Intelligence, at the Information System Department, the Faculty of Computers & Artificial Intelligence, Helwan University

Supervised by:

[Dr. Abdullah Yassin]

June 2025

Abstract

Secure to Conquer (STC) is a next-generation Secure Banking System designed to merge intelligent automation with modern cybersecurity practices. The project integrates advanced tools and technologies across web development, machine learning, and security domains to deliver a trusted and efficient banking environment. Key technologies include **ASP.NET Core** for backend services, **machine learning models** for predictive decision-making, and **FastAPI** for seamless frontend-backend interaction. Security is reinforced through the integration of **Wazuh SIEM** for real-time threat detection and log analysis, as well as **VirusTotal API** for proactive malware scanning. By aligning cutting-edge development frameworks with enterprise-grade security tools, STC provides a scalable and resilient foundation for secure digital banking operations.

Introduction

Secure to Conquer (STC) — a Secure Banking System designed to strike an effective balance between **performance, usability, and cybersecurity** across modern financial operations. STC integrates multiple components spanning **machine learning**, **backend development**, and **advanced security tools**, forming a resilient and intelligent infrastructure for secure digital banking.

The backend is built on a modular, API-driven architecture using **ASP.NET Core**, supporting key functionalities such as secure user authentication (including OTP and social login), role-based access control, transaction management, real-time communication via SignalR, cheque generation in PDF format, and file scanning. This structure ensures maintainability and scalability without compromising performance.

On the intelligence side, STC incorporates two machine learning models: a **logistic regression model** for credit approval prediction, enabling users to assess eligibility before visiting the bank, and an **XGBoost-based model** for detecting malicious URLs. The latter is deployed using **FastAPI**, allowing secure, real-time predictions through a CORS-enabled REST API.

To address increasing security demands without introducing excessive complexity, the system integrates the **Wazuh SIEM platform** to automate threat detection, log analysis, and behavioral monitoring through modules like rootcheck and syscheck. It also maps

Windows Defender logs to the **MITRE ATT&CK framework**, enhancing threat intelligence capabilities. Additionally, the backend connects to the **VirusTotal API**,

using SHA-256 hashing to verify uploaded files for malware — ensuring proactive protection without affecting system responsiveness and mitigate other malicious attacks

Ultimately, **Secure to Conquer (STC)** demonstrates how a well-architected solution can maintain a strategic balance between **robust security**, **system performance**, and **architectural complexity**, creating a secure, responsive, and user-focused banking environment.

Keywords

SOC (Security Operations Center), SIEM Solution (Security Information and Event Management.),Wazuh , Dinner time attack , Infrastructure, VirusTotal , MITRE ATT&CK framework

Acknowledgement

We would like to express our deepest gratitude to **Dr. Abdullah Yassin** for his invaluable support and guidance in shaping our understanding of key **security concepts**, which played a critical role in the foundation of this project. His expertise and insights greatly enhanced the security layers of our system including building SIEM solution , VirusTotal Integration and his continuous motivation for our potentials.

We are also sincerely thankful to **Dr. Mostafa Adel** for his encouragement and constructive feedback. After presenting our project idea to him, his motivation and technical discussions—particularly around **role-based security**—inspired us to take the project further and build a more robust and ambitious solution.
Their combined mentorship was instrumental in the successful development of **Secure to Conquer (STC)**, and we truly appreciate their time, support, and dedication.

كلية الحاسبات والذكاء الإصطناعي
Faculty of Computers & Artificial Intelligence

Table of contents

Table of Contents

Abstract ***2***

Introduction ***2***

Keywords ***3***

Acknowledgment ***3***

Chapter 1: Malware Analysis **5**

1.1 Malware's types and classification 5

1.2 Malware's circulation 6

1.3 Malware's infection 15

1.4 Malware's concealment 20

1.5 Malware's payload capabilities 23

1.6 Malware's Dynamic analysis Vs Malware's static analysis 29

1.7 Malware's real-life examples 33

Chapter 2: SIEM Configuration and Analysis **40**

2.1 What is SIEM? 40

2.2 Benefits of SIEM? 42

2.3 How the SIEM works? 43

2.4 what is open source SIEM solutions ? 48

2.5 How open source SIEM tools work? 53

2.6 What is Wazuh? 54

2.7 How is Wazuh works? 56

2.8 How to build Wazuh? 58

2.9 Key features of Wazuh's dashboard 63

2.10 How Wazuh agent works? 65

2.11 What is Putty? How it works? 70

2.12 Monitoring and alerting setup report 74

2.13 Windows defender configurations 77

2.14 Wazuh's alerting configuration and integration with VirusTotal 77

2.15 Wazuh’s Bruteforce attack detector 85
2.16 Wazuh’s Malicious commands detector 87
2.17 Wazuh’s malicious network intrusion commands detector 91

Chapter 3: Prevention strategy and training.....95
3.1 Malware prevention strategy 95
3.2 Wazuh’s exe used for prevention strategy 100
3.3 Phishing mails shape 114

Chapter 4: SOC analyst final formal report.....116
4.1 Final SOC analyst report 116

Chapter 5: Software assistance in action 124
5.1 Software diagrams for main banking functionalities 124
5.1 Closer look to main security backend related functionalities 139
5.1 Additional backend concerns taken into consideration 152
5.1 Smart machine learning agents in action 154
5.1 Frontend important visualizations and functionalities 170

Chapter 6: Future Learning and recommendations 176

6.1 Future learnings to apply 176
6.1 Important recommendations for production environment and real-life deployment.....178

Glossary

SIEM Solution: A Security Information and Event Management (SIEM) solution collects, analyzes, and correlates logs from various sources to detect and respond to security threats in real time.

Dinner Time Attack: A Dinner Time Attack exploits times when users are distracted, or systems are less monitored (like during lunch/dinner hours) to perform unauthorized actions or breaches.

SOC (Security Operations Center): A SOC is a centralized team and facility that continuously monitors, detects, analyzes, and responds to cybersecurity incidents across an organization.

IOCs (Indicators of Compromise): IOCs are artifacts or pieces of forensic data (like IPs, file hashes, or URLs) that indicate a potential breach or malicious activity within a system.

Siamese CNN Model: A Siamese Convolutional Neural Network is a type of neural network with two identical subnetworks used for comparing two inputs, often used in tasks like face verification or similarity detection.

Chapter 1: Malware Analysis

1.1 Malware Type and Classifications

Malware, or malicious software, refers to any software that infiltrates a computer system without the user's knowledge or consent, subsequently performing harmful and unwanted actions. The term encompasses a wide variety of damaging software programs. As security measures have advanced to combat malware, the complexity of these threats has also increased, leading to the emergence of numerous distinct malware variants, such as the ZeuS malware. However, there is no universally accepted classification system for these various instances, and many existing classifications simply list different types of malwares (e.g., viruses) rather than grouping similar instances into broader categories. This lack of standardization makes Cybersecurity science more broad to understand it's concepts. And into different categories and we will classify them according to their behaviors in this chapter as an important introductory chapter for main Cybersecurity's concepts

1.2 Malware's Circulation

Two types of malwares have the primary trait of circulation. These are viruses and worms.

1.2.1 Virus

A biological virus is an agent that reproduces within a host cell, commandeering the cell's functions to produce numerous copies of itself. Once a cell is infected, it rapidly generates thousands or even millions of identical virus copies—like the polio virus, which can create over a million replicas within a single cell. Biologists often assert that the primary purpose of viruses is to replicate themselves. Similarly, a computer virus is malicious code that self-replicates on the same machine. In strict terms, a computer virus can reproduce itself or a modified version without any human assistance.

Note

Strictly speaking, virus and malware are not interchangeable terms. A virus is only one type of malware

In addition to their self-replicating nature, both biological and computer viruses share the ability to spread rapidly and cause widespread damage. Biological viruses can be transmitted through air, bodily fluids, or surfaces, infecting entire populations if not controlled. Similarly, computer viruses can spread through infected files, email attachments, malicious websites, or removable storage devices. Once active, they may corrupt data, steal sensitive information, degrade system performance, or disable essential functions. Some sophisticated computers viruses even evade detection by mutating their code—much like how biological viruses evolve to resist immune responses or treatments. Despite their destructive capabilities, the study of viruses has led to advancements in science and cybersecurity. For example, understanding viral mechanisms have contributed to vaccine development in medicine and the design of advanced antivirus and threat detection systems in computing.

Note

The first macro virus emerged in 1995, primarily targeting Microsoft Word documents and becoming the most common type of virus until 2000, when Microsoft disabled macros by default in its Office products. Despite this, macro viruses have seen a resurgence as threat actors find new methods to deceive victims into enabling macros, allowing these viruses to execute.

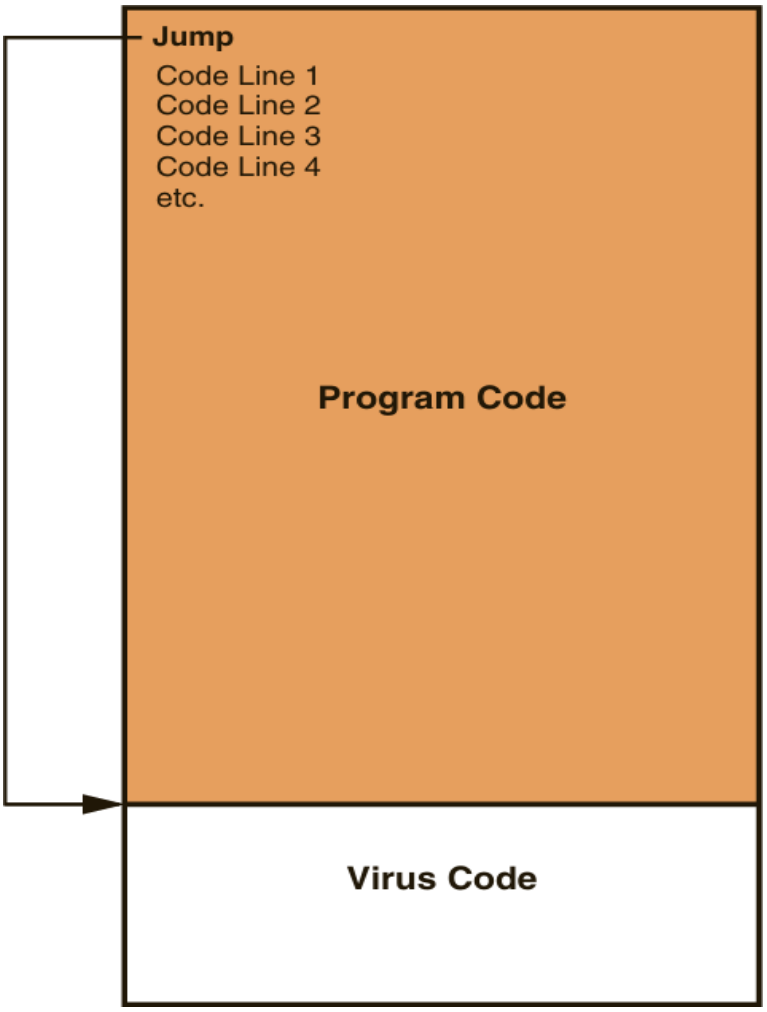


Fig [1-1] Describes Appender Infection

However, these types of viruses could be detected by virus scanners relatively easily. Most viruses today go to great lengths to avoid detection; this type of virus is called an armored virus. Some of the armored virus infection techniques include:

Swiss cheese infection: which involves a dual-layered approach. Instead of a single jump instruction leading to the main virus code, these viruses first encrypt their malicious code, rendering it more challenging for scanners to identify. They then fragment the decryption engine into multiple components, scattering these pieces.

throughout the infected program's code. Upon execution, the program reassembles these fragments to reveal the original virus, creating a convoluted path for detection, **Swiss cheese infection** derives its name from the resemblance to Swiss cheese, which is characterized by numerous holes and gaps. In this context, the "holes" refer to the way the armored virus splits and distributes its code throughout the host program. Just as Swiss cheese has irregularly placed holes, the virus fragments are scattered at various points within the code, making it difficult for antivirus software to detect the malicious components. A Swiss cheese infection is shown in Figure 1-2.

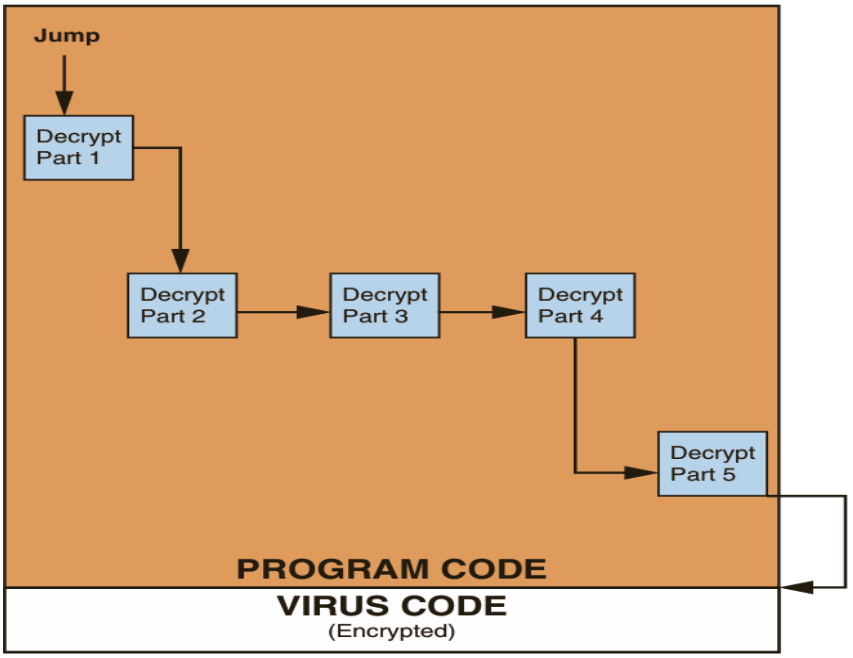


Fig [1-2] Describes Swiss Cheese Infection

split infection in this method the virus itself is divided into several segments, which are interspersed randomly within the host program's code. To further obscure its presence, these segments may include extraneous "garbage" code, further disguising the virus's true intent. This strategic placement not only complicates detection efforts but also heightens the risk of infection, A split infection virus is shown in Figure 1-3.

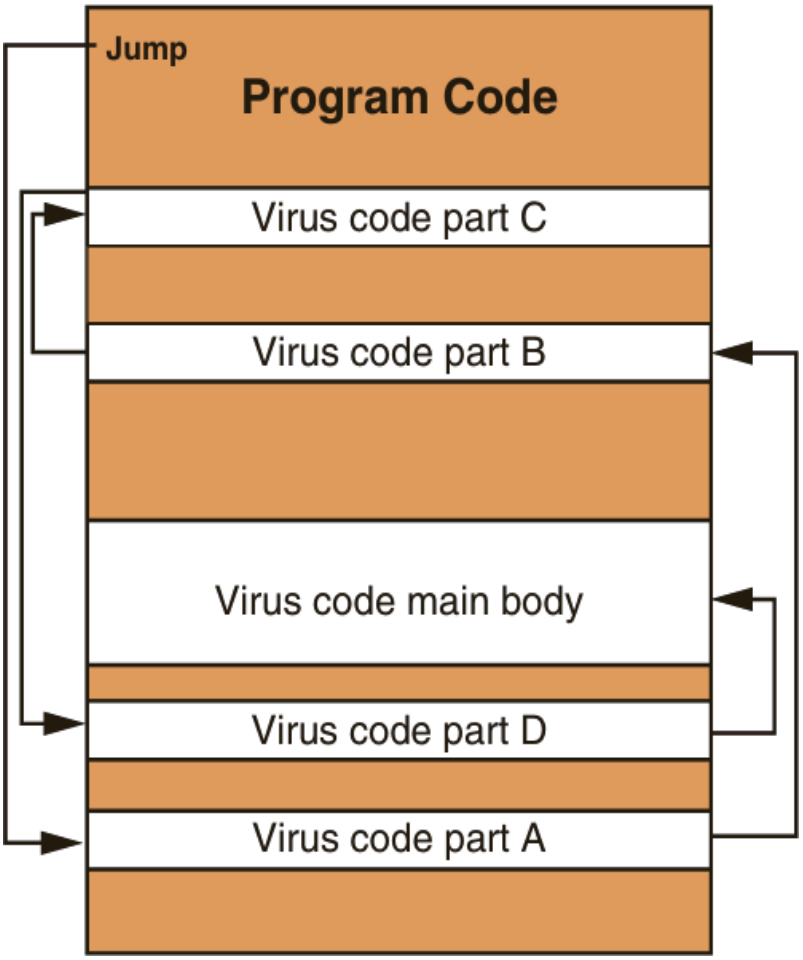


Fig [1-3] Describes Split Infection

Mutation is a critical characteristic of certain computer viruses, allowing them to evade detection and enhance their survival. Unlike simple viruses that merely conceal themselves within files, some possess the ability to alter their code dynamically. **Oligomorphic viruses** modify their internal structure to one of a limited set of predefined mutations upon execution, whereas **polymorphic viruses** entirely change their form each time they are activated. The most advanced, **metamorphic viruses**, can rewrite their own code, generating a logical equivalent that appears different with every execution. And figure 1-4 show the difference between polymorphic and Metamorphic.

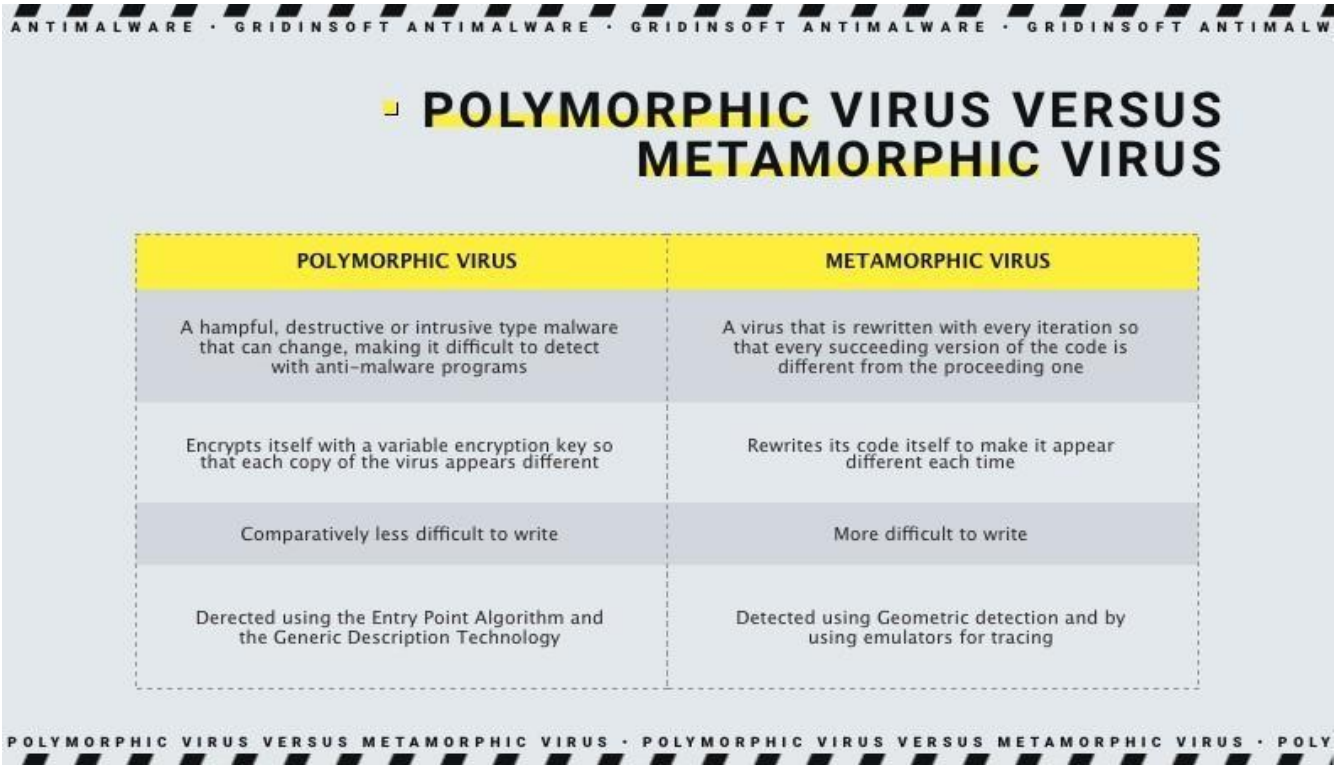


Fig [1-4] Describes the difference between Polymorphic and Metamorphic virus

At the end, when an infected program is initiated or a compromised file is accessed, the virus executes two primary functions. Firstly, it releases a payload designed to execute malicious actions. Modern viruses are significantly more destructive than their early counterparts, which primarily displayed nuisance messages; contemporary variants can corrupt or delete files, prevent applications from launching, exfiltrate sensitive data, instigate repeated system crashes, and disable security settings.
Secondly, upon execution, the virus replicates itself by embedding its code into additional files, but this replication is confined to the original host computer. Viruses lack the capability for autonomous transmission to other systems; they depend on user actions for dissemination. For instance, a user may inadvertently share an infected file via email or transfer it using a USB flash drive. Upon reaching a new host computer, the virus begins the infection cycle anew. Thus, a virus requires two essential components for propagation: a file to which it attaches and a human agent to facilitate its transfer to other systems.

Note

Sometimes a virus will remain dormant for a period before unleashing its payload

1.2.2 Worm

A second category of malware specifically designed for propagation is the **worm**. Worms are malicious programs that exploit computer networks to replicate themselves, often referred to as network viruses. These programs are engineered to infiltrate a computer via the network, taking advantage of vulnerabilities in applications or operating systems on the host machine. Once a worm successfully exploits a vulnerability on one system, it actively seeks out other computers on the network with the same weaknesses.
In their early iterations, worms were relatively harmless, designed primarily for rapid dissemination without causing damage to the infected systems. However, they could significantly degrade network performance, as their rapid replication consumed available resources. Modern worms, on the other hand, often deliver a harmful payload to the systems they infect, akin to traditional viruses. Their actions can include deleting files on the infected computer or enabling remote control by an attacker, thereby posing a considerable threat to both individual systems and network integrity.

Note

Although viruses and worms are said to be automatically self-replicating, where they replicate is different. A virus self-replicates on the host computer but does not spread to other computers by itself. A worm self-replicates between computers (from one computer to another)

Action	Virus	Worm
What does it do?	Inserts malicious code into a program or data file.	Exploits a vulnerability in an application or operating system
How does it spread to other computers?	User transfers infected files to other devices	Uses a network to travel from one computer to another
Does it infect a file?	Yes	No
Does there need to be user action for it to spread?	Yes	No

1.3 Malware's Infection

1.3.1 Trojans

Named after the legendary Trojan horse from Greek mythology, which allowed Greek soldiers to secretly infiltrate the city of Troy, a computer Trojan operates in a similar deceptive manner. It disguises itself as a legitimate application, fooling users into installing it under the belief that it will perform harmless or useful tasks. However, alongside the advertised function, it secretly executes malicious activities, such as:

1- Collecting sensitive information like credit card details and passwords.
2- Transmitting this information to attackers via remote servers.

A particularly dangerous subtype of Trojan is the Remote Access Trojan (RAT). In addition to its standard Trojan functionality, a RAT allows attackers unauthorized control over the infected device. Once inside, the threat actor can monitor the user's actions, access files, modify settings, and even use the infected computer to spread the infection across a network.

1.3.2 Ransomware

Ransomware is an aggressive form of malware that blocks access to a device or its files until a ransom is paid. After embedding itself into the system, it locks down functions, often launching immediately upon restart to prevent circumvention.

Early Ransomware (Blocker Ransomware): Initially, ransomware focused on freezing the entire computer, displaying a full-screen message that impersonated a legitimate entity, such as law enforcement or software providers. These warnings claimed that the user had committed illegal activities or that critical system issues needed resolution, prompting immediate payment of a "fine" or fee to restore access.

Modern ransomware has abandoned pretense. Now, it simply locks the system, demanding payment for its release. This model has become so lucrative that

ransomware attacks have skyrocketed, with billions of dollars lost each year. Notably, even if victims pay, there's only a limited chance they will successfully recover their files. As you can see in figure 1-5 and figure 1-6 the shape of ransomware infection.

Fig [1-5] and Fig [1-6] Shows the shape of ransomware infection.

1.3.3 Crypto-malware

Building on the foundations of ransomware, crypto malware takes the attack a step further. Instead of just blocking access to a system, it encrypts critical files so that they cannot be opened or used without an encryption key. Victims are told that they must pay for this decryption key, with the threat of permanent data loss if they refuse.
The encryption process begins with the malware connecting to the attacker's Command & Control (C&C) server, where it receives an encryption key. Once the files are locked, the malware sends the decryption key back to the C&C server, and victims can only retrieve it after paying the ransom, and you can see its shape in figure 1-7.

Recent Advances in Crypto malware:

- It now targets not just the local computer but any connected devices and network storage, which could affect entire enterprises.
- Some forms also infect mobile devices, expanding the scope of potential damage

Fig [1-7] Shows crypto-malware infection

Feature	Trojans	Ransomware	Crypto-Malware
Primary Trait	Deception (masquerades as legitimate apps)	Locks system access	Encrypts files, making them unusable
Malicious Activity	Steals data (passwords, credit cards)	Blocks device until ransom is paid	Encrypts files; demands ransom for decryption
Method of Infection	User unknowingly installs malicious software	Embeds into system, launches on boot	Encrypts files after receiving key from C&C
Severity	Medium: Can be detected and removed	High: Prevents system use until payment	Critical: Loss of data unless ransom is paid
Subtypes	Remote Access Trojan (RAT)	Blocker ransomware	Advances include file & network encryption
Impact on Networks	Can spread to other devices in network	Typically isolated to one device	Can infect multiple devices & network storage
Target Devices	Computers and networks	Computers and mobile devices	Computers, networks, cloud storage, mobile devices

Each of these malware types poses significant risks, with Trojans focusing on stealth and data theft, ransomware on extortion, and crypto malware presenting the most destructive potential through encryption. By understanding their differences, users can take appropriate steps to protect their systems.

1.3 Malware's Concealment

1.4.1 Definition of Concealment

In the field of malware security, concealment refers to the deliberate act of hiding malicious code or activities from detection by security tools and system users. Malware developers use various techniques to ensure that their software operates in the background without raising alarms, such as obfuscating code, embedding malicious components in legitimate files, or using rootkits to mask processes. This concealment is intentional, designed to prolong the malware's presence and effectiveness on a compromised system, often to steal sensitive data or cause damage without immediate detection. Unintentional concealment can also occur when malware exploits unknown vulnerabilities, allowing it to remain hidden for extended periods, sometimes only discovered after significant harm has been done.

1.4.2 Types of Concealment

- Physical Concealment: can be seen in hardware-based attacks where malicious devices or implants are hidden in physical environments (e.g., USB sticks with malware).

- Informational Concealment: is more prominent, as malware often hides critical information from security systems. Techniques such as obfuscation, encryption, and fileless malware allow attackers to keep malicious code and data hidden from antivirus programs and intrusion detection systems.

- Emotional Concealment: can be compared to the way phishing attacks exploit human psychology by hiding the true intent of malicious communications, creating a false sense of trust or urgency, thus leading users to reveal sensitive data or install harmful software without realizing the threat. These layers of concealment make it challenging for cybersecurity professionals to identify and mitigate risks effectively.

1.4.3 Concealment in Law

- Fraudulent Concealment: refers to deliberate actions taken by malicious software, such as rootkits, to hide its presence or the presence of other malware on a system. Rootkits are specifically designed to access lower layers of the operating system, often using undocumented functions, to modify system behavior and evade detection. This manipulation allows rootkits to conceal files, processes, and activities from antivirus programs and operating systems, similar to how fraudulent concealment in law involves intentionally hiding facts to deceive others.

- Legal Consequences: of malware-related concealment can be severe, leading to fines or imprisonment for those involved in the creation and dissemination of such malicious software. Just as in legal cases, where withholding critical information leads to penalties, malware authors face harsh penalties when their deceit is uncovered, particularly when the malware causes substantial damage to systems or sensitive data.

1.4.4 Concealment in Cybersecurity

- Network Concealment involves techniques like VPNs, proxy servers, and Tor to hide or mask network traffic, making it difficult for cybersecurity tools to detect or trace back malicious activity.
- Stealth Malware: such as rootkits, epitomizes advanced concealment by embedding itself deep within a system's operating layers. Rootkits can alter the operating system’s functioning, making malicious files invisible to security tools and scanners. For instance, when a rootkit infects a computer, it hides both its presence and the presence of other malware by accessing lower layers of the operating system, making these files undetectable by antivirus software.

1.4.5 Military Concealment

- Camouflage: can be likened to the stealthy techniques used by malware to hide its presence and evade detection, similar to how armies use camouflage to make personnel, equipment, or positions less visible on the battlefield. Malware, like rootkits, employs advanced "camouflage" by embedding itself deep within a computer's operating system, making malicious files or processes invisible to security tools.

- Deception Tactics: Just as soldiers use deception tactics such as decoys or false information to mislead the enemy, malware employs strategies like misleading system logs, altering file timestamps, or hiding in legitimate processes to deceive antivirus software and users. For example, rootkits access lower layers of the operating system to conceal their presence and that of other malware, making it impossible for scanning software to detect them. This "military-style" concealment allows the malware to continue operating in the background, much like a hidden army quietly advancing, undetected by its adversaries.

1.5 Malware's Payload Capabilities

In cybersecurity, "payload" refers to the malicious code or actions that are delivered and executed after an exploit or vulnerability is triggered. This can include malware, ransomware, or other harmful programs designed to damage, steal, or manipulate data. In software, the term "payload" often refers to the portion of a program or message that performs the intended function, as opposed to the surrounding code or metadata. In networking, "payload" is the data being transmitted within a packet, excluding the headers and metadata used for routing and transmission. Essentially, it's the core information or executable part that fulfills the purpose of a cyber operation or communication.

Spyware

Spyware is tracking software installed without the user's knowledge or consent. It typically monitors user activity in secret, gathering information using the computer's existing resources and programs. This data is then collected and distributed without the user's approval, often involving personal or sensitive information. One particularly malicious type of spyware is a keylogger.

Keyloggers quietly record every keystroke a user types on their keyboard, enabling attackers to sift through the captured data for valuable details such as passwords, credit card numbers, or personal information. And Software Keyloggers shown in Figure 1-8

Fig [1-8] Shows Software Keylogger.

Early keyloggers were hardware devices that were inserted between the computer's keyboard connection and the USB port, as shown in Figure 1-9. These devices looked like ordinary plugs and were often connected to the USB port at the back of the computer, making them difficult to detect. Since they are physical devices, they are not scanned by antimalware software, allowing them to operate undetected. However, because attackers must return to physically retrieve the hardware keylogger to access the captured information, hardware keyloggers are not commonly used today.

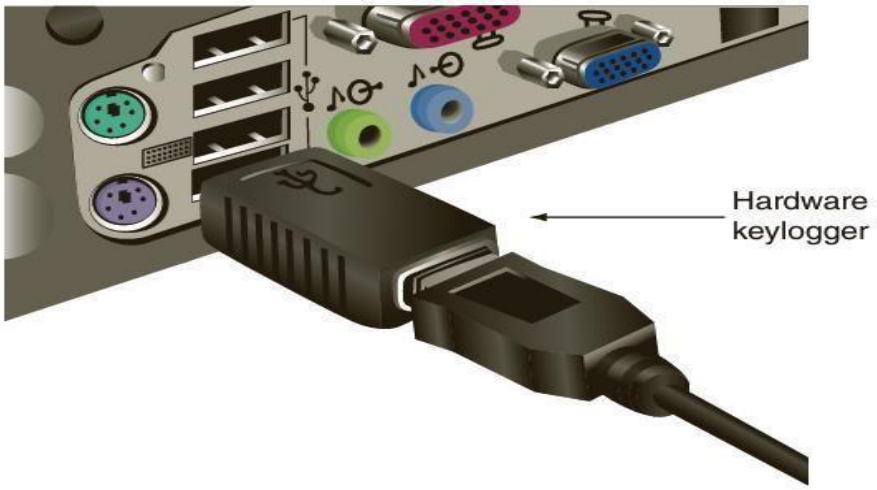


Fig [1-9] Shows hardware Keylogger.

Note

Not all spyware is necessarily malicious. For example, spyware monitoring tools can help parents keep track of the online activities of their children

Adware

Adware delivers advertising content in ways that are unexpected and unwanted by users. Once installed, adware typically displays banners, pop-up ads, or opens new browser windows at random intervals. Users generally dislike adware for several reasons:

- It can display inappropriate content, such as gambling or pornography.
- Frequent pop-up ads disrupt productivity.
- These ads can slow down a computer or even cause crashes, potentially leading to data loss.
- The sheer volume of ads is often annoying.

As online advertising has increased, users have responded by installing ad-blocking software in their browsers. Ad blocking grew by 41% worldwide within a year, reaching 198 million users, while U.S. ad blocking rose by 48%, impacting 45 million users. It's estimated that ad blocking costs website publishers around $22 billion annually. To address this, marketers have started implementing paywalls (requiring users to pay to access content without ads), friendly reminders about the purpose of ads (highlighting that ads fund free content), and warnings that content may be blocked if ad blockers are detected.

Logic Bomb

One common type of malware used for data deletion is a logic bomb. A logic bomb is malicious code embedded within a legitimate program that remains inactive until triggered by a specific event. Once activated, the logic bomb can delete data or carry out other harmful actions. For instance, a government employee in Maryland attempted to destroy data on over 4,000 servers by planting a logic bomb set to detonate 90 days after his termination.

A notable recent attack using a logic bomb targeted three banks and two media companies in South Korea. The malware contained four files, including AgentBase.exe, which initiated the attack. Embedded within the file was a hexadecimal string (4DAD4678) representing the date and time of the attack:

March 20, 2013, at 2:00 PM. Once the system clock hit 2:01 PM, the logic bomb activated, overwriting the hard drive and master boot record on Microsoft Windows computers, causing them to reboot and become unusable. The malware also had a module to delete data from remote Linux machines. Detecting logic bombs before they are triggered is challenging. They are often hidden within large programs containing tens of thousands of lines of code, and a trusted insider can easily add a few lines of malicious code without raising suspicion. Additionally, these programs are not routinely scanned for malicious content.

Backdoor

Some types of malwares aim to alter a system's security settings to enable more dangerous attacks. One such type of malware is known as a backdoor. A backdoor provides unauthorized access to a computer, program, or service, bypassing normal security protections. Once installed, it allows attackers to return later and evade security measures.

Note

Developers often create legitimate backdoors to access a program or device regularly without being interrupted by repeated password prompts or security checks. These backdoors are intended to be removed once the application is complete. However, in some cases, backdoors have been left in place, allowing attackers to exploit them and bypass security.

Botnet

A popular malware payload today allows an infected computer to be remotely controlled by an attacker to carry out attacks. These infected machines, known as bots or zombies, are often part of a larger network called a botnet, which is controlled by a bot herder. Botnets can consist of hundreds, thousands, or even millions of compromised computers.

Bot computers receive instructions from the bot herder through a command-and-control (C&C) structure. This communication can occur in several ways, including:

- Bots may automatically log in to a website run by the bot herder, where commands are posted for them to interpret and execute.
- Bots can log in to third-party websites, allowing the bot herder to remain unaffiliated with the site.
- Commands can be delivered through blogs, Twitter posts, or Facebook notes, using specially coded instructions.
- Increasingly, bot herders use a "dead drop" C&C method, where they create a Google Gmail account and leave commands in a draft email that is never sent. Bots log in to the account, read the draft, and receive the instructions. Since the email is never sent, there is no record of it, and Gmail's encryption ensures the transmission remains private.

The scale of botnets is staggering. According to the FBI’s cyber division, 18 computers are infected and added to botnets every second, resulting in hundreds of millions of compromised machines each year. Some botnets control 3 to 4 million bots, while another botnet used for email spam was responsible for sending up to 60 billion emails daily.

1.6 Malware's Dynamic Analysis vs. Static Analysis

1.6.1 What's the Difference?

Dynamic analysis involves analyzing a program while it is running, allowing for the examination of its behavior in real-time. This method is useful for detecting runtime errors and performance issues. On the other hand, static analysis involves examining the code without executing it, focusing on identifying potential bugs and vulnerabilities before the program is run. While dynamic analysis provides more accurate results, static analysis is more efficient and can be performed earlier in the development process. Both methods have their strengths and weaknesses, and a combination of both is often used to ensure comprehensive testing of software.

Attribute	Dynamic Analysis	Static Analysis
Timing	Occurs during runtime	Occurs before runtime
Code execution	Code is executed	Code is not executed
Performance impact	May slow down the system	No performance impact
Identifying bugs	Can find runtime bugs	Can find syntax errors and potential bugs

Automation	Can be automated	Cannot be automated

1.6.2 Further Details

1.6.2.1 Introduction

Dynamic analysis and static analysis are two common techniques used in software testing to identify defects and vulnerabilities in software applications. While both methods aim to improve the quality and security of software, they differ in their approach and the types of issues they can uncover. In this article, we will compare the attributes of dynamic analysis and static analysis to help you understand their strengths and weaknesses.

1.6.2.2 Dynamic Analysis

Dynamic analysis, also known as black-box testing, involves executing the software application and observing its behavior in real- time. This technique simulates the actual runtime environment of the software and can uncover issues related to performance, memory leaks, and security vulnerabilities that may only manifest during execution. Dynamic analysis tools typically include profilers, debuggers, and fuzzers that help testers identify and diagnose problems in the software.

- Executes the software application to observe its behavior.
- Simulates the actual runtime environment.

- Uncover issues related to performance, memory leaks, and security vulnerabilities.
- Includes profilers and debuggers.

1.6.2.2 Static Analysis

Static analysis, on the other hand, is a white box testing technique that involves examining the source code or binary of the software without executing it. This method focuses on identifying issues such as coding errors, security vulnerabilities, and compliance violations by analyzing the code structure, syntax, and dependencies. Static analysis tools use algorithms and rules to scan the codebase and generate reports on potential issues that may exist in the software.

- Examines the source code or binary without executing it.
- Focuses on identifying coding errors, security vulnerabilities, and compliance violations.
- Analyzes the code structure, syntax, and dependencies.
- Uses algorithms and rules to scan the codebase.

1.6.2.3 Comparison

Dynamic analysis and static analysis have their own strengths and weaknesses when it comes to software testing. Dynamic analysis is effective at uncovering runtime issues and performance bottlenecks that may not be apparent during static analysis. It can also help identify security vulnerabilities that only manifest when the software is running. However, dynamic analysis can be time-consuming and may not provide a comprehensive view of all possible issues in the software. On the other hand, static analysis is great for identifying coding errors and security vulnerabilities early in the development process. It can help developers catch issues before they become critical problems and improve the

overall quality of the codebase. Static analysis is also faster than dynamic analysis since it does not require the software to be executed. However, static analysis may produce false positives and may not catch all runtime issues that dynamic analysis can uncover.

1.6.2.4 Conclusion

In conclusion, both dynamic analysis and static analysis are valuable techniques in software testing that offer unique benefits to developers and testers. Dynamic analysis is great for uncovering runtime issues and security vulnerabilities that may only manifest during execution, while static analysis is effective at identifying coding errors and security vulnerabilities early in the development process. By combining both techniques in a comprehensive testing strategy, developers can ensure that their software is of the highest quality and security standards.

1.7 Malware's Real-life examples

1.7.1 The Target Data Breach (2013)

One of the most infamous real-life examples of a malware and social engineering attack occurred during the Target data breach in 2013.
Cybercriminals used a phishing attack to trick a third-party vendor, gaining access to Target's network. Once inside, the attackers deployed malware to capture credit card information from over 40 million customers at point-of-sale terminals. This attack combined social engineering to manipulate an unsuspecting user into granting access and malware to stealthily extract sensitive financial data from Target's systems, all while remaining undetected for several weeks.

