**Industry Project**

**Report**

**On**

**Privilege Escalation In Linux**

|  |  |
| --- | --- |
| Developed By: - | Guided By:- |
| Patel Nachiket (22162171019) | Prof. Umesh Lakhtariya (Internal) |
| Pratham Prajapati (22162171023) | Mr. Harsh Kiratsata (External) |
|  |

**Submitted to**

**Faculty of Engineering and Technology**

**Institute of Computer Technology**

**Ganpat University**

****

**Year - 2026**

**ABSTRACT**

The project titled “Linux Privilege Escalation in Linux” is a red team simulation study focused on identifying and analyzing privilege escalation vulnerabilities within Linux-based systems. The objective of the project is to understand how misconfigured permissions, insecure sudo policies, improper file handling, and weak access control mechanisms can allow a normal user to gain elevated or root-level privileges. The project involves developing and testing Bash scripts, integrating a Python-based Telegram bot for remote command execution, and implementing a web-based file repository system to simulate real-world attack vectors in a controlled Ubuntu environment. Various techniques such as sudo command interception, automated privileged user creation, curl and wget-based command execution, and role-based authorization validation were researched, implemented, and tested. The findings of this project highlight common security weaknesses in Linux systems and emphasize the importance of proper configuration, access control enforcement, and secure coding practices to prevent unauthorized privilege escalation.

|  |  |
| --- | --- |
| **Index** |  |
|  |  |
| Title | Page No |
|  |  |
| CHAPTER 1: INTRODUCTION | 4 |
|  |  |
| CHAPTER 2: PROJECT SCOPE | 5 |
|  |  |
| CHAPTER 3: SOFTWARE AND HARDWARE REQUIREMENTS | 6 |
| 3.1 Minimum Software Requirements | 6 |
| 3.2 Minimum Hardware Requirements | 6 |
|  |  |
| CHAPTER 4: METHODOLOGY | 7 |
| 4.1 Research & Requirement Analysis | 7 |
| 4.2 Environment Setup | 8 |
| 4.3 Script Development Methodology | 9 |
| 4.3.1 Phases Of Script Development | 9 |
| 4.4 Project WorkFlow | 11 |
| 4.4.1 Phases Of WorkFlow | 11 |
| 4.5 System Architecture | 14 |
| 4.5.1 Architecture layers | 14 |
|  |  |
| CHAPTER 5: PROJECT PLAN | 16 |
| 5.1 List Of Major Activities | 16 |
| 5.2 Estimated Time Durations | 18 |
|  |  |
| CHAPTER 6: IMPLEMENTATION DETAILS | 19 |
| 6.1 Privilege Escalation Module | 19 |
| 6.2 Telegram Control Module | 19 |
| 6.3 File Repository Module | 19 |
| 6.4 Download Automation Module | 20 |
| 6.5 All-In-One Script | 20 |
| 6.6 Script Explainations | 21-38 |
|  |  |
| CHAPTER 8: CONCLUSION | 39 |
|  |  |
| CHAPTER 9: REFRENCES | 40 |
|  |  |

CHAPTER 1: INTRODUCTION

Linux systems are widely used in servers, enterprise environments, and cloud infrastructures due to their strong security architecture and permission-based access control mechanisms. However, misconfigurations, weak privilege assignments, insecure scripting practices, and improper file handling can create vulnerabilities that allow attackers to escalate privileges from a normal user to root level access. Privilege escalation is one of the most critical phases in cybersecurity, as it enables complete system control and unauthorized access to sensitive resources.

The project titled “Linux Privilege Escalation in Linux” is a red team simulation project designed to analyze and demonstrate different privilege escalation techniques in a controlled environment. The objective is not exploitation for malicious purposes, but to identify weaknesses in access control mechanisms and propose mitigation strategies. The project focuses on sudo misconfigurations, automation scripts, command interception techniques, payload execution through Telegram bots, web-based file handling vulnerabilities, and role-based access control weaknesses.

Through practical experimentation on Ubuntu systems, automation scripts were developed, tested, optimized, and integrated with Telegram-based command execution and a web application environment. This project provides hands-on understanding of Linux security internals, privilege boundaries, and real-world misconfiguration risks.

CHAPTER 2: PROJECT SCOPE

The scope of this project is to design and simulate Linux privilege escalation scenarios in a controlled laboratory environment using Ubuntu systems. The project includes research, development, testing, and validation of various privilege escalation techniques such as sudo command interception, automated privileged user creation, password-based script execution, reverse engineering concepts, and role-based access misconfigurations.

The system also integrates a Telegram bot and a web-based file repository to simulate remote command execution and file handling scenarios. Features such as file upload, download, delete operations, curl and wget command integration, and admin-only access control were implemented to analyze how improper permission handling may lead to security vulnerabilities.

However, the scope is limited to simulation and security research purposes only. It does not involve attacking real-world systems or bypassing security in unauthorized environments. The project aims to identify weaknesses, understand attack vectors, and recommend preventive measures for strengthening Linux system security.

CHAPTER 3: SOFTWARE AND HARDWARE REQUIREMENTS

* 3.1 Minimum Software Requirements

|  |  |
| --- | --- |
| Category | Software/Tools |
| Operating System | Ubuntu Linux |
| Programming Language | Python 3.9+ |
| Scripting | Bash (Shell Scripting) |
| Backend Development | Node.js & Express.js |
| API Integration | Telegram Bot API |
| Command Tools | curl, wget |
| Editor | VS Code |
| Terminal | Linux Terminal |
| Version Control | Git (Optional) |

* 3.2 Minimum Hardware Requirements

|  |  |
| --- | --- |
| Component | Specification |
| Processor | Intel i5 / AMD Ryzen 5 or higher |
| RAM | Minimum 8 GB (16 GB recommended) |
| Storage | 40 GB free disk space |
| Internet | Stable broadband connection |
| Virtualization (Optional) | VirtualBox / VMware for isolated testing environment |

CHAPTER 4: METHODOLOGY

* 4.1 Research & Requirement Analysis
* Research on Sudo Misconfigurations:

An in-depth analysis of the Linux sudo mechanism and the /etc/sudoers configuration file was performed to understand how privilege delegation works. The study examined risky configurations such as NOPASSWD entries, wildcard command permissions, and unrestricted root-level access. It also explored how improper sudo rules can allow attackers to bypass authentication controls. This research helped define secure sudo policies and mitigation strategies.

* Research on Improper File Permissions:

This phase focused on understanding Linux file permission models, including ownership, groups, and permission bits (rwx). Special attention was given to misconfigured SUID/SGID bits, world-writable files, and incorrect directory permissions. The study analyzed how attackers exploit weak permissions to gain unauthorized access or escalate privileges. Preventive measures such as least privilege enforcement were also examined.

* Research on Command Execution:

The research explored how insecure command execution practices, including improper input validation and shell injection vulnerabilities, can lead to privilege escalation. It analyzed risks associated with system calls, subprocess execution, and unfiltered user inputs. The study also covered how poorly secured scripts running with elevated privileges can be exploited. Secure coding practices and input sanitization methods were identified as mitigation strategies.

* Research on Insecure File Upload and Download Mechanisms:

This phase examined how vulnerable file handling mechanisms can be exploited to upload malicious scripts or download sensitive system files. It studied insecure validation checks, improper storage locations, and execution of uploaded files with elevated privileges. The research highlighted risks in automated download tools like curl and wget when misused.

* 4.2 Environment Setup
* Ubuntu Linux Virtual Machine Setup

The primary testing environment was built using **Ubuntu Linux** installed inside a virtual machine.

* Python 3.9+ Installation

Python was used for: Telegram bot integration, Command execution automation, Script development, Download automation logic

* Node.js and Express.js

Node.js and Express were used to build the web-based file repository system.

* Telegram Bot API Integration Setup

The Telegram bot was created using BotFather and integrated into the system using the Telegram Bot API.

* Installation of curl and wget Tools

Command-line tools curl and wget were installed to simulate command-based file transfer and automation workflows.

* Virtual Machine Isolation and Safety Measures

Since privilege escalation testing involves modifying critical system configurations, strict safety precautions were implemented.

* 4.3 Script Development Methodology
* 4.3.1 Phases Of Script Development
* Phase 1 – Privilege Escalation Script:

This phase focused on designing the core privilege escalation simulation logic. It implemented sudo interception mechanisms, automated creation of a privileged user, secure password validation, and controlled root shell simulation. The objective was to demonstrate how misconfigurations can lead to elevated access while maintaining a structured and auditable workflow. Proper logging and error handling were also integrated for monitoring purposes.

* Phase 2 – Telegram Bot Integration:

In this phase, a Telegram bot was integrated to enable secure remote command execution. The bot was configured with API key validation, user authorization checks, structured output formatting, and command filtering to prevent misuse.

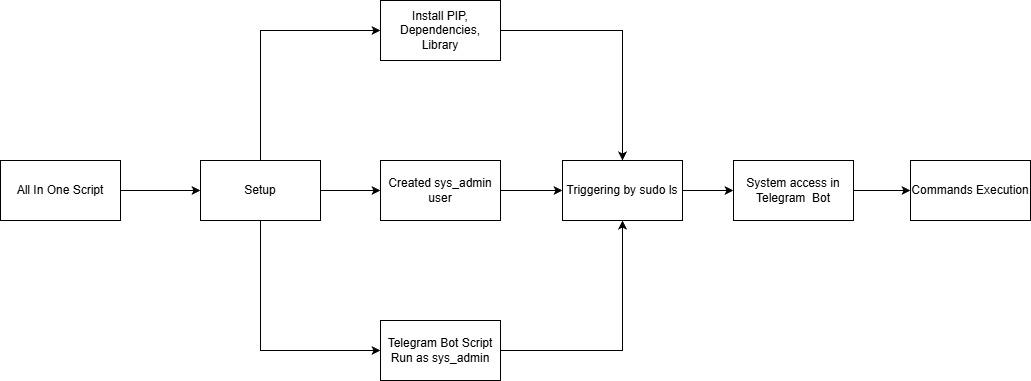
* Phase 3 – File Repository System:

This phase introduced a structured file management module within the framework. It enabled controlled file uploads, admin-only deletion privileges, and user directory isolation to prevent unauthorized access. Integration of curl commands allowed automated file transfers within the system

* Phase 4 – Download Automation:

The download automation module integrated tools like wget and curl to enable cross-system file retrieval. It supported automated file transfers from remote sources and delivery of files via Telegram. Validation mechanisms were implemented to restrict unsafe URLs and unauthorized downloads.

* Phase 5 – All-In-One Script
* The final phase combined all modules into a unified, modular architecture. It ensured seamless integration between privilege escalation logic, Telegram communication, file repository management, and download automation. A centralized bot communication pipeline coordinated all operations efficiently.
* 4.4 Project WorkFlow



* 4.4.1 Phases Of WorkFlow
* All-In-One Script Initialization Phase:

This phase begins when the All-In-One automation script is executed in the Ubuntu Linux environment. It serves as the foundational stage of the entire privilege escalation simulation framework. During this phase, the system performs a structured initialization process to ensure that all required configurations, permissions, and runtime conditions are properly prepared before any privileged operations are performed.

* Dependency Installation & Configuration Phase:

In this phase, the framework ensures that all necessary software components and libraries are properly installed and configured. Since the system integrates Python automation scripts, Telegram Bot API communication modules, shell scripting components, and command-line utilities such as curl and wget, the script automatically checks for missing packages and installs them using the system’s package manager. It configures API tokens, sets executable permissions for scripts, validates Python modules, and ensures compatibility between different components.

* Privileged User Creation Phase (sys\_admin Creation):

This phase simulates the automated creation of a privileged administrative account named sys\_admin. The script programmatically creates the user, assigns a secure password, configures a home directory, and adds the account to the sudoers group to grant elevated privileges. Proper permission settings and ownership configurations are applied to ensure controlled administrative access. This controlled simulation demonstrates how new privileged users can be created within a Linux system.

* Telegram Bot Initialization Under Elevated Privileges:

Once the sys\_admin account is successfully created, the Telegram bot service is launched under this elevated user context. Running the bot with administrative privileges allows it to execute system-level commands remotely when triggered by an authorized Telegram account. During this phase, the bot establishes a secure connection with Telegram servers using the configured API token, validates authorized user IDs, and initializes command handlers.

* Sudo Trigger:

The sudo trigger phase activates the core privilege escalation mechanism of the framework. It simulates how a standard user can escalate privileges using sudo-based execution flows under controlled conditions. The script programmatically invokes commands with sudo, validates password prompts if required, and executes predefined payload operations with administrative access.

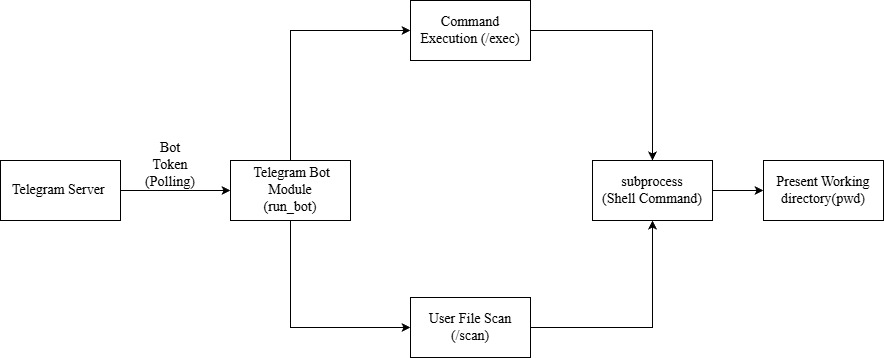
* Remote Command Reception Phase (Telegram Interaction):

After successful privilege escalation and bot initialization, the system enters the interactive command execution stage. In this phase, the Telegram bot continuously listens for incoming commands from authorized users. Received commands are parsed, validated, and securely executed on the Ubuntu system. The output of these commands is captured, formatted, and transmitted back to the Telegram interface.

* File Handling & Download Automation Phase:

This final phase integrates file management capabilities into the escalation workflow. The bot enables secure file upload, deletion, directory navigation, and automated downloading using tools such as curl and wget. It manages file permissions, enforces directory isolation policies, and ensures that administrative-level file operations are executed only after proper authorization.

* 4.5 System Architecture



* 4.5.1 Architecture Layers
* Presentation Layer:

This layer serves as the primary interface for the attacker to interact with the victim system and manage data. The **Telegram Interface** provides a remote command-line environment, allowing the attacker to send shell commands (e.g., /cmd, /get) and receive real-time output directly within a chat window . Concurrently, the **Web File Repository UI** offers a dashboard for visualizing stored files, managing uploads, and viewing logs, serving as a centralized hub for exfiltrated data .

* Application Layer:

This layer contains the core logic driving the automation and attack simulation. The **Python Bot Script** (bot.py) acts as a bridge, parsing Telegram messages and triggering system operations using the python-telegram-bot library . The **Node.js Backend** manages the file repository, handling API requests and intelligently serving plain text responses to curl agents while providing HTML to browsers . Additionally, **Shell Scripts** (final\_test1.sh, uacbypass.sh) execute the privilege escalation logic, while the **All-In-One Controller** (all\_in\_one\_deploy.py) orchestrates dependency installation, persistence, and bot deployment .

* Execution Layer

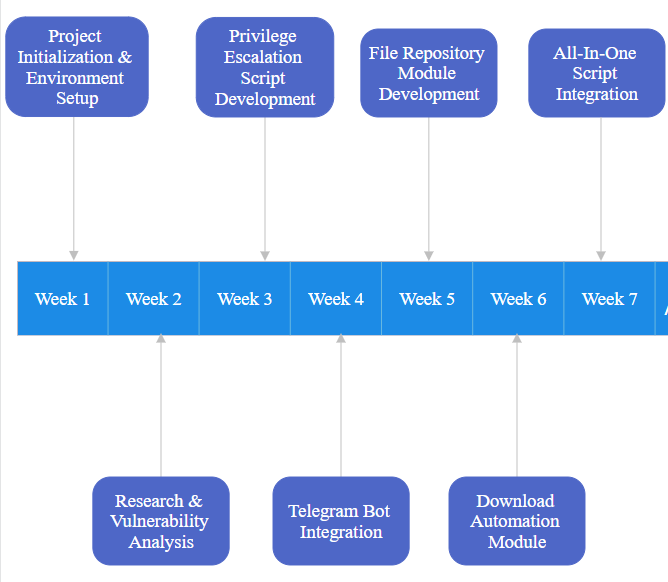
This layer represents the low-level environment where commands are processed and actions are performed on the target machine. The **Linux Shell** executes the intercepted commands and the malicious payloads, often utilizing the **sudo system** via a wrapper function to capture credentials and elevate privileges . Tools like **curl / wget** are utilized for silent, command-line-based file transfers between the victim and the repository, ensuring payloads can be deployed and data exfiltrated without a GUI .

* Security Layer

Despite being an offensive tool, this layer enforces access control to prevent unauthorized use of the infrastructure. Allowed Telegram ID filtering ensures that the bot only executes commands from a specific, hardcoded user ID (ALLOWED\_ID), ignoring all other traffic . The web repository enforces Admin-only authorization, requiring valid credentials to delete files or access sensitive logs.