Video Explanation: (196) Target 2013 Data Breach Explained - YouTube

1.7.2 The Target Data Breach (2013)

The WannaCry ransomware attack in 2017 was a global cyberattack that affected thousands of organizations, including hospitals, businesses, and government agencies. WannaCry spread through a vulnerability in Microsoft's Windows operating system, encrypting files on infected computers and demanding ransom payments in Bitcoin to unlock them. In this case, the malware didn't rely on sophisticated social engineering but exploited a known system vulnerability. However, many victims were tricked into opening malicious email attachments that initiated the attack. The rapid spread of the malware across networks demonstrated how damaging a malware attack can be when paired with weak cybersecurity defenses and social engineering to trigger initial infection.

Video Explanation: WANNACRY: The World's Largest Ransomware Attack (Documentary) - YouTube

1.7.3 The Twitter Bitcoin Scam (2020)

In 2020, several high-profile Twitter accounts, including those of Barack Obama, Elon Musk, and Bill Gates, were compromised in a social engineering attack that led to a widespread Bitcoin scam.
Hackers used social engineering tactics to trick Twitter employees into giving access to internal tools. Once the attackers gained control, they tweeted messages from the compromised accounts, promising to double any Bitcoin sent to a specific address. While this attack didn't involve malware, it was a stark example of how social engineering can be leveraged to exploit trusted platforms and cause financial damage on a massive scale, affecting both individuals and the companies involved.

Video Explanation: The Teenager Who Hacked Twitter And Stole Millions In Bitcoin (youtube.com)

1.7.4 Stuxnet (2010)

Stuxnet is a highly targeted worm designed to infiltrate industrial control systems, specifically those in Iranian nuclear facilities. It spread via infected USB drives and targeted PLCs (programmable logic controllers), causing physical damage by altering the speed of centrifuges in nuclear plants. This attack, believed to be state- sponsored by the US and Israel, significantly set back Iran's nuclear program and marked the first publicly known instance of malware engineered to harm physical infrastructure. Defense strategies include isolating critical systems from the internet through air-gapping and raising cybersecurity awareness in industrial environments.

Video Explanation: Stuxnet: The Cyber Weapon That Destroyed Iran's Nuclear Program (youtube.com)

1.7.5 Emotet (2014)

Emotet, first identified in 2014 as a banking Trojan, initially targeted financial institutions but later evolved into a botnet that infected a wide range of industries. It spread primarily through phishing emails containing malicious attachments or links, harvesting banking credentials and facilitating the distribution of other malware, such as ransomware. Emotet's widespread impact included affecting businesses and organizations globally, with its resilience making it difficult to detect and remove. Over time, it became a robust botnet infrastructure responsible for significant financial losses. Effective defenses include strong email filtering, robust anti-malware solutions, and thorough employee education on phishing threats.

Video Explanation: What is Emotet? (youtube.com)

1.7.6 Zeus (2007)

Zeus, also known as Zbot, is a notorious Trojan horse first identified in 2007, primarily designed to target Windows systems for the purpose of stealing banking information. Zeus operated by infiltrating a victim's computer and silently monitoring their activities, particularly focusing on capturing sensitive financial data through keylogging and form-grabbing techniques. Keylogging allowed Zeus to record every keystroke made by the user, enabling it to capture login credentials, passwords, and other personal information as they were typed. Form-grabbing, on the other hand, intercepted data submitted through web forms, such as those used on banking websites, before it could be encrypted and securely transmitted. Zeus was highly effective due to its stealthy nature, often spreading through phishing emails, malicious downloads, and compromised websites. It infected millions of computers worldwide, creating a vast botnet used

to siphon off billions of dollars from bank accounts. The malware was so successful that it gave rise to various offshoots and derivatives, making Zeus one of the most impactful pieces of malware in cybercrime history. Despite numerous takedowns and arrests related to Zeus, its legacy continues to influence modern banking Trojans.

Video Explanation: The First Virus to Infiltrate Banks and Steal Millions, Zeus.exe (youtube.com)

1.7.7 AgentTesla (2014)

AgentTesla, first discovered in 2014, is a powerful and widely used spyware and keylogger designed to steal sensitive information from compromised systems. This malware primarily targets Windows environments and is often distributed through phishing campaigns, which entice users to download malicious attachments or click on links that execute the malware. Once installed, AgentTesla operates stealthily in the background, capturing keystrokes, taking screenshots, and logging clipboard data. It is particularly effective at stealing login credentials, including those for email accounts, VPNs, and other critical applications. AgentTesla is also capable of extracting information from web browsers, email clients, and FTP servers. The malware communicates the stolen data back to the attacker via HTTP POST requests or through SMTP, making it difficult to detect. Its modular design allows it to be customized for specific attacks, and it has been sold as a "malware-as-a-service" on underground forums, making it accessible to a wide range of cybercriminals. AgentTesla has been involved in numerous high-profile attacks, often targeting businesses to gain access to corporate networks and steal valuable data, leading to significant financial and reputational damage for the victims. Despite efforts to curb its spread, AgentTesla remains a persistent threat due to its adaptability and the ongoing demand for its capabilities among cybercriminals.

Video Explanation: Agent Tesla (youtube.com)

1.7.8 NotPetya (2017)

NotPetya, discovered in June 2017, is a form of wiper malware that initially masqueraded as ransomware, targeting Windows systems and causing widespread devastation across the globe. Originating in Ukraine, NotPetya exploited a vulnerability in Microsoft Windows known as EternalBlue, which allowed it to propagate rapidly within networks. Unlike traditional ransomware, NotPetya had a destructive purpose; it aimed to wipe out data rather than encrypt it for ransom, rendering infected systems inoperable. Once executed, NotPetya encrypted files on the infected machine, but it also corruptly overwrote the Master File Table (MFT), making data recovery nearly impossible without extensive backups.

The malware spread quickly through various vectors, including compromised software updates and the use of legitimate administrative tools like PsExec. NotPetya's impact was particularly severe, affecting major corporations, government entities, and critical infrastructure worldwide, leading to billions of dollars in damages.

Notable victims included Maersk, Merck, and FedEx, all of which faced significant operational disruptions due to the malware's rampage. The sophistication of NotPetya highlighted the growing trend of cyber warfare, with many cybersecurity experts attributing its development to state-sponsored actors, particularly in the context of geopolitical tensions. The aftermath of NotPetya underscored the importance of robust cybersecurity measures, regular backups, and the need for organizations to prepare for advanced persistent threats that could lead to catastrophic data loss.

Video Explanation : NotPetya Attack (youtube.com)

Note

The Master File Table (MFT) is a critical component of the NTFS (New Technology File System) used by Windows operating systems. It acts as a database that contains information about every file and directory on the NTFS volume, including file names, sizes, locations, attributes, and data runs. Each file and directory have a corresponding entry in the MFT, allowing the operating system to quickly access and manage files

Chapter 2: SIEM Configuration and Monitoring

2.1 What's SIEM?

SIEM (Security Information and Event Management) is a comprehensive security solution that combines two essential aspects of cybersecurity: Security Information Management (SIM) and Security Event Management (SEM). The primary function of a SIEM is to provide organizations with real-time visibility into their security environment by collecting, analyzing, and correlating security data from multiple sources.

Key Components of SIEM: -

2.1.1 Log Collection:

SIEM systems gather log data from various sources like servers, applications, network devices, firewalls, intrusion detection systems (IDS), and more. These logs contain records of events, errors, and user activity.

2.1.2 Event Correlation:

The SIEM platform correlates the data by identifying patterns or relationships between different events and logs. This helps in detecting security threats that may not be evident in individual logs but become clear when data from various sources is combined.

2.1.3 Real-Time Monitoring:

SIEM continuously monitors network activity, logs, and security events in real time. It identifies abnormal behavior and potential security threats like unauthorized access, malware activity, or policy violations.

2.1.4 Incident Detection and Response:

Based on the analysis of logs and event data, SIEM detects security incidents and can automatically trigger alerts to security teams. In some cases, SIEM can automate the response by taking predefined actions (e.g., blocking an IP address or isolating a compromised system).

2.1.5 Alerting and Reporting:

SIEM systems generate alerts based on predefined rules and thresholds, enabling security teams to respond to potential threats quickly. The system also produces detailed reports for compliance, audits, and forensics.

2.1.6 Threat Intelligence Integration:

Many SIEMs integrate threat intelligence feeds to stay updated with the latest cyber threats, vulnerabilities, and attack techniques. This allows the system to detect emerging threats more efficiently.

2.1.7 Compliance and Audit Support:

SIEM helps organizations comply with various regulatory requirements, such as GDPR, HIPAA, and PCI-DSS, by logging and reporting security events, ensuring that they meet audit requirements and maintain security best practices.

2.2 Benefits Of SIEM

- Centralized Security Management: SIEM systems provide a single platform for monitoring and managing security data across the entire IT infrastructure.

- Improved Incident Response: SIEM solutions enable faster detection of security threats and reduce the response time by automating alerting and response mechanisms.

- Enhanced Threat Detection: By correlating data from multiple sources, SIEM systems can identify advanced and complex attacks that may go unnoticed otherwise.

- Compliance: SIEM helps organizations meet regulatory compliance by maintaining security logs and generating reports for auditors.

2.2.1 Use Case Example:

Imagine a company that has hundreds of servers, firewalls, and endpoints generating logs. A SIEM system collects logs from all these devices and correlates them to detect suspicious activities, such as multiple failed logins attempt across different systems, which could indicate a brute-force attack. It then generates an alert for the security team to investigate, preventing potential breaches.

2.2.2 Conclusion:

SIEM is a vital tool for modern cybersecurity management, offering visibility, real-time detection, incident response, and compliance support. By centralizing security data and applying intelligent analysis, it helps organizations detect and respond to threats before they cause significant damage.

2.3 How The SIEM Works?

2.3.1 Data Collection:

- SIEM gathers logs and event data from various sources, such as:

o Network devices (routers, switches).

o Servers (Windows, Linux, etc.).

o Security tools (firewalls, IDS/IPS, antivirus).

o Applications (web applications, databases).

o Cloud environments.

- This data includes login attempts, traffic patterns, system errors, access records, and more.
- Data is collected via log agents installed on these devices or through syslog protocols.

2.3.2 Data Normalization:

- Collected data comes in different formats, so the SIEM system normalizes it to ensure consistency.
- Normalization involves converting different log formats into a standard format that can be analyzed. For example, a login attempt on a Windows server might look different from one on a Linux server, but normalization makes them comparable.

2.3.3 Data Aggregation:

- The SIEM system aggregates similar logs or events into a unified view.
- Instead of processing individual log events separately, the system combines related events (e.g., repeated failed login attempts) to make analysis more efficient and reduce unnecessary data.

2.3.4 Data Correlation:

- The correlation engine is the core component of SIEM. It examines multiple events across different systems and correlates them to detect security incidents. For example, a single failed login attempt might not be a problem, but multiple failed login attempts followed by a successful login could indicate a brute-force attack.

- Correlation rules are defined based on security policies, threat intelligence, or known attack patterns (like recognizing attempts to exploit vulnerabilities).

2.3.5 Threat Detection:

- SIEM uses correlation rules and threat intelligence feeds to detect potential security threats in real-time.

- If the system detects an anomaly, such as abnormal login times, unusual network traffic, or multiple authentication failures, it flags this as a potential security incident.

2.3.6 Alerting

- When a security incident or anomaly is detected, the SIEM system triggers an alert.
- Alerts are sent to security teams through emails, dashboards, or even integrated ticketing systems.
- SIEM can categorize alerts based on severity (e.g., low, medium, high), allowing teams to prioritize which incidents to respond to first.

2.3.7 Incident Response:

- Many SIEM systems have automated response capabilities. Based on predefined rules, the system can take actions such as:
  - Blocking IP addresses.
  - Isolating infected machines from the network.
  - Stopping specific processes or services.

- Incident response teams can also manually investigate the alerts to determine if further action is needed.

2.3.8 Log Storage and Retention:

- SIEM systems store logs for future analysis, audits, and compliance reporting.
- Logs are stored for a defined period (e.g., months or years), based on organizational policies or compliance requirements (such as PCI-DSS, HIPAA).
- Having historical data helps with investigations and forensics to understand how a breach or attack occurred.

2.3.9 Reporting and Dashboards:

- SIEM systems provide customizable reports and dashboards for security monitoring.
- These reports are used for:
  - Compliance auditing (meeting regulatory requirements).
  - Detecting trends over time.
  - Security posture reviews.

2.3.10 Threat Intelligence Integration:

- SIEM systems often integrate with threat intelligence feeds to stay updated on the latest vulnerabilities, malware, and attack vectors.
- These feeds help detect new types of threats, allowing the SIEM to identify and respond to emerging risks in real-time.

-

2.3.11 Examples: How SIEM Works in a Cybersecurity Incident:

1. Data Collection: Logs from a firewall show that there's been an unusually high number of inbound traffic requests to a web server.

2. Data Normalization and Correlation: The SIEM system correlates these firewall logs with logs from the web server, which show multiple failed login attempts followed by a successful one from the same IP address.

3. Alerting: The SIEM generates an alert, flagging the sequence of events as a potential brute-force attack followed by unauthorized access.

4. Incident Response: Based on predefined rules, the SIEM system could automatically block the suspicious IP address or alert the incident response team for immediate investigation.

5. Reporting: Security analysts can review the logs and correlation data via the SIEM dashboard, generate a report for further analysis, and take appropriate actions to prevent further damage.

2.4 Open Source SIEMs:

There are several open-source SIEM (Security Information and Event Management) tools available that offer robust features for log management, threat detection, and security analytics. Here are some popular ones:

2.4.1 ELK Stack (Elastic Stack)

- Components: Elasticsearch, Logstash, and Kibana.

- Features:
  - Real-time log collection and analysis.
  - Customizable dashboards and visualization.
  - Supports integrations with Beats and other log shippers.

- Use Case:
Widely used for log management and as a SIEM tool by configuring alerting (e.g., through Elastic's free tier or integrations).

- Link: Elastic Stack: (ELK) Elasticsearch, Kibana & Logstash | Elastic

2.4.2 Wazuh (we already use)

- Features:
  - Intrusion detection, log analysis, and incident response.
  - File integrity monitoring (FIM) and vulnerability detection.
  - Real-time alerting and compliance monitoring.

- Use Case:
Open-source platform built on top of Elastic Stack; acts as a SIEM solution.

- Link: Wazuh - Open Source XDR. Open Source SIEM.

2.4.3 Graylog

- Features:
  - o Centralized log collection and management.
  - o Customizable alerts and dashboards.
  - o Integrates with Elasticsearch for data storage.

- Use Case: Known for its performance in managing large-scale log data.

- Link: SIEM, Log Management & API Protection (graylog.org)

2.4.4 Security Onion

- Features:
  - o Intrusion detection, network monitoring, and log management.
  - o Uses a combination of tools like Zeek (formerly Bro), Suricata, and Elasticsearch.
  - o Incident response and threat hunting capabilities.

- Use Case: Designed for network security monitoring and intrusion detection but can also serve as a SIEM.

- Link: Security Onion Solutions

2.4.5 The Hive

- Features:
  - o Incident response platform with case management and collaboration.
  - o Can integrate with Cortex for threat intelligence analysis and automated response.
  - o Threat indicators and IoC (Indicators of Compromise) management.

- Use Case: Incident response platform that works well with other SIEM tools.

- Link: StrangeBee

2.4.6 Apache Metron

- Features:
  - o Real-time security data analytics.
  - o Pluggable for custom threat intelligence feeds.
  - o Built on top of Apache Hadoop and supports large-scale processing.

- Use Case: Suitable for organizations that need big data security analytics.

- Link: Apache Metron Big Data Security

2.5.7 SIEMonster

- Features:
  - o End-to-end open-source SIEM solution built on a range of open-source tools. to Scalable, from small business to large enterprises.
  - o Incorporates Elastic Stack, Wazuh, and others for monitoring and detection.

- Use Case: Provides a complete SIEM solution with multi-tenant and dashboard features.

- Link: Home - SIEMonster

2.5.8 AlienVault OSSIM

- Features:

o Combines SIEM and threat detection with vulnerability assessment.

o Log management, asset discovery, and network monitoring. o Real-time event correlation and security monitoring.

- Use Case: A solid open-source platform for small to mid-sized organizations.

- Link: Enhance Security with OSSIM | LevelBlue (att.com)

2.5.9 Prelude SIEM

- Features:

o Open-source hybrid SIEM.

o Real-time data collection, correlation, and normalization.

o Focuses on interoperability with other security tools.

- Use Case: Designed to be an extensible and interoperable SIEM solution.

- Link: Overview - PRELUDE SIEM (prelude-ids.org)

2.5.10 Snort (with additional tools)

- Features:
  - o Primarily a network intrusion detection system (NIDS).
  - o Can be integrated with other tools like ELK Stack or Wazuh for log management and threat detection.

- Use Case: Use alongside SIEM platforms for network-based threat detection.

- Link: [Snort - Network Intrusion Detection & Prevention System]

These tools provide a wide range of options for different needs, from large-scale log management and monitoring to complete SIEM solutions with integrated threat detection and response capabilities. Depending on your organization's size, infrastructure, and specific needs, you can choose the one that best fits your environment and different open-source SIEMs shows in figure 2-1.

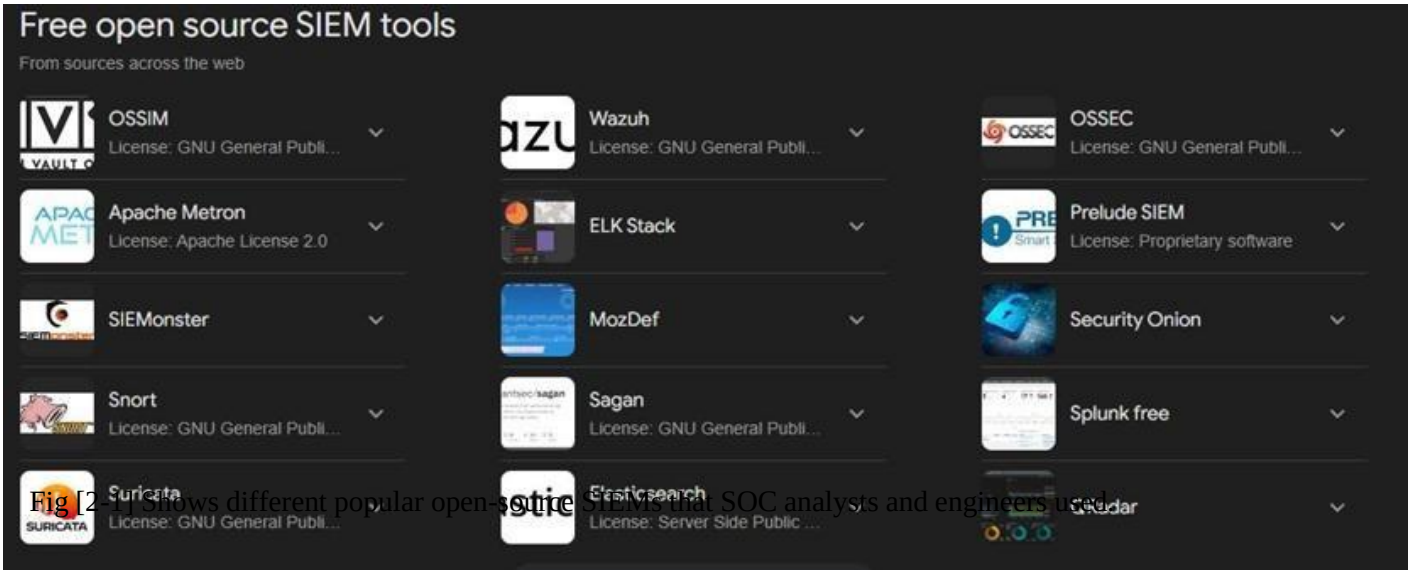


Fig [2-1] Shows different popular open-source SIEMs that SOC analysts and engineers use.

2.5 How Open-Source SIEM Tools Work?

- Log Collection: Most tools (e.g., ELK, Graylog, Wazuh) collect logs from various sources such as system logs, network devices, and applications.

- Log Analysis: Logs are processed, normalized, and correlated to detect anomalies or suspicious patterns.

- Threat Detection: Signature-based tools like Snort or behavior-based tools like Zeek help identify potential threats in the data.

- Visualization: Dashboards and visualizations (e.g., Kibana, Metron Dashboard) provide real-time monitoring and analytics.

- Alerting: Customizable alerts notify you of security incidents or anomalies in real- time.

- Incident Response: Some tools like TheHive focus on managing security incidents, while others (e.g., Wazuh, Security Onion) integrate incident response capabilities into their core functions.

These SIEM tools provide a combination of log management, threat detection, and incident response through efficient collection, analysis, and real-time monitoring of security data.

2.6 What's Wazuh?

Wazuh is an open-source security monitoring and data analysis system, designed to collect and analyze data from various systems to provide a comprehensive view of security threats and compliance. It is considered the successor to OSSEC, having been developed and enhanced to include advanced capabilities in log analysis, threat detection, and incident response

.

Wazuh Features:

2.6.4 Log Management

- Collects logs from multiple devices such as servers, applications, and network devices.
- Allows for the analysis of these logs to detect unusual activities or potential threats.

2.6.5 Threat Detection

- Relies on predefined security rules to detect intrusion attempts, vulnerabilities, or malicious activities.
- Provides capabilities to detect malware, breaches, and unauthorized access attempts.

2.6.6 Compliance Monitoring

- Assists in compliance with security and privacy standards such as GDPR, HIPAA, and PCI DSS, by monitoring systems and generating the required reports.

2.6.7 SIEM Integration

- Integrates with SIEM tools like Elastic Stack (ELK) and Splunk to provide deeper log and threat analysis.

2.6.8 Multi-host Monitoring

- Can monitor hundreds or thousands of devices and systems from a central location.
- Operates on a client/server model, where agents collect data and send it to the central server for analysis.

2.6.9 Alerting

- Can send real-time alerts when suspicious activities or threats are detected.
- Supports integration with other alerting systems such as Slack, Email, and Webhook.

2.7 How Wazuh Works?

2.7.4 Agents

- Agents are installed on various systems (servers, desktops, applications) to collect logs and data.

2.7.5 Server

- The central server collects data from agents, analyzes it using predefined security rules, and generates reports and alerts based on the findings.

2.7.6 Dashboard

- Uses a Kibana-based interface (visual tool) to display information and analysis in an easy-to-understand and visual format.

Wazuh Use Cases:

- Network Security Monitoring: Monitoring networks and detecting security threats.
- Incident Response: Providing alerts and quick responses to security incidents.
- Log Analysis: Comprehensive log analysis for security improvement and compliance reporting.

Wazuh is an effective and powerful tool for monitoring systems and security in organizations, both large and small, providing a comprehensive and analytical view of security events and helping to enhance overall security, and Figure 2-2 Show Wazuh's Dashboard

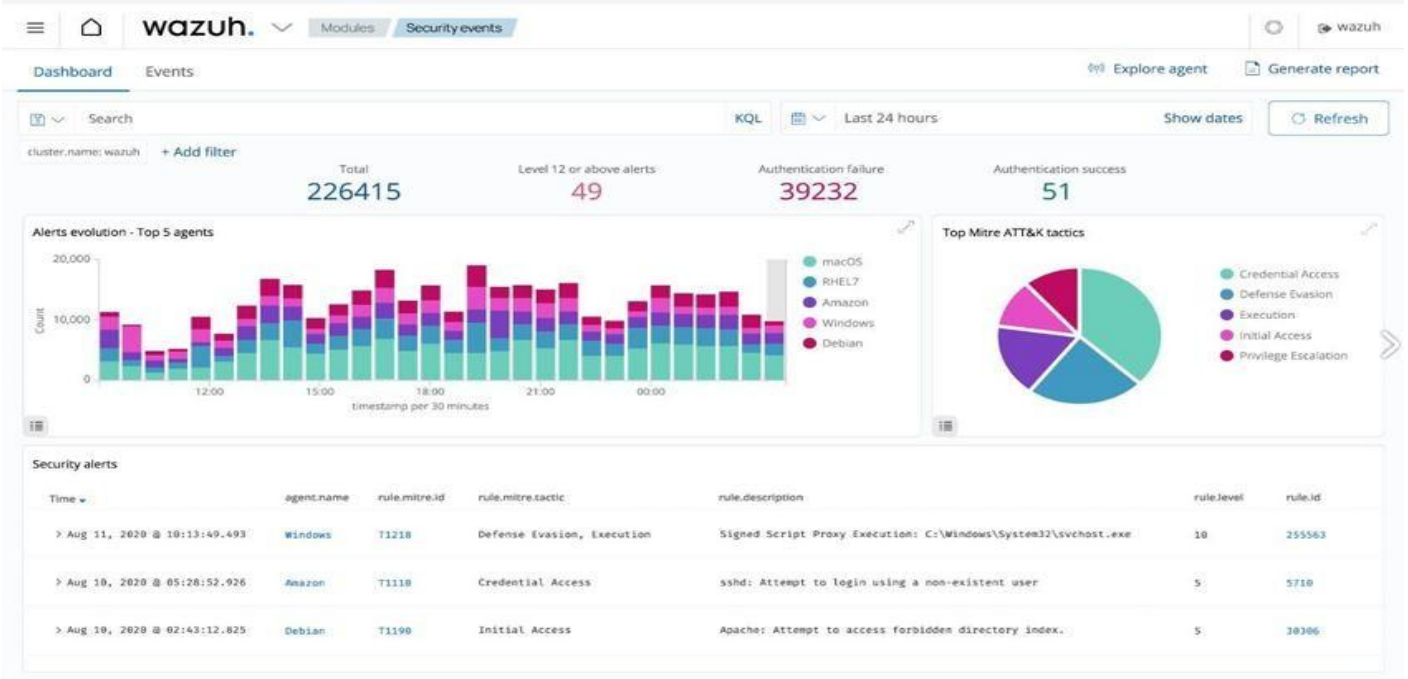


Fig [2-2] Shows Wazuh's Dashboard, Describes the alerts specially monitored security alerts.

2.8 How to Build Wazuh?

Building a Wazuh SIEM involves setting up the Wazuh server, agents, and an interface for data visualization. Here's a step-by-step guide on how to build Wazuh SIEM:

2.8.4 Prerequisites:

1- A Linux server (Ubuntu, Debian, CentOS, etc.).

2- Minimum hardware requirements:
o 2 CPU cores.
o 4 GB of RAM.
o 20 GB of storage (may vary depending on log volume).

o 3- Root or sudo access.

2.8.5 Install Wazuh Server:

The Wazuh server is responsible for collecting data from agents, analyzing it, and generating alerts. You can install it on a Linux server. Follow these steps:

Download the Wazuh OVA File

1- Visit the Wazuh website and download the latest Wazuh OVA file. The OVA contains preconfigured components like Wazuh Manager, Wazuh Indexer (formerly Elasticsearch), and the Wazuh Dashboard (formerly Kibana).

2- Install VMware Workstation or VMware Player If you don’t have VMware Workstation or VMware Player installed, download and install the appropriate version for your OS:

- VMware Workstation Player (Free for personal use) or VMware Workstation Pro.

3. Import Wazuh OVA into VMware

- Open VMware Workstation or Player: o
  Open VMware, go to File > Open.

- Select the OVA file:
  o Browse and select the Wazuh OVA file you downloaded and click Open.

4. Review Virtual Machine Settings:
o Review the OVA's configuration (RAM, CPU, disk size, etc.). You can adjust the settings (e.g., increasing RAM or CPUs) based on your system's resources and requirements.

5. Import the Virtual Machine:
o Click Import to begin importing the OVA. This process may take a few minutes depending on your system.

6. Start the Wazuh Virtual Machine
- Once the import is complete, you will see the Wazuh VM in the list.
- Select the Wazuh VM and click Start (or play virtual machine).
- The VM will boot, and Wazuh services (Wazuh Manager, Wazuh Indexer, and Wazuh Dashboard) will automatically start. It may take a couple of minutes for all services to be fully operational.

Three Main Commands to Run: -

```
[wazuh-user@wazuh-server ~]$
[wazuh-user@wazuh-server ~]$ sudo systemctl start wazuh-manager
[wazuh-user@wazuh-server ~]$ sudo systemctl start wazuh-dashboard
[wazuh-user@wazuh-server ~]$ sudo systemctl start wazuh-indexer
[wazuh-user@wazuh-server ~]$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP gr
oup default qlen 1000
    link/ether 00:0c:29:e8:b1:da brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.231/24 brd 192.168.1.255 scope global dynamic eth0
       valid_lft 85956sec preferred_lft 85956sec
    inet6 fe80::20c:29ff:fee8:b1da/64 scope link
       valid_lft forever preferred_lft forever
[wazuh-user@wazuh-server ~]$
[wazuh-user@wazuh-server ~]$
[wazuh-user@wazuh-server ~]$ sudo systemctl start wazuh-manager
[wazuh-user@wazuh-server ~]$ sudo systemctl start wazuh-dashboard
[wazuh-user@wazuh-server ~]$ sudo systemctl start wazuh-indexer
[wazuh-user@wazuh-server ~]$
```

Fig [2-3] Shows Three Main Commands and Components of Wazuh dashboard.

- ***sudo systemctl start wazuh-manager***
- ***sudo systemctl start wazuh-dashboard***
- ***sudo systemctl start wazuh-indexer***

And these Commands Shows in Figure 2-3.

Once the Wazuh VM is running, it will display its IP address (usually via DHCP). To access the Wazuh Dashboard:

1. Open a web browser.
2. Navigate to:

Fig [2-4] Shows the Recommended IP and Port to connect your host to wazuh server

https://<WAZUH_VM_IP>:5601
Replace <WAZUH_VM_IP> with the actual IP of the Wazuh VM.

3. Bypass any SSL warnings (self-signed certificates).

4. Login using default credentials:
o Username: admin
o Password: admin

Fig [2-5] Shows Wazuh Cardinalates Login to appear dashboard

2.9 Key Features of Wazuh Dashboard:

2.9.4 Real-time Monitoring

- Provides real-time monitoring of your systems and security events. It allows you to view the status of agents, analyze logs, and detect anomalies as they occur.

2.9.5 Security Events Visualization

- The dashboard allows users to create customizable visualizations and dashboards to track events, visualize trends, and monitor various security metrics such as threats, vulnerabilities, and compliance status.

2.9.6 Alert Management

- Alerts generated by the Wazuh Manager are displayed in the dashboard, were users can sort, filter, and investigate them. You can drill down into specific alerts to analyze the details of a potential security incident.

2.9.7 Threat Detection and Responses

- The Wazuh Dashboard provides a central platform to respond to security incidents, such as monitoring active responses or configuring automated responses to specific threats.

2.9.8 Compliance Monitoring

- Wazuh helps organizations comply with various regulatory frameworks (such as PCI DSS, GDPR, and HIPAA). The dashboard offers predefined compliance reports and tools to help monitor compliance status across the infrastructure.

2.9.9 File Integration Monitoring (FIM)

- The dashboard displays file changes in real-time, helping monitor critical system files and directories for any unauthorized modifications, which may indicate a security breach.

2.9.10 Vulnerability Detection

- With integrated vulnerability detection features, the Wazuh Dashboard helps users identify vulnerabilities in software packages, operating systems, and network configurations across their infrastructure.

2.9.11 Active Threat Hunting

- It allows security teams to actively hunt for potential threats or indicators of compromise by querying data and logs from multiple endpoints.

2.9.12 Log Data and Analysis

- Through integration with Wazuh Indexer (formerly Elasticsearch), the dashboard provides advanced search functionality, allowing users to perform full text searches logs, filter data, and analyze specific events.

Connect your network devices (such as servers, endpoints, and firewalls) to the Wazuh SIEM (Security Information and Event Management) system, you need to configure the Wazuh agents on those devices, which will then report their log data and events to the Wazuh Manager, and Figure 2-6 shows key features of Wazuh.

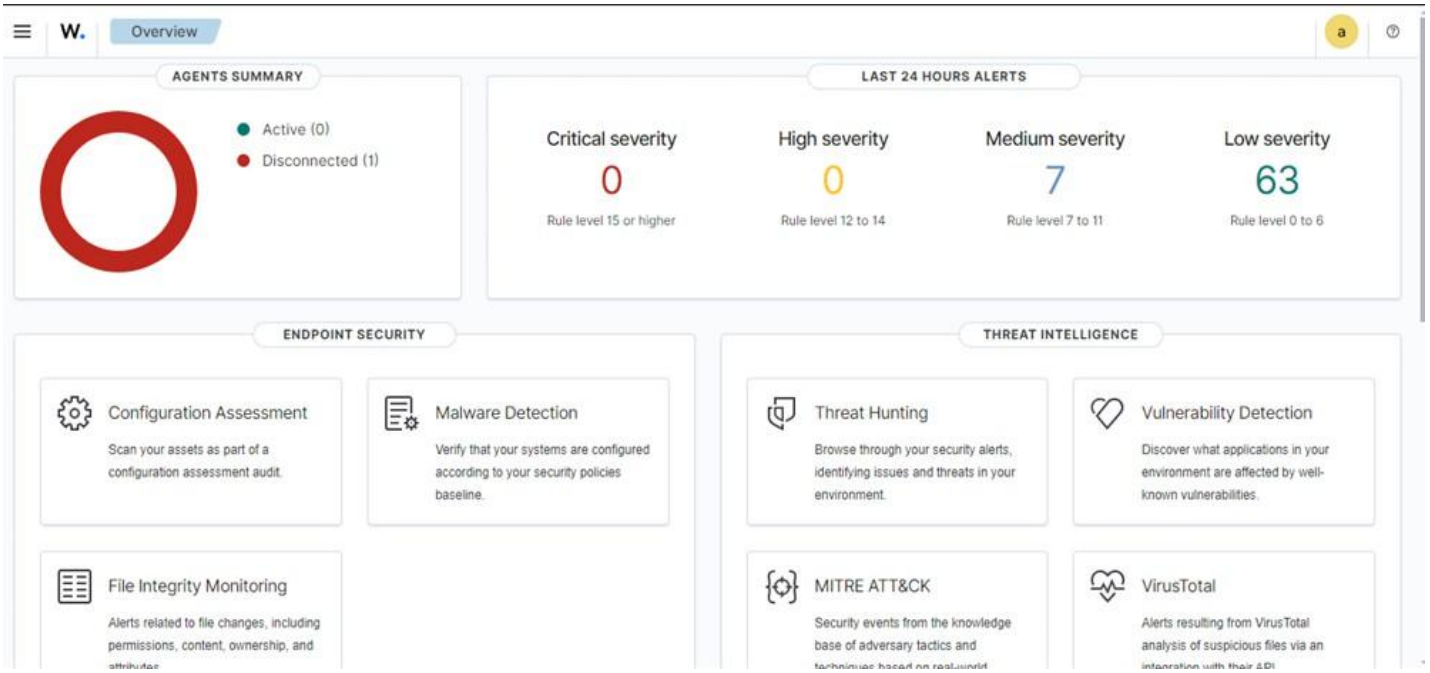


Fig [2-6] Shows Wazuh key Features and Functions.

2.10 How Wazuh-agent Works?

1- After installation, the Wazuh Agent continuously monitors the endpoint for various types of security-related activities, such as file changes, network activity, or log entries.

2- It sends the collected data to the Wazuh Manager using a secure communication protocol.

3- The Wazuh Manager processes and analyzes this data, generates alerts if necessary, and makes it available for visualization in the Wazuh Dashboard.

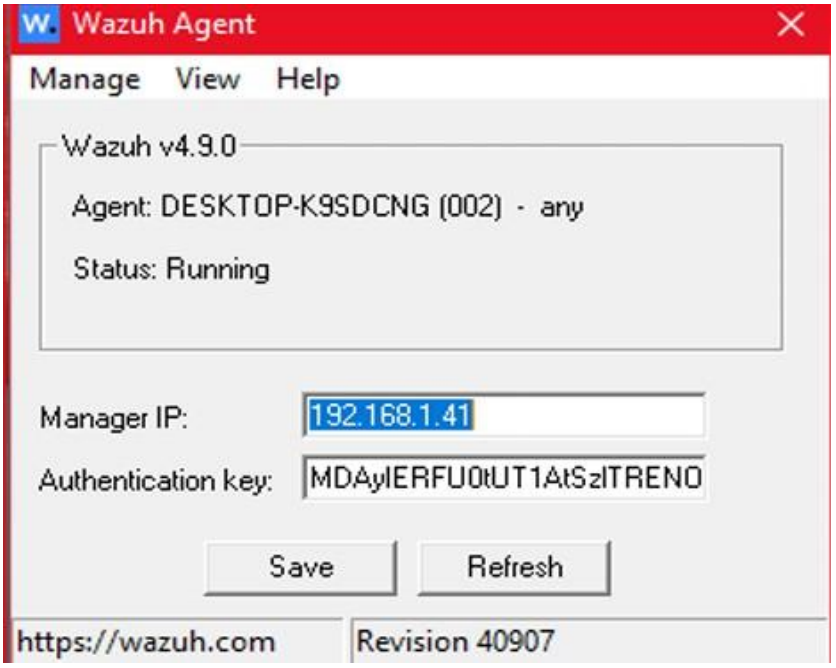


Fig [2-7] Shows Wazuh Agent Interface

Authentication key on the Wazuh Agent is a unique token used to securely register and authenticate the agent with the Wazuh Manager. This key is generated by the Wazuh Manager and is required to establish a trusted communication channel between the agent and the manager. The authentication key ensures that only authorized agents can connect to the Wazuh Manager and send logs or security data.

Key Points about the Authentication Key:

1. Purpose:

o The authentication key is used to securely register the Wazuh Agent with the Wazuh Manager.

o It helps prevent unauthorized agents from connecting to the Wazuh Manager.

2. One-Time Use:

o The key is a one-time-use token. After an agent has been registered using the authentication key, the key is no longer needed for subsequent communications.

3. Agent-Manager Communication:

o Once the agent is successfully registered, all communications between the agent and the Wazuh Manager occur over a secure channel (using TLS encryption), ensuring the confidentiality and integrity of the data transmitted.

4. Agent Registration Process:

o After installing the Wazuh Agent on a device, you need to register it with the Wazuh Manager by obtaining an authentication key from the manager.

o This key is generated using the manage_agents tool on the Wazuh Manager, and you copy it to the Wazuh Agent during the registration process.

To Get This Authentication Key Follow This Steps:

PuTTY is a free and open-source terminal emulator, network file transfer application, and SSH client. It is widely used for establishing secure, remote connections to servers and network devices, especially in environments running Linux, Unix, or other command line-based operating systems.

Key Features of PuTTY:

1. SSH Client:

o PuTTY primarily serves as an SSH (Secure Shell) client. SSH is a protocol used to securely connect to remote machines over an encrypted channel. This is commonly used by system administrators to manage remote servers and devices.

2. Telnet Client:

o In addition to SSH, PuTTY can also be used as a Telnet client, which is an older, less secure protocol for remote connections. SSH is preferred due to its encryption.

3. Serial Console:

o PuTTY can be used to connect to serial ports, making it useful for configuring network devices such as switches, routers, and firewalls that require console access.

4. Network File Transfers:

o PuTTY supports SCP (Secure Copy Protocol) and SFTP (SSH File Transfer Protocol) for transferring files securely between local and remote machines.

5. Customizable Interface:

o PuTTY provides options to customize the appearance of the terminal, such as changing font size, color schemes, and window behavior.

6. Port Forwarding:

o PuTTY supports SSH tunneling or port forwarding, allowing users to securely route traffic from one network port to another over an SSH connection.

7. Cross-Platform:

o While originally designed for Windows, PuTTY is also available on Linux and macOS. However, many Linux users prefer native tools like ssh for terminal access, but PuTTY is still popular for Windows users.

Common Use Cases:

1. Remote Server Administration:

o PuTTY is widely used by system administrators to connect to and manage remote Linux or Unix servers from their local machines. SSH provides a secure connection, allowing administrators to execute commands and perform configurations.

2. Network Device Configuration:

o PuTTY is used to connect to network devices like routers, switches, and firewalls via SSH or a serial interface for initial setup and configuration.

3. File Transfers:

PuTTY's support for SCP and SFTP allows secure file transfers between machines,

commonly used in remote file management or system backups.