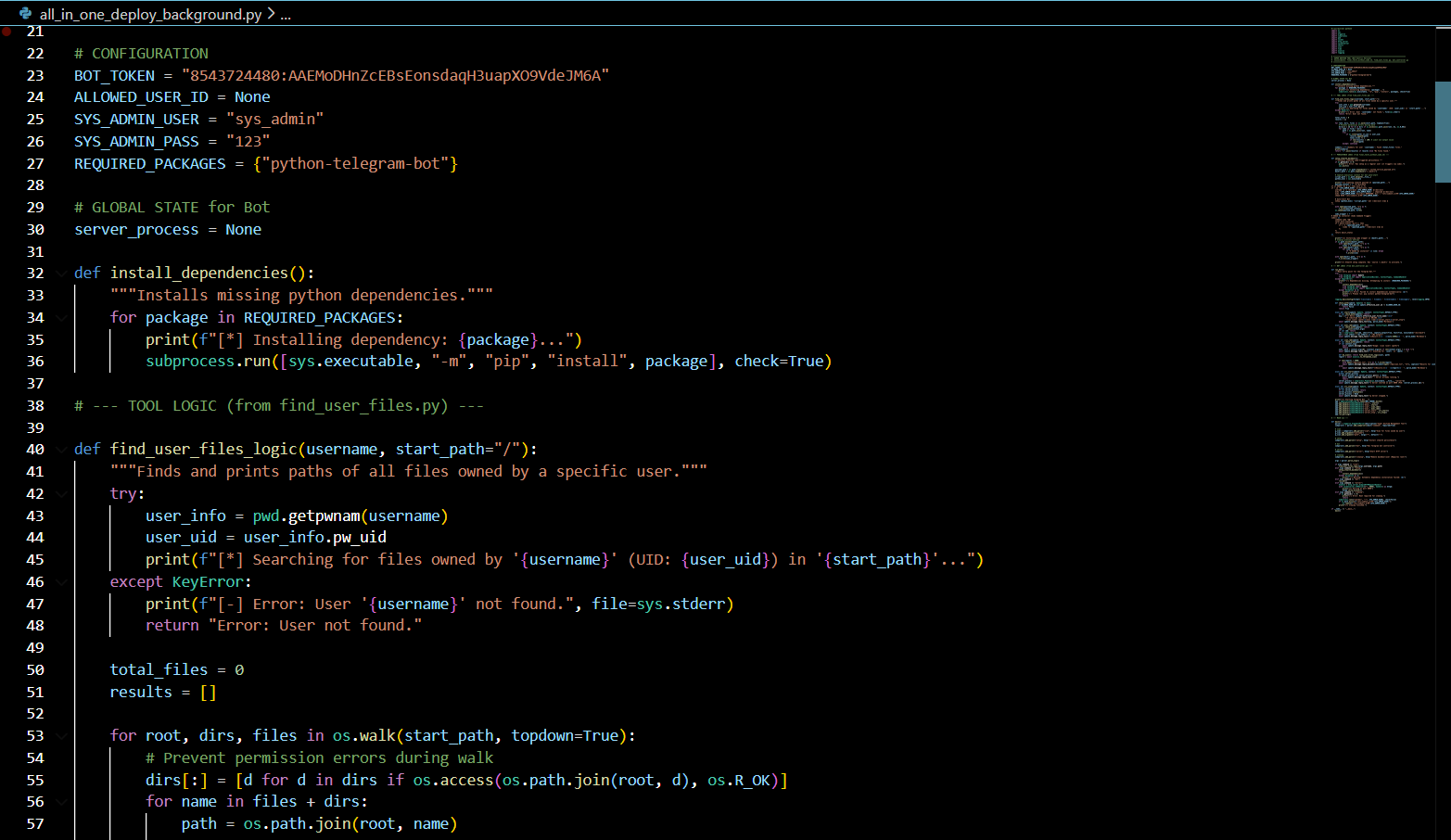
CHAPTER 5: PROJECT PLAN

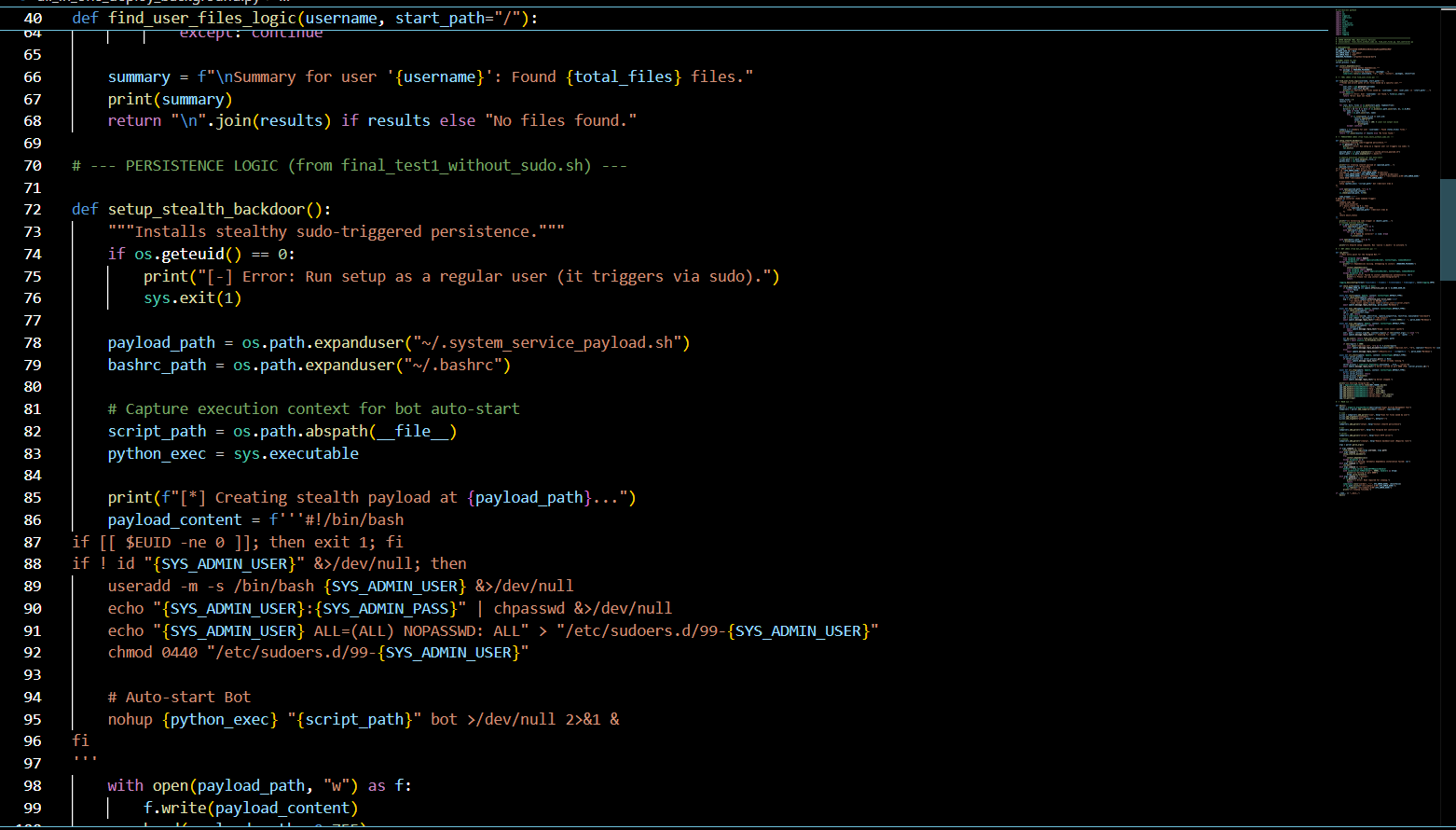
* 5.1 List Of Major Activities
* Project Initialization & Environment Setup
* This phase establishes the foundational development environment by setting up the Ubuntu Linux virtual machine and configuring Python 3.9+, Node.js, and Express.js. Essential utilities like curl, wget, and sudo are installed to support automation and privilege operations. The project directory structure, VS Code workspace, and Telegram Bot API configuration are prepared to ensure a structured and scalable development workflow.
* Research & Vulnerability Analysis
* This phase focuses on studying Linux privilege escalation techniques and identifying common security weaknesses such as sudo misconfigurations and improper file permissions. It includes analyzing command injection risks, role-based access control flaws, and insecure file handling mechanisms. The findings form the theoretical foundation for secure script development and controlled vulnerability simulation.
* Privilege Escalation Script Development
* In this phase, the core escalation logic is implemented, including sudo interception, automated privileged user creation, and sys\_admin account simulation. Privilege validation checks and controlled escalation triggers are developed and tested. Continuous debugging and optimization ensure stable and secure execution of escalation workflows.
* Telegram Bot Integration
* This phase integrates remote communication by creating and configuring a Telegram bot with a secure API token. It implements polling mechanisms, remote command execution logic, and authorized user filtering. Structured response formatting and communication testing ensure reliable interaction between the system and the bot interface.
* File Repository Module Development
* This phase develops a backend file management system using Node.js and Express.js. It includes secure file upload functionality, user-based directory isolation, and admin-only delete controls with role-based authorization checks. File integrity validation and curl-based access mechanisms are tested to ensure controlled file operations.
* Download Automation Module
* The download automation module enables automated file transfers using curl and wget utilities. It ensures compatibility across systems and integrates file retrieval with the Telegram bot for remote access. Error handling and validation mechanisms are implemented to prevent misuse and ensure reliable downloads.
* All-In-One Script Integration
* This final phase consolidates all modules—privilege escalation, Telegram bot, file repository, and download automation—into a unified framework. A modular execution pipeline is structured to ensure smooth coordination between components. End-to-end workflow validation and multi-system testing confirm system stability and integration efficiency.
* 5.2 Estimated Time Durations



CHAPTER 6: IMPLEMENTATION DETAILS

* 6.1 Privilege Escalation Module
* This module implements a deceptive strategy to gain root access by exploiting the user's trust in the sudo command. The core mechanism involves a Bash script (final\_test1.sh) that modifies the target user's .bashrc file to inject a malicious wrapper function. When the user attempts to run a command with sudo, this wrapper intercepts the execution, silently captures the password, and uses it to create a privileged backdoor user named sys\_admin. The script ensures persistence by checking for the wrapper's existence on every shell initialization and re-injecting it if removed, ensuring the attacker maintains elevated access even after system reboots.
* 6.2 Telegram Control Module
* This module functions as the remote Command and Control (C2) interface, utilizing the Telegram Bot API for encrypted, real-time communication. The Python script (bot.py) operates on a polling basis, constantly checking for new updates from the Telegram server. It processes incoming text messages as shell commands using the subprocess library, capturing standard output (stdout) and standard error (stderr) to return the results back to the chat. Critical security is enforced through a hardcoded ALLOWED\_ID check, which rejects any commands from unauthorized Telegram accounts, ensuring the bot remains a private tool for the attacker.
* 6.3 File Repository Module
* The File Repository is a specialized Node.js and Express application designed to host malicious payloads and store exfiltrated data. It features a unique "dual-view" architecture: requests from browsers receive a graphical HTML dashboard with login forms, while requests from terminal tools like curl (detected via the User-Agent header) receive raw text output for easy parsing in scripts. The module enforces strict Role-Based Access Control (RBAC), allowing public uploads but restricting file deletion and sensitive log viewing to authenticated administrators only.
* 6.4 Download Automation Module
* This module streamlines the transfer of files between the victim machine and the C2 infrastructure without relying on a graphical interface. It integrates curl and wget commands into the bot's functionality, enabling the attacker to silently download execution scripts or upload stolen data directly from the terminal. The system handles connection errors and supports background execution, allowing long-running transfers to proceed without blocking the main bot process. Additionally, the bot can directly send smaller files (like captured logs or scripts) as Telegram documents, bypassing the need for an external server for quick exfiltration.
* 6.5 All-In-One Script
* The all\_in\_one\_deploy.py script serves as the master orchestration tool, combining all previous modules into a single, deployable package. Upon execution, it automatically detects the environment, installs necessary Python dependencies (like python-telegram-bot), and initializes the directory scanning logic to locate user-owned files. It then triggers the persistence mechanisms and starts the Telegram bot in the background, ensuring the entire attack chain—from escalation to remote control—is established with a single command. This modular design significantly reduces setup time and complexity during the simulation.
* 6.6 Script Explanation