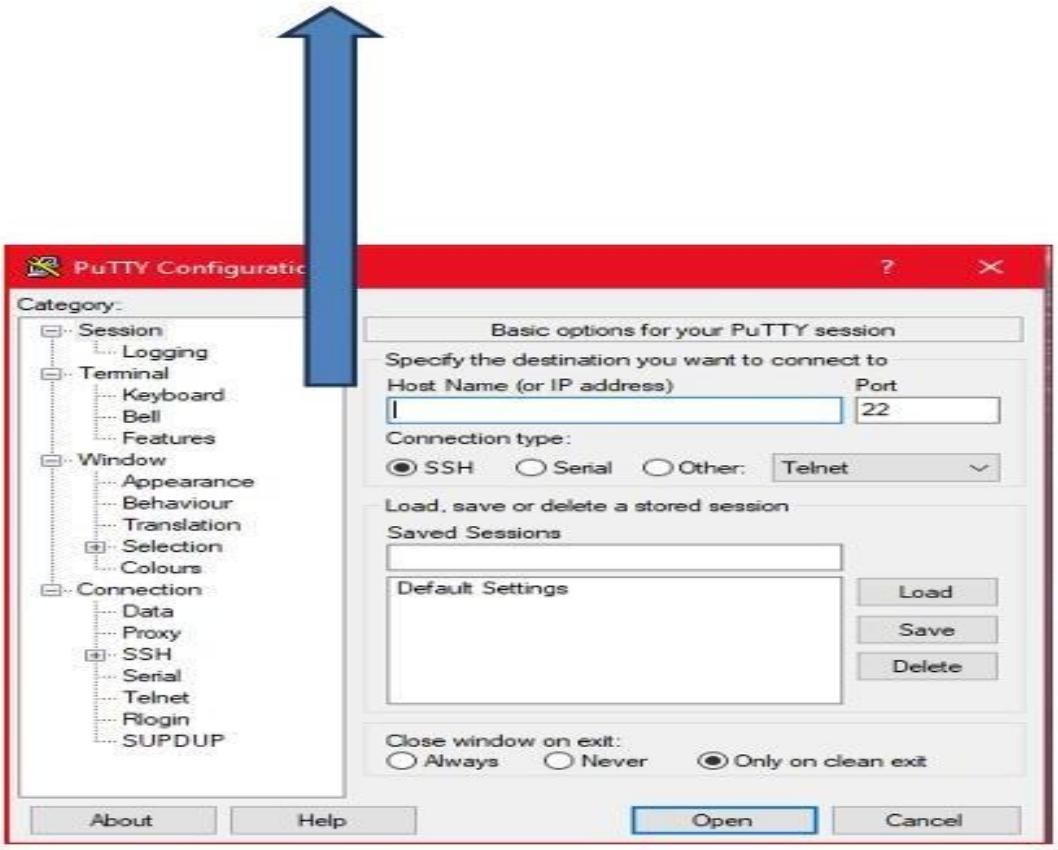
4. Secure Tunneling:

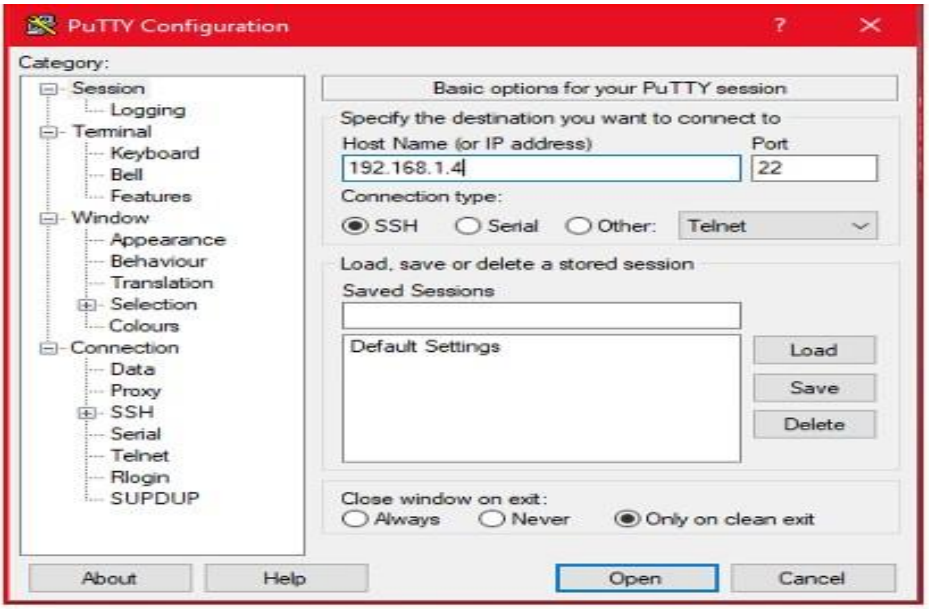
o PuTTY's port forwarding feature is often used to create secure tunnels over SSH, enabling secure connections to services that are otherwise not encrypted (e.g., accessing a database through an SSH tunnel).

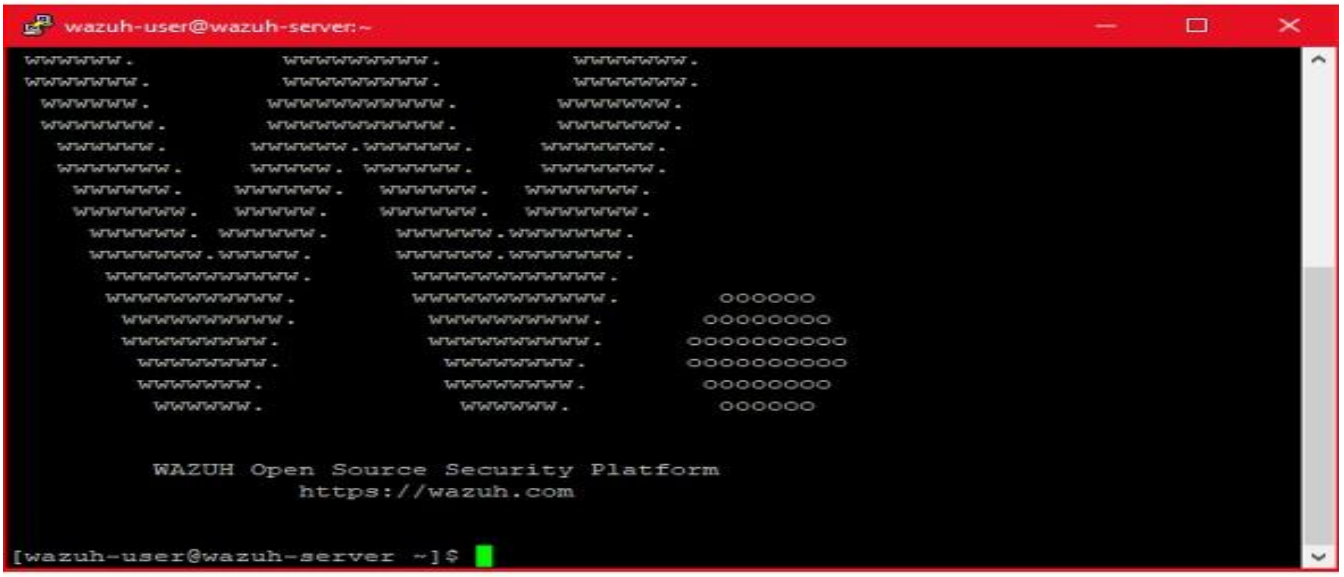
2.11 What is putty? How to use it?

1- Open Putty and Choose SSH as Connection type.

You Need To Fill The ip Of Wazuh (OVA Server)







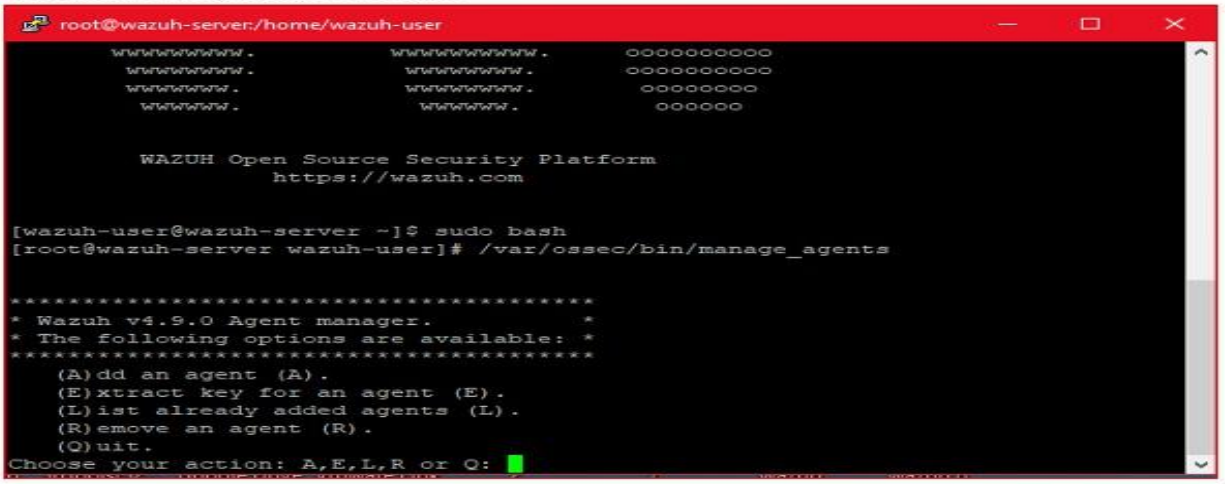
Username : wazuh-user

Password: wazuh

2- Write Server IP and connect to Wazuh-server.

We User sudo bash to get into root

/VAR/OSSEC/BIN/MANAGE_AGENTS



To Get Connet With Wazuh Agent / We Choose A And named it with your name prefer and with the ip of your ipv4 address

---

3- Write “sudo bash” to start writing up commands in server to add your agent.

4- Get Authentication Key to your added Agent.

- **To get the ip of id:001 we must go first to command prompt to get our ip :**

2.12 Monitoring and Alerting Setup Report

Introduction

- Overview: This section provides a brief introduction to the purpose of the monitoring and alerting setup. It highlights why monitoring is essential for the system's performance, security, and uptime.

- Scope: Define what areas or components the monitoring will cover (e.g., servers, network, applications, security, databases).

Objectives:

- o Ensure system performance and availability.
- o Detect anomalies, incidents, or failures.
- o Enable real-time or near-real-time alerting for critical events.
- o Maintain logs for future analysis and auditing.

System Environment Overview

Infrastructure: Describe the infrastructure being monitored (e.g., cloud, on-premises, hybrid) and the primary technologies used.

- o Example: Linux-based servers, cloud VMs, containerized applications, etc.

Log Collection:

o Centralized log collection through Wazuh.
o Log sources: system logs, security logs, application logs.

Alerting Setup

Alert Types:
o Critical: System or application downtime, security breaches, disk full.
o Warning: CPU usage above 90%, application response time degraded.
o Info: Daily status checks, backup completed successfully.

Thresholds:
o Define thresholds for each metric that trigger alerts.
o Example: CPU > 90% for 5 minutes triggers a warning alert.

Alert Channels:
o Email: For critical alerts to system administrators.
o SMS: For urgent security or system downtime alerts.
o Slack/Teams: For team collaboration and incident response.

Escalation Policies:

o Define the workflow for escalating critical issues (e.g., from Level 1 support to Level 2 and 3).
o Example: If the issue is not resolved in 15 minutes, notify the senior team.

Security Monitoring and Compliance

Intrusion Detection:
o Using Wazuh to monitor for unauthorized access attempts and suspicious behavior.

Compliance Monitoring:
o Log and audit trails for compliance with regulations (e.g., GDPR, PCI DSS).

File Integrity Monitoring:
o Detect and alert changes to critical system files.

Testing and Validation

Regular Testing: How often tests are performed on the monitoring and alerting system to ensure they work as expected (e.g., monthly).

Simulated Incidents: Conduct periodic simulated incidents to verify alerting and response workflows.

Threshold Fine-Tuning: Adjust thresholds based on system behavior over time. (Network Baseline)

Future Improvements

Automation: Plans for automating incident response through scripts or machine learning.
Scalability: Plans for scaling the monitoring system as infrastructure grows.
Advanced Analytics: Use of predictive analytics or machine learning to detect anomalies early.

2.13 Windows defender (Anti-Virus) Configuration

Configuration:

To collect Windows Defender logs, you need to set up the Wazuh agent either through centralized configuration or locally by editing the **C:\Program Files (x86)\ossec-agent\ossec.conf** file. Centralized configuration enables you to share instructions with multiple agents.

To enable the collection of Windows Defender logs, add the following block to the configuration file:

```
<localfile>
  <location>Microsoft-Windows-Windows  Defender/Operational</location>
  <log_format>eventchannel</log_format>
</localfile>
```

Decoders and Rules

Wazuh provides pre-built decoders for Microsoft Windows logs, including those from Windows Defender, so there's no need to create additional decoders for these logs. Additionally, rules for Windows Defender are included and can be found at /var/ossec/ruleset/rules/0600-win-wdefender_rules.xml on the Wazuh server and this appears in figure

2.14 Wazuh's Alerting Configuration & Integration With Virus Total

To configure Wazuh to monitor near real-time changes in the ***C:\Users\<USER_NAME>\Downloads*** directory to check any file installed in downloads, follow these steps. This process includes installing necessary packages and creating an active response script to remove malicious files:

1. Locate the <syscheck> block in the Wazuh agent configuration file, typically found at ***C:\Program Files (x86) \ossec-agent\ossec.conf.***

2. Ensure that the ***<disabled> tag within the <syscheck> block is set to no***. This step activates the Wazuh File Integrity Monitoring (FIM) module, allowing it to track directory changes.

3. Add an entry inside the <syscheck> block to specify the directory you want to monitor in near real-time. For this use case, configure Wazuh to monitor the ***C:\Users\<USER_NAME>\Downloads*** directory by replacing ***<USER_NAME>*** with the appropriate username

> ***Note***
>
> Username of your windows client you can get it from "whoami" command and take the second partition after slash symbol for example: -
>
> ```
> C:\Users\dell>whoami
> desktop-1gfugr7\dell
> ```
>
> The username of this client is: dell, so directory of downloads will be in this path C:\Users\dell\Downloads

4. Download the Python executable installer from the official Python website here , After the download is complete, run the Python installer. Be sure to check

the following options:

5.

- **Add Python 3.X to PATH** (This adds the interpreter to the system's execution path)

```
> pip install pyinstaller
> pyinstaller --version
```

Once the installation is finished, open PowerShell as an administrator and use pip to install PyInstaller.

6. PyInstaller is used here to convert the remove-threat.py active response Python script into a standalone executable application that can be run on a Windows endpoint.

7. To convert the remove-threat.py script into a Windows executable, open PowerShell as an administrator and run the following command:

```
> pyinstaller -F \path_to_remove-threat.py
```

8. Transfer the remove-threat.exe executable file to the C:\Program Files (x86)\ossec-agent\active-response\bin directory.

9. To apply the changes, restart the Wazuh agent by running the following in admin PowerShell

```
> Restart-Service -Name wazuh
```

9. After finishing your configuration in Wazuh-agent , now go to Wazuh-server after connecting on it from Putty.

Follow these steps on the Wazuh server to set up ***VirusTotal*** integration. This process will also enable and execute the active response script whenever a suspicious file is detected:

10. Add the following configuration to the ***/var/ossec/etc/ossec.conf*** file on the Wazuh server to enable VirusTotal integration. Replace ***<YOUR_VIRUS_TOTAL_API_KEY> with your VirusTotal API key***. This configuration will trigger a VirusTotal query whenever any of the rules in the FIM syscheck group are activated:

```
<ossec_config>
  <integration>
    <name>virustotal</name>
    <api_key><YOUR_VIRUS_TOTAL_API_KEY></api_key> <!-- Replace with your VirusTotal API key -->
    <group>syscheck</group>
    <alert_format>json</alert_format>
  </integration>
</ossec_config>
```

Note

You Should create an account in VirusTotal and after that click in your username in navbar and navigate to API key and get your API Key from here , and note that ***The free VirusTotal API rate limits requests to four per minute***. If you have a ***premium VirusTotal API key***, with a high frequency of queries allowed, you can ***add more rules besides these two. You can configure Wazuh to monitor more directories besides C:\Users\<USER_NAME>\Downloads***.

Append the following blocks to the Wazuh server ***/var/ossec/etc/ossec.conf*** file. This enables active response and triggers the remove-threat.sh script when VirusTotal flags a file as malicious:

```
<ossec_config>
  <command>
    <name>remove-threat</name>
    <executable>remove-threat.sh</executable>
    <timeout_allowed>no</timeout_allowed>
  </command>

  <active-response>
    <disabled>no</disabled>
    <command>remove-threat</command>
    <location>local</location>
    <rules_id>87105</rules_id>
  </active-response>
</ossec_config>
```

Note

Local Indicates that the file location of remove-threat and the malicious files and action will be loaded locally in the device which configured to be Wazuh-agent, and rule id: 87105 used for VirusTotal integration.

10. Add the following rules to the Wazuh server ***/var/ossec/etc/rules/local_rules.xml*** file to alert about the active response results.

```
<group name="virustotal,">
  <rule id="100092" level="12">
     <if_sid>657</if_sid>
     <match>Successfully removed threat</match>
     <description>$(parameters.program) removed threat located at $(parameters.alert.data.virustotal.source.file)</description>
  </rule>

  <rule id="100093" level="12">
    <if_sid>657</if_sid>
    <match>Error removing threat</match>
    <description>Error removing threat located at $(parameters.alert.data.virustotal.source.file)</description>
  </rule>
</group>
```

Level 12: OSSEC uses a **level system** to categorize the severity of alerts (from 0 to 15). A level of 12 indicates a relatively high-severity alert, suggesting that the rule relates to a significant event, such as the successful removal of a threat and level 15 used for data exfiltration cases.

if_sid (If Signature ID): This line specifies that this rule should only be evaluated if a previous alert with sid (signature ID) 657 has been triggered.

➔This means that **rule 100092** will only be considered if a previous event related to rule 657 has occurred. The 657 might represent a rule that detects a specific threat or triggers a VirusTotal check.

Match Condition: This specifies the condition that triggers the rule. The rule will be activated when the text **"Successfully removed threat"** appears in the alert message or error message appeared.

$(parameters.program): This placeholder likely refers to the program or script that executed the threat removal.

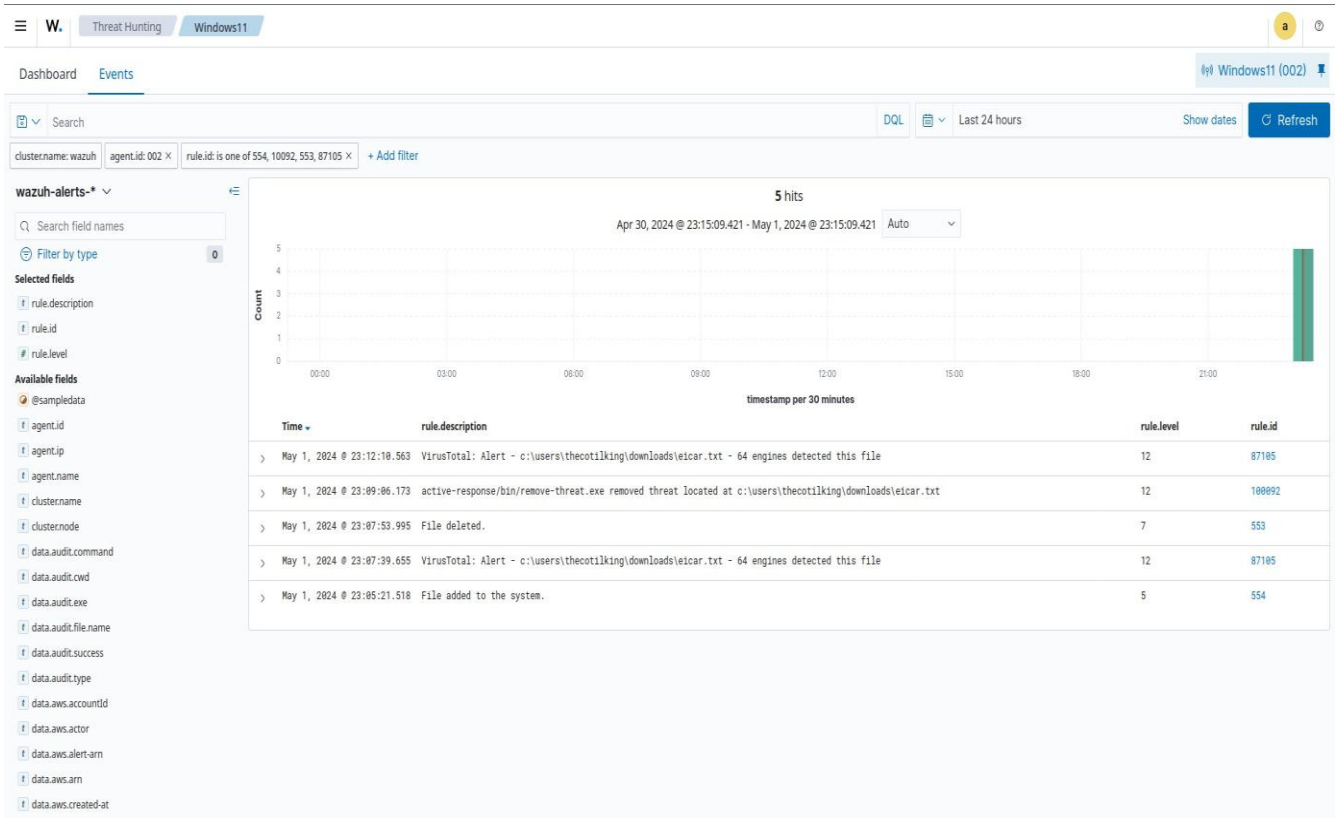
$(parameters.alert.data.virustotal.source.file): This placeholder refers to the location or name of the file that was flagged by VirusTotal and subsequently removed.
Purpose: The description will contain information about which program removed the threat and the specific file that was addressed. It makes the alert message more informative for the user or analyst monitoring OSSEC.

Rule 100092: Triggered when a threat is successfully removed after a VirusTotal scan.
Rule 100093: Triggered when an error occurs during the attempt to remove a threat flagged by VirusTotal.

11. Restart the Wazuh manager to apply the configuration changes.

12. Start your **Attack emulation** to test your configuration.

13. Turn off real-time protection in the Windows Security app and then download an EICAR test file to the C:\Users\<USER_NAME>\Downloads folder by running following commands.



Fi [2-7] Shows the Alerts that come from malicious file downloaded in your Wazuh-agent.

14. Check the results of your ***near real-time anti-malware*** as shown in Figure 2-7.

Note

Now you can ask how malicious file has been removed from your Wazuh-agent or what's the source code of remove-threat.py or how it been implemented in python, all these concerns and question will be discussed in chapter 3 on what's our malware prevention strategy.

2.15 Wazuh Bruteforce attack detector

2.15.1 Configuration

On the attacker endpoint, install Hydra and use it to execute the brute-force attack:

- *sudo apt update.*
- *sudo apt install -y hydra.*

2.15.2 Attack Emulation

1- Create a text file with random number of passwords.

2- Run Hydra from the attacker endpoint to execute brute-force attacks against the RHEL endpoint. To do this, replace <RHEL_IP> with the IP address of the RHEL endpoint and run the command below:

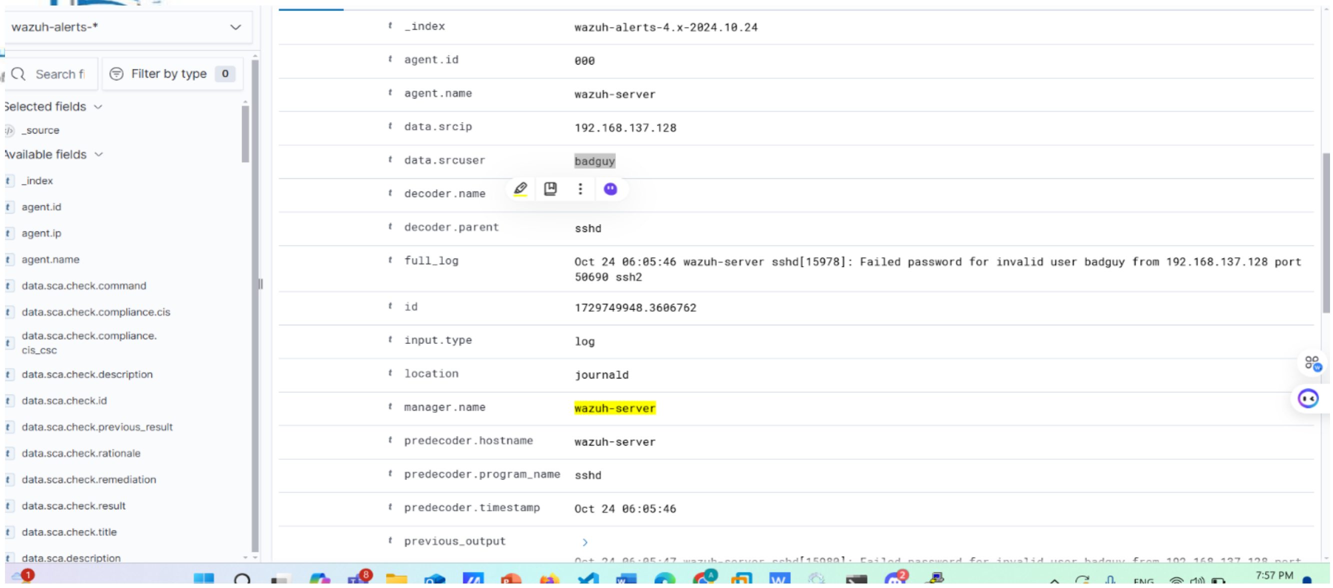
sudo hydra -l badguy -P <PASSWD_LIST.txt> <RHEL_IP> ssh

3- Run Hydra from the attacker endpoint to execute brute-force attacks against the

4- Windows endpoint. To do this, replace <WINDOWS_IP> with the IP address of the Windows endpoint and run the command below:

sudo hydra -l badguy -P <PASSWD_LIST.txt> rdp://<WINDOWS_IP>

4 - You can visualize the alert data in the Wazuh dashboard. To do this, go to the **Threat Hunting** module and add the filters in the search bar to query the alerts as shown in fig 2-8 and fig 2.9



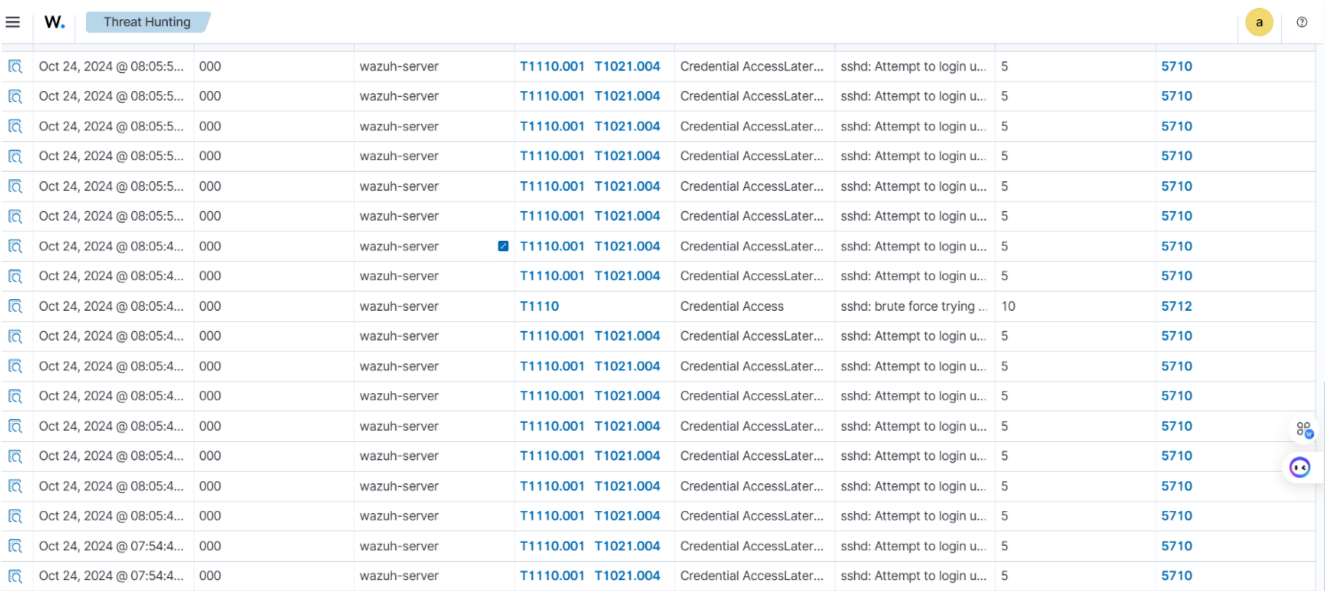


Fig [2-8] & Fig [2-9] Describe VirusTotal’s Bruteforce attack alert in threat hunting section

2.16 Wazuh' Malicious Command Detector

- ***2.16.1 Configuration***

In Ubuntu endpoint:

1-Install, start and enable Auditd if it's not present on the endpoint:

- ***sudo apt -y install auditd***
- ***sudo systemctl start auditd***
- ***sudo systemctl enable auditd***

2- As the root user, execute the following commands to append audit rules to /etc/audit/audit.rules file:

- ***echo "-a exit,always -F auid=1000 -F egid!=994 -F auid!=-1 -F arch=b32 -S execve -k audit-wazuh-c" >> /etc/audit/audit.rules***

- ***echo "-a exit,always -F auid=1000 -F egid!=994 -F auid!=-1 -F arch=b64 -S execve -k audit-wazuh-c" >> /etc/audit/audit.rules***

5- Reload the rules and confirm that they are in place:

- sudo auditctl -R /etc/audit/audit.rules
- sudo auditctl -l

6- Command line response of this output to ensure that you install auditd tool

- ***-a always,exit -F arch=b32 -S execve -F auid=1000 -F egid!=994 -F auid!=-1 -F key=audit-wazuh-c***
- ***-a always,exit -F arch=b64 -S execve -F auid=1000 -F egid!=994 -F auid!=-1 -F key=audit-wazuh-c***

7- Add the following configuration to the Wazuh agent /var/ossec/etc/ossec.conf file. This allows the Wazuh agent to read the auditd logs file

```
<localfile>
  <log_format>audit</log_format>
  <location>/var/log/audit/audit.log</location>
</localfile>
```

8- Restart the Wazuh agent:
   - sudo systemctl restart wazuh-agent.

On-Wazuh-Server:

Perform the following steps to create a CDB (Constant Database list) list of malicious programs and rules to detect the execution of the programs in the list :-

1- Look over the key-value pairs in the lookup file /var/ossec/etc/lists/audit-keys.
   - audit-wazuh-w:write
   - audit-wazuh-r:read
   - audit-wazuh-a:attribute
   - audit-wazuh-x:execute
   - audit-wazuh-c:command

2- Create a CDB list /var/ossec/etc/lists/suspicious-programs and fill its content with the following:
   - ncat:yellow
   - nc:red
   - tcpdump:orange

3- Add the list to the <ruleset> section of the Wazuh server /var/ossec/etc/ossec.conf file:
   - <list>etc/lists/suspicious-programs</list>

4- Create a high severity rule to fire when a "red" program is executed. Add this new rule to the /var/ossec/etc/rules/local_rules.xml file on the Wazuh server:

```
<group name="audit">
  <rule id="100210" level="12">
      <if_sid>80792</if_sid>
  <list field="audit.command" lookup="match_key_value" check_value="red">etc/lists/suspicious-program
    <description>Audit: Highly Suspicious Command executed: $(audit.exe)</description>
      <group>audit_command,</group>
  </rule>
</group>
```

Attack emulation

On the Ubuntu endpoint, install and run a "red" program netcat:

- sudo apt -y install netcat
- nc -v

and wazuh will show these alerts in its dashboard to alert that server face a severe malicious command attack.

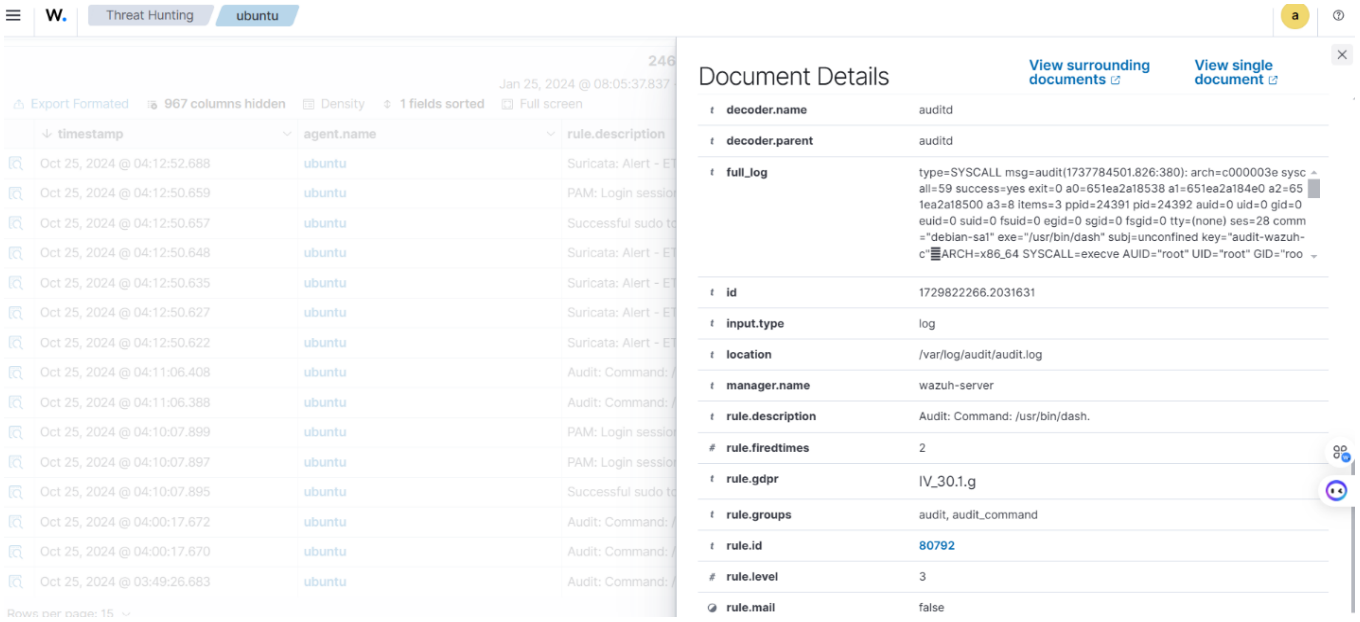


Fig [2-10] Shows malicious command detection using netcat

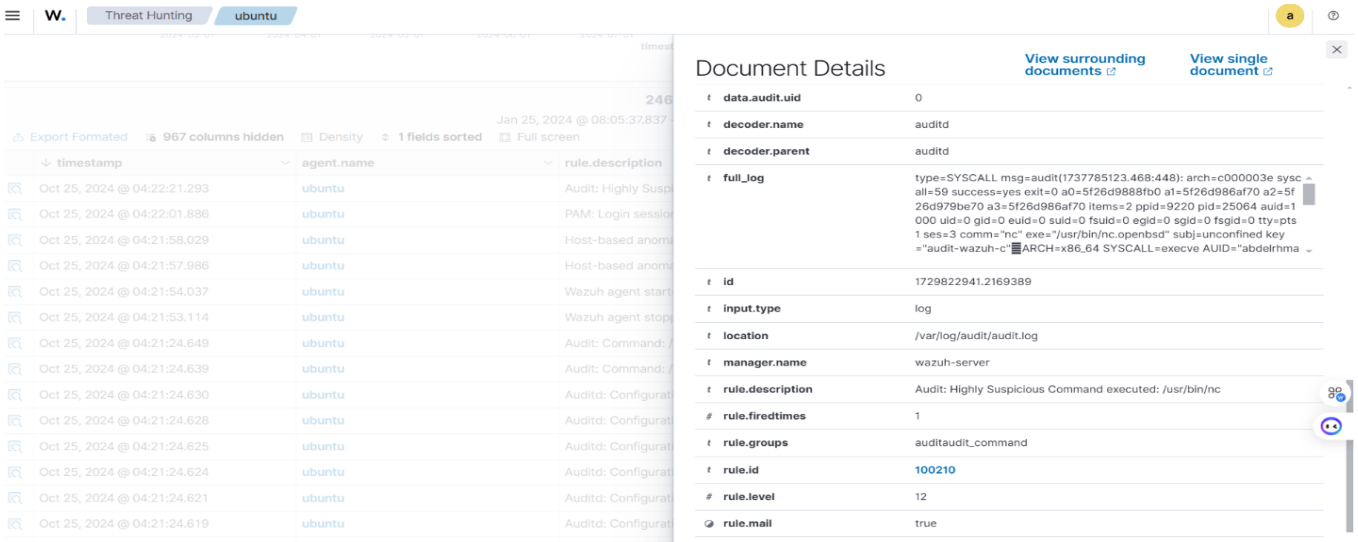


Fig [2-11] Shows malicious command detection using netstat

2.17 Wazuh's Network Intrusion Command Detector

2.17.1 Configuration

1- Install Suricata on the Ubuntu endpoint. We tested this process with version 6.0.8, and it can take some time:

- ***sudo add-apt-repository ppa:oisf/suricata-stable***
- ***sudo apt-get update***
- ***sudo apt-get install suricata -y***

2- Download and extract the Emerging Threats Suricata ruleset:

- ***cd /tmp/ && curl -https://rules.emergingthreats.net/open/suricata-6.0.8/emerging.rules.tar.gz***
- **sudo tar -xvzf emerging.rules.tar.gz && sudo mkdir /etc/suricata/rules && sudo mv rules/*.rules /etc/suricata/rules/**
- ***sudo chmod 640 /etc/suricata/rules/*.rules***

3- Modify Suricata settings in the ***/etc/suricata/suricata.yaml*** file and set the following variables:

```
HOME_NET: "<UBUNTU_IP>"
EXTERNAL_NET: "any"

default-rule-path: /etc/suricata/rules
rule-files:
- "*.rules"

# Global stats configuration
stats:
enabled: yes

# Linux high speed capture support
af-packet:
  - interface: enp0s3
```

Note

interface represents the network interface you want to monitor. Replace the value with the interface name of the Ubuntu endpoint. For example, enp0s3..