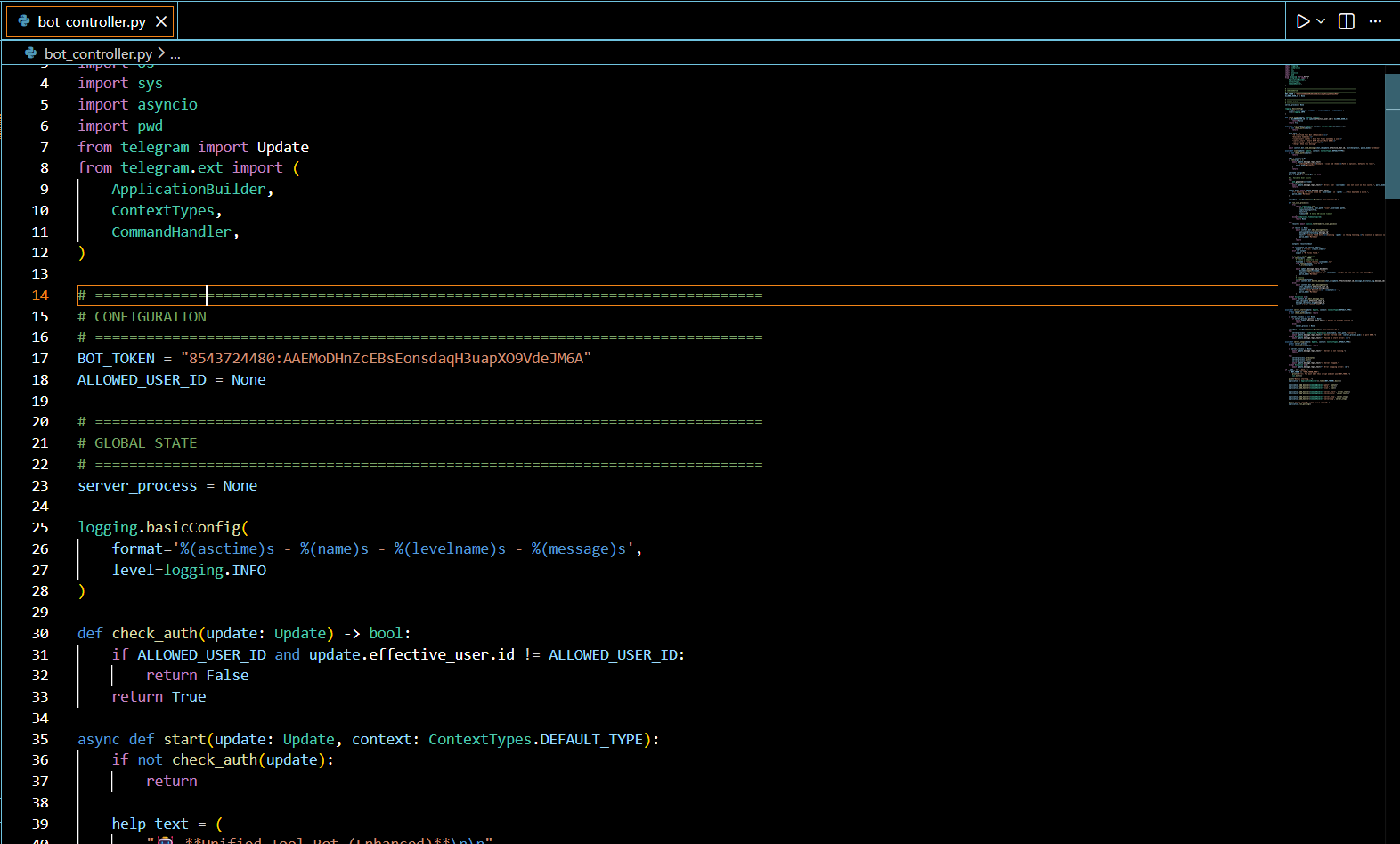








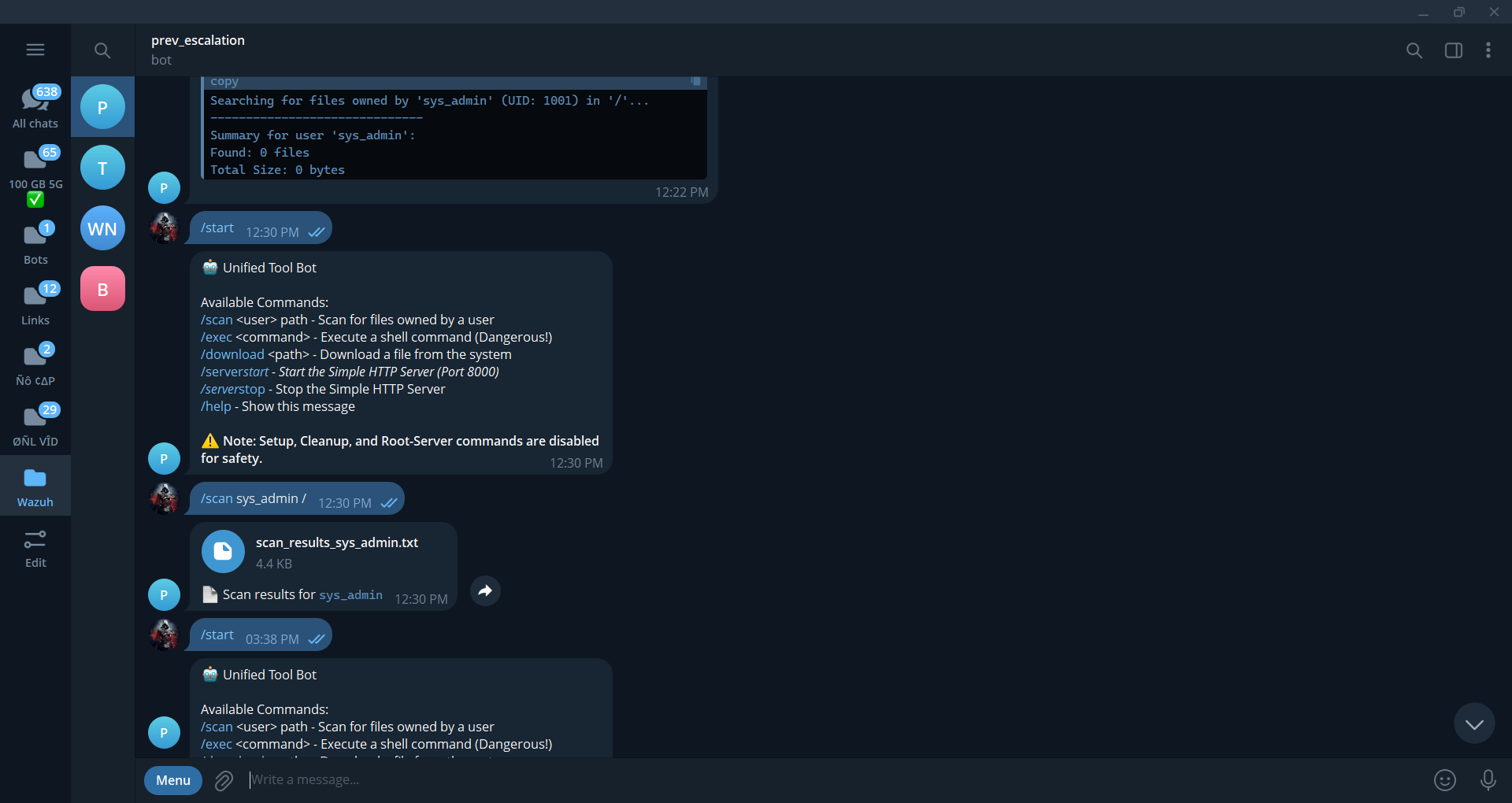
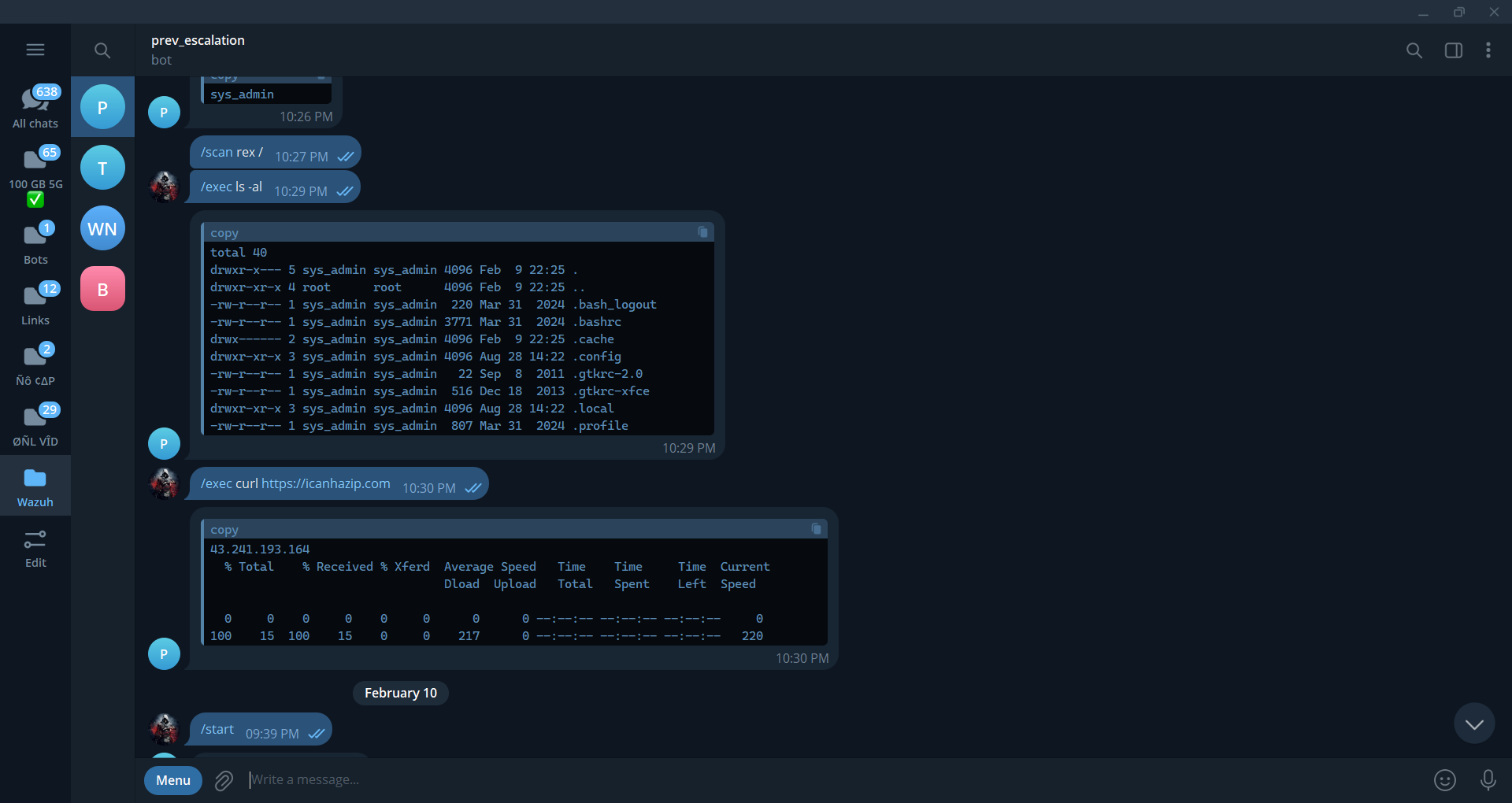
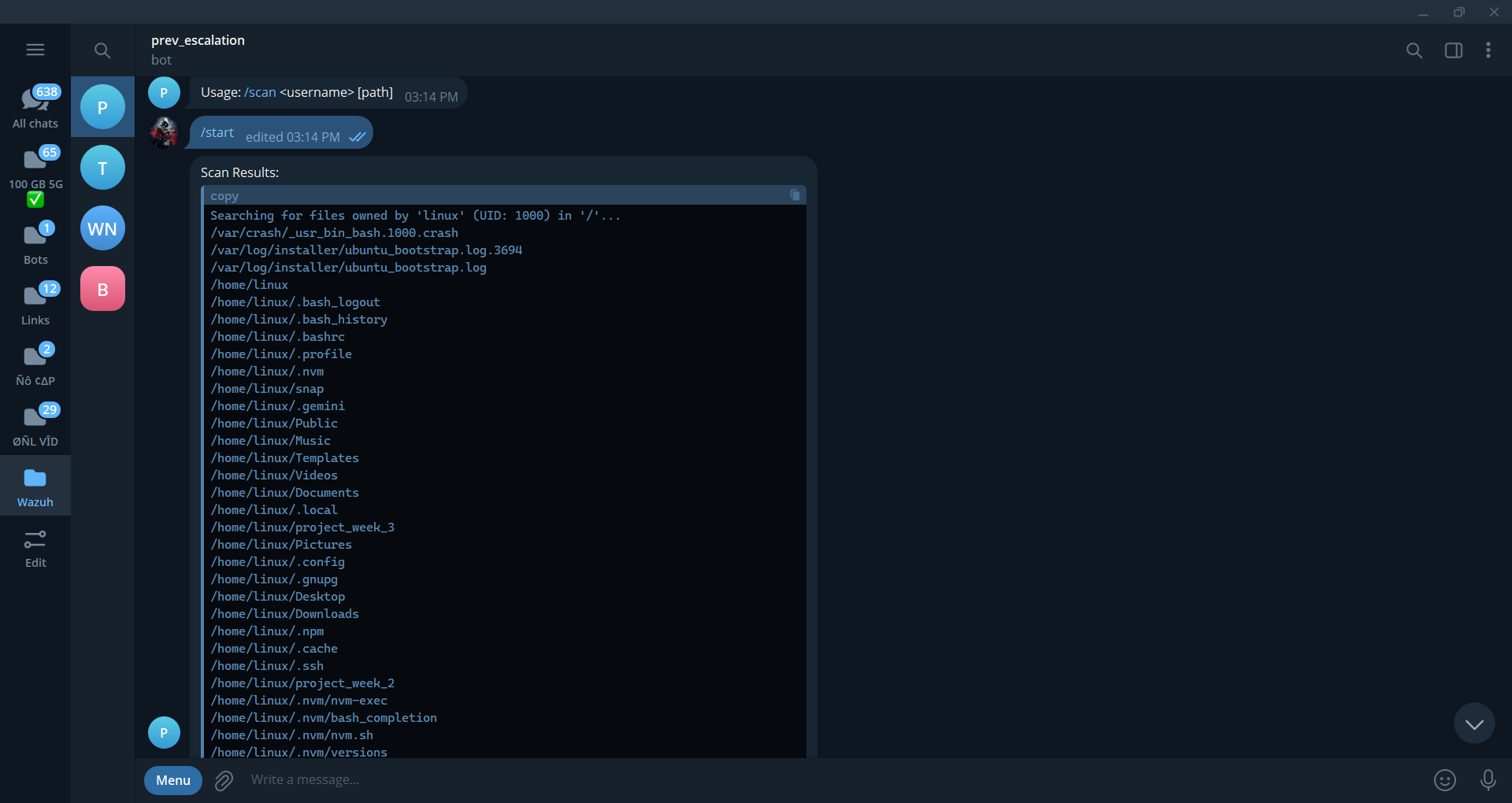
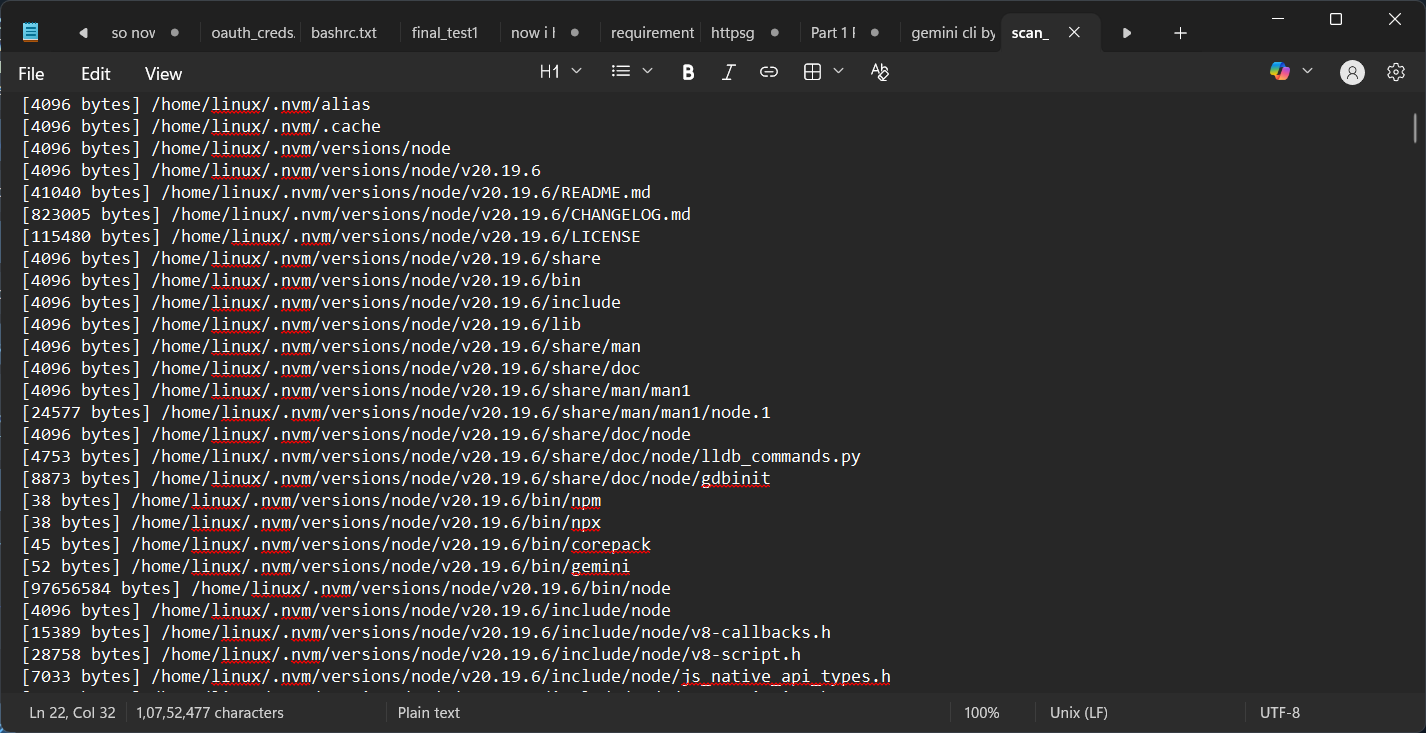