4- Use ***ifconfig*** command to see if Suricata tool activated successfully and you will see This output: -

Output

```
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::9ba2:9de3:57ad:64e5  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:14:65:bd  txqueuelen 1000  (Ethernet)
        RX packets 6704315  bytes 1268472541 (1.1 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4590192  bytes 569730548 (543.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

5- Restart the Suricata service:
   - sudo systemctl restart suricata

6- Add the following configuration to the /var/ossec/etc/ossec.conf file of the Wazuh agent. This allows the Wazuh agent to read the Suricata logs file:

```
<ossec_config>
  <localfile>
    <log_format>json</log_format>
    <location>/var/log/suricata/eve.json</location>
  </localfile>
</ossec_config>
```

7- Restart the Wazuh agent to apply the changes:
   - sudo systemctl restart wazuh-agent

2.17.2 Attack emulation

1- Ping the Ubuntu endpoint IP address from the Wazuh server:

3 ping -c 20 "<UBUNTU_IP>"

2- Alerts will be shown in Wazuh dashboard as shown in Fig [2-12] and Fig [2-13] Using Ping and Nmap.

Jan 23, 2024 @ 06:58:25.393 - Jan 23, 2025 @ 06:58:25.393

Export Formated | 662 columns hidden | Density | 1 fields sorted | Full screen

timestamp	rule.mitre.id	rule.mitre.tactic	rule.description	rule.level	rule.id
Oct 25, 2024 @ 02:44:39.959			Suricata: Alert - GPL ICMP_INFO ...	3	86601
Oct 25, 2024 @ 02:35:27.308				4	11
Oct 25, 2024 @ 02:35:27.271			Suricata: Alert - ET SCAN Potenti...	3	86601
Oct 25, 2024 @ 02:35:27.270			Suricata: Alert - ET SCAN Suspici...	3	86601
Oct 25, 2024 @ 02:35:27.263			Suricata: Alert - ET SCAN Suspici...	3	86601
Oct 25, 2024 @ 02:35:27.263			Suricata: Alert - ET SCAN Suspici...	3	86601
Oct 25, 2024 @ 02:35:27.263			Suricata: Alert - ET SCAN Suspici...	3	86601
Oct 25, 2024 @ 02:29:37.645			Suricata: Alert - ET POLICY Possi...	3	86601

Fig [2-12] Shows malicious network intrusion command using nmap-scan

Oct 25, 2024 @ 02:45:51.232			Suricata: Alert - ET POLICY Possi...	3	86601
Oct 25, 2024 @ 02:44:59.135			Suricata: Alert - GPL ICMP_INF	3	86601
Oct 25, 2024 @ 02:44:59.135			Suricata: Alert - GPL ICMP_INFO PING *NIX		86601
Oct 25, 2024 @ 02:44:56.987			Suricata: Alert - GPL ICMP_INFO ...	3	86601
Oct 25, 2024 @ 02:44:56.984			Suricata: Alert - GPL ICMP_INFO ...	3	86601
Oct 25, 2024 @ 02:44:55.145			Suricata: Alert - GPL ICMP_INFO ...	3	86601
Oct 25, 2024 @ 02:44:54.792			Suricata: Alert - GPL ICMP_INFO ...	3	86601
Oct 25, 2024 @ 02:44:52.976			Suricata: Alert - GPL ICMP_INFO ...	3	86601
Oct 25, 2024 @ 02:44:52.624			Suricata: Alert - GPL ICMP_INFO ...	3	86601
Oct 25, 2024 @ 02:44:50.895			Suricata: Alert - GPL ICMP_INFO ...	3	86601
Oct 25, 2024 @ 02:44:50.468			Suricata: Alert - GPL ICMP_INFO ...	3	86601
Oct 25, 2024 @ 02:44:50.454			Suricata: Alert - GPL ICMP_INFO ...	3	86601

Fig [2-13] Shows malicious network intrusion command using ping

Chapter 3: Prevention Strategy and Training

3.1 Malware prevention strategy

3.1.1 General Prevention Strategies

A multi-layered defense strategy offers robust protection against different stages of the malware lifecycle by implementing various security measures across endpoint, network, email, web, and data protection domains. Here's a detailed explanation of each component:

1. Endpoint Protection

Endpoints are often the primary target for malware. Ensuring robust endpoint protection helps to defend against direct attacks and compromises.

- **Antivirus and Anti-malware Software**: Deploying reputable antivirus software on all endpoints helps detect and remove known malware. Regularly updating the virus definitions ensures the software can recognize the latest threats, while heuristic scanning allows for detecting unknown or new malware based on suspicious behavior patterns.

- **Application Whitelisting**: This approach allows only pre-approved applications to execute on endpoints. It prevents unauthorized software from running, which can significantly reduce the risk of malware execution. By using policies that define the allowed applications, suspicious software can be blocked before it has a chance to infect the system.

- **Patch Management**: Ensuring that operating systems, applications, and firmware are kept up to date is crucial for closing security gaps. An automated patch management system can prioritize critical updates and patch known vulnerabilities (like those identified by CVE numbers) that malware often exploits. It helps to limit the attack surface by eliminating software bugs that could be leveraged by attackers.

- **User Account Control (UAC)**: Enforcing the principle of least privilege limits the ability of users and applications to perform tasks that could compromise the system. For instance, users should only have the necessary permissions to perform their job functions, preventing malware from gaining administrative access to execute harmful actions or spread within the network.

2. Network Security

Network security controls protect against unauthorized access, lateral movement, and data exfiltration.

- **Firewall Configuration**: Firewalls serve as a frontline defense, controlling traffic based on predetermined security rules. By blocking unauthorized connections and restricting traffic to trusted IP addresses, ports, and services, firewalls help to prevent external threats from infiltrating the network. Stateful inspection allows the firewall to monitor active connections and decide whether to allow or deny traffic based on the state of the connection, while intrusion prevention capabilities can detect and block malicious traffic.

- **Network Segmentation**: Segmenting the network into different zones (e.g., DMZ, internal network, secure zones) helps contain potential threats and minimizes the damage in case of a breach. For example, placing sensitive data in a secure zone with stricter controls, while using ACLs to restrict access between zones, can prevent attackers from easily moving between network areas.

- **Intrusion Detection/Prevention Systems (IDS/IPS)**: IDS/IPS solutions monitor network traffic for indications of malicious activity, such as known attack signatures or deviations from normal traffic patterns (anomalies). While IDS detects and alerts about potential threats, IPS can take automated actions to block or contain suspicious traffic,
- thus, reducing the risk of successful attacks.

3. Email Security

Email remains a major vector for delivering phishing attacks and malware, making email security essential.

- **Email Filtering**: Email filtering solutions block spam, phishing attempts, and emails containing malicious attachments or links. This reduces the likelihood that a user will accidentally open a malicious email or click on a dangerous link.

- **Attachment Sandboxing**: This technology executes email attachments in a controlled environment (sandbox) to determine if they behave maliciously before being delivered to the recipient. If malicious behavior is detected, the attachment is quarantined or blocked.

- **Email Authentication Protocols**: Enforcing email authentication protocols like DMARC, SPF, and DKIM helps prevent email spoofing and ensures that the email sender is legitimate. These protocols verify that incoming emails are from trusted sources and have not been altered during transit, reducing the chances of successful phishing attacks.

4. Web Security

Web-based threats such as drive-by downloads, malicious websites, and phishing can be mitigated with web security measures.

- **Web Filtering**: Web filtering solutions are crucial for restricting access to high-risk websites, reducing exposure to online threats such as malware and phishing. To further enhance security, advanced backend techniques can be employed, including application-level guard controls, secure password hashing with salting (using algorithms like bcrypt or Argon2), and robust encryption using AES-256 with custom modifications. These modifications can involve custom key derivation functions, cipher mode enhancements,

key rotation, and splitting to make decryption nearly impossible for attackers. Additionally, code obfuscation and integrity checks can protect encryption routines from reverse-engineering, creating a multi-layered defense that goes beyond conventional measures to safeguard sensitive data and web applications.

- **SSL/TLS Inspection**: As a significant amount of web traffic is encrypted, it is important to inspect SSL/TLS traffic to identify hidden threats. SSL/TLS inspection decrypts traffic for inspection, allowing security tools to detect and block malware concealed in encrypted web traffic.

- **Content Delivery Network (CDN)**: Using a CDN for web application delivery not only improves performance but also provides built-in security features like DDoS protection and a web application firewall (WAF). The WAF helps protect web applications from common attacks such as SQL injection, cross-site scripting (XSS), and other vulnerabilities.

5. Data Protection and Encryption

Protecting data is critical for reducing the impact of data breaches and preventing data loss.

- **Data Loss Prevention (DLP)**: DLP solutions monitor and control the movement of sensitive data, preventing unauthorized data transfers or leakage. For instance, DLP can block or log attempts to send confidential files outside the organization, ensuring data security policies are enforced.

- **Encryption**: Encrypting data at rest and in transit ensures that even if data is intercepted or stolen, it cannot be easily accessed or used by unauthorized individuals. Using strong encryption algorithms, such as AES-256, makes it difficult for attackers to decrypt the data.

- **Access Control Policies**: Enforcing strict access control policies limits who can access sensitive data. Policies can be based on roles and responsibilities, ensuring that only authorized users can access certain data. Role-based access control (RBAC) and multi-factor authentication (MFA) are effective measures to further secure sensitive information.

Each layer in this defense strategy complements the others, making it harder for malware to achieve its objectives. By integrating these measures, organizations can reduce risks, detect threats earlier, and respond more effectively to potential incidents.

3.2 Wazuh's exe file used for prevention strategy (remove-threat.py)

Do you remember remove-threat.py which we change it to exe to run it with defined VirusTotal rules to add alerts to our client logs and remove malicious file and we explain this part step-by-step in chapter 2 and you can review it here , now we will explain every part in this code.

```
import os
import sys
import json
import datetime

if os.name == 'nt':
    LOG_FILE = "C:\\Program Files (x86) \\ossec-agent\\active-response\\active-responses.log"
else:
    LOG_FILE = "/var/ossec/logs/active-responses.log"

ADD_COMMAND = 0
DELETE_COMMAND = 1
CONTINUE_COMMAND = 2
ABORT_COMMAND = 3

OS_SUCCESS = 0
OS_INVALID = -1

class message:
    def __init__(self):
        self.alert = ""
        self.command = 0

def write_debug_file(ar_name, msg):
    with open(LOG_FILE, mode="a") as log_file:
        log_file.write(str(datetime.datetime.now().strftime('%Y/%m/%d %H:%M:%S')) + " " + ar_name + ": " + msg +"\n")

def setup_and_check_message(argv):

    # get alert from stdin
    input_str = ""
    for line in sys.stdin:
        input_str = line
        break
```

```
    try:
        data = json.loads(input_str)
    except ValueError:
        write_debug_file(argv[0], 'Decoding JSON has failed, invalid input format')
        message.command = OS_INVALID
        return message

    message.alert = data

    command = data.get("command")

    if command == "add":
        message.command = ADD_COMMAND
    elif command == "delete":
        message.command = DELETE_COMMAND
    else:
        message.command = OS_INVALID
        write_debug_file(argv[0], 'Not valid command: ' + command)

    return message


def send_keys_and_check_message(argv, keys):

    # build and send message with keys
    keys_msg = json.dumps({"version": 1,"origin":{"name": argv[0],"module":"active-response"},"command":"check_keys","parameters":{"keys":keys}})

    write_debug_file(argv[0], keys_msg)

    print(keys_msg)
    sys.stdout.flush()

    # read the response of previous message
```

```
 input_str = ""
    while True:
        line = sys.stdin.readline()
        if line:
            input_str = line
            break

    # write_debug_file(argv[0], input_str)

    try:
        data = json.loads(input_str)
    except ValueError:
        write_debug_file(argv[0], 'Decoding JSON has failed, invalid input format')
        return message

    action = data.get("command")

    if "continue" == action:
        ret = CONTINUE_COMMAND
    elif "abort" == action:
        ret = ABORT_COMMAND
    else:
        ret = OS_INVALID
        write_debug_file(argv[0], "Invalid value of 'command'")

    return ret


def main(argv):

    write_debug_file(argv[0], "Started")

    # validate json and get command
    msg = setup_and_check_message(argv)
```

```
if msg.command < 0:
        sys.exit(OS_INVALID)

    if msg.command == ADD_COMMAND:
        alert = msg.alert["parameters"]["alert"]
        keys = [alert["rule"]["id"]]
        action = send_keys_and_check_message(argv, keys)

        # if necessary, abort execution
        if action != CONTINUE_COMMAND:

            if action == ABORT_COMMAND:
                write_debug_file(argv[0], "Aborted")
                sys.exit(OS_SUCCESS)
            else:
                write_debug_file(argv[0], "Invalid command")
                sys.exit(OS_INVALID)

        try:
            file_path =
msg.alert["parameters"]["alert"]["data"]["virustotal"]["source"]["file"]
            if os.path.exists(file_path):
                os.remove(file_path)
            write_debug_file(argv[0], json.dumps(msg.alert) + " Successfully removed
threat")
        except OSError as error:
            write_debug_file(argv[0], json.dumps(msg.alert) + "Error removing
threat")



    else:
        write_debug_file(argv[0], "Invalid command")

    write_debug_file(argv[0], "Ended")
```

```
if __name___== "__main__":
    main(sys.argv)
```

This is the whole code, seems complicated! but let us explain it block by block: -

1- Imports and Global Variable Definitions

```
import os
import sys
import json
import datetime
if os.name == 'nt':
    LOG_FILE = "C:\\Program Files (x86) \\ossec-agent\\active-response\\active-responses.log"
else:
    LOG_FILE = "/var/ossec/logs/active-responses.log"

ADD_COMMAND = 0
DELETE_COMMAND = 1
CONTINUE_COMMAND = 2
ABORT_COMMAND = 3

OS_SUCCESS = 0
OS_INVALID = -1
```

This block imports the necessary modules:

- **os** is used for operating system-dependent functionalities.
- **sys** allows interaction with the system, like reading and writing data from standard input/output.

- **json** handles JSON encoding and decoding.
- **datetime** is used for timestamping log entries.
- This determines the location of the log file based on the operating system. For **Windows (nt)**, a different file path is used compared to Unix-based systems.
- These constants define commands and status codes used throughout the script:
- **ADD_COMMAND, DELETE_COMMAND**, etc., represent different command types.
- **OS_SUCCESS and OS_INVALID** indicate whether the operation succeeded or encountered an error.

2. Message Class Definition

```
class message:
    def __init__(self):
        self.alert = ""
        self.command = 0
```

The message class defines a structure to store alert details and the associated command. It initializes alert as an empty string and command as zero.

3. Logging Function

```
def write_debug_file(ar_name, msg):
```

```
with open(LOG_FILE, mode="a") as log_file:
        log_file.write(str(datetime.datetime.now().strftime('%Y/%m/%d %H:%M:%S')) + "
" + ar_name + ": " + msg +"\n")
```

- opens the `LOG_FILE` in append mode (`"a"`), ensuring that new log entries are added to the end of the file without overwriting existing content.
- It writes a log entry that includes the current timestamp, the name of the script (`ar_name`), and the actual message (`msg`) provided as input.
- The timestamp is formatted as `YYYY/MM/DD HH:MM:SS` to ensure consistency.

4. Message Setup and Validation

```
def setup_and_check_message(argv):

    # get alert from stdin
    input_str = ""
    for line in sys.stdin:
        input_str = line
        break



    try:
        data = json.loads(input_str)
    except ValueError:
        write_debug_file(argv[0], 'Decoding JSON has failed, invalid input format')
        message.command = OS_INVALID
        return message

    message.alert = data

    command = data.get("command")
```

```
    if command == "add":
        message.command = ADD_COMMAND
    elif command == "delete":
        message.command = DELETE_COMMAND
    else:
        message.command = OS_INVALID
        write_debug_file(argv[0], 'Not valid command: ' + command)

    return message.
```

This function reads and validates the incoming message:

Reading Input: It reads the input from standard input (`sys.stdin`) line by line, expecting a JSON-formatted string containing alert information.

JSON Decoding: It attempts to parse the input string using `json.loads()`. If parsing fails (due to an invalid JSON format), it logs an error and sets the `command` to `OS_INVALID`.

Command Checking: If parsing is successful, it checks the `"command"` field in the JSON data:

- If the command is `"add"`, it sets `message.command` to `ADD_COMMAND`.
- If the command is `"delete"`, it sets `message.command` to `DELETE_COMMAND`.
- If the command is neither of these, it logs an invalid command message and sets the command to `OS_INVALID`.

5. Sending and Validating Keys

```
def send_keys_and_check_message (argv, keys):

    # build and send message with keys
    keys_msg = json.dumps({"version": 1,"origin":{"name": argv[0],"module":"active-response"},"command":"check_keys","parameters":{"keys":keys}})

    write_debug_file(argv[0], keys_msg)

    print(keys_msg)
    sys.stdout.flush()

    # read the response of previous message
    input_str = ""
    while True:
        line = sys.stdin.readline()
        if line:
            input_str = line
            break

    # write_debug_file(argv[0], input_str)

    try:
        data = json.loads(input_str)
    except ValueError:
        write_debug_file(argv[0], 'Decoding JSON has failed, invalid input format')
        return message

    action = data.get("command")

    if "continue" == action:
        ret = CONTINUE_COMMAND
    elif "abort" == action:
        ret = ABORT_COMMAND
    else:
        ret = OS_INVALID
```

```
write_debug_file(argv[0], "Invalid value of 'command'")
return ret
```

This function builds and sends a message to validate keys:

Building the Message: It creates a JSON message containing details about the origin, command, and parameters. The `keys` parameter is the list of keys to be validated.

Logging and Sending: It logs the message to the debug file and sends it via `stdout`.

Reading the Response: The function waits for a response by reading from `stdin`. It expects a valid JSON response.

Decoding the Response: It tries to parse the response JSON. If parsing fails, it logs an error.

Validating the Command in Response: It checks the `"command"` field in the response:

- If the command is `"continue"`, it returns `CONTINUE_COMMAND`.
- If the command is `"abort"`, it returns `ABORT_COMMAND`.
- If neither, it logs an invalid command error and returns `OS_INVALID`.

6. Main Function for Execution

```
def main(argv):

    write_debug_file(argv[0], "Started")

    # validate json and get command
    msg = setup_and_check_message(argv)

    if msg.command < 0:
        sys.exit(OS_INVALID)

    if msg.command == ADD_COMMAND:
        alert = msg.alert["parameters"]["alert"]
        keys = [alert["rule"]["id"]]
        action = send_keys_and_check_message(argv, keys)

        # if necessary, abort execution
        if action != CONTINUE_COMMAND:

            if action == ABORT_COMMAND:
                write_debug_file(argv[0], "Aborted")
                sys.exit(OS_SUCCESS)
            else:
                write_debug_file(argv[0], "Invalid command")
                sys.exit(OS_INVALID)

        try:
            file_path =
msg.alert["parameters"]["alert"]["data"]["virustotal"]["source"]["file"]
            if os.path.exists(file_path):
                os.remove(file_path)
            write_debug_file(argv[0], json.dumps(msg.alert) + " Successfully removed
threat")
        except OSError as error:
            write_debug_file(argv[0], json.dumps(msg.alert) + "Error removing
threat")
```

```
    else:
        write_debug_file(argv[0], "Invalid command")

    write_debug_file(argv[0], "Ended")

    sys.exit(OS_SUCCESS)
```

The main function coordinates the entire execution:

Start Logging: It logs "Started" at the beginning.

Message Validation: It calls setup_and_check_message() to validate the incoming message.

Command Execution: If the command is invalid, it exits. If the command is ADD_COMMAND, it retrieves the alert information and extracts keys for validation.

Key Validation: It calls send_keys_and_check_message() to validate the keys. If the response requires aborting, it logs "Aborted" and exits.

Threat Removal: If the command is valid and allowed to proceed, it attempts to remove a file specified in the alert. If the file is removed successfully, it logs the success; otherwise, it logs an error.

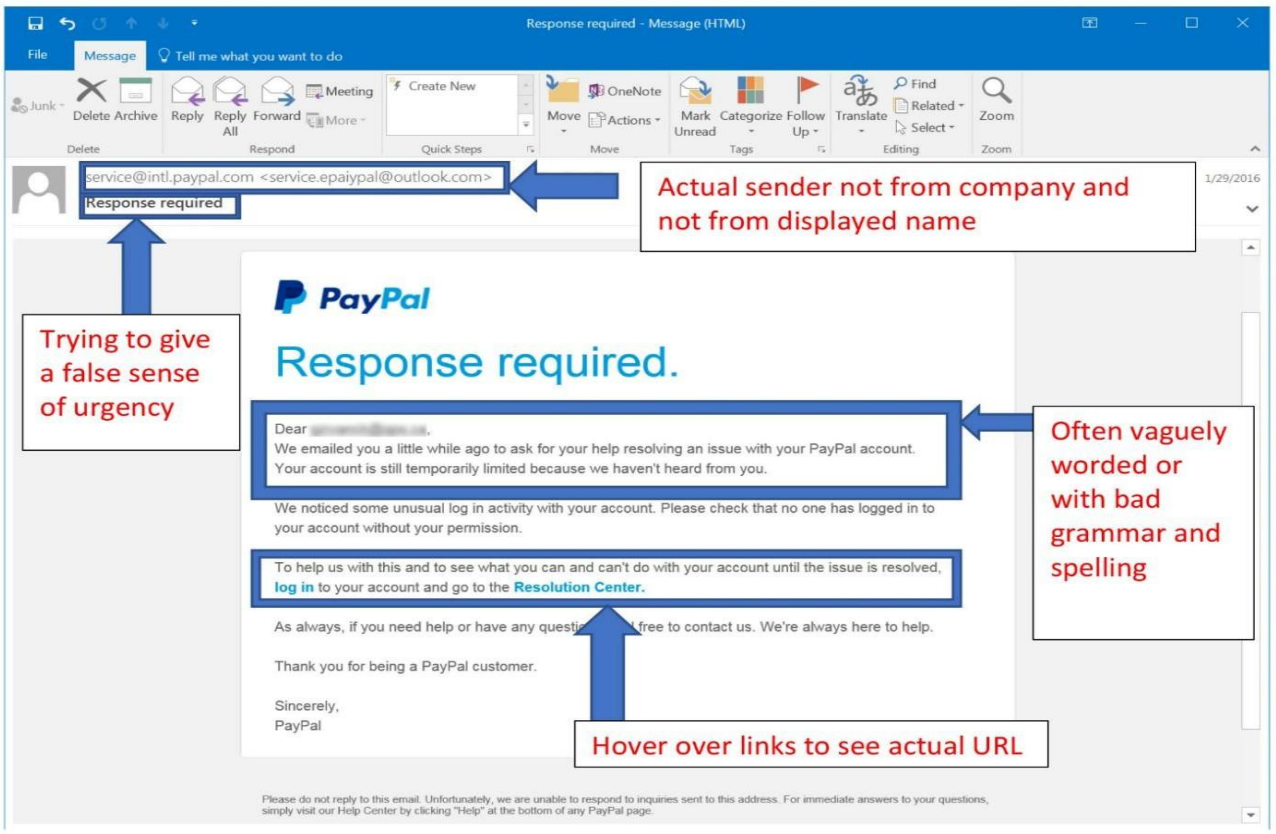
End Logging: It logs "Ended" at the end and exits with a success status.

7. Script Execution Entry Point