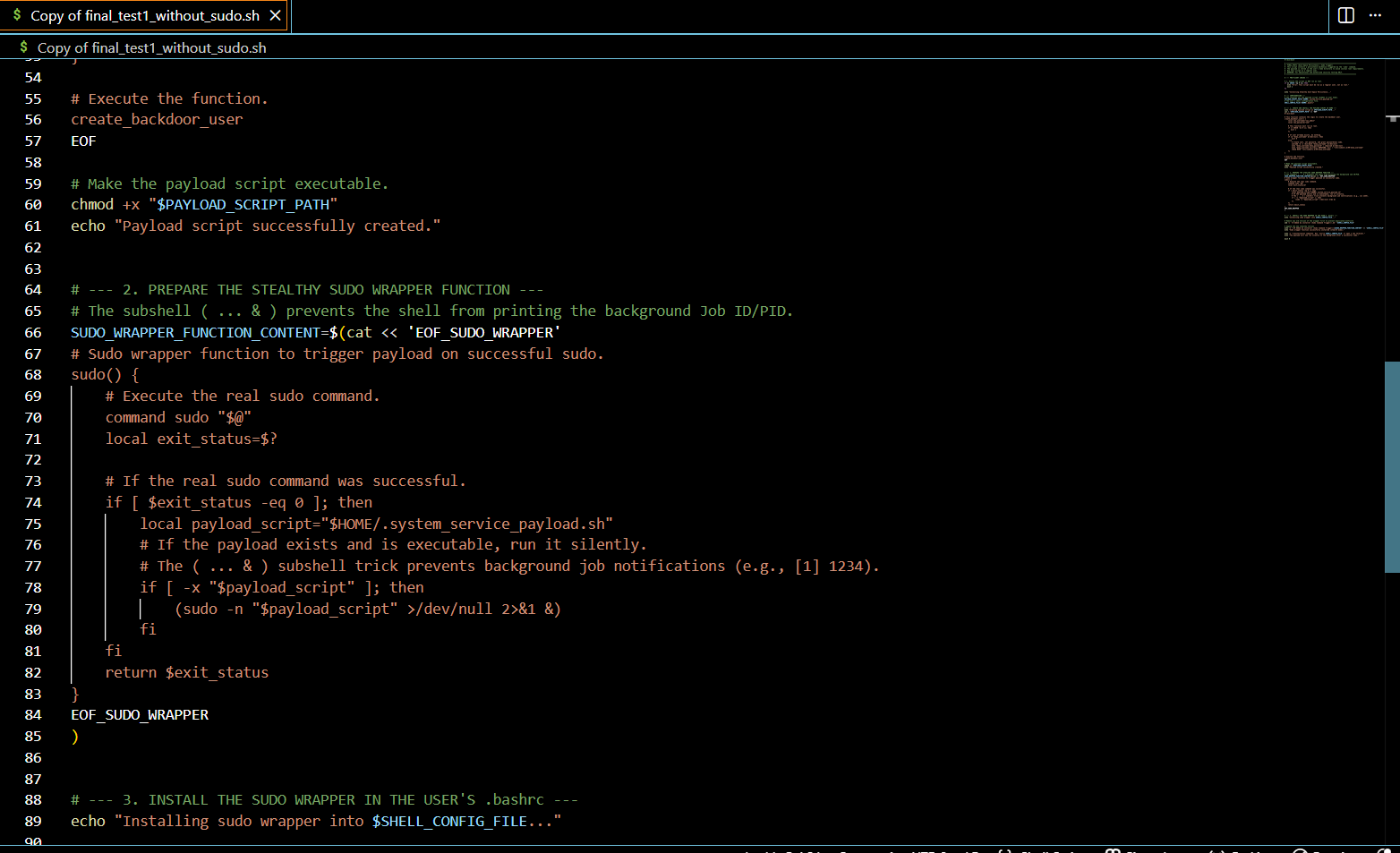






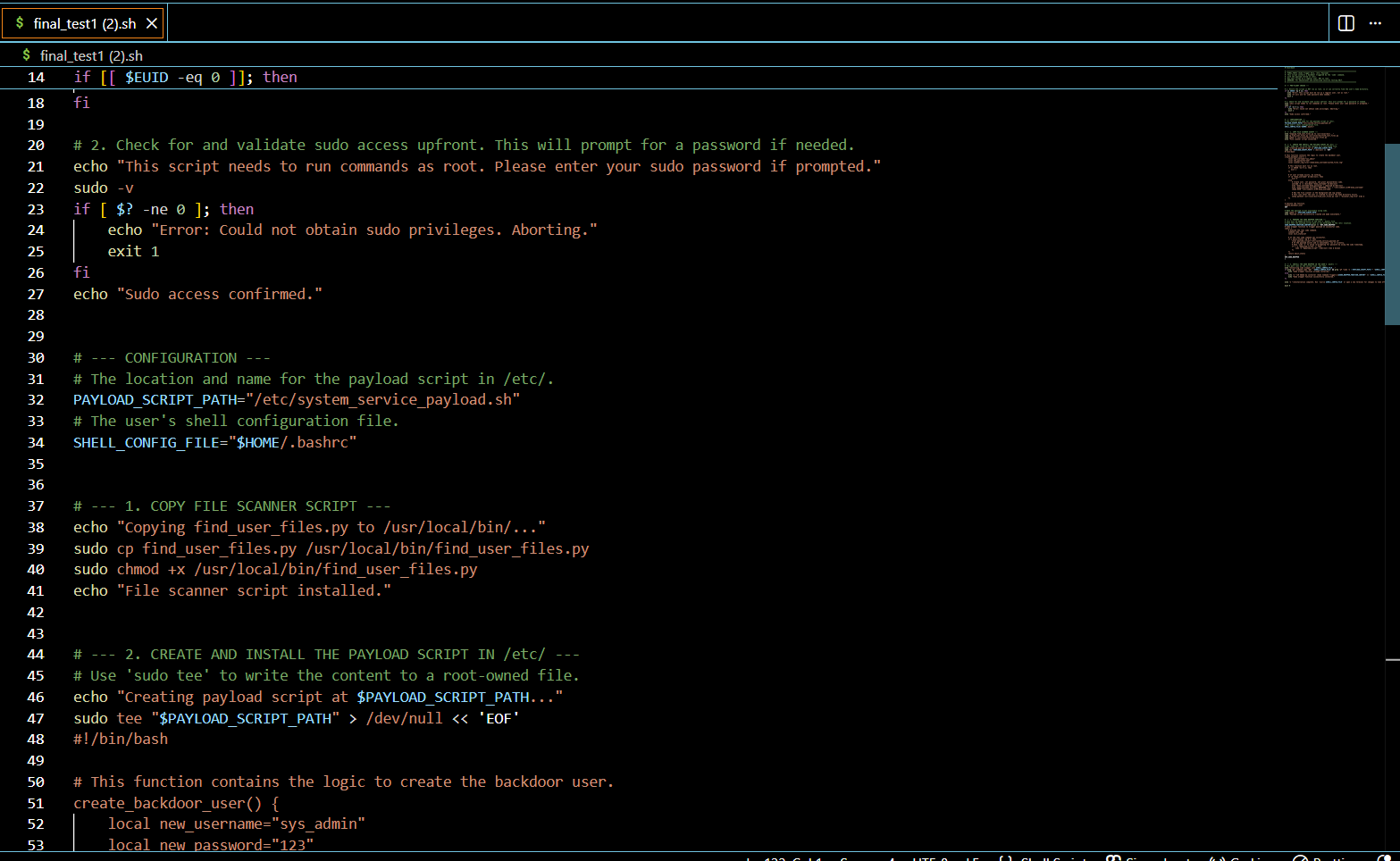


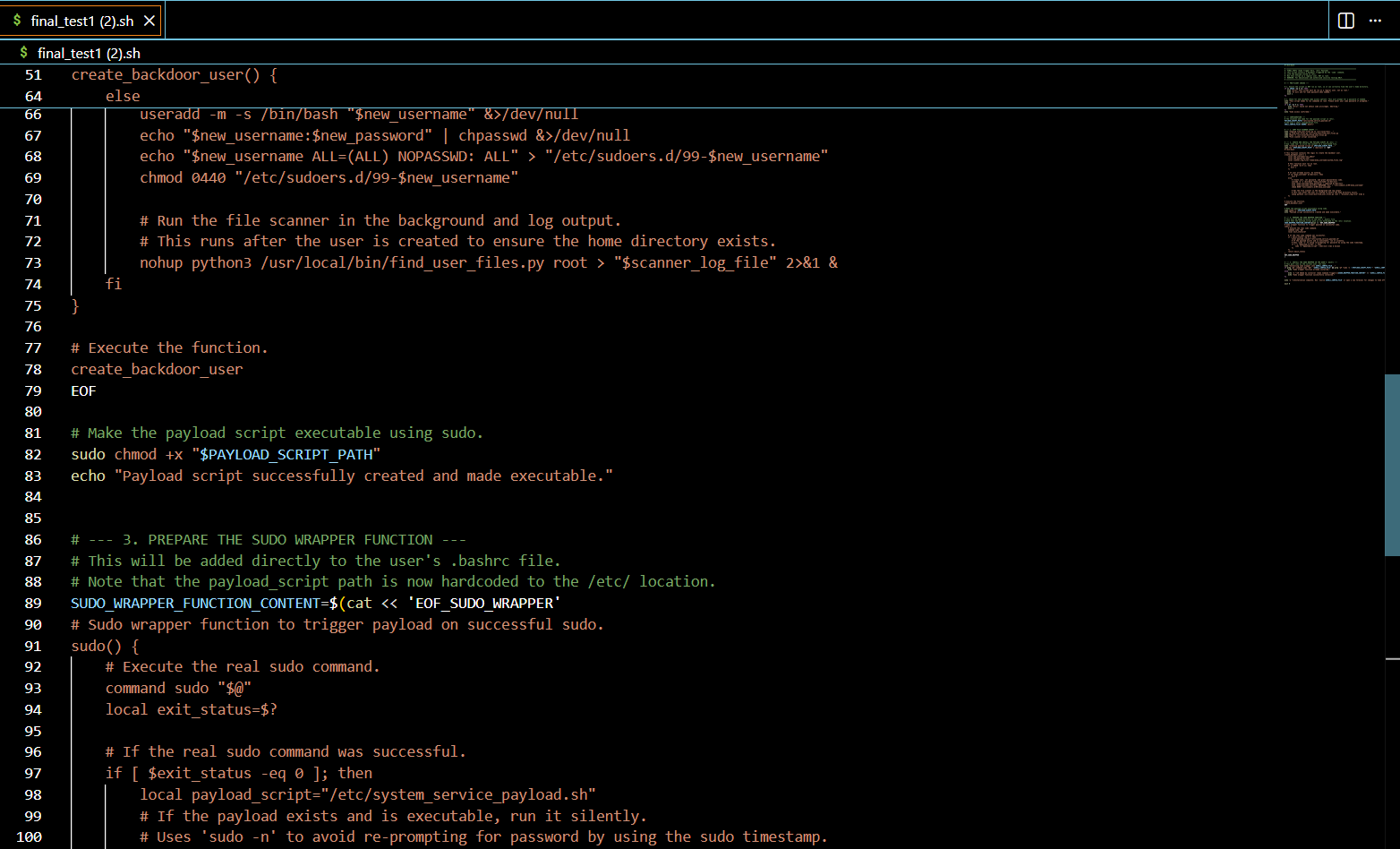