```
if __name__ == "__main__":
    main(sys.argv)
```

This ensures that the script runs the `main()` function when executed as a standalone program. It passes the command-line arguments to `main()`. This block allows the script to be used as a standalone executable or imported as a module without running the `main()` function automatically.

3.3 Phishing mail shapes



Even if you think the email is legitimate, if it is not something you are expecting it is a good idea to contact the person you believe to be the sender “out-of-band,” or by another method than clicking “reply” or any links in the email. For example, call a phone number you know belongs to the institution or person or go directly to their website by typing in the URL.

***original image taken from phishing.org**

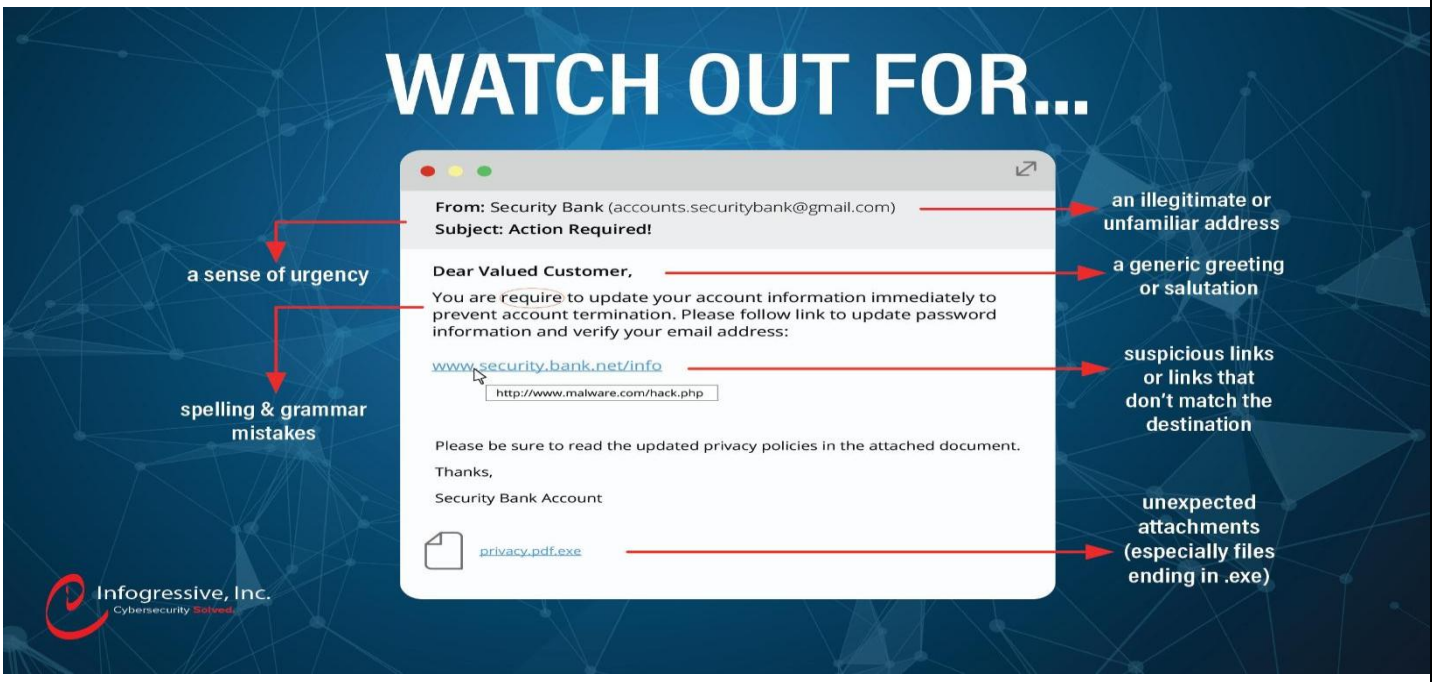


Fig [3-1] & Fig [3-2] Shows phishing mails anatomy that contains a malicious attachment

.

Chapter 4: SOC Analyst Final Report

4.1 Final Report: Malware Analysis and Incident Response

4.1.1 Malware Analysis Report

Incident Response Framework

- Preparation:
  - Ensured up-to-date antivirus and antimalware definitions.
  - Enabled logging and SIEM monitoring for detecting anomalous activities.
  - Prepared incident response teams and playbooks to handle malware incidents.

- Identification:
  - Used threat intelligence and hash-based detection to identify known malware.
  - Detected suspicious activities such as unusual file executions, network connections, and behavioral indicators consistent with malware.
  - Hash values used for identification:
    - MD5: 6685c433705f558c5535789234db0e5a (Trojan)
    - MD5: 141812f77bdef659d1bc5c1403c5e094 (AgentTesla)
    - MD5: 7299c49dd85069e47d6514ab5e10c264 (Trojan)

- Containment:
    - Implemented containment strategies such as network isolation, disabling affected accounts, and removing infected devices.
    - Used host-based firewalls to block malicious IP addresses:
        - 173.231.198.30, 5.135.143.133, 70.38.21.229

- Eradication:
    - Removed malware using endpoint security tools.
    - Deleted or quarantined affected files.
    - Cleaned registry entries and startup items.

- Recovery:
    - Restored systems from clean backups.
    - Monitored the network for any signs of reinfection.
    - Validated the integrity of restored systems.

- Lessons Learned:
    - Identified gaps in the incident response process and developed strategies to improve detection, response time, and prevention measures.
    - Updated playbooks to include new attack vectors and indicators of compromise (IOCs).

Simulation Outcomes

- Scenario Description: A simulated SOC environment was created to replicate a real-world malware incident involving multiple attack vectors.

- Simulated Attacks:
    - Initial Compromise: A phishing email with an infected attachment (MS Word document leveraging CVE-2017-11882) was used.
    - Lateral Movement: Exploited vulnerabilities such as CVE-2017-16995 for privilege escalation using JuicyPotato.EXE.
    - Command and Control (C2) Communication: Malicious traffic was generated to domains such as mail.davaobay.com.ph and andaluciabeach.net.

- Incident Response Simulation:
    - The SOC team followed the incident response framework, conducting detection, containment, and eradication procedures.
    - Multiple tools were used during the simulation, including SIEM alerts, antivirus, network monitoring, and manual analysis.
- Evaluation Metrics:
    - Detection Time: 20 minutes
    - Response Time: 45 minutes
    - Eradication Success: 95%
    - Reinfection Cases: 0

Malware Analysis Details

- Sample 1: Trojan Analysis (MD5: 6685c433705f558c5535789234db0e5a)
    - Behavior: Used Mshta.exe to execute malicious JavaScript.
    - Indicators: Command-line execution, network traffic to external IPs.

- Sample 2: AgentTesla Analysis (MD5: 141812f77bdef659d1bc5c1403c5e094)
    - Behavior: Credential harvesting from email clients and browsers.
    - Indicators: Communication to C2 server at 173.231.198.30.

- Sample 3: Exploit Analysis (MD5: 7299c49dd85069e47d6514ab5e10c264)
    - Exploit: CVE-2017-11882 used to execute arbitrary code.
    - Indicators: Execution of EQNEDT32.EXE followed by network traffic.

Prevention Strategy

- Policies and Procedures:
    - Implemented stricter email filtering and phishing detection.
    - Regularly patched and updated vulnerable software.

- Educated employees on the dangers of phishing and social engineering.

- Technical Measures:
  - Deployed advanced endpoint detection and response (EDR) tools.
  - Configured firewalls to block known malicious IP addresses and domains.
  - Conducted regular vulnerability scanning and penetration testing.
  - In our case we use VirusTotal API and python script to remove detected malicious files.

Acknowledge to user what happened

To inform User what happen to his host and why his host has been contained and what happened to his device to avoid it in the future.

Mail format

Subject: Notification of Host Containment Due to Security Incident

Dear [User's Name],

We hope this message finds you well. We are writing to inform you that your computer, identified as [Host Name], has been temporarily contained from the network. This action was taken as a precautionary measure in response to a detected security incident involving potentially harmful activity.

What Happened?

During routine monitoring, our Security Operations Center (SOC) identified unusual behavior originating from your computer. Our analysis revealed evidence suggesting the presence of a malware infection, which could pose a risk to the network and other devices. The suspected threat involves malicious software that may have been introduced through a phishing email or compromised download.

Why Was Your Host Contained?

To prevent the potential spread of malware and safeguard sensitive data, it was necessary to isolate your device from the network. This containment action helps to:

- Limit the impact of the malicious activity.
- Prevent unauthorized access to other network resources.
- Allow our SOC team to investigate and remediate the issue safely.

Next Steps

1. Our SOC team will continue analyzing the incident to determine the extent of the threat and eliminate any malicious components.
2. Your device will undergo a thorough security check and cleaning. We will notify you once it is safe to reconnect to the network.
3. We recommend you refrain from using your device until you receive further instructions from our IT team.

Please rest assured that we are taking all necessary steps to resolve the issue quickly and minimize any disruption. If you have any questions or need assistance, do not hesitate to contact the IT support team at [IT Support Contact Information].

We appreciate your understanding and cooperation during this time.

Sincerely,

[SOC Member Name.]

[His Position]

[Company name]

[Contact Information]

Future Recommendations

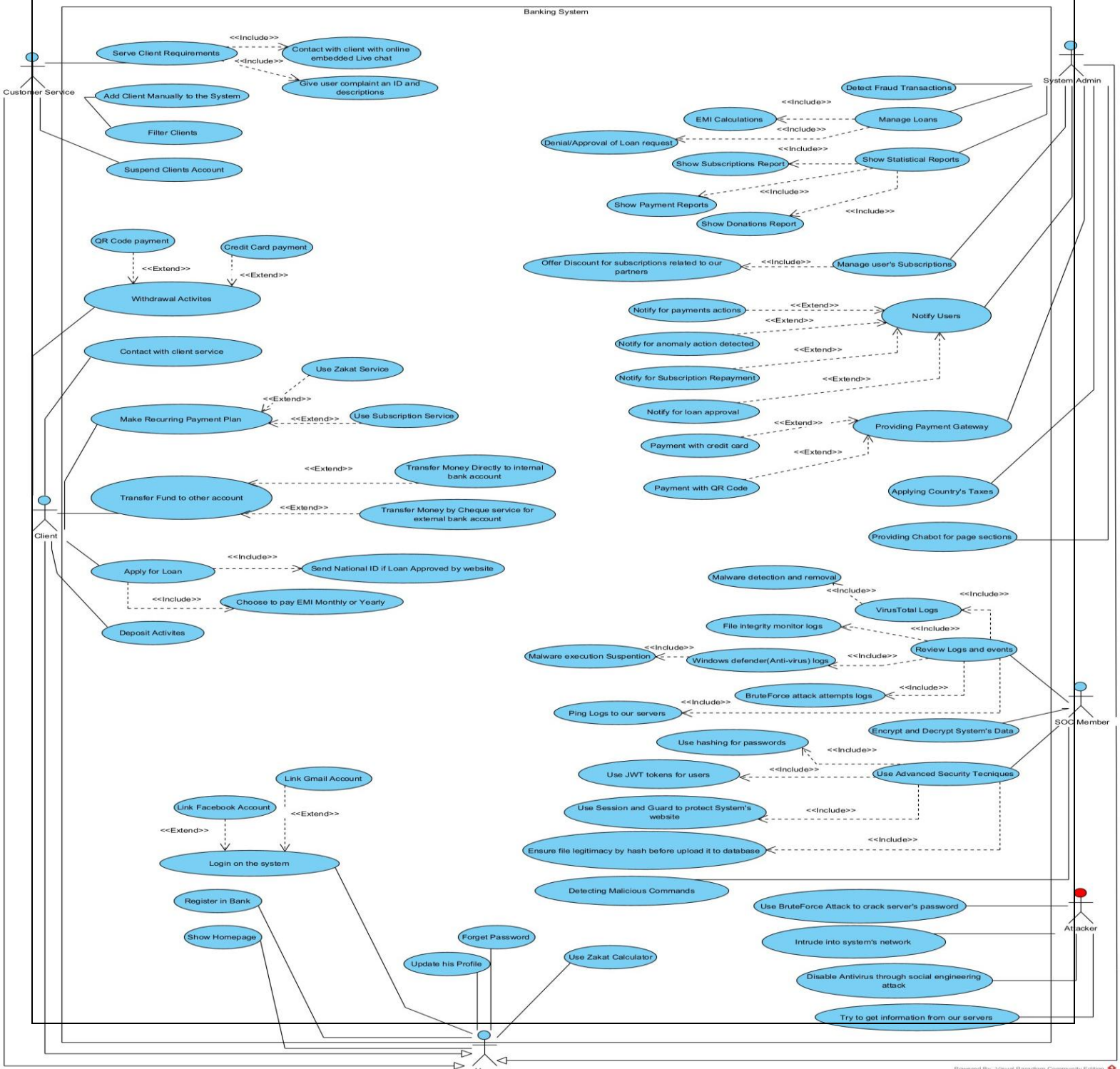
- Strengthen User Awareness Training:
    - Continue training users on recognizing phishing attempts and proper reporting procedures.
- Enhance Detection Capabilities:
    - Invest in more advanced SIEM and behavioral analysis tools.

- Routine Simulation Drills:
    - Conduct regular simulation drills to test and improve incident response procedures.

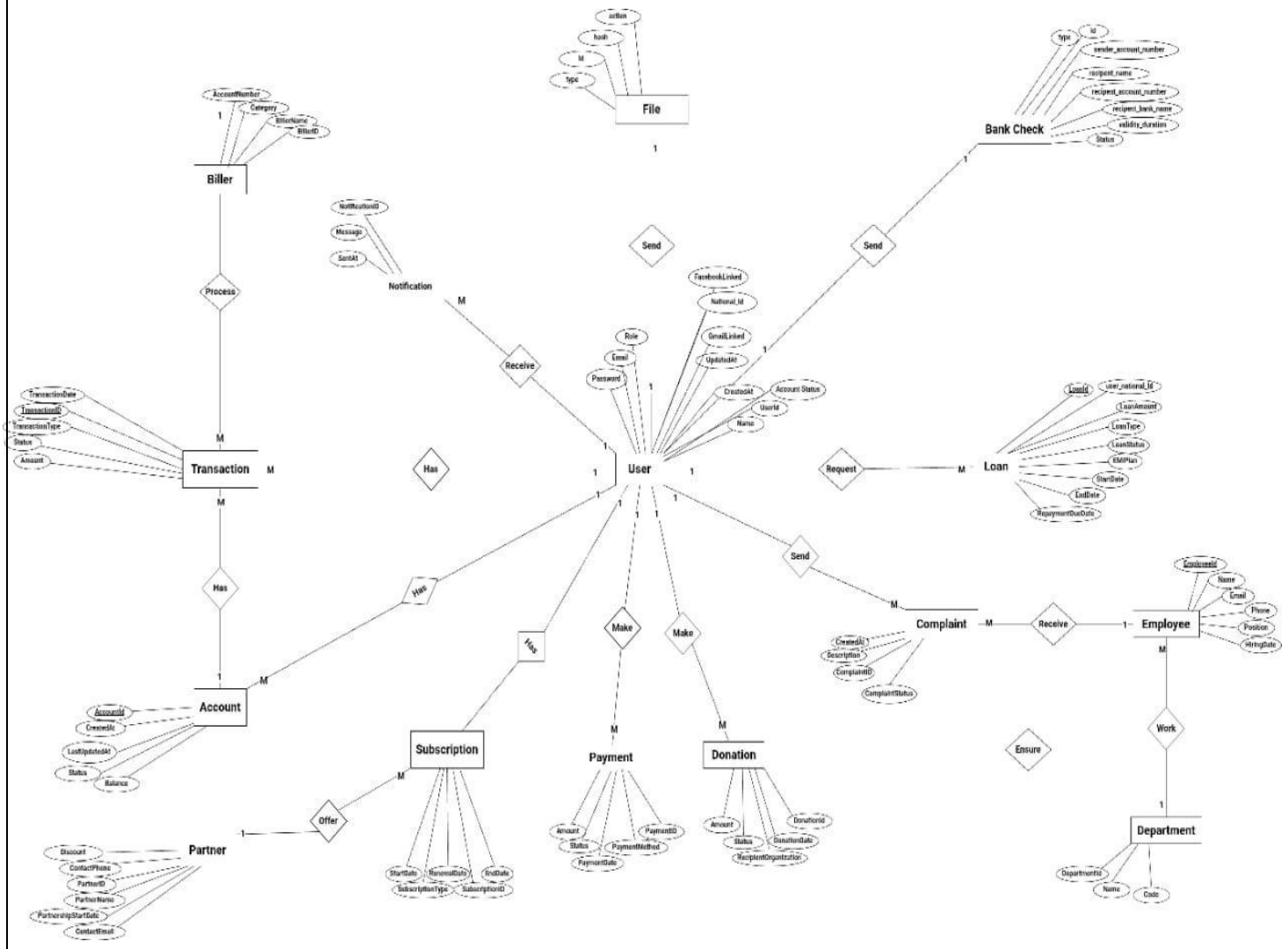
Chapter 5: Software assistance in action

5.1 Software Diagrams for main banking functionalities

5.1.1 Use case diagram

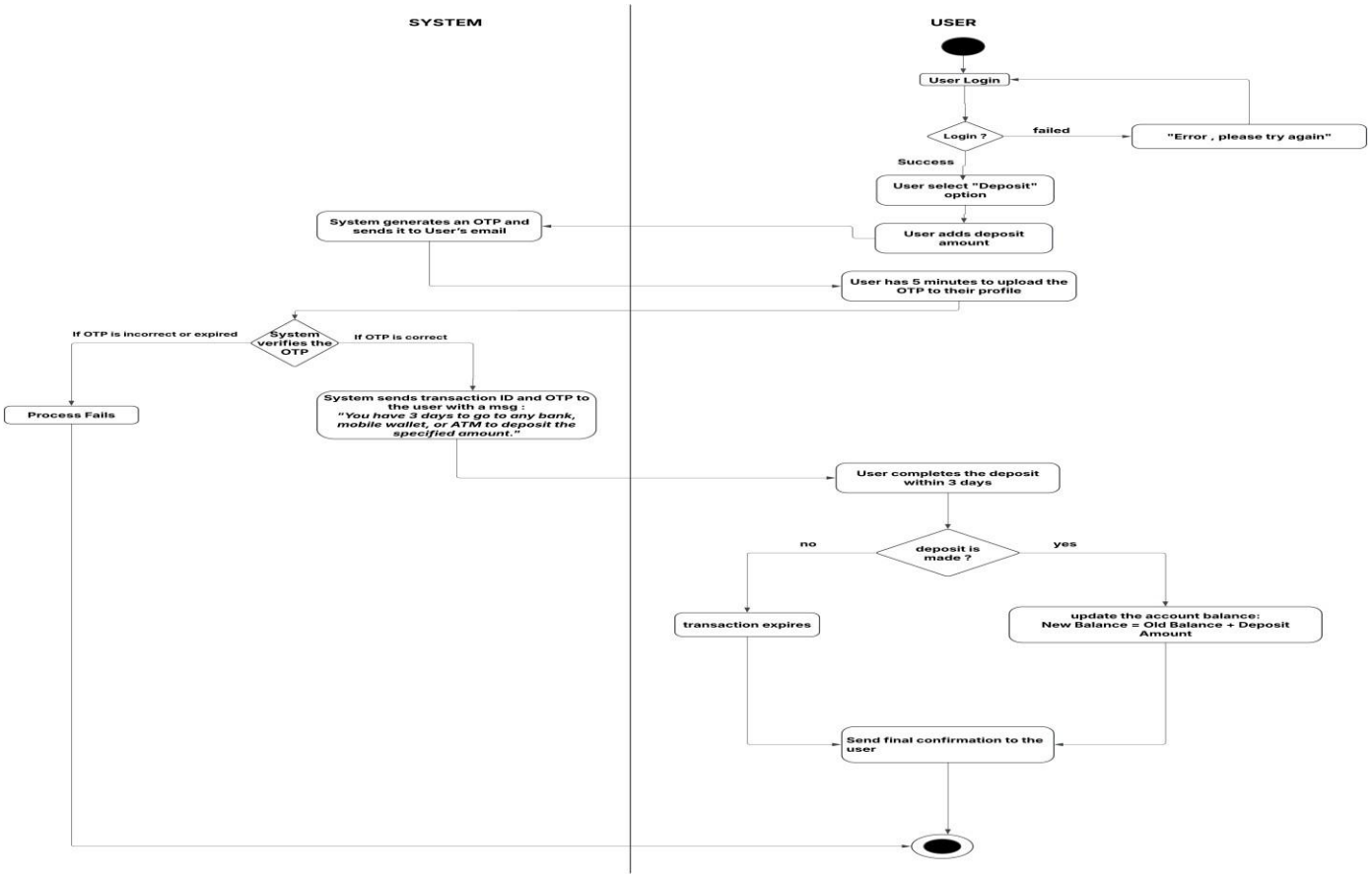


5.1.2 Entity relationship diagram

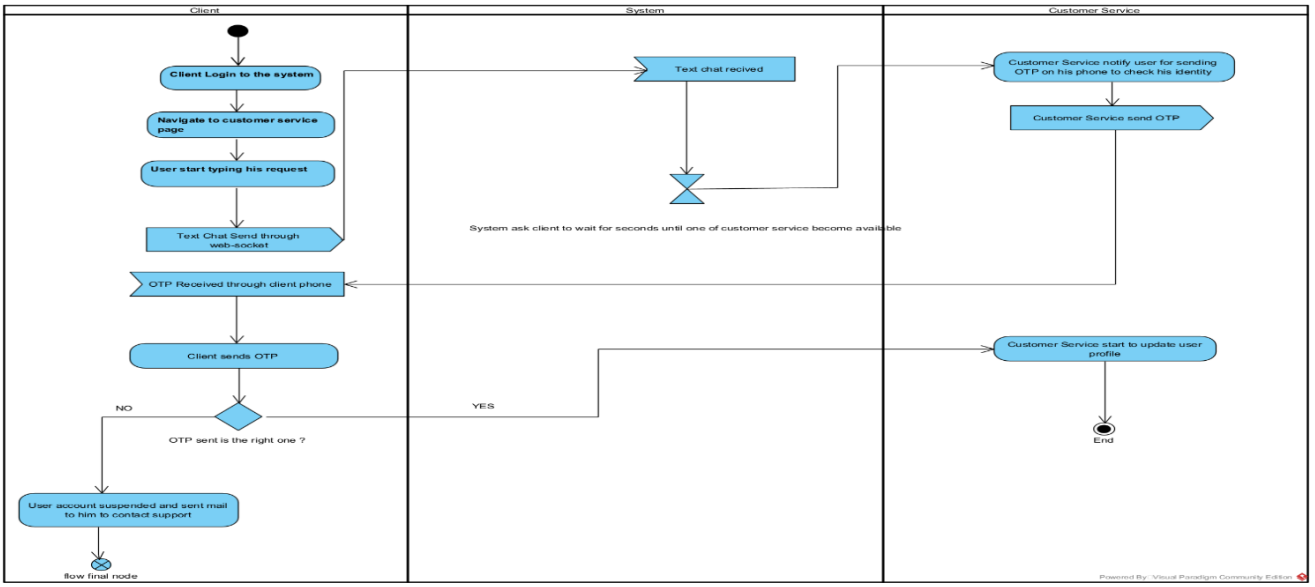


5.1.3Activity diagrams

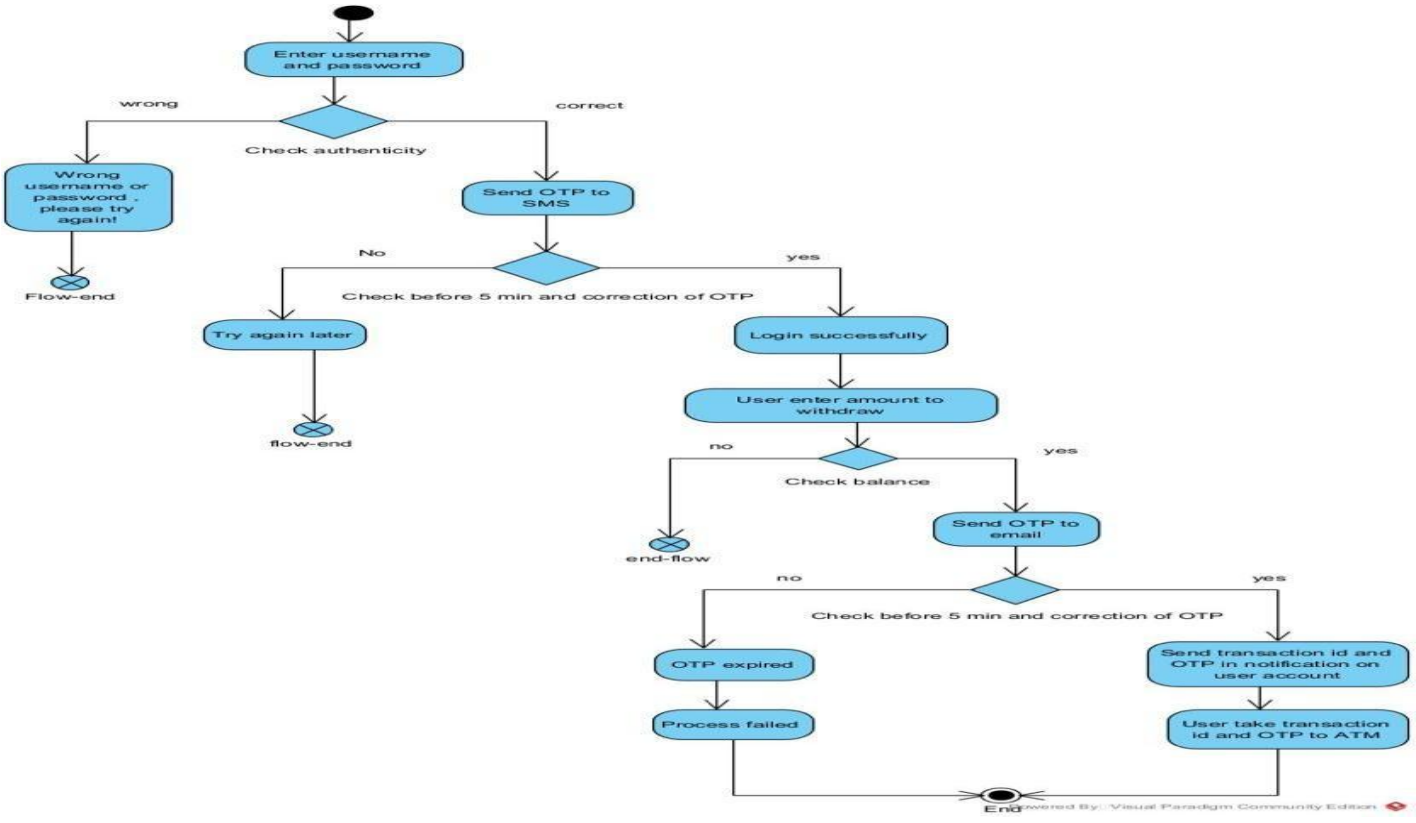
5.1.3.1 Client’s deposit activities



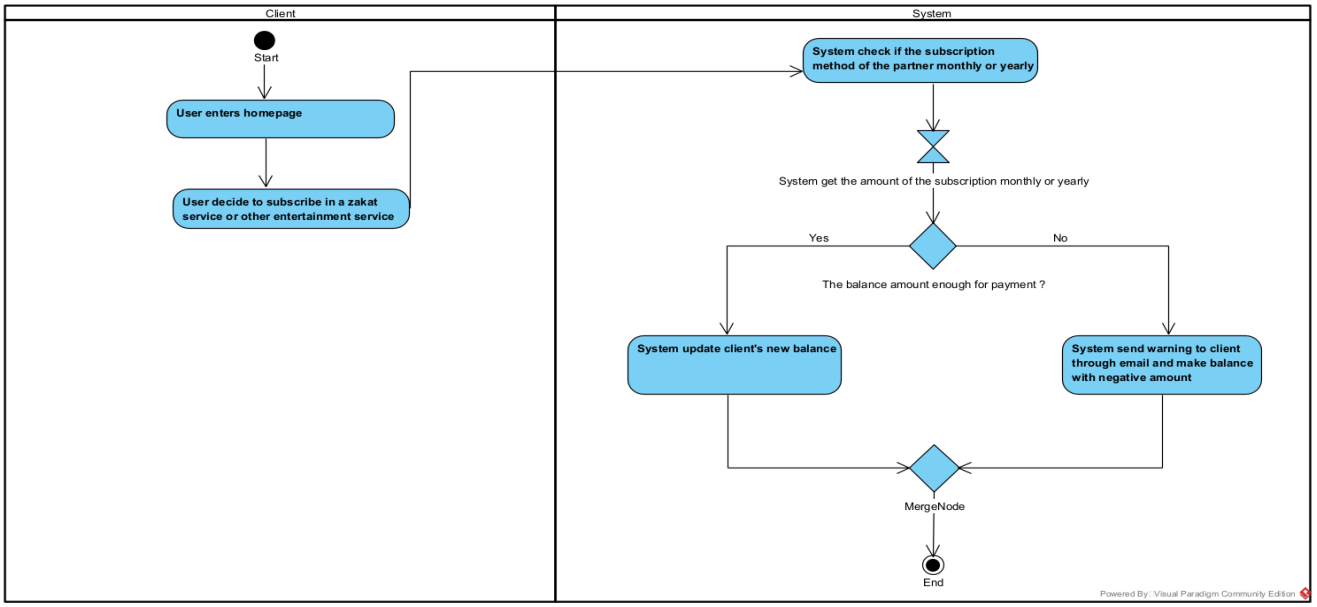
5.1.3.2 Real-time chat activity



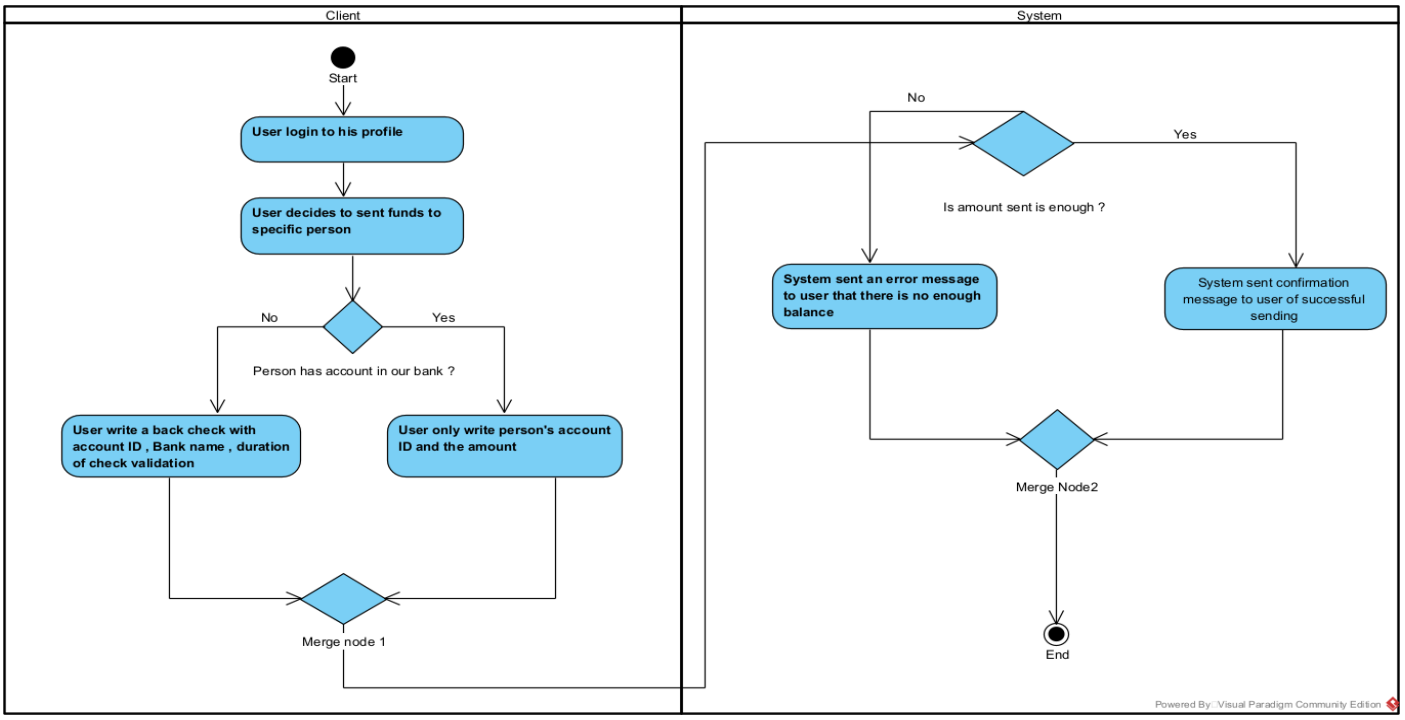
5.3.1.3 Client's Withdrawal activity



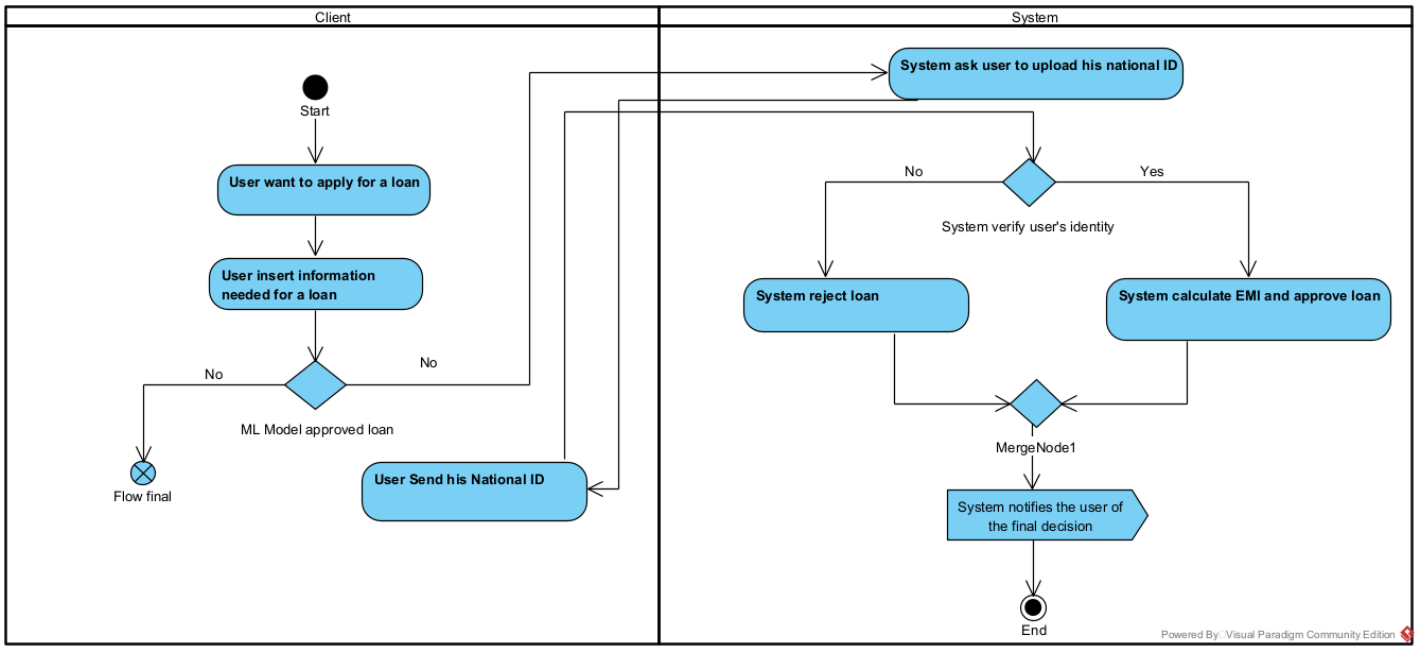
5.3.1.4 Recurring Client's payment



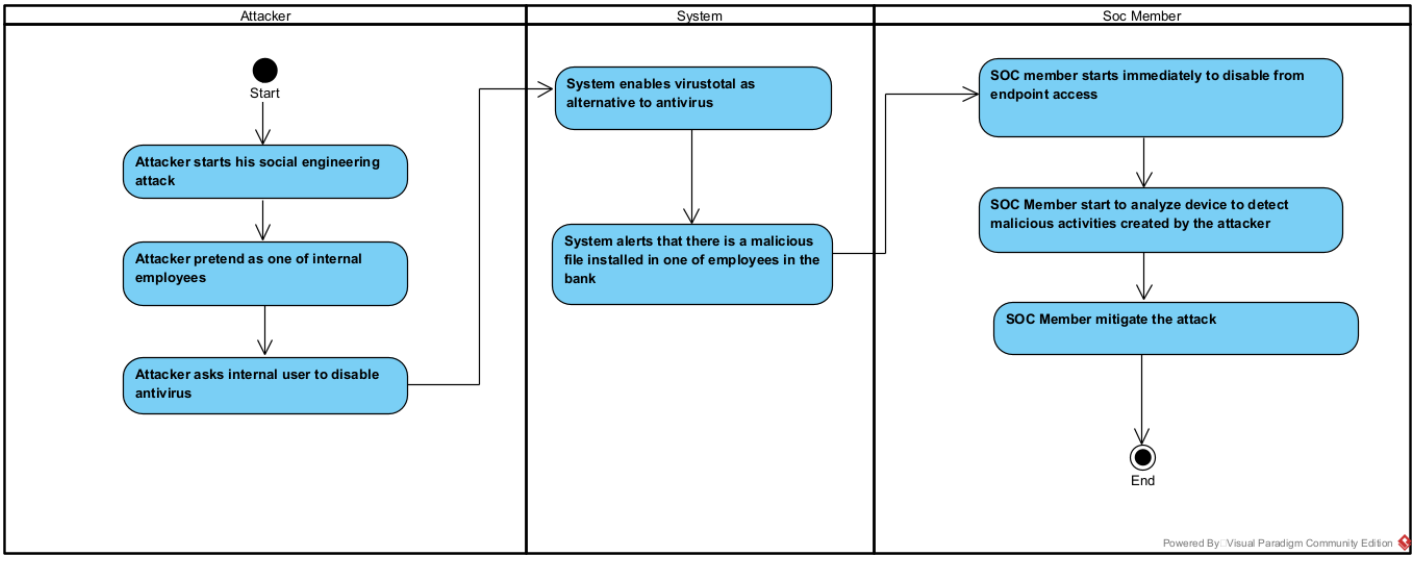
5.1.3.5 Client transfer fund to another client



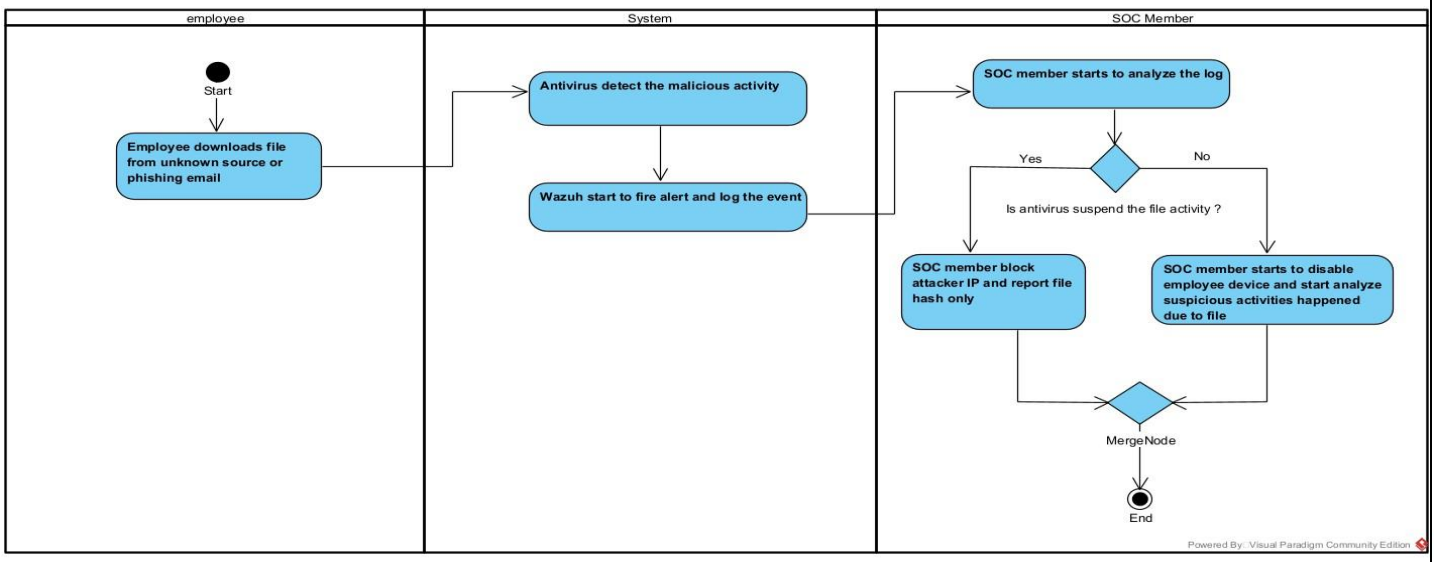
5.1.3.6 Client applies for loan



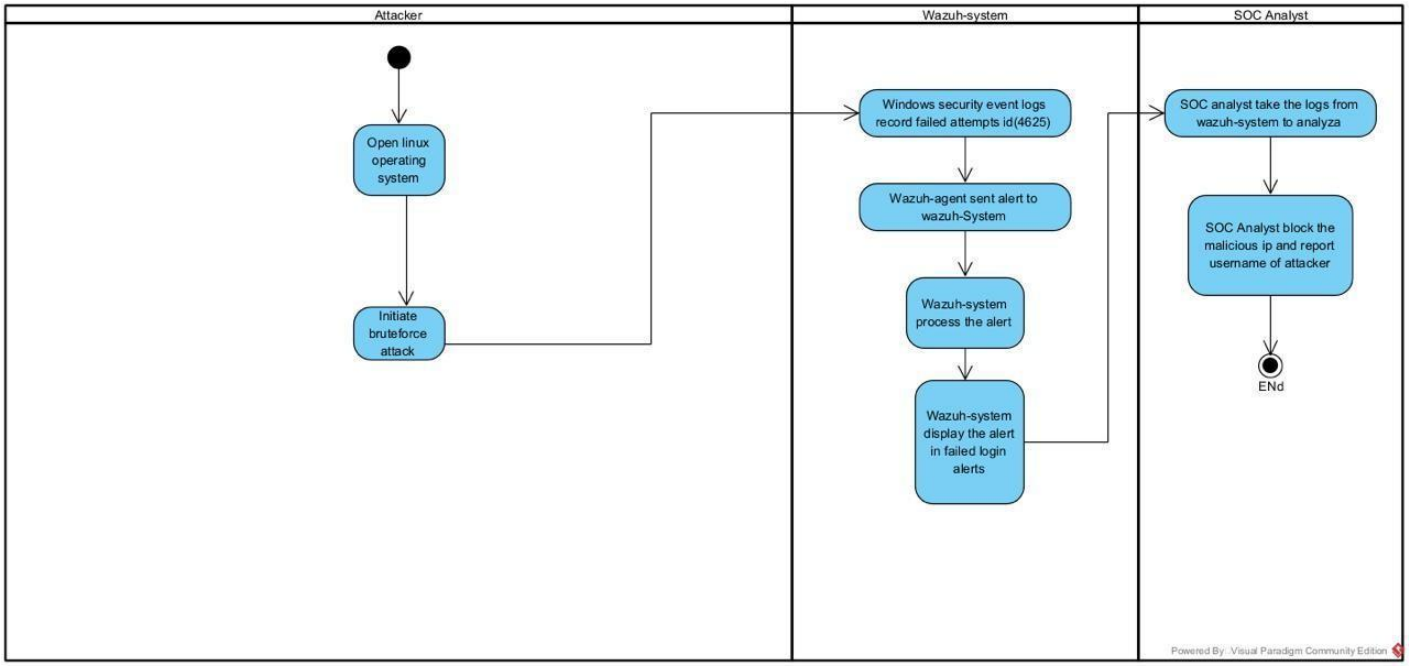
5.1.3.7 Virus total prevention to phishing and social-engineering attack



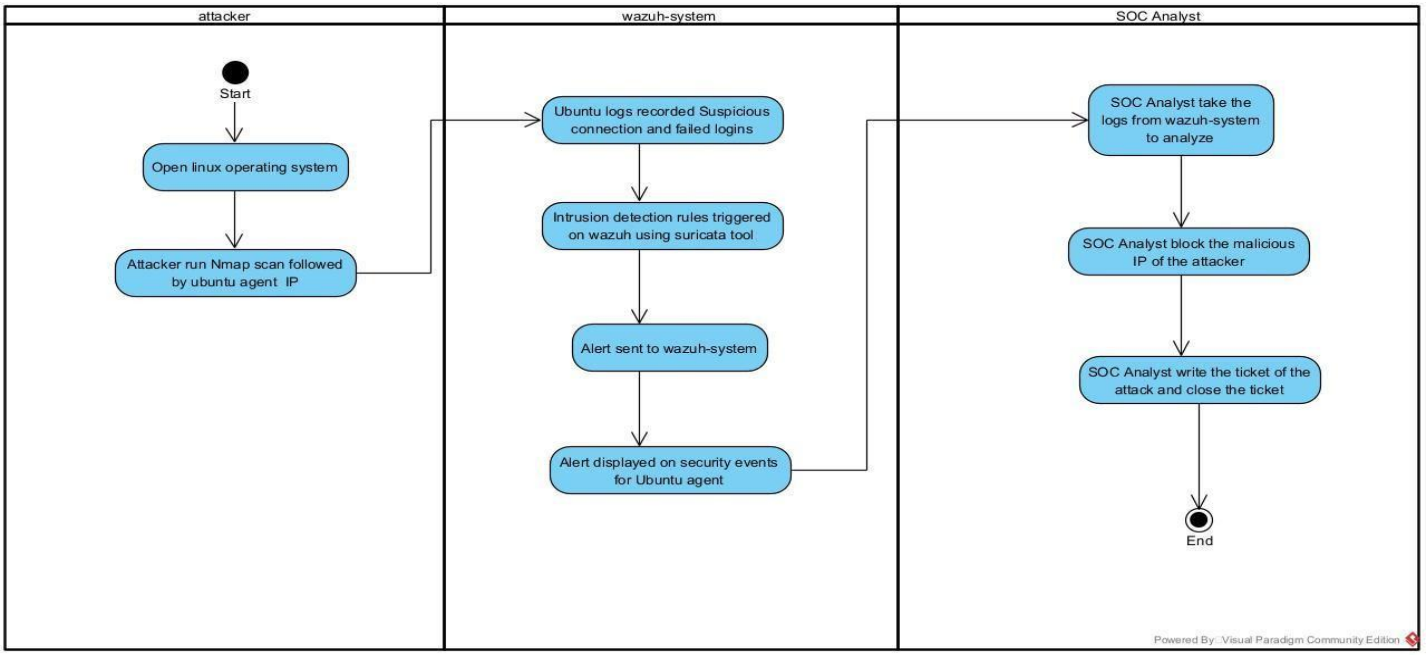
5.1.3.8 Antivirus Logs



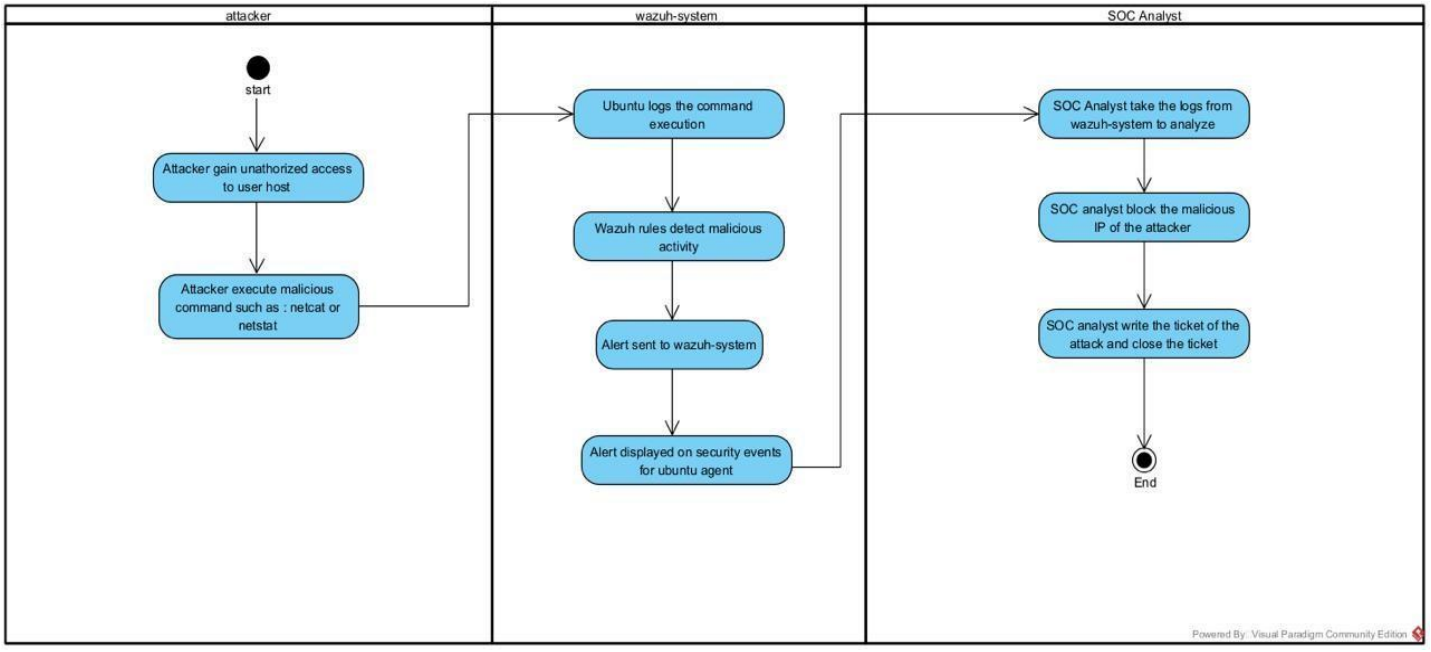
5.1.3.9 Brute-force attack prevention



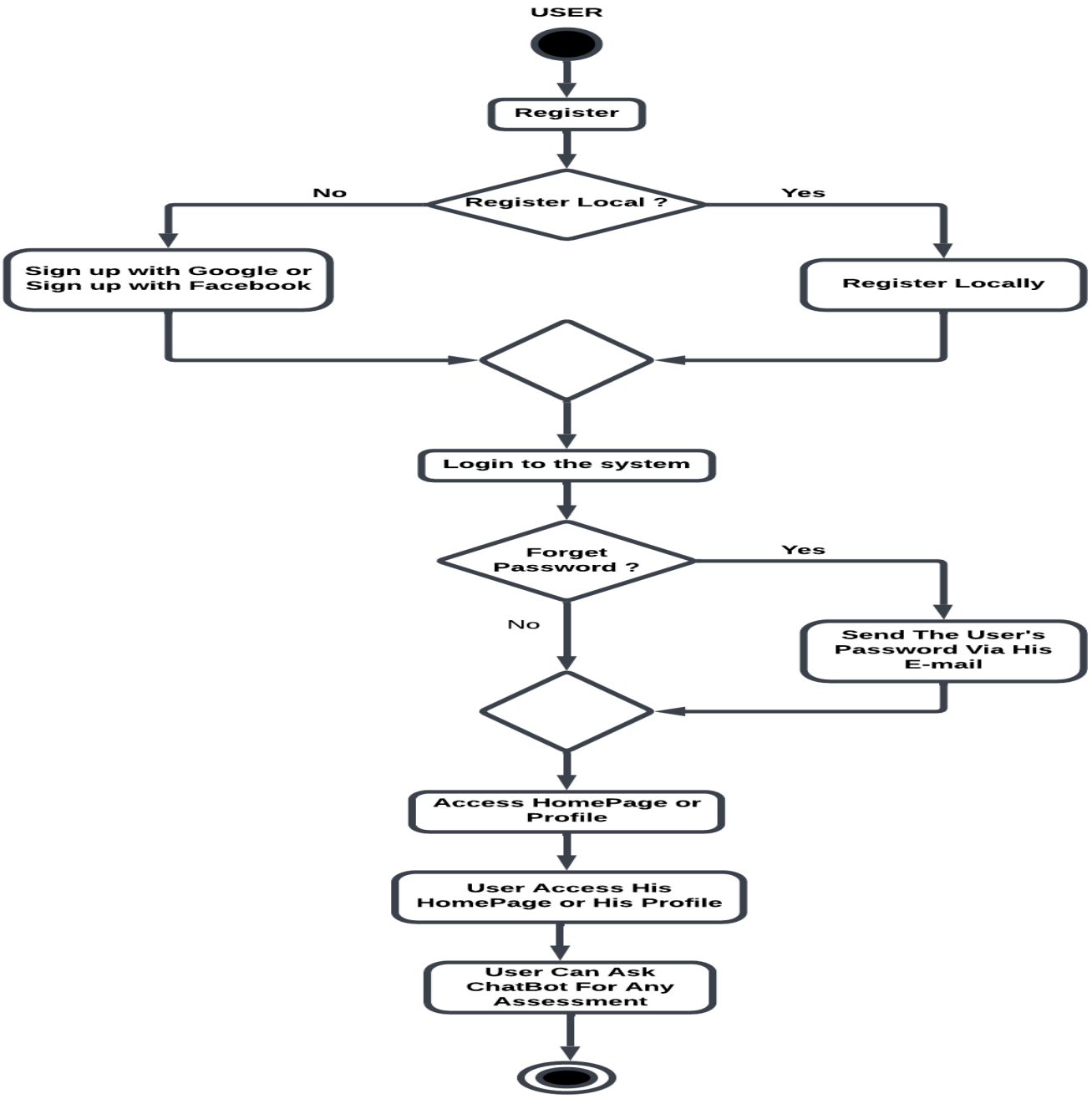
5.1.3.10 Network Intrusion attack Prevention



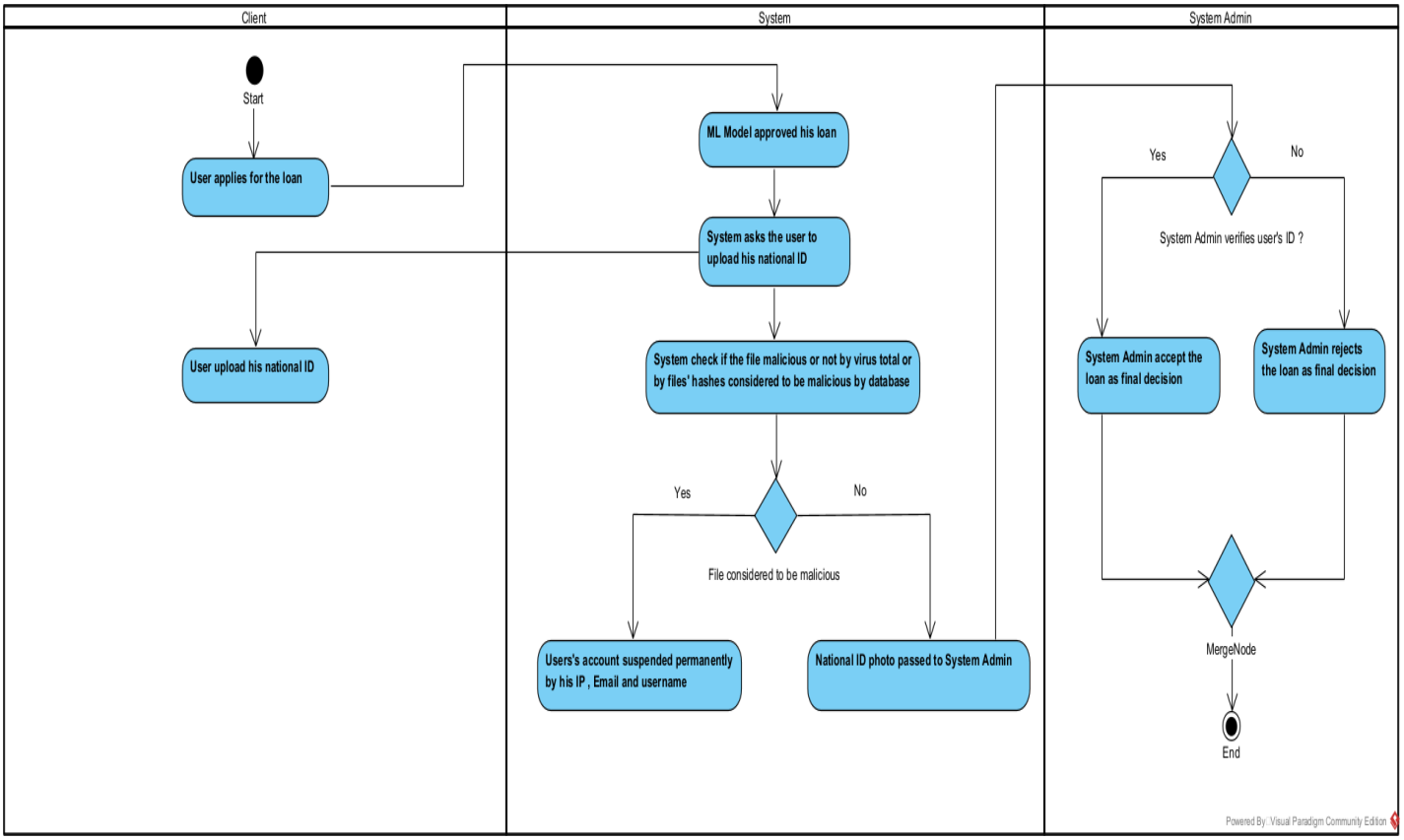
5.1.3.11 Detecting malicious commands



5.1.3.12 Client registers to the system

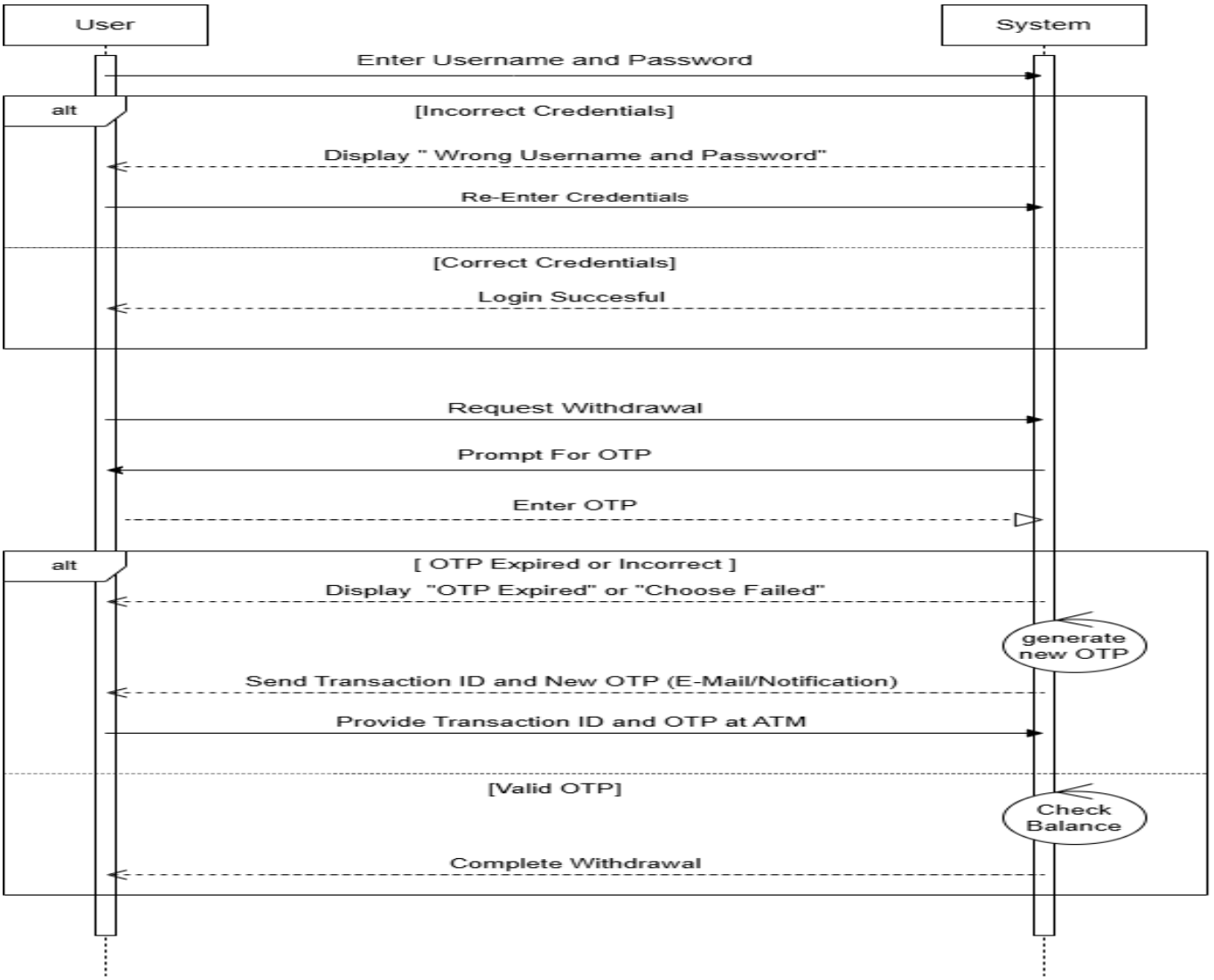


5.1.3.13 Check files' legitimacy by its hash

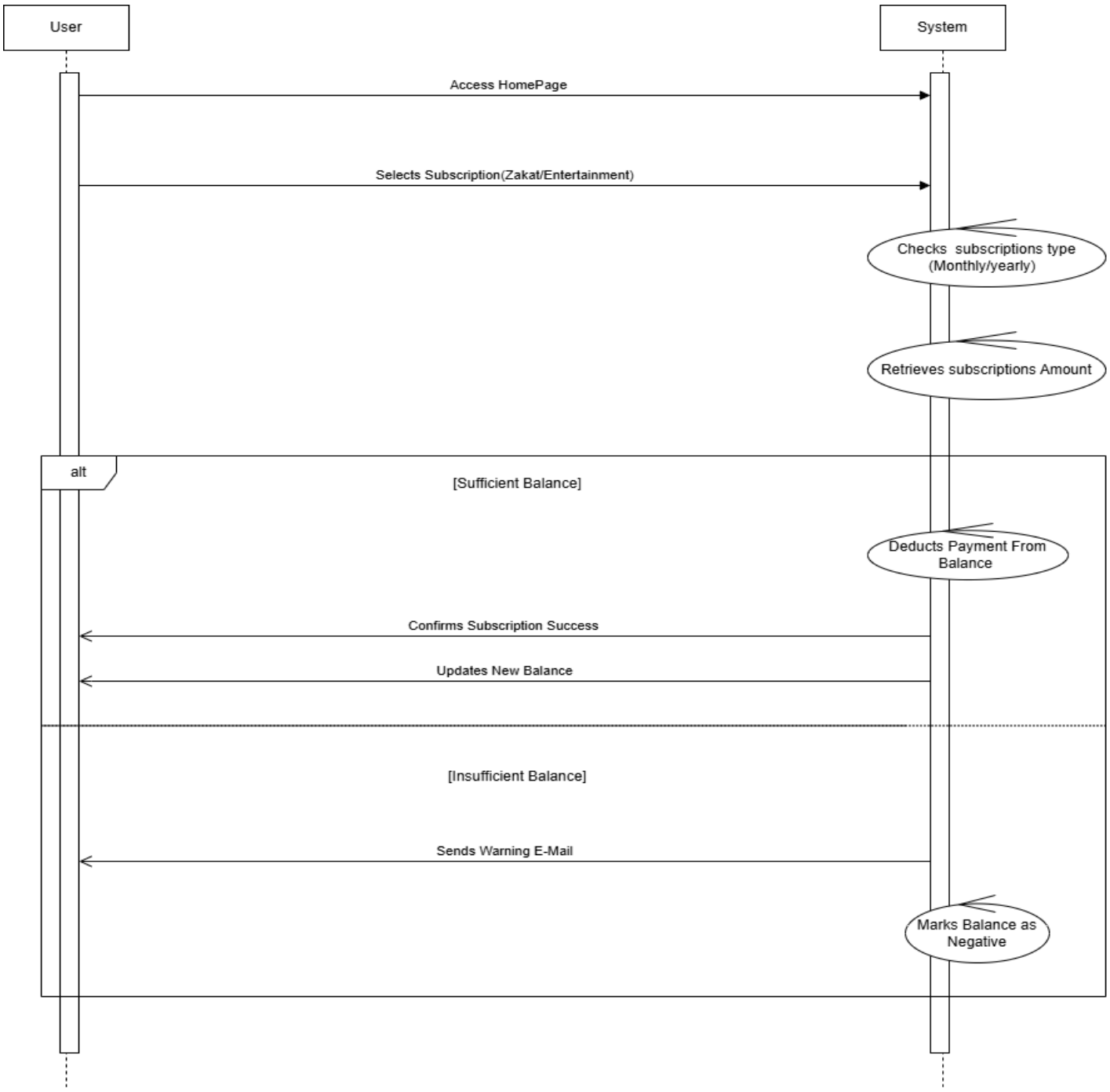


5.1.4 Sequence diagrams

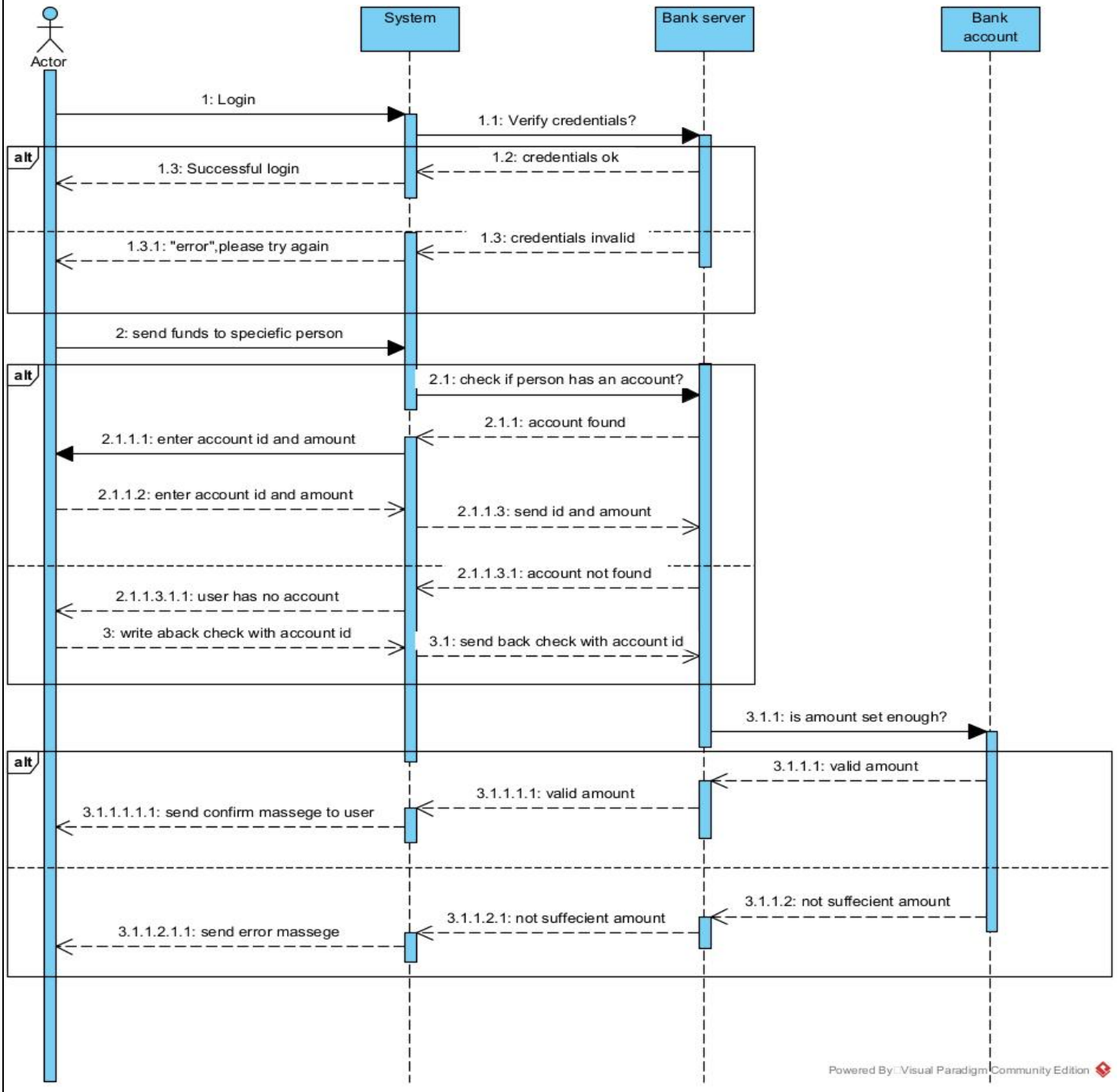
5.1.4.1 Client’s withdrawal functionality sequence



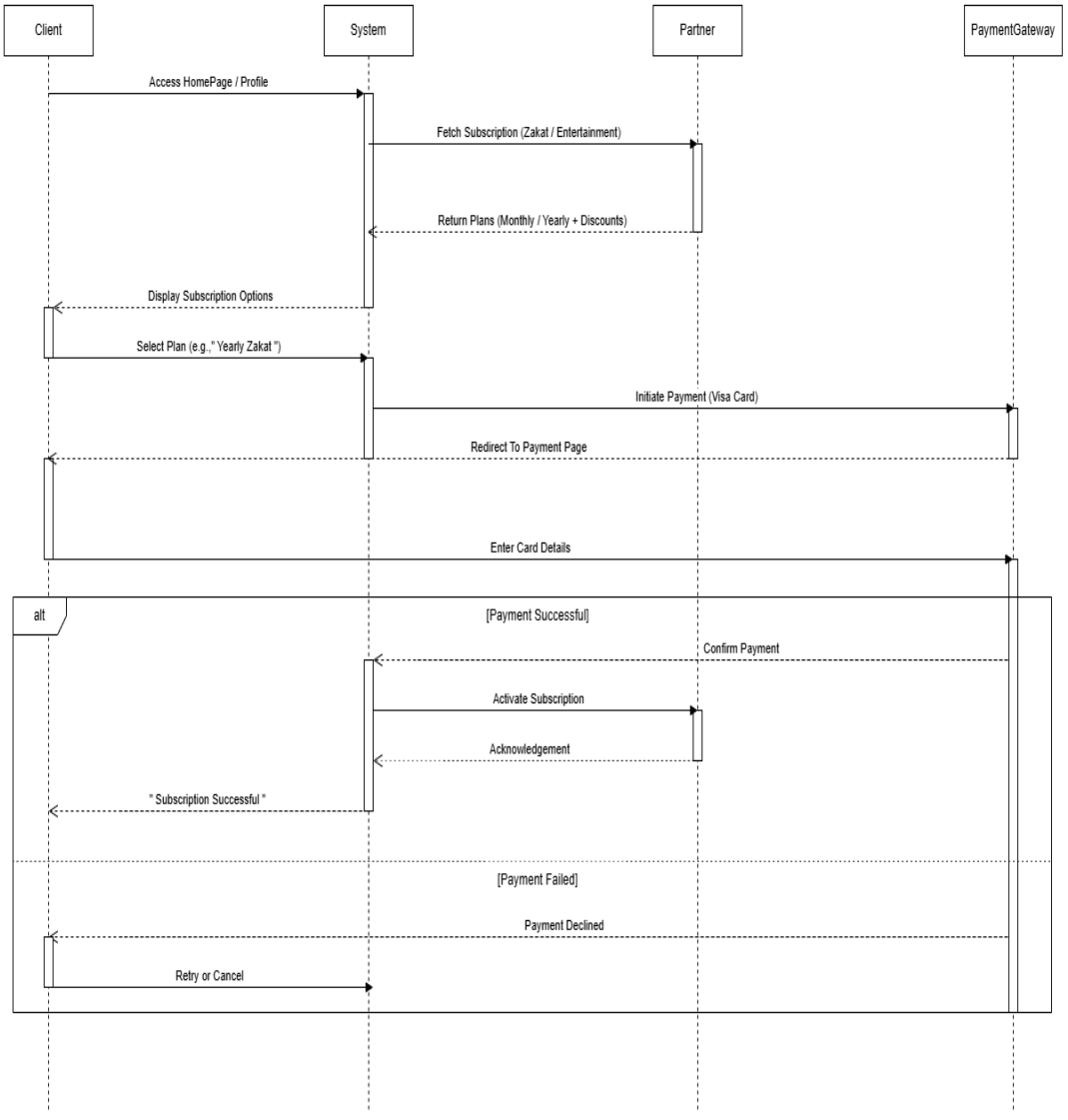
5.1.4.2 Client’s Subscriptions functionality sequence



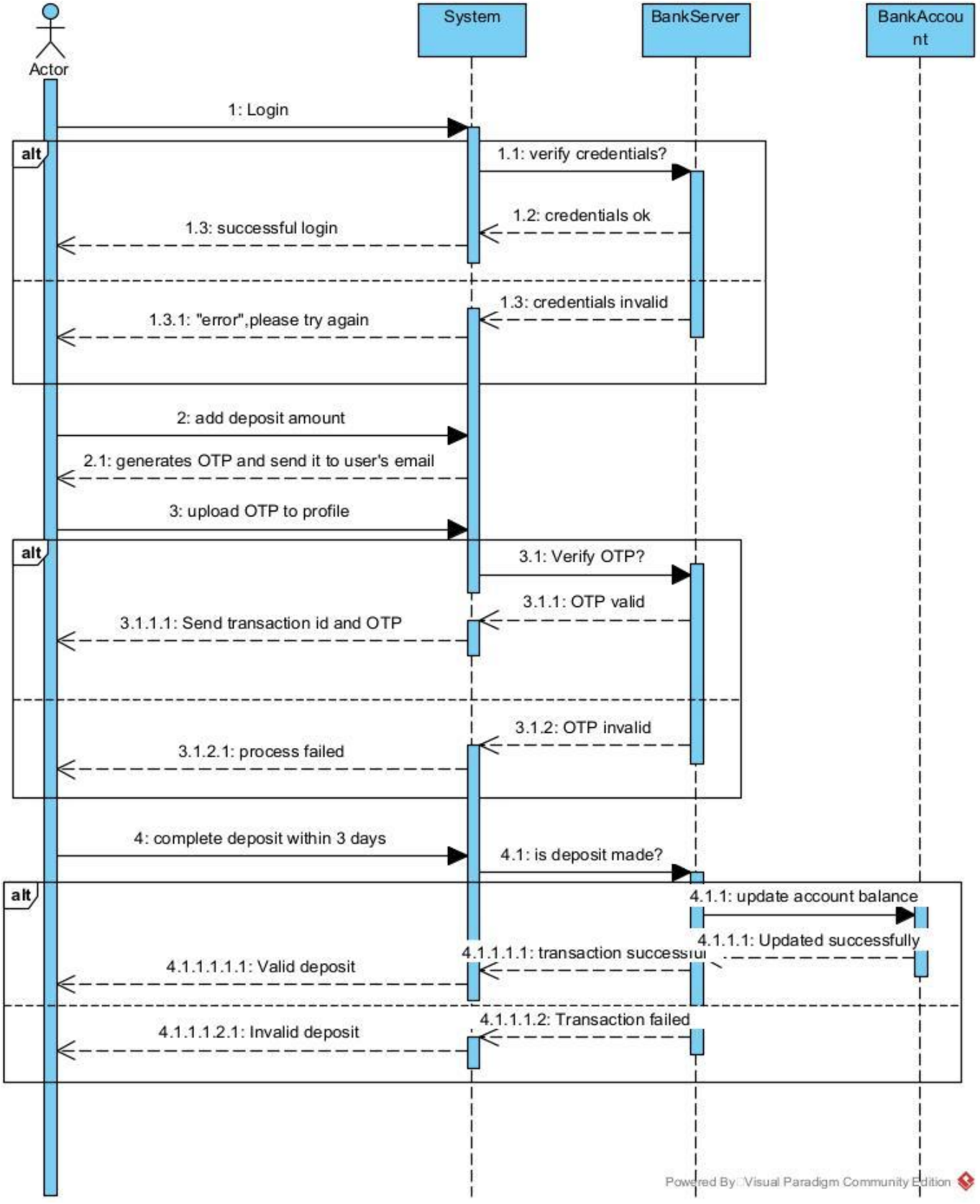
5.1.4.3 Client transfer funds functionality sequence



5.1.4.4 Client’s payment functionality sequence



5.1.4.5 Client’s deposits functionality sequence



5.2 Closer look to main security backend concern functionalities.

5.2.1 Two Factor Authentication

Two-Factor Authentication (2FA) is a security process that requires users to verify their identity using two different factors before accessing an account or system. These factors fall into three main categories:

1. Something You Know – A password, PIN, or security question.
2. Something You Have – A mobile phone, security token, smart card, or authenticator app.
3. Something You Are – Biometrics like a fingerprint, facial recognition, or retina scan.

How 2FA Works:

1. A user enters their username and password (first factor).
2. The system prompts for a second factor, such as a one-time code sent to their phone or generated by an authenticator app.
3. After entering the correct second factor, the user gains access to the account.

N.B: This is the scenario we already use.

Benefits of 2FA:

- Adds an extra layer of security, making it harder for attackers to access accounts.
- Protects against password leaks and phishing attacks.
- Reduces unauthorized access even if passwords are compromised.

Common 2FA Methods:

- SMS or Email Codes – A temporary code sent to the user's phone or email.
- Authenticator Apps (Google Authenticator, Microsoft Authenticator)
- Generate time-based one-time passwords (TOTP).
- Hardware Tokens (YubiKey, RSA SecurID) – A physical device generates authentication codes.
- Biometrics – Fingerprint, facial recognition, or retina scan.

In our project's implementation we already use two factors which are password "something you know" and sending OTP through mobile phone "something you have" as we know using two different factors falling in two different categories make your security implementation concerns used in a best practice way.

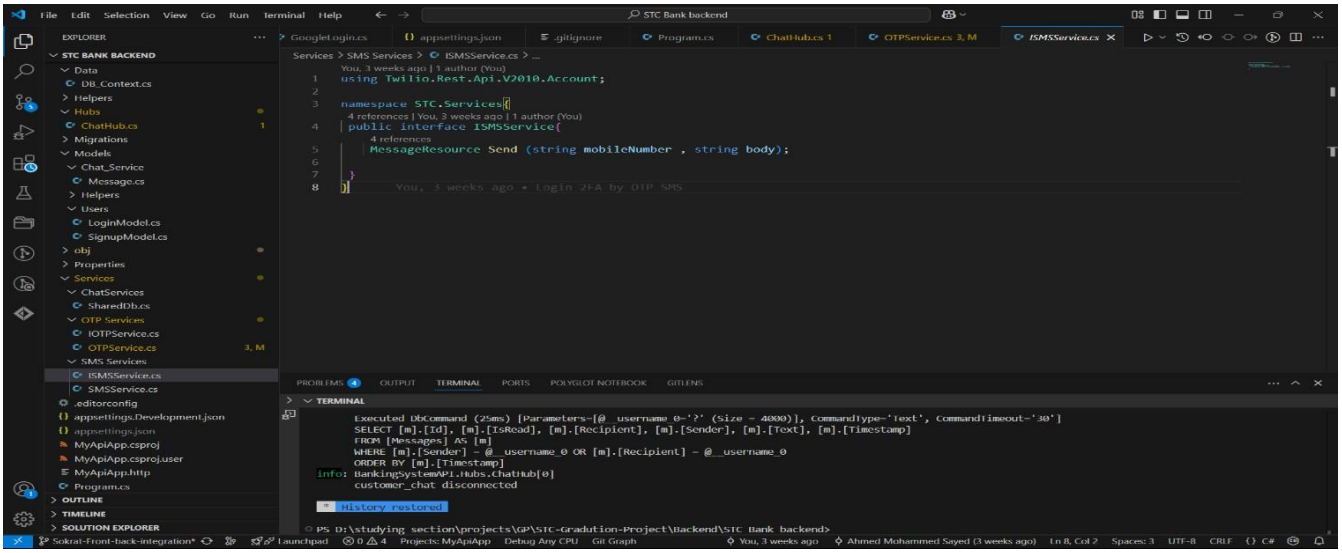


Fig [5-1] Twilio Service Interface

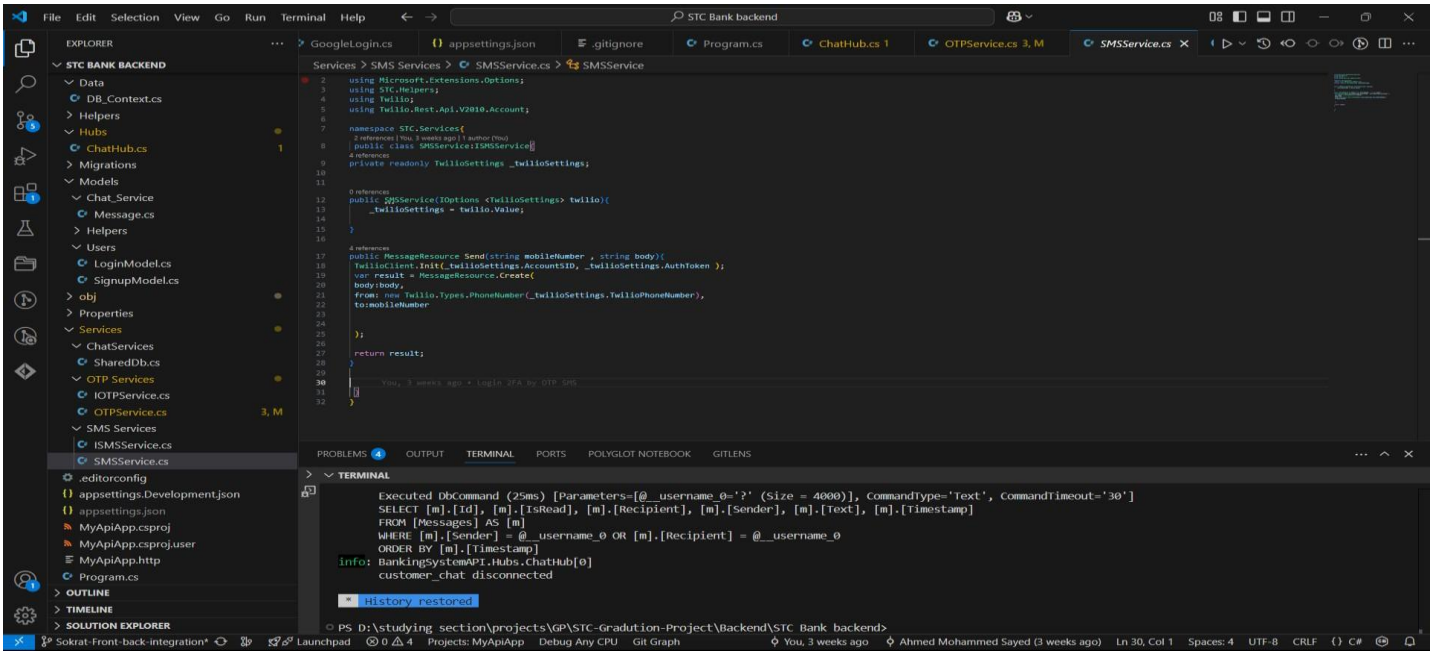


Fig [5-2] Twilio Service Implementation

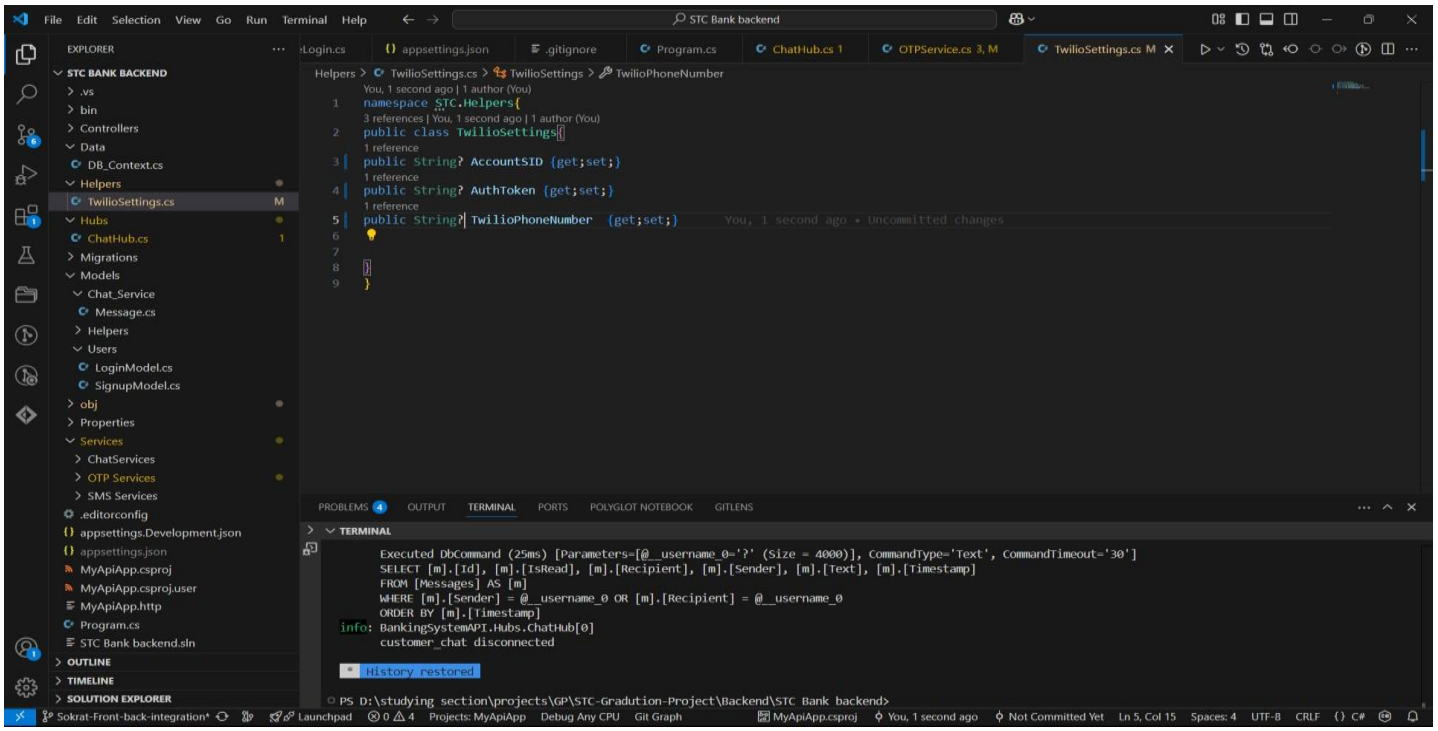


Fig [5-3] Twilio Model Implementation