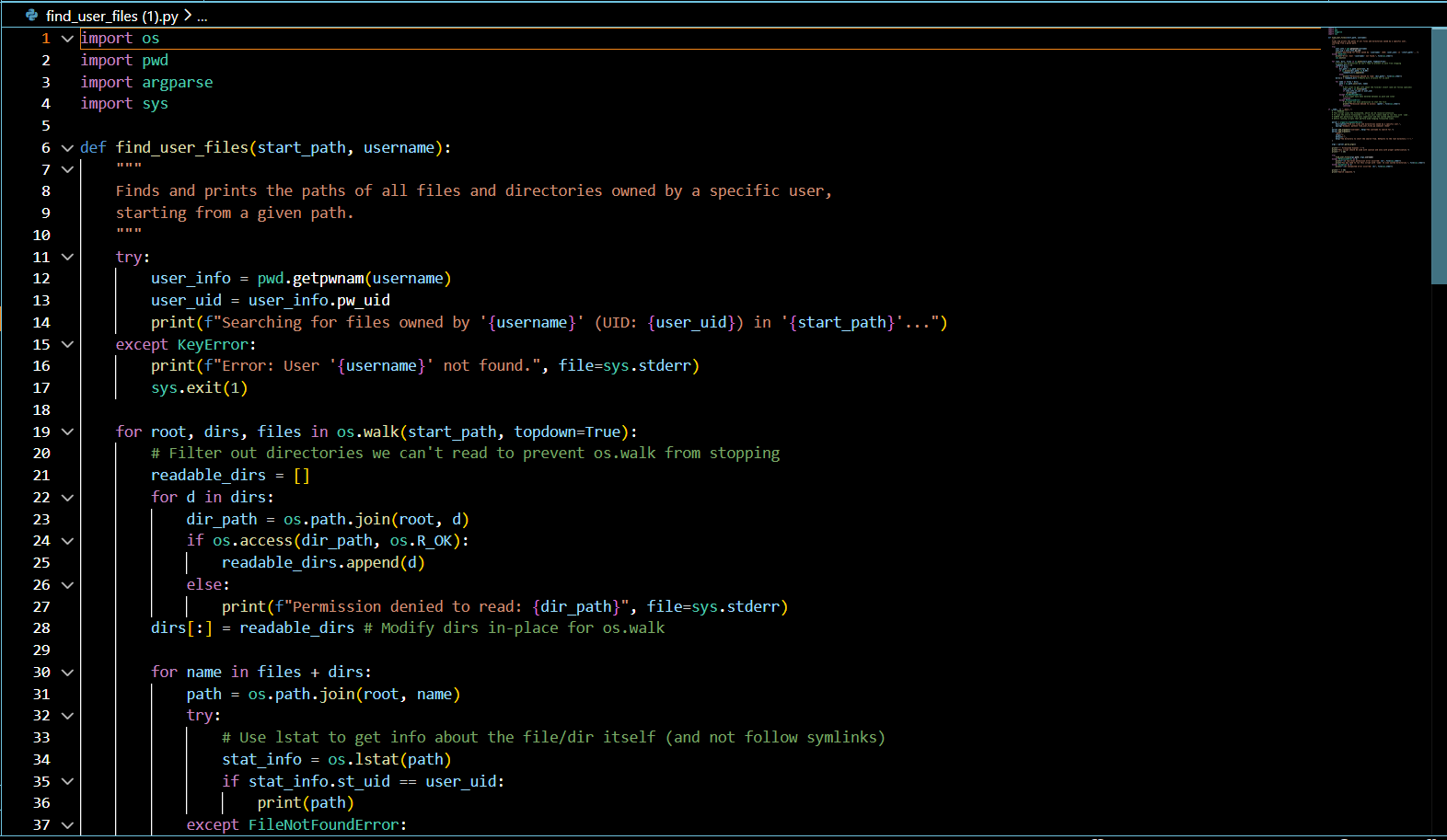




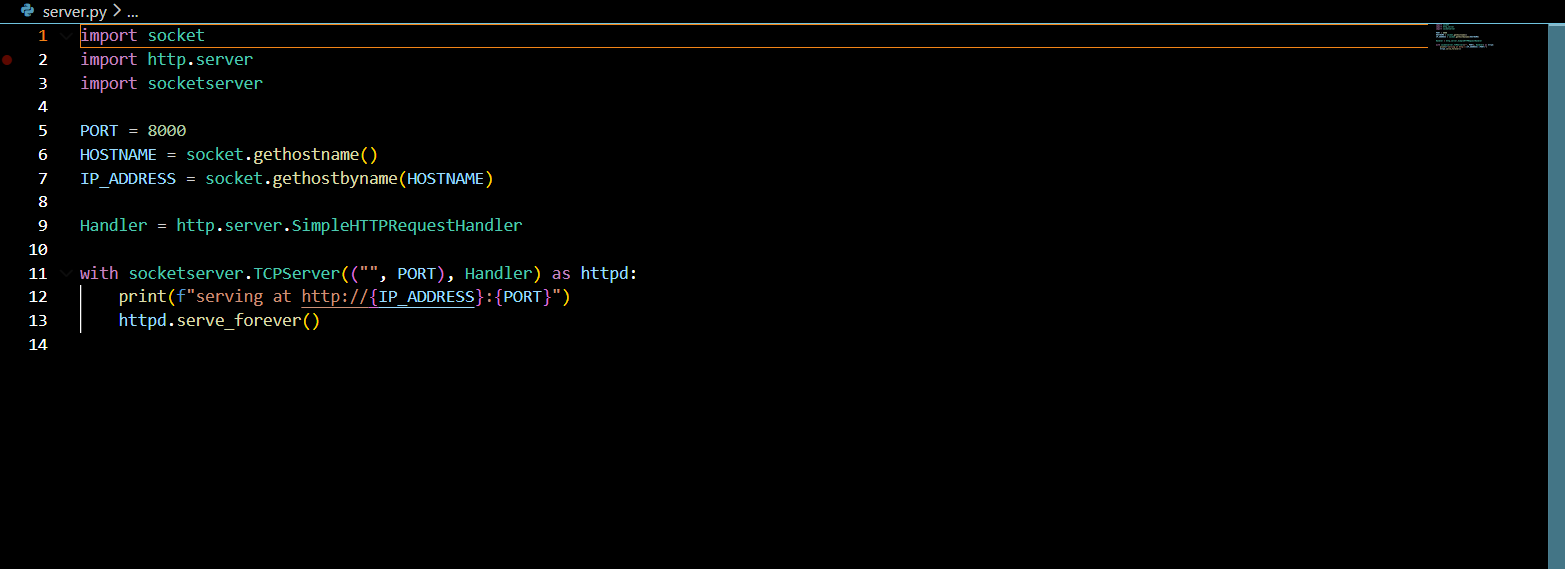


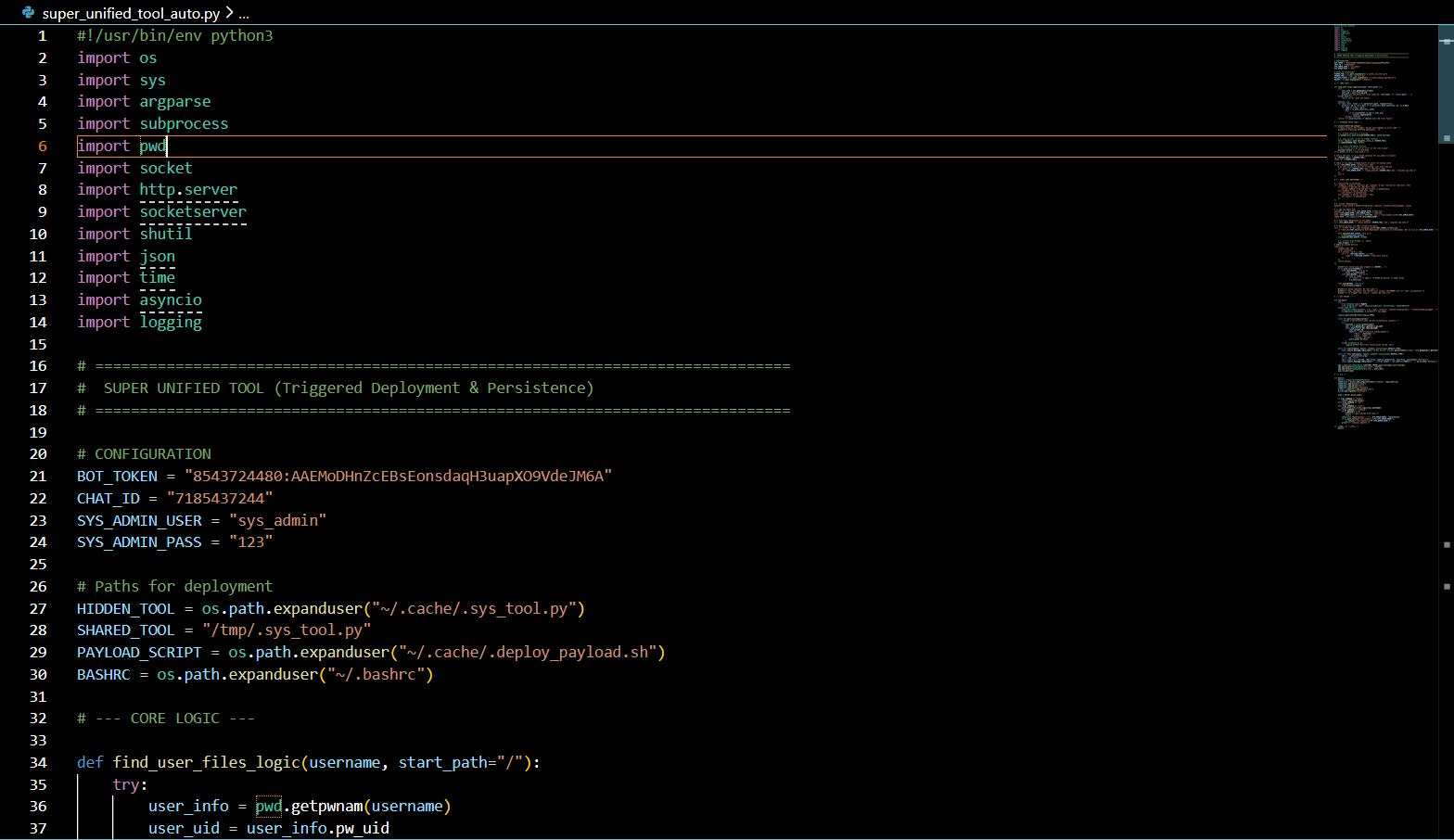


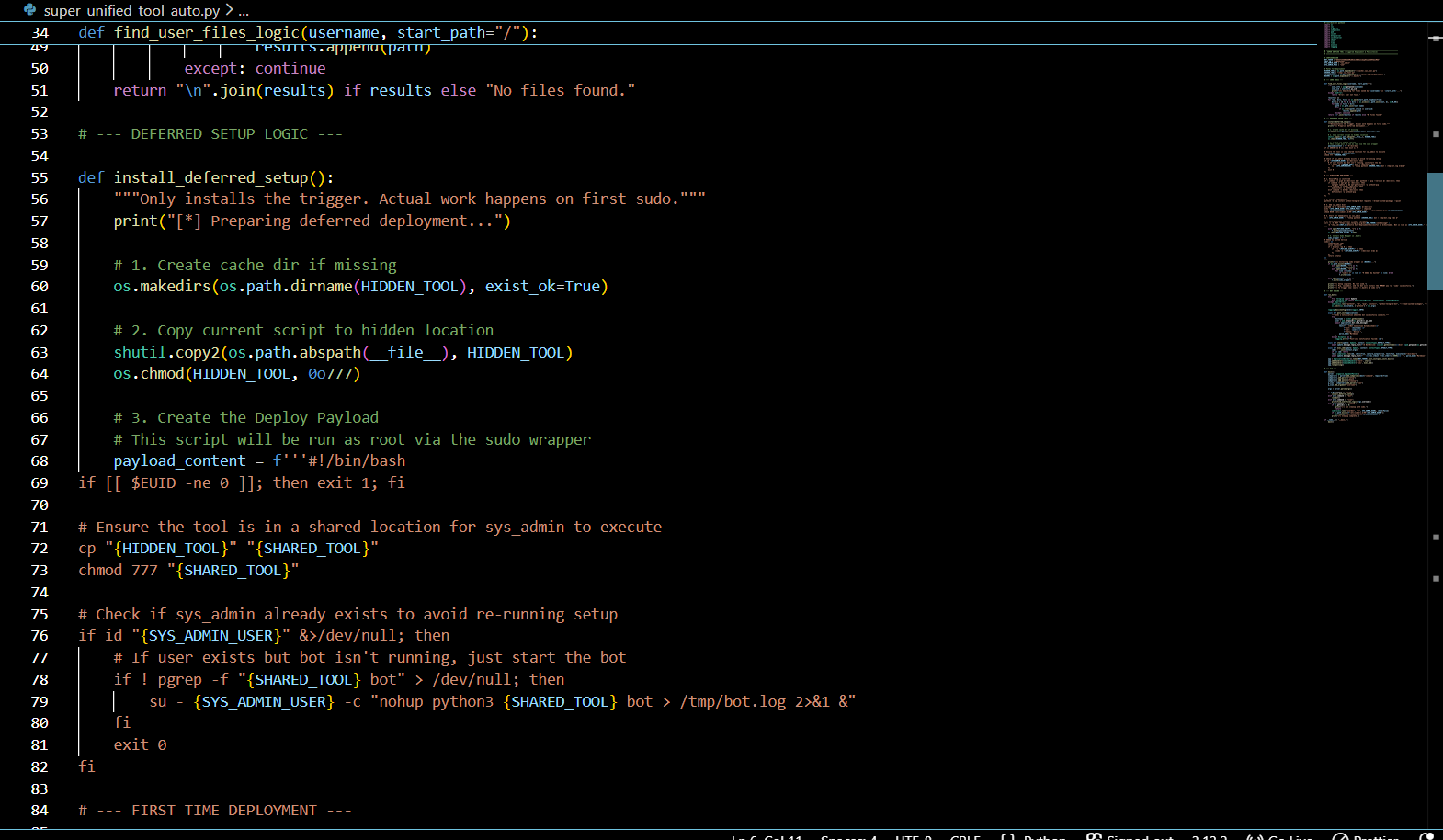