```
_smsService.Send(user.PhoneNumber, $"Your OTP is {otp}. It expires in 5 minutes");
```

Fig [5-4] Login Controller implementation for sending OTP to user once correct login process.

5.2.2 One-Time Password

A One-Time Password (OTP) is a security feature that generates a unique, temporary password for user authentication. OTPs are used in two-factor authentication (2FA) and multi-factor authentication (MFA) to enhance security by preventing unauthorized access.

Types of OTPs

1. Time-Based One-Time Password (TOTP)
    - The password is valid for a short period (e.g., 30 or 60 seconds).
    - Generated using a secret key and the current time.
    - Example: Google Authenticator, Microsoft Authenticator.

2. HMAC-Based One-Time Password (HOTP)
    - The password is valid until it is used.
    - Generated using a secret key and a counter that increments after each use.
    - Example: Some banking systems use HOTP-based tokens.

How OTP Works

1. A user attempts to log in to a service.
2. The system generates an OTP and sends it via SMS, email, authenticator app, or hardware token.
3. The user enters the OTP within the valid timeframe.
4. The system verifies the OTP and grants access.

Why OTP is Secure

- Unique and Temporary – Each OTP is different and expires quickly.
- Prevents Password Reuse – Even if intercepted, the OTP cannot be reused.
- Phishing and Man-in-the-Middle Attack Resistance – More secure than static passwords.

Our Implementation

We use a GUID (Globally Unique Identifier) is a 128-bit identifier used to uniquely distinguish objects, records, or entities in computing systems. GUIDs are widely used in databases, software development, and security applications to

generate OTP and we remove hyphens from it to send it as a naturally sequence six numbers to easily used by the user and we linked it with timestamp which is 5 minutes that can be used during this period “TOTP”.

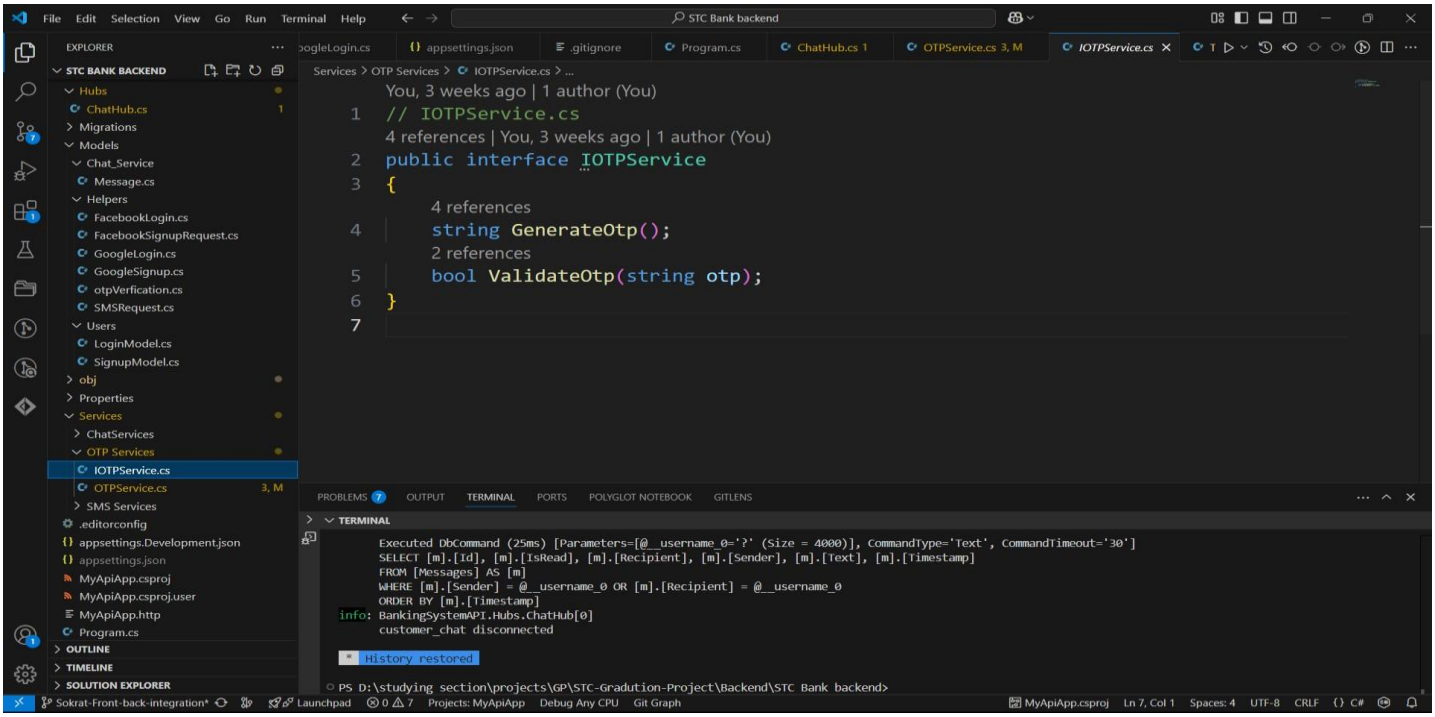


Fig [5-5] OTP Service Interface.

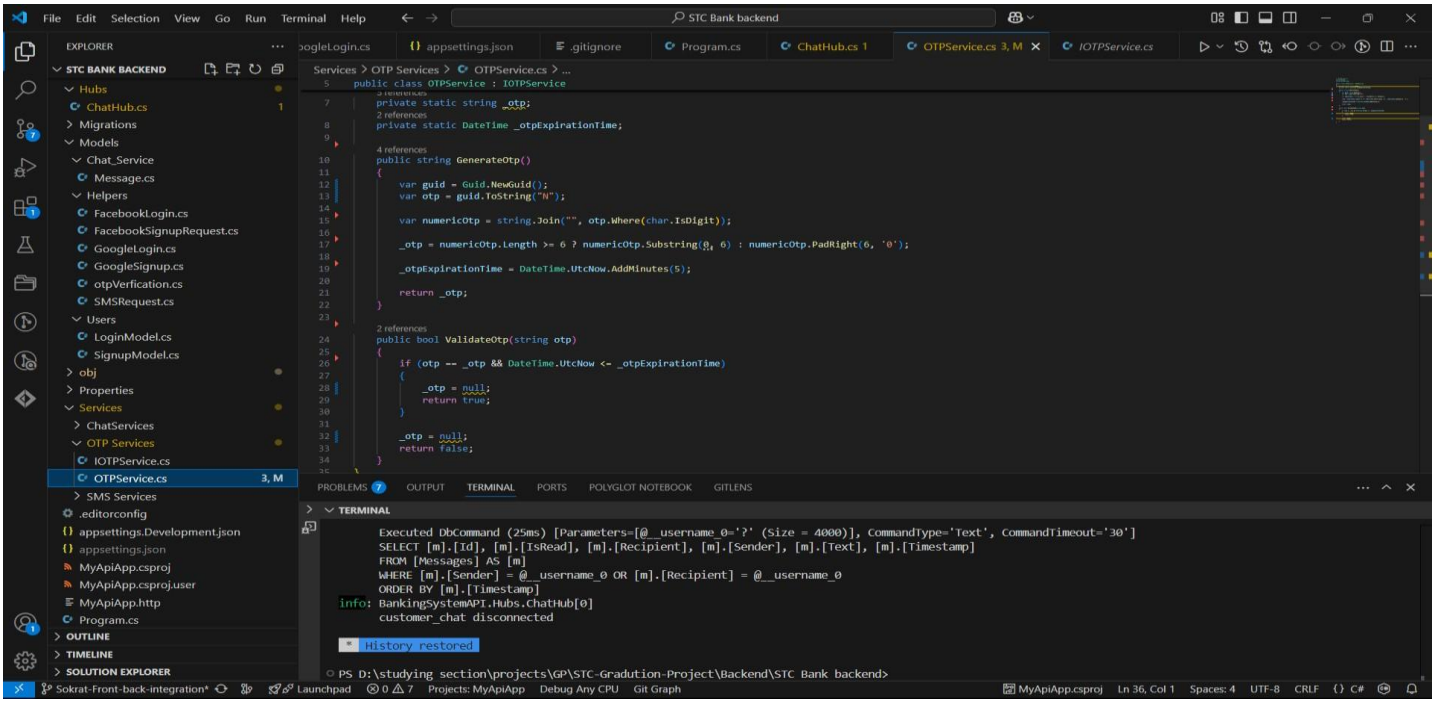


Fig [5-6] OTP Service Implementation using GUID without hyphens

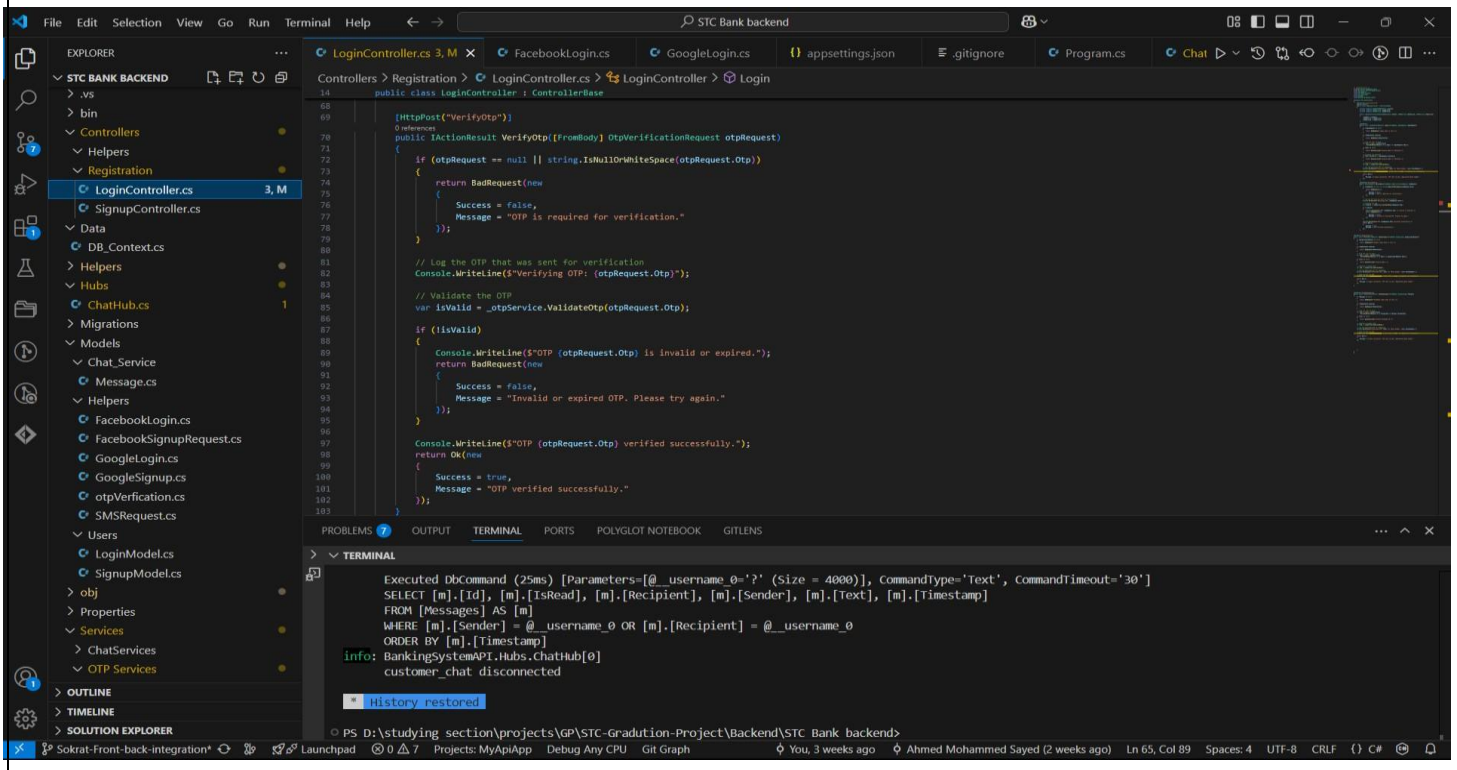


Fig [5-7] Check If TOTP Expired/Valid or not

5.2.3 VirusTotal Malicious File detector integration

Integrating VirusTotal with a .NET website provides an effective way to detect malicious files uploaded by users. When someone uploads a file, the system first calculates its hash (such as SHA-256) and checks it against VirusTotal's database to see if it has been previously identified as harmful. If the file is unknown, the system can upload it to VirusTotal for a full scan. Based on the results, the application can decide whether to allow the file, flag it as suspicious, or block it entirely. This process helps protect the website from malware and ensures that only safe files are accepted, adding an important layer of security to any file upload feature.

5.2.3.1 Implementation

We have implemented VirusTotal's services using an API key to analyze files uploaded to our website, especially when users submit a national ID image as part of the loan approval process. Before storing any file in the database, we generate its hash using the SHA-256 algorithm. Although SHA-256 is considered deprecated for secure password storage, it is still widely used by VirusTotal for file comparison purposes. If the uploaded file is detected as malicious, the system blocks the upload and automatically suspends the user. While we use SHA-256 for file integrity checks, we understand it is not suitable for secure password hashing (Deprecated). For passwords, we currently use salted MD5, though more secure alternatives like bcrypt or Argon2 are generally recommended give us number of the file submitted, it's IOCs (**Indicators of Compromise**) and number of vendors detect it a malicious one implementation shown in Fig [5-8] , Fig [5-9] and Fig [5-10]

Note

Remember that VirusTotal API integration is limited
We recommend you get paid service if you want
To apply this code in a sensitive real-time service in
Your enterprise.

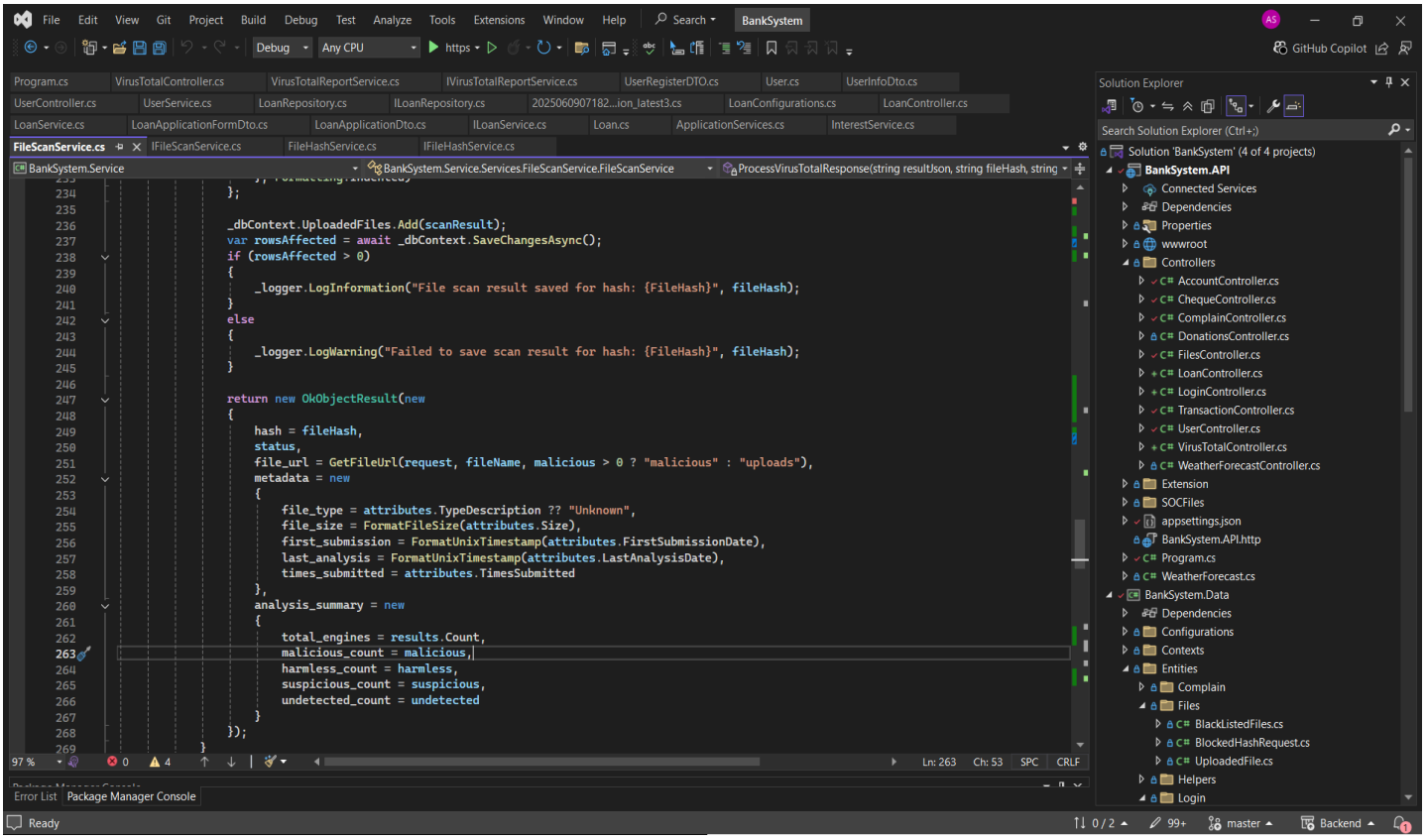


Fig [5-8] VirusTotal implementation indicating IOCs, file's status and last analysis result.

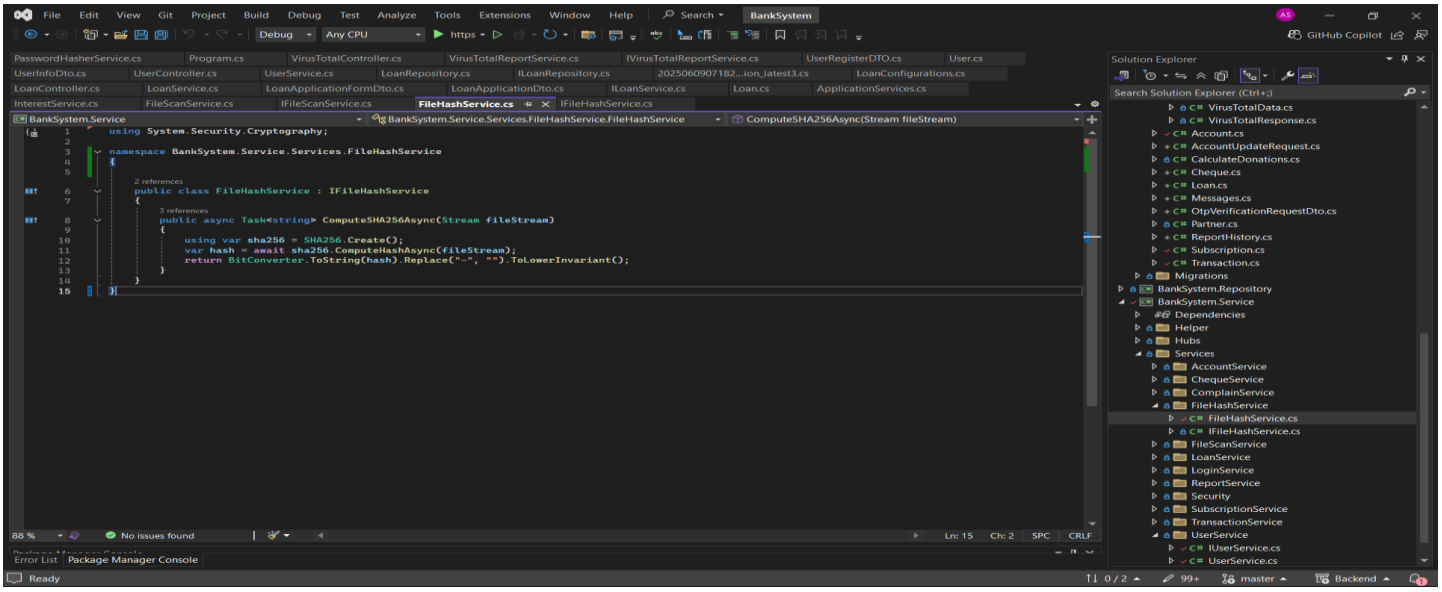
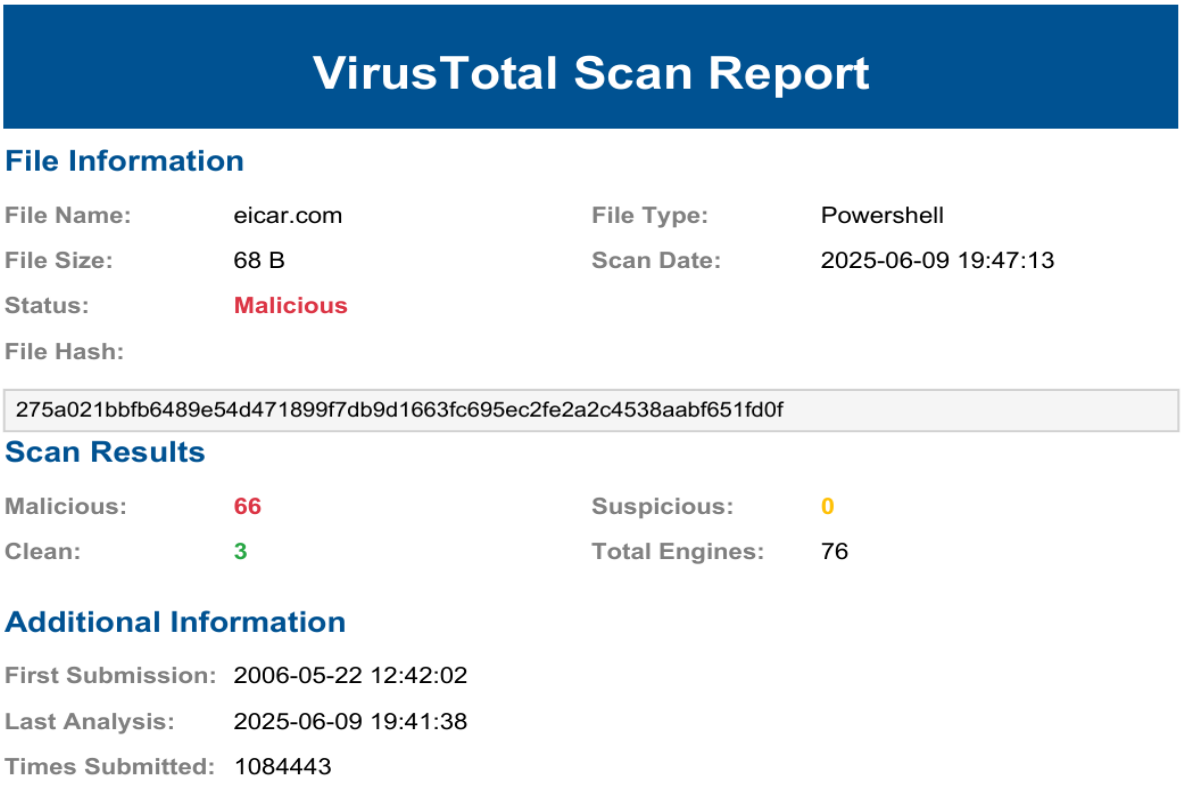


Fig [5-9] File's SHA-256 hash implementation



VirusTotal Scan Report

File Information

File Name:	eicar.com	File Type:	Powershell
File Size:	68 B	Scan Date:	2025-06-09 19:47:13
Status:	Malicious		

File Hash:

275a021bbfb6489e54d471899f7db9d1663fc695ec2fe2a2c4538aabf651fd0f

Scan Results

Malicious:	66	Suspicious:	0
Clean:	3	Total Engines:	76

Additional Information

First Submission:	2006-05-22 12:42:02
Last Analysis:	2025-06-09 19:41:38
Times Submitted:	1084443

Fig [5-10] VirusTotal's backend report indicated most important file IOCs

5.2.4 JSON Web Tokens (JWT), Authorization, Authentication, hashing and Identity

In modern .NET applications, authentication and authorization are essential components of secure system design. Authentication is the process of verifying a user's identity, typically through a username and password, while authorization determines what actions an authenticated user is allowed to perform. To enhance security, .NET uses password hashing—transforming passwords into irreversible strings using algorithms like PBKDF2 or SHA-256 combined with salting—to prevent storing passwords in plain text. For stateless and scalable authentication, .NET commonly implements JSON Web Tokens (JWT). When a user successfully logs in, the server generates a JWT containing encoded claims such as user ID and roles. This token is then sent to the client, which includes it in the header of each request. The server verifies the token's signature and claims to authenticate the user and authorize access to specific resources. Role-based authorization can be enforced using attributes like [Authorize (Roles = "Admin")], restricting endpoints to users with roles. JWTs are configured in the application using middleware settings that define validation parameters, signing keys, and token lifetimes. Together, hashed passwords and JWT-based authentication provide a secure and efficient foundation for managing access control in .NET web applications.

Implementation: -

In .NET, authentication and authorization are implemented using built-in services and middleware. For authentication, user passwords are hashed using the Password Hasher<TUser> class, which applies strong hashing algorithms with salting before storing them in the database. During login, the input password is verified against the hashed value. Once authenticated, a JSON Web Token (JWT) is generated using the JwtSecurityTokenHandler class, which encodes user information (claims) and signs it is using a secret key.
To implement JWT-based authentication, the AddAuthentication() and AddJwtBearer() methods are configured in Program.cs or Startup.cs, setting token validation parameters like the signing key and expiration. The generated JWT is sent to the client, which includes it in the Authorization header on each request. On the server side, middleware automatically validates the token and makes the user's identity available.

For authorization, attributes like [Authorize] or [Authorize (Roles = "Admin")] are used to restrict access to controllers or actions based on the user's role or claims. This setup allows developers to easily manage who can access what parts of the application in a clean and secure way and this shown in Fig [5-11] and Fig [5-12]

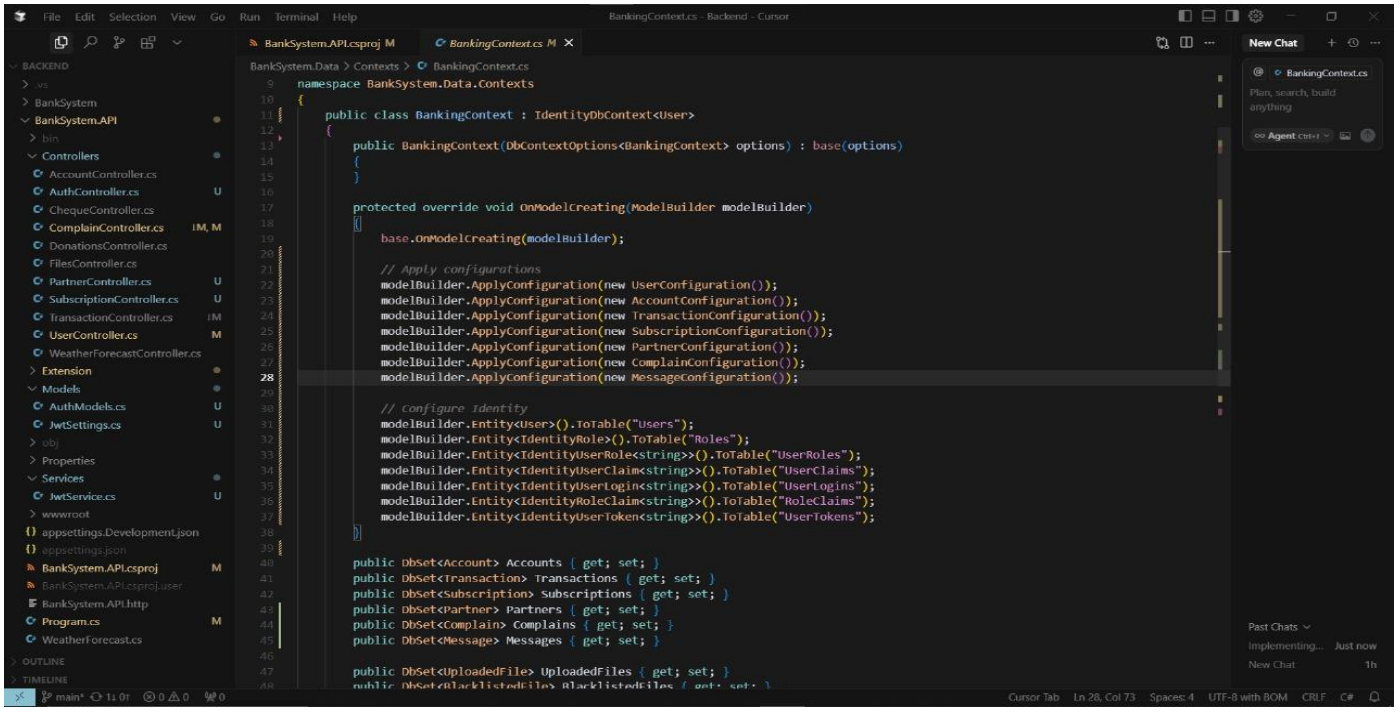


Fig [5-11] Identity Context that show user role-based authorization, Claims, JWT

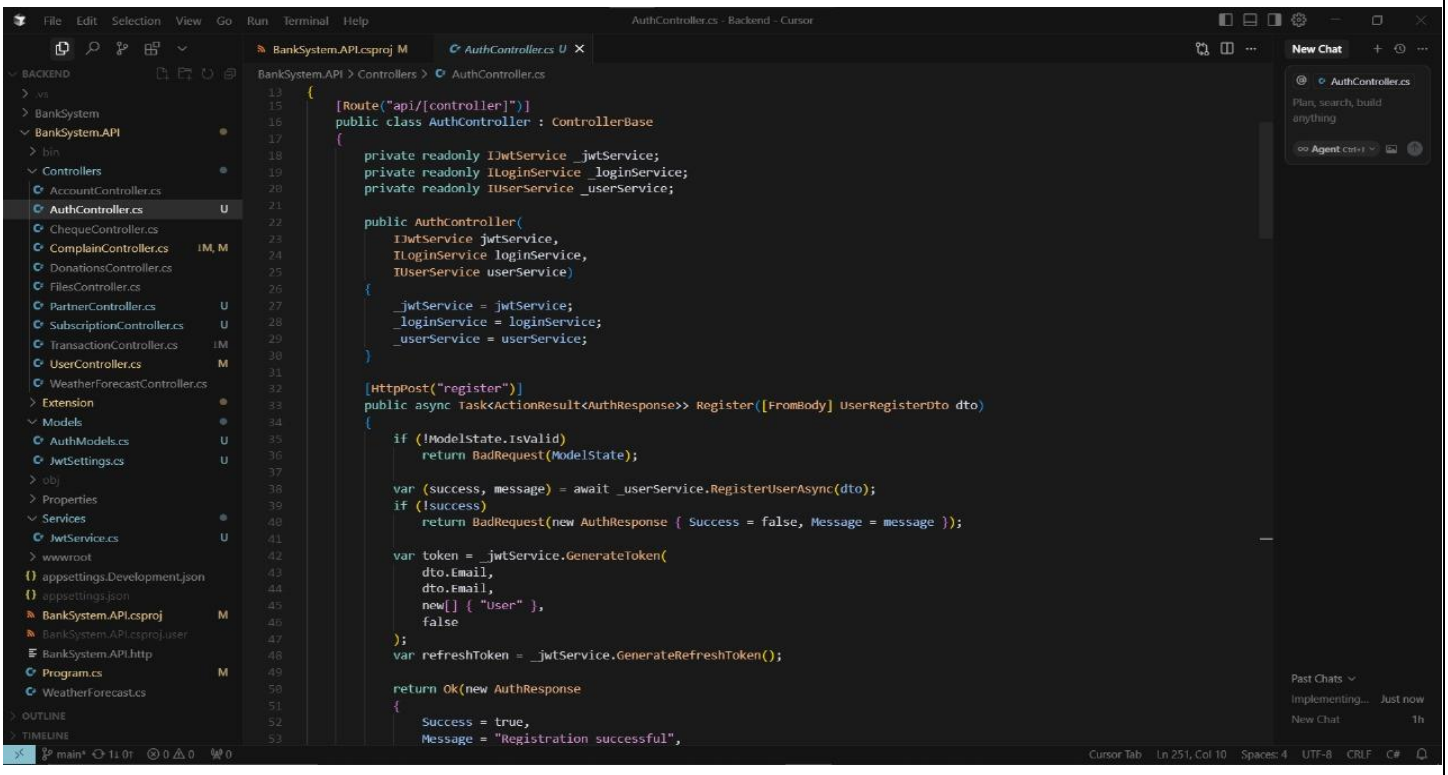


Fig [5-12] Identity JWT implementation for tokens and refresh tokens to identify user roles.

5.3 Additional backend concerns taken into considerations

5.3.1 Hybrid backend architecture (Clean architecture + N-tier architecture)

Our project adopts a hybrid architecture by combining **Clean Architecture** with the **N-Tier Architecture** model to ensure modularity, scalability, and maintainability. The solution is structured into clearly defined tiers: the **Presentation Layer** (ASP.NET Core Web API), the **Service/Application Layer**, the **Domain/Data Layer**, and the **Infrastructure Layer**. At the top, the Presentation Layer is responsible for handling incoming HTTPS requests from the frontend and acts as the system's entry point. This layer does not contain any business logic but delegates operations to the Service Layer via **Dependency Injection (DI)**. The Service Layer acts as the brain of the application, encapsulating the core business logic, use cases, and workflows. It processes input data, applies validations, and enriches it with additional metadata such as user context, timestamps, or status flags. This processed data is then passed to the Domain/Data Layer, which defines the application's entities and data transfer objects (DTOs), and is finally handled by the Infrastructure Layer, which manages interactions with databases, file systems, and third-party services like VirusTotal. By integrating the principles of Clean Architecture with the tiered separation of N-Tier Architecture, our project maintains a strict direction of dependency flow (inward only), promotes interface-based communication, and adheres to the SOLID principles especially the **Dependency Inversion Principle** resulting in a testable, flexible, and maintainable system.

5.3.2 Email Service and transactions reports

To ensure compliance with international and regional banking regulations, our system implements a robust email notification service that informs users of every transaction in real time. This aligns with standards such as PSD2 Federal Financial Institutions Examination Council Guidelines), FFIEC (Federal Financial Institutions Examination Council Guidelines) , and Central Bank guidelines, which mandate immediate user awareness of account activity to enhance security and transparency. The service also supports Two-Factor Authentication (2FA) by delivering OTPs for identity verification, reducing the risk of unauthorized access. Additionally, detailed transaction reports are generated in PDF format using PdfSharpCore and sent to users to support financial tracking, auditing, and regulatory audit trail requirements. This approach not only strengthens user trust but also ensures ful

compliance with data integrity and consumer protection standards and this shown in Fig [5 - 13] and transaction report shape in Fig [5-14] shows report shape.

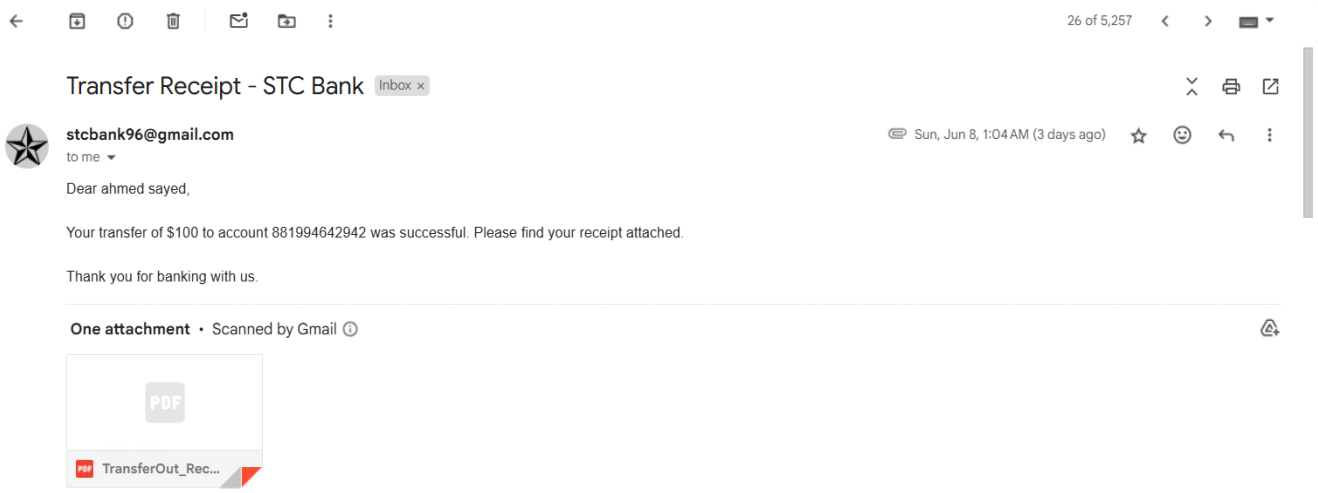


Fig [5-13] Shows email Service and transaction report service integration.

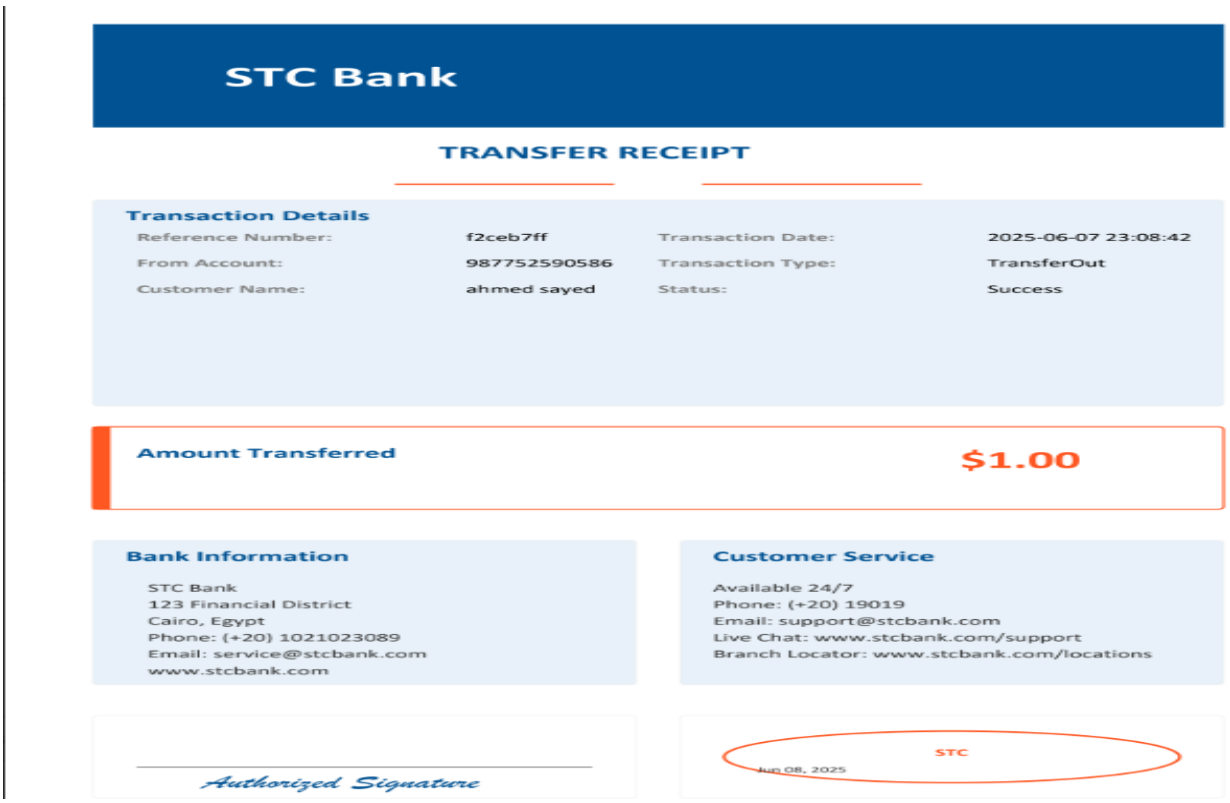


Fig [5-14] Report transfer transaction shape.

5.4 Smart Machine learning agents in action

5.4.1 Loan approval Model

5.4.1.1 preprocessing: -

we began by loading the loan approval dataset and performing several preprocessing steps to prepare the data for machine learning. First, we cleaned the dataset by removing any unnecessary whitespace from the column names and dropping the loan_id column, as it does not contribute to the prediction process. To handle categorical data, we applied label encoding to the education, self_employed, and loan_status columns, converting their string values into numeric codes that the model can understand.After ensuring the data was clean and properly formatted, we scaled the feature columns using a standard scaler. This normalization step is important because it brings all the features onto a similar scale, which can significantly improve the performance of many machine learning algorithms. We then explored the data visually, using plots to examine the distribution of loan approvals and the education levels of applicants. Pairplots and correlation heatmaps helped us understand the relationships between different features and identify any patterns that might influence loan approval decisions.

5.4.1.2 Model Implementation: -

we selected all relevant features except for the target variable, loan_status, which we aimed to predict. The data was split into training and testing sets, with 80% used for training and 20% reserved for evaluating the model's performance. We chose a Random Forest Classifier for this task, as it is well-suited for classification problems and can handle both numerical and categorical data effectively. After training the model, we evaluated its performance using several metrics. The model achieved an impressive accuracy of 97.78% on the test set, indicating that it was able to correctly predict loan approvals and rejections in most cases. We also examined the confusion matrix and a detailed classification report, which provided insights into the model's precision, recall, and F1-score for both approved and rejected loans. These metrics confirmed that the model performed well across both classes, making it a reliable tool for predicting loan approval outcomes and classification report and confusion matrix shown in Fig [5-15] and Fig [5-16].

Remember that

- Recall tells you **how many actual positives** your model correctly identified, Recall = True positives / True positive + False Negative.
- Precision measures how many of the **predicted positives** are actually correct, Precision = True positives / True positive + False Positive.

- Support is the **number of actual occurrences of each class** in your dataset.
- The F1 Score is the harmonic mean of precision and recall. It balances both false positives and false negatives, F1 Score = 2 * (precision * recall / precision + recall)
- Accuracy = Total positives / Total Negatives.

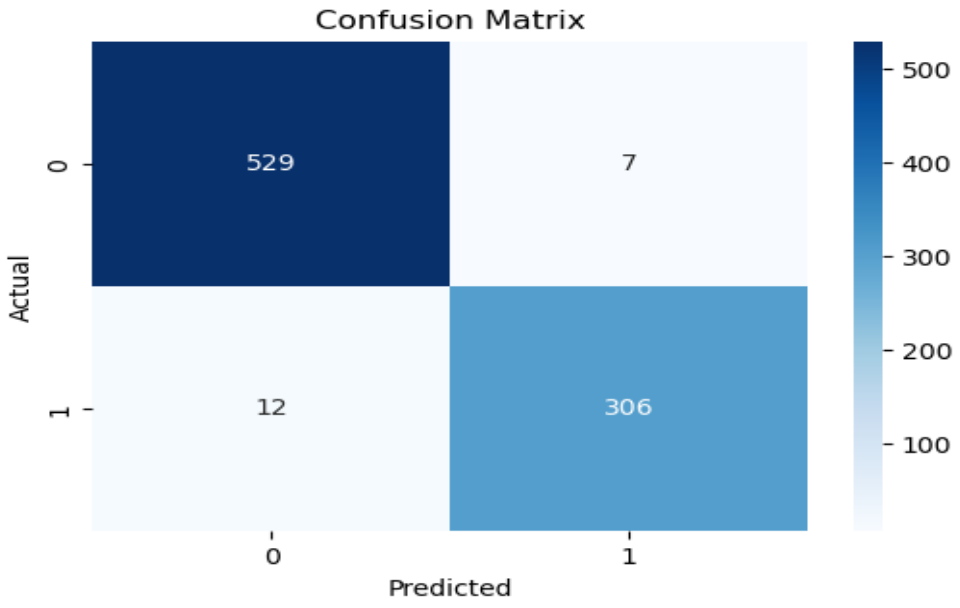


Fig [5-15] Loan Model confusion matrix.

```
Classification Report:

              precision    recall  f1-score   support

           0       0.98      0.99      0.98       536
           1       0.98      0.96      0.97       318

    accuracy                           0.98       854
   macro avg       0.98      0.97      0.98       854
weighted avg       0.98      0.98      0.98       854
```

Fig [5-15] Loan Model classification report.

5.4.1.3 Model Pros and Cons: -

Pros: - It predicted loan approval well with good accuracy, support and F1 score .

Cons: - It biased and depend on CIBIL_Score (Score measure trustfulness of the user range from 300 to 900) so we mitigate this by giving the user average score at his first loan and make minimum loan amount 1000 dollar.

5.4.2 Malicious URL detector

5.4.2.1 preprocessing: -

The preprocessing stage involved several crucial steps to prepare the dataset for machine learning. First, the Domain column was removed because it acted purely as an identifier and did not contribute meaningful information for classification. Following this, the dataset was shuffled using random sampling (frac=1) and index resetting to ensure that the training and testing sets would have a representative and unbiased distribution of both benign and malicious URLs. After shuffling, the features (X) and target labels (y) were separated. The Label column was used as

the output class (1 for malicious, 0 for benign), while the remaining columns represented various URL characteristics and were used as input features. These preprocessing steps ensured the data was clean, well-distributed, and ready for training a machine learning model.

5.4.2.2 Model Implementation: -

For the model implementation, the dataset was first split into training and testing subsets using an 80:20 ratio via the train_test_split function from Scikit-learn, ensuring that model performance could be evaluated on unseen data. An XGBoost Classifier was chosen for this task because of its high accuracy, speed, and capability to handle structured tabular data effectively. The model was initialized with specific hyperparameters including a learning_rate of 0.4 and a max_depth of 7, which control how quickly the model learns and the complexity of each decision tree respectively. The classifier was then trained using the training data (X_train, y_train) through the fit() method. After training, the model was saved using the joblib library and exported to a file named xgb_model.pkl, allowing it to be reused later for making predictions or deploying it as part of an API and measures of the model and confusion matrix shown in Fig [5-16] and ROC (Receiver Operating Characteristic Curve.) shown in Fig [5-17].

5.4.2.3 Model features meaning: -

The model uses 16 input features derived from various characteristics of the URLs to determine whether they are malicious or benign. **Have_IP** indicates whether the URL contains a raw IP address rather than a traditional domain name, which is often a sign of phishing. **Have_At** checks for the presence of the "@" symbol in the URL, which attackers sometimes use to mislead users into thinking the URL is safe**. URL_Length** measures the total length of the URL; unusually long URLs can be a red flag. **URL_Depth** refers to how many subdirectory levels are present in the path of the URL, with deeper structures sometimes used to mimic legitimate websites.

Redirection detects the use of multiple forward slashes (//) in the path, a common trick in phishing attempts to obfuscate the actual target. **https_Domain** checks whether the protocol used in the domain part of the URL is **HTTPS**, which is a basic indicator of site legitimacy, although not foolproof. **TinyURL** identifies if the URL is shortened using a TinyURL-like service; attackers often use these to hide the real destination. **Prefix_Suffix** looks for hyphens in the domain name, which is an unusual trait in legitimate domains but common in phishing links.

DNS_Record evaluates whether the domain has a valid DNS record; a missing record could indicate a fake or temporary domain. Web_Traffic measures the traffic rank of the website using tools like Alexa, as malicious websites often have low or no traffic. **Domain_Age** calculates how long the domain has existed; newer domains are more likely to be suspicious. **Domain_End** checks how soon the domain is set to expire—short-lived domains are often used in phishing schemes.

iFrame detects whether the website uses invisible iframe tags, a technique used to load malicious content without the user noticing. **Mouse_Over** monitors if the actual link shown changes when a user hovers their mouse over it, which can trick users into clicking harmful links. **Right_Click** identifies whether right-clicking has been disabled on the webpage—this is sometimes done to prevent users from inspecting the page source or copying URLs. Finally, **Web_Forwards** counts how many times the browser is redirected before reaching the final destination, as excessive redirection is often used to disguise malicious intent.

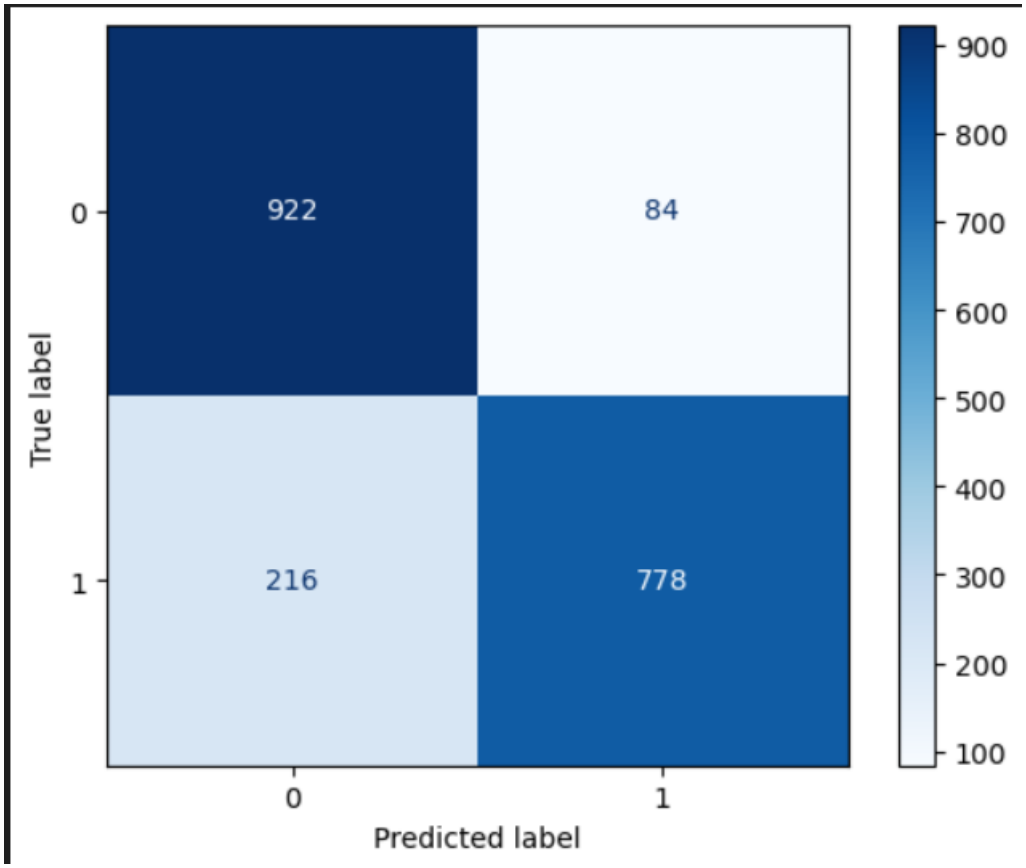


Fig [5-16] malicious URL detector agent confusion matrix.

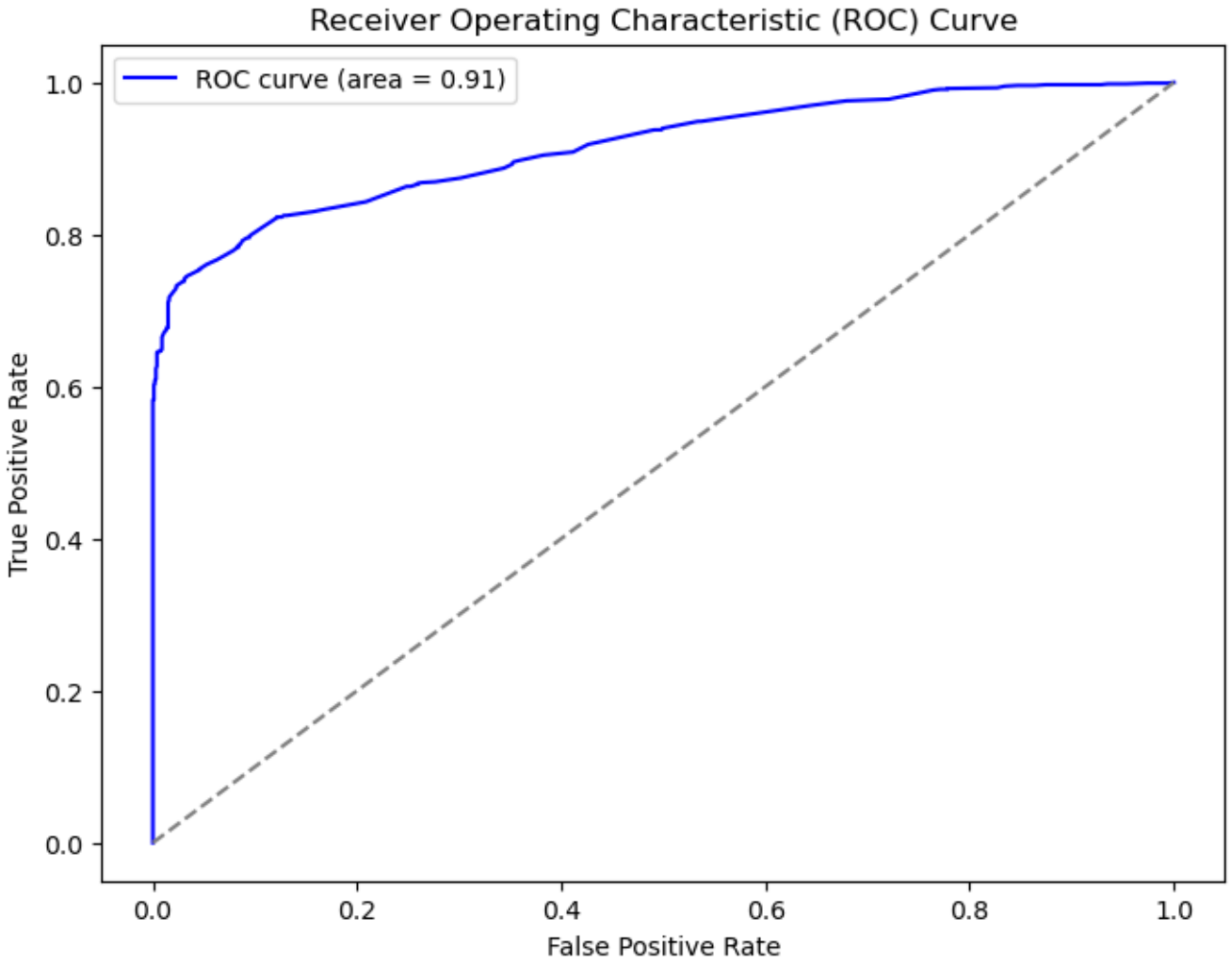


Fig [5-17] malicious URL detector ROC Curve.

5.4.2.4 Model Pros and Cons: -

Pros: - It predicted URL status well with good accuracy, support and F1 score.

Cons: - Sometimes it predict some short URLs as a malicious one due to enormous number of shortened tiny malicious links nowadays, but it’s fine because false negative is acceptable (predict it’s malicious but actually it’s benign) , but false positive is most danger cause (predict it’s benign but actually it’s malicious) . in addition to this we use VirusTotal Malicious link detector to use both so it’s totally fine.

5.4.3 Face-recognition Model (Most Important one for security concern)

5.4.3.1 Setting up environment: -

5.4.3.1.1 Enable GPU memory growth for fast implementation using CUDA installer to run model in GPU

```python
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

5.4.3.1.2 Loading many Images dataset of client's images

The dataset consists of three categories: **anchor** (reference images), **positive** (images of the same person as the anchor), and **negative** (images of different people).

5.4.3.1.2.1 Loading many Images dataset of client's images

5.4.3.1.2.2 List all .jpg images in each category (anchor, positive, negative).

```python
POS_PATH = os.path.join('data', 'positive')
NEG_PATH = os.path.join('data', 'negative')
ANC_PATH = os.path.join('data', 'anchor')
```

5.4.3.1.2.3 Take a fixed number of samples (300 in this case) to keep the dataset size manageable.

```python
anchor = tf.data.Dataset.list_files(ANC_PATH+'\*.jpg').take(300)
positive = tf.data.Dataset.list_files(POS_PATH+'\*.jpg').take(300)
negative = tf.data.Dataset.list_files(NEG_PATH+'\*.jpg').take(300)
```

5.4.3.2 Image Preprocessing: -

Images are preprocessed to ensure consistency and compatibility with the Siamese Network. The preprocessing function performs the following:

- **Decoding**: Converts .jpg files into tensors.
- **Resizing**: Adjusts images to a uniform size of 105x105 pixels. This size retains essential facial features while reducing computational load.
- **Normalization**: Scales pixel values to the range [0, 1] to improve model training stability.

5.4.3.2.1 Normalize pixel values from 255 to the range [0, 1]: -

```python
def preprocess(file_path):
    byte_img = tf.io.read_file(file_path)
    img = tf.io.decode_jpeg(byte_img, channels=3)
    img = tf.image.resize(img, (105, 105))
    img = img / 255.0

    return img
```

Python

5.4.3.2.2 The Siamese Network requires pairs of images (anchor-positive and anchor-negative) with corresponding labels (1 for positive pairs, 0 for negative pairs):

5.4.2.2.1 Combine both positive and negative files to make them in one dataset

```python
positives = tf.data.Dataset.zip((anchor, positive, tf.data.Dataset.from_tensor_slices(tf.ones(len(anchor)))))
negatives = tf.data.Dataset.zip((anchor, negative, tf.data.Dataset.from_tensor_slices(tf.zeros(len(anchor)))))
data = positives.concatenate(negatives)
```

Python

5.4.2.2.3 Apply preprocessing to both images in a pair

```python
def preprocess_twin(input_img, validation_img, label):
    return(preprocess(input_img), preprocess(validation_img), label)
```

Python

```python
res = preprocess_twin(*exampple)
```

Python

5.4.2.2.4 Shuffle the data to prevent the model from learning patterns in data order (to Avoid overfitting)

```python
data = data.map(preprocess_twin)
data = data.cache()
data = data.shuffle(buffer_size=1024)
```

Python

```python
samples = data.as_numpy_iterator()
samp = samples.next()
```

Python

5.4.2.2.5 Display image and print label and compare with other randomized sample in data

```
data = data.map(preprocess_twin)
data = data.cache()
data = data.shuffle(buffer_size=1024)
```
Python

```
samples = data.as_numpy_iterator()
samp = samples.next()
```
Python

```
plt.imshow(samp[1])
```
Python

```
samp[2]
```
Python

```
1.0
```

5.4.3.3 Model Implementation: -

The model implemented in this project is a Siamese Neural Network, specifically designed for face verification tasks. Unlike traditional classifiers that assign labels to individual inputs, a Siamese network compares pairs of images and learns to determine whether the two images belong to the same person or different people. It is particularly useful in one-shot learning scenarios, where only a few examples per class are available.

The backbone of the Siamese network is a **Convolutional Neural Network (CNN)** that serves as a **feature extractor**, also known as the embedding model. This CNN takes a face image as input and transforms it into a high-dimensional **embedding vector** that captures the unique features of the face.

Inputs

- **Shape**: (105, 105, 3)
- Represents a color image with 105×105 pixels and 3 color channels (RGB).

Convolutional Layers

The CNN consists of multiple convolutional layers that progressively extract features from the image.

1. First Convolutional Block

- **Conv2D** with 64 filters of size **10×10**, activation: **ReLU**.
- Followed by **MaxPooling2D** with pool size (2×2), used for down sampling.
- Captures **low-level features** like edges and textures.

2. Second Convolutional Block

- **Conv2D** with 128 filters of size **7×7**, activation: **ReLU**.
- Followed by **MaxPooling2D** (2×2).
- Captures **mid-level features** such as facial contours and shapes.

3. Third Convolutional Block

- **Conv2D** with 128 filters of size **4×4**, activation: **ReLU**.
- Followed by **MaxPooling2D** (2×2).
- Extracts **more abstract patterns** in the face structure.

4. Fourth Convolutional Block

- **Conv2D** with 256 filters of size **4×4**, activation: **ReLU**.
- Not followed by pooling—this is the **deepest layer**, capturing highly abstract representations.

Flattening and Dense Layer

- The output of the final convolutional layer is **flattened** into a 1D vector.
- This is passed to a **Dense layer** with **4096 units** and **sigmoid** activation.
- The result is the **embedding vector** that represents the input image in a learned feature space.

The Model architecture shown in Fig [5-18] to understand the process more.

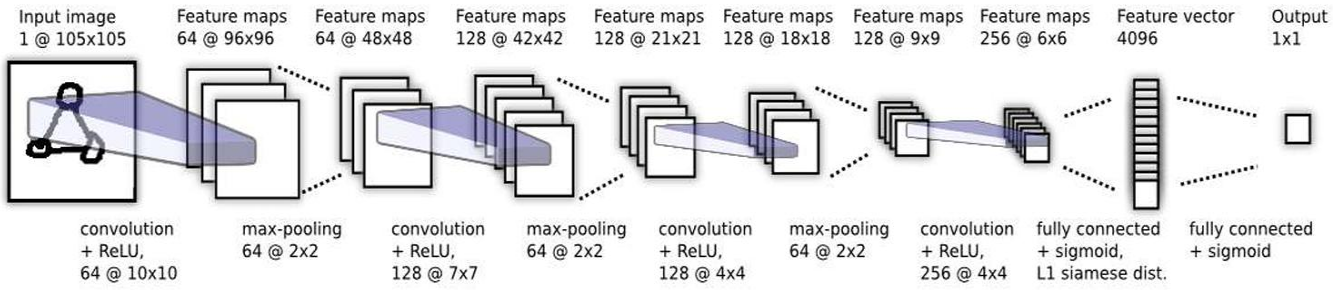


Fig [5-18] Siamese Network Model architecture

Note: -

Convolutional Layer

A convolutional layer applies small filters (e.g., 10×10) across the image to detect patterns like edges, lines, and textures. Each filter produces a feature map highlighting where a specific feature appears in the image.

ReLU Activation

The Rectified Linear Unit (ReLU) introduces non-linearity by converting negative values to zero, helping the model learn complex features beyond linear relationships.

Max Pooling

Max pooling reduces the spatial dimensions of feature maps by selecting the **maximum value** from each small region (e.g., 2×2 window). This operation:

- Reduces computational cost.
- Provides spatial invariance.
- Retains the most prominent features.

Flattening

Flattening converts the 2D feature maps into a 1D vector so that it can be fed into the fully connected (dense) layer.

Dense Layer

A dense (fully connected) layer connects every neuron from the previous layer to each neuron in the current layer. In this case, the 4096-dimensional output encodes the full face information as a feature vector.

5.4.3.4 Model final touch: -

After completing the implementation and testing phase (as illustrated in Figure 5-19), the model was converted into an executable (.exe) using PyInstaller. This executable was then deployed as a Windows service to run continuously in the background. The service is configured to trigger facial recognition scans every **5 minutes**, enhancing security by mitigating risks such as **launch-time attacks** and **piggybacking attacks**.

Key Improvements:

- **Automated Execution**: The model operates as a persistent background service, eliminating manual intervention.
- **Regular Security Checks**: Frequent scans (every 5 minutes) ensure timely detection of unauthorized access attempts.
- **Attack Mitigation**: Specifically addresses **launch-time** (delayed unauthorized access) and **piggybacking** (tailgating) threats.

This deployment approach ensures robust, real-time security monitoring while optimizing system performance.

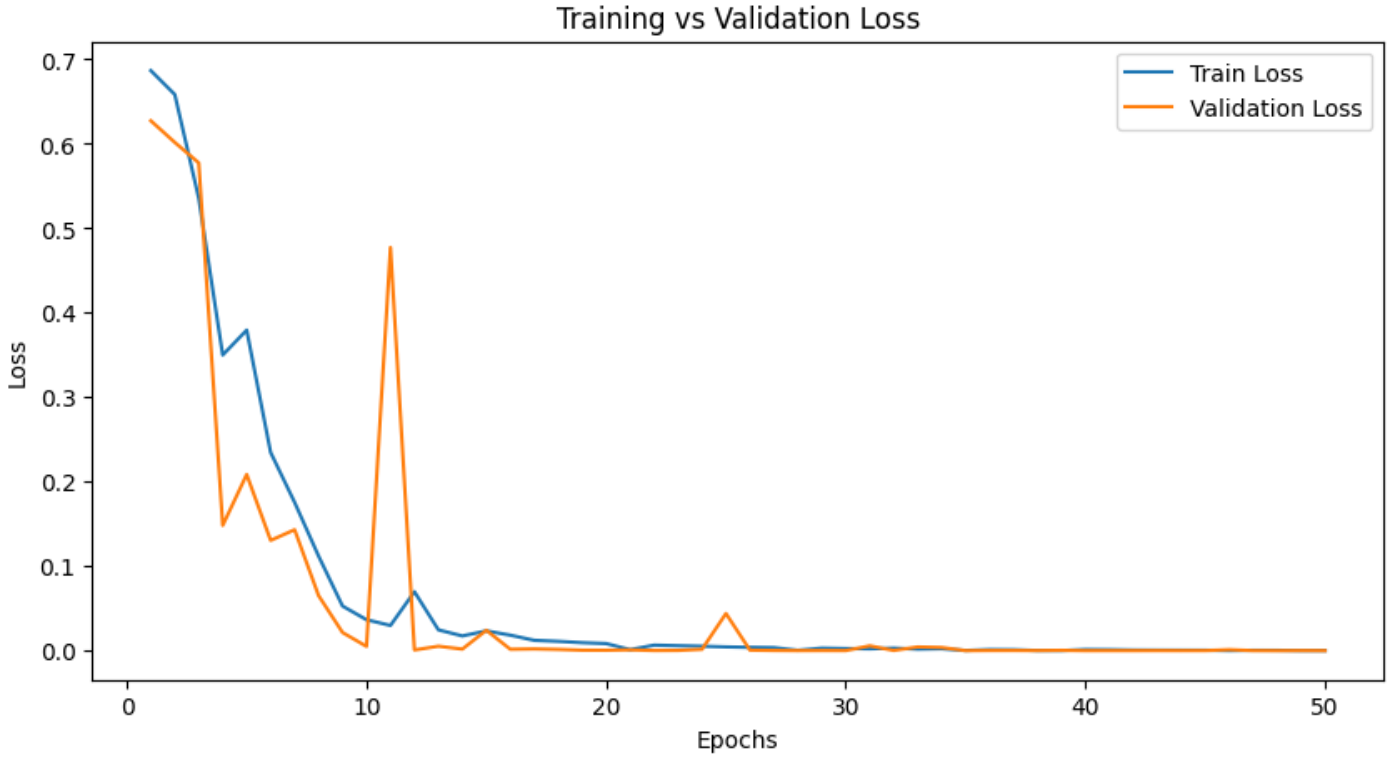


Fig [5-19] Shows loss between validation loss and testing for Siamese face recognition model

5.4.4 Another Machine learning models and Concepts in action

5.4.4.1 Credit Card Approval

We use this model to enhance user experience by allowing applicants to check their eligibility for a credit card before visiting a bank branch. The **Credit Approval Model** notebook outlines a machine learning pipeline that predicts whether a credit application is likely to be approved based on a set of applicant features. During the **preprocessing** stage, missing values were addressed to maintain data integrity, categorical variables were converted into numerical form using LabelEncoder, and numerical features were standardized using StandardScaler to ensure consistency across inputs an essential step for algorithms like RandomForestClassifier. In the **implementation** phase, the dataset was divided into training and testing sets to assess model performance, and a Logistic Regression model from scikit-learn was trained. The model's effectiveness was evaluated using accuracy metrics and a confusion matrix to distinguish between correct and incorrect classifications. Overall, the results show that the model is a strong baseline solution for preliminary credit approval checks, with opportunities for future improvements using more sophisticated models and hyperparameter tuning, it’s closer to processing of Loan classification approval with minimum change and its confusion matrix shown in Fig [5-20]

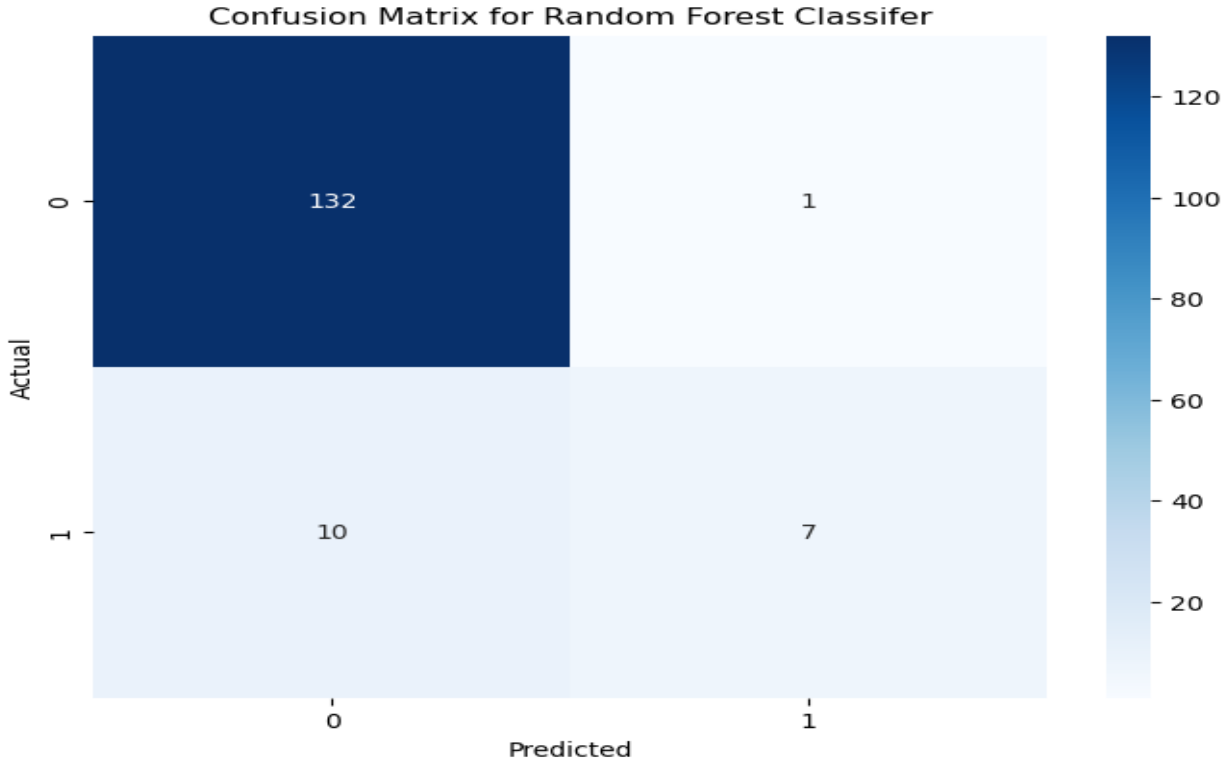


Fig [5-20] Shows confusion matrix of credit card approval model

5.4.4.2 FastAPI for front-end integration

We utilize FastAPI to expose the backend machine learning model in a way that allows seamless integration with the front-end interface. The FastAPI application serves as an intermediary, enabling real-time interaction between the frontend and the trained model by handling HTTP POST requests to perform predictions. This setup allows users to submit data from the frontend and instantly view the model's output without the need to run the model manually. For secure communication, we implement **CORS (Cross-Origin Resource Sharing)** configuration, explicitly permitting requests only from the frontend running on localhost (typically localhost:5174), ensuring that only the intended client can interact with the backend API. This approach not only improves efficiency by allowing local execution and testing but also maintains a controlled and secure communication channel between frontend and backend.

5.5 *Frontend important visualization and functionalities*

5.5.1 Chatbot

The chatbot is implemented to allow users to ask common or simple questions without the need to contact customer services directly. It is designed to handle frequently asked queries in a conversational manner, enhancing accessibility and responsiveness. This approach improves overall user experience by offering a fast, intuitive interface and promotes ease of use without hesitation. The chatbot contributes to a more streamlined and user-friendly interaction using Gemini API, reducing service load while supporting better user interface capabilities and it’s visualization shown in Fig [5-23]

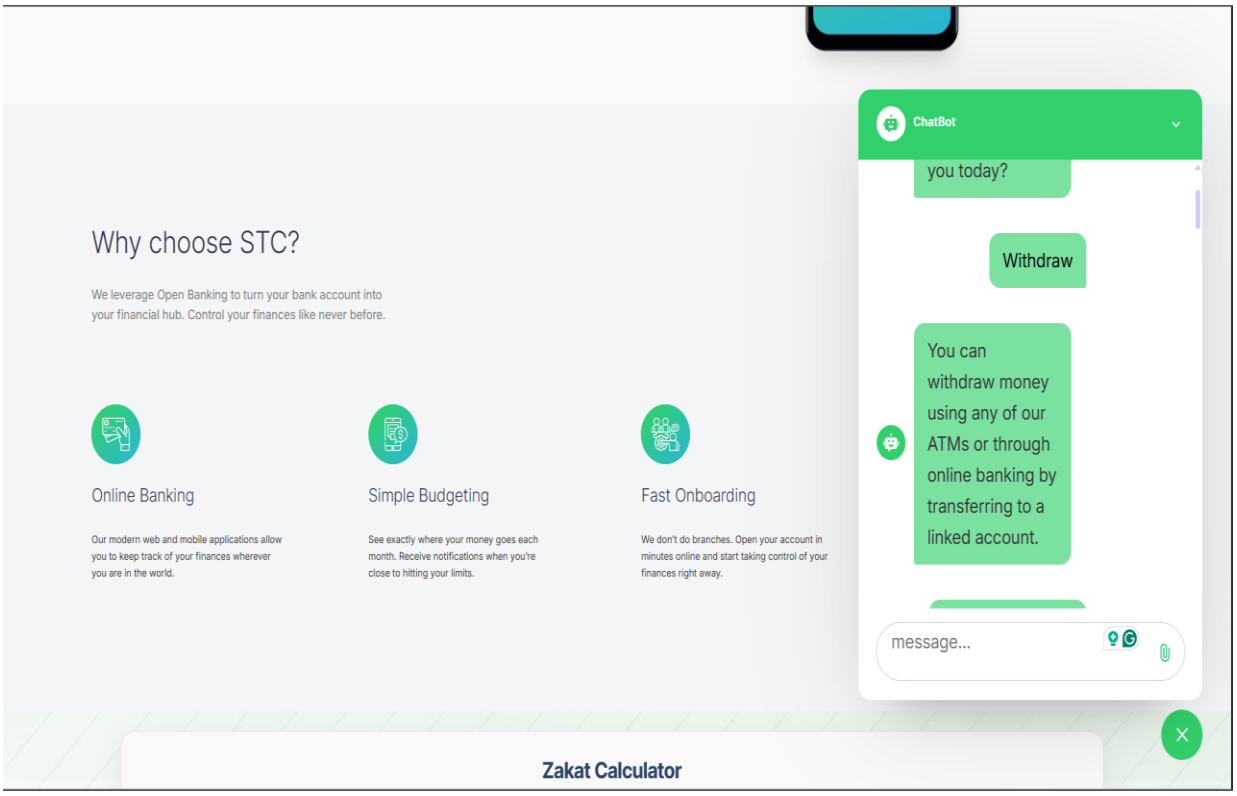


Fig [5-24] Shows how user can easily interact with chatbot.

5.5.2 Important security concerns visualization

Our system implements robust security measures including One-Time Passwords (OTPs) for secure transaction verification and integrates with VirusTotal for threat detection, all managed through our backend services. These security features are seamlessly exposed to users through our API tier in a hybrid backend architecture, with intuitive visualizations presented in the frontend interface as demonstrated in the architecture diagrams shown earlier. This approach ensures secure transactions while maintaining excellent usability through clear, accessible security information displays and it shown in the figures below (Fig [5-25] , Fig [5-26] , Fig [5-27] and Fig [5-28]).

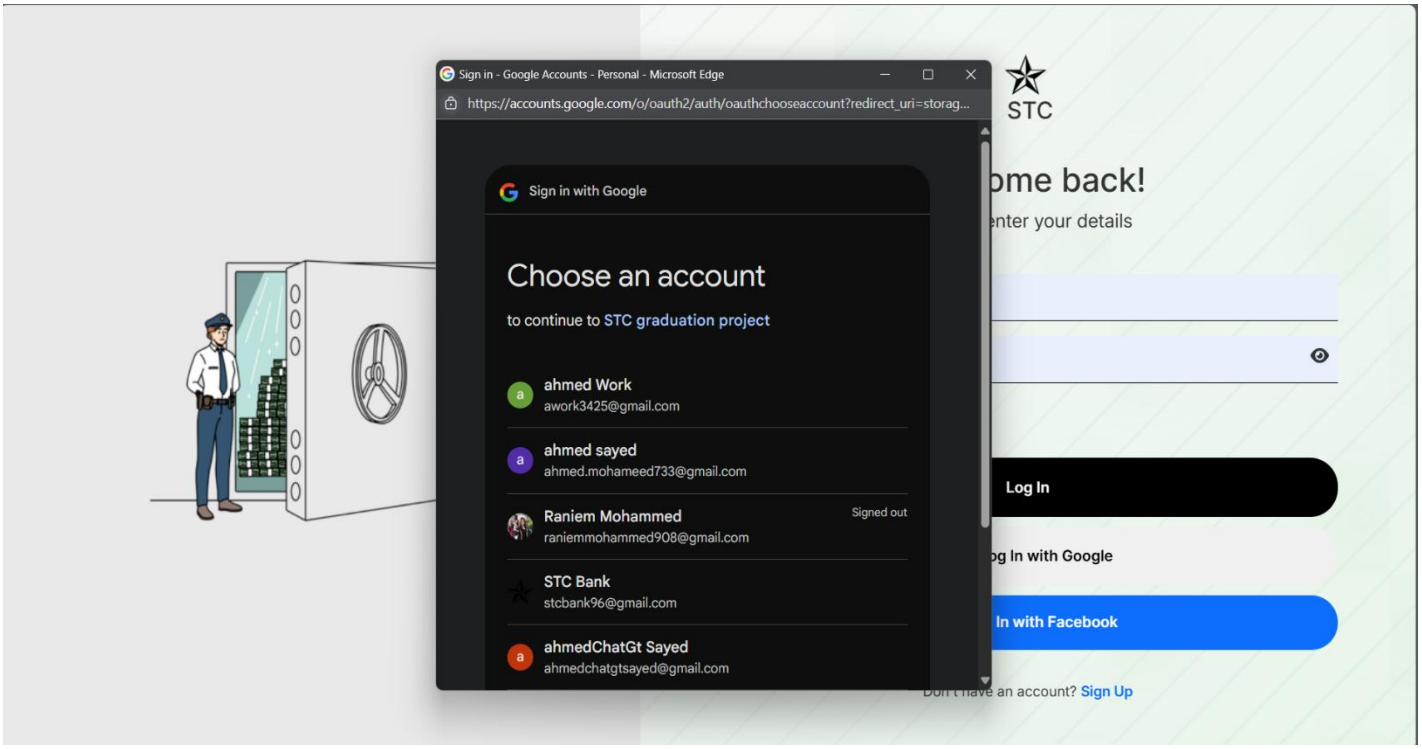


Fig [5-25] Shows how user can login with his Facebook and google account safely

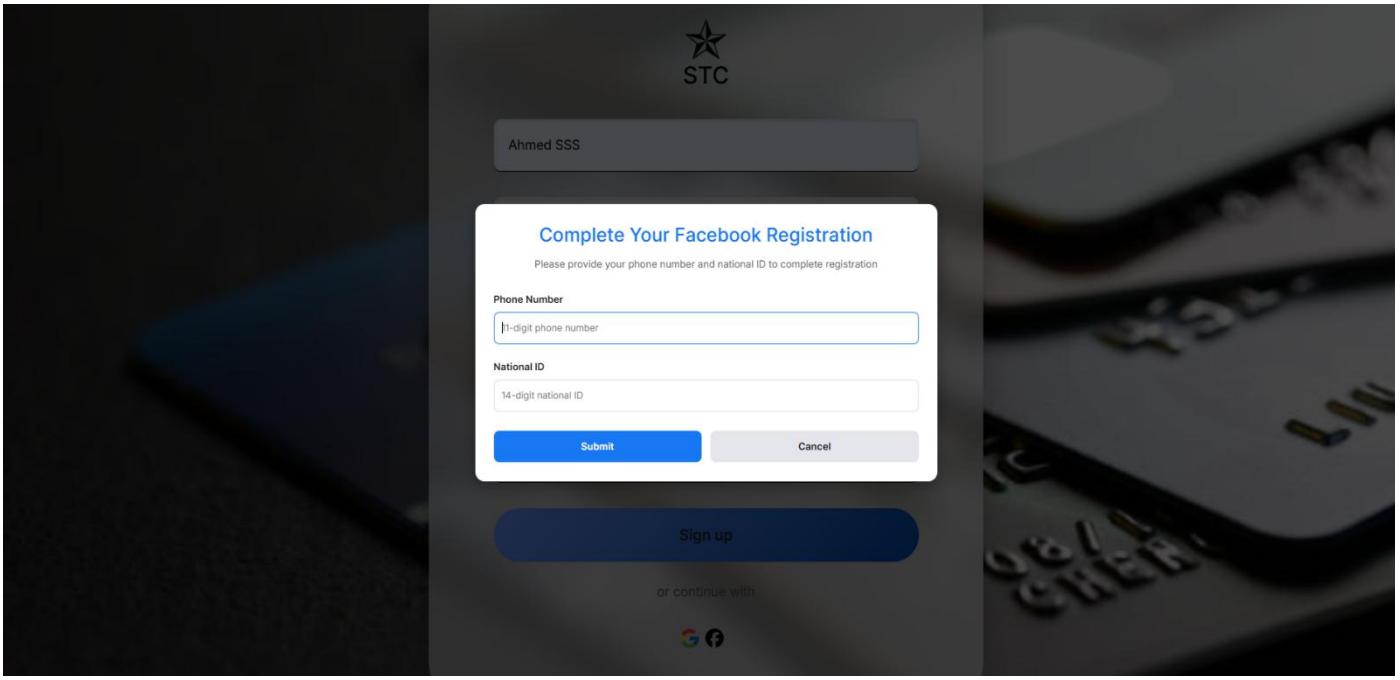
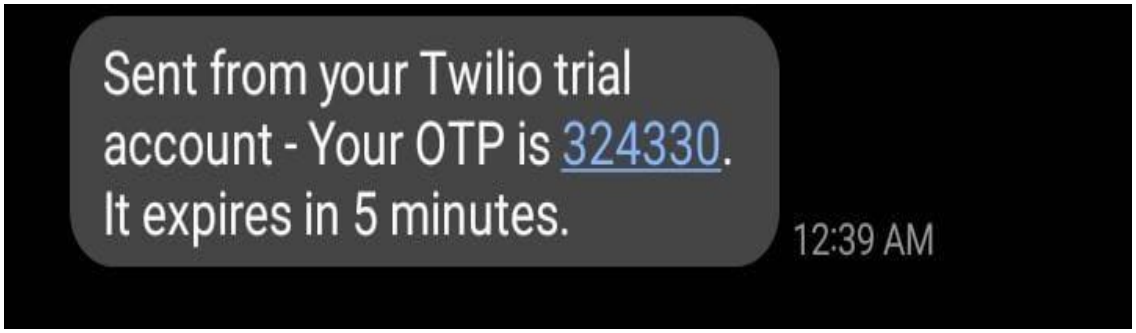


Fig [5-26] Shows how user can signup with his Facebook providing his phone number and National ID for security concerns.



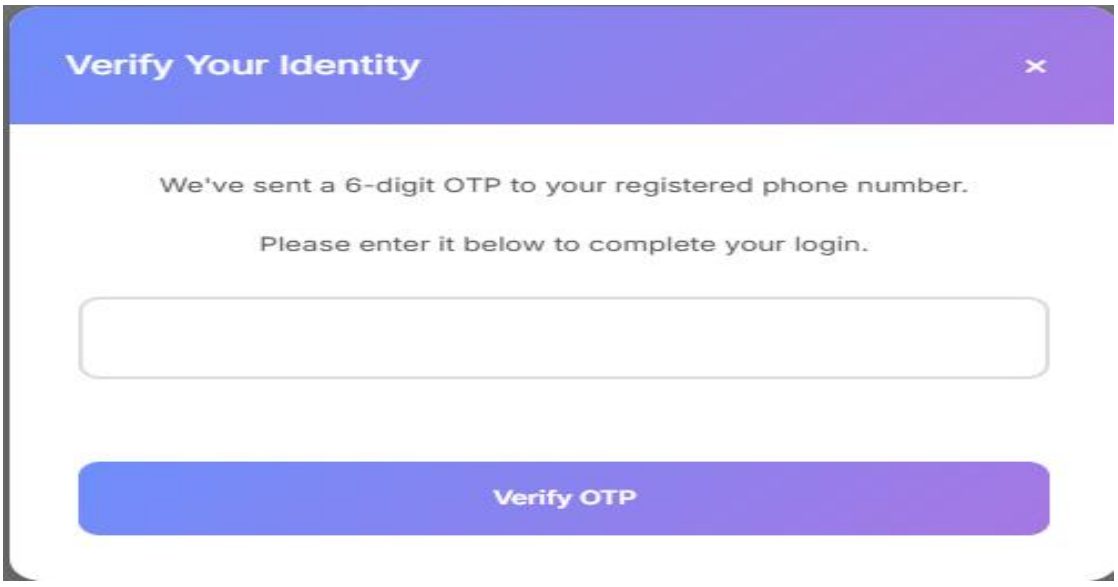


Fig [5-27] & Fig [5-28] Shows how OTP interface as an interface in the system.

Chapter 6 Future learning and recommendations

6.1 Future learning to apply

While we have implemented a fraud detection model, its effectiveness is currently limited by the accuracy of user location data. Free geolocation services, such as Google Maps APIs or IP-based methods, often yield location estimates with deviations of several kilometers, which reduces their reliability for use in location-sensitive fraud detection. To address this limitation, we recommend integrating hardware-based GPS modules or advanced geolocation software capable of delivering high-precision coordinates. This enhancement would significantly improve the model's input quality and overall prediction accuracy, as demonstrated in Fig [6-1] and Fig [6-2]. Since the system is still in the deployment and testing phase and operates under a zero-budget constraint using only open-source and free APIs, the model currently tends to classify most transactions as benign. This behavior highlights the need for more accurate data and further real-world validation before production deployment.

```
<class 'pandas.core.frame.DataFrame'>
Index: 194501 entries, 398918 to 375392
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   trans_date_trans_time  194501 non-null  object
 1   cc_num                 194501 non-null  int64
 2   merchant               194501 non-null  object
 3   category               194501 non-null  object
 4   amt                    194501 non-null  float64
 5   first                  194501 non-null  object
 6   last                   194501 non-null  object
 7   gender                 194501 non-null  int32
 8   street                 194501 non-null  object
 9   city                   194501 non-null  object
 10  state                  194501 non-null  object
 11  zip                    194501 non-null  int64
 12  lat                    194501 non-null  float64
 13  long                   194501 non-null  float64
 14  city_pop               194501 non-null  int64
 15  job                    194501 non-null  object
 16  dob                    194501 non-null  object
 17  trans_num              194501 non-null  object
 18  unix_time              194501 non-null  int64
 19  merch_lat              194501 non-null  float64
...
 21  is_fraud               194501 non-null  int64
 22  is_fraud_cat           194501 non-null  object
```

Fig [6-1] Describes the highlighted latitude and longitude dependence of fraud classification Model.

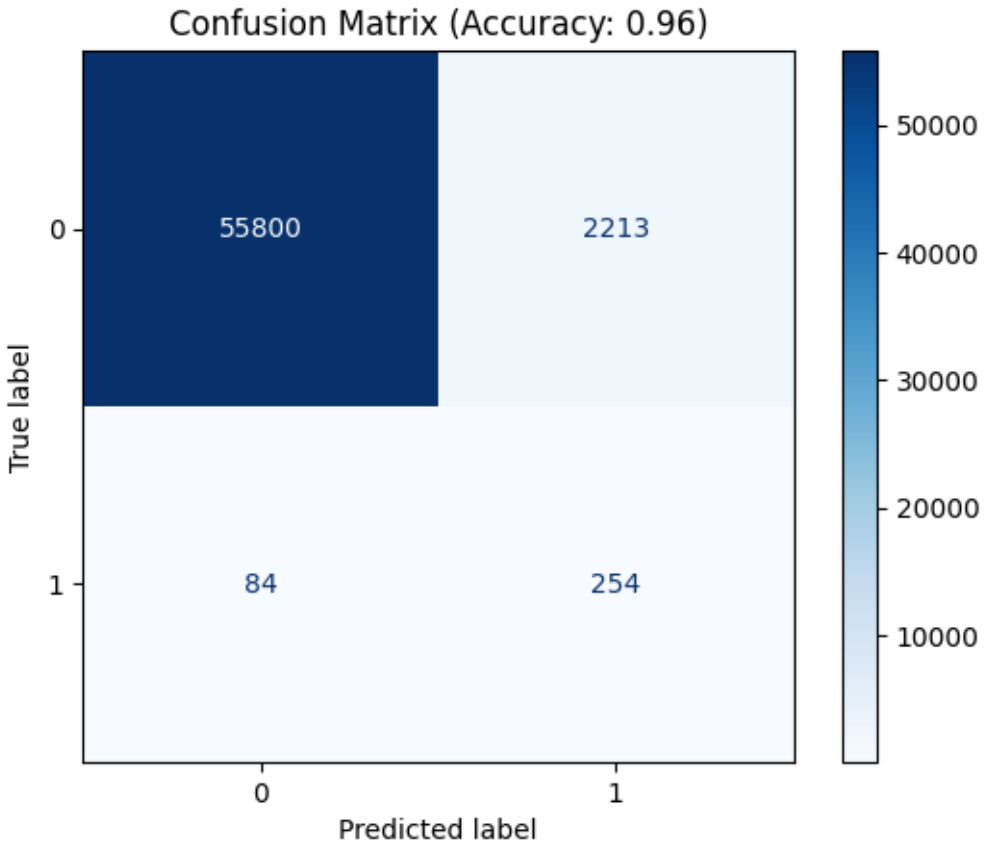


Fig [6-2] Describes Confusion matrix of fraud detection model

6.2 Important recommendation for production environment and real-life deployment

1- Secure licensed access to VirusTotal and Twilio APIs to strengthen threat detection and communication security mechanisms.

2- Avoid containerizing core transactional logic until reliable, real-time container orchestration frameworks are validated to uphold transactional consistency, durability, and fault tolerance.

3- Implement a fraud detection model to identify and flag anomalous or suspicious transactions based on behavioral and contextual patterns.

4- Balance security and performance—prioritizing security at the cost of all system resources can degrade performance and introduce unnecessary complexity. Consider the organization's risk appetite; in banking systems, it is often justifiable to trade off certain modern approaches like Docker and large-scale distributed processing to maintain tighter security controls.

5- Use HTTPS with modern TLS protocols for all client-server communications to protect data in transit and prevent MITM (Man-in-the-Middle) attacks.

6- Log all access to sensitive operations (e.g., fund transfers, account updates) for auditability and compliance with financial regulations.

7- Ensure GDPR/PCI-DSS compliance where applicable, especially for handling User PII (Personally Identifiable Information.), transaction history, and fraud investigation data.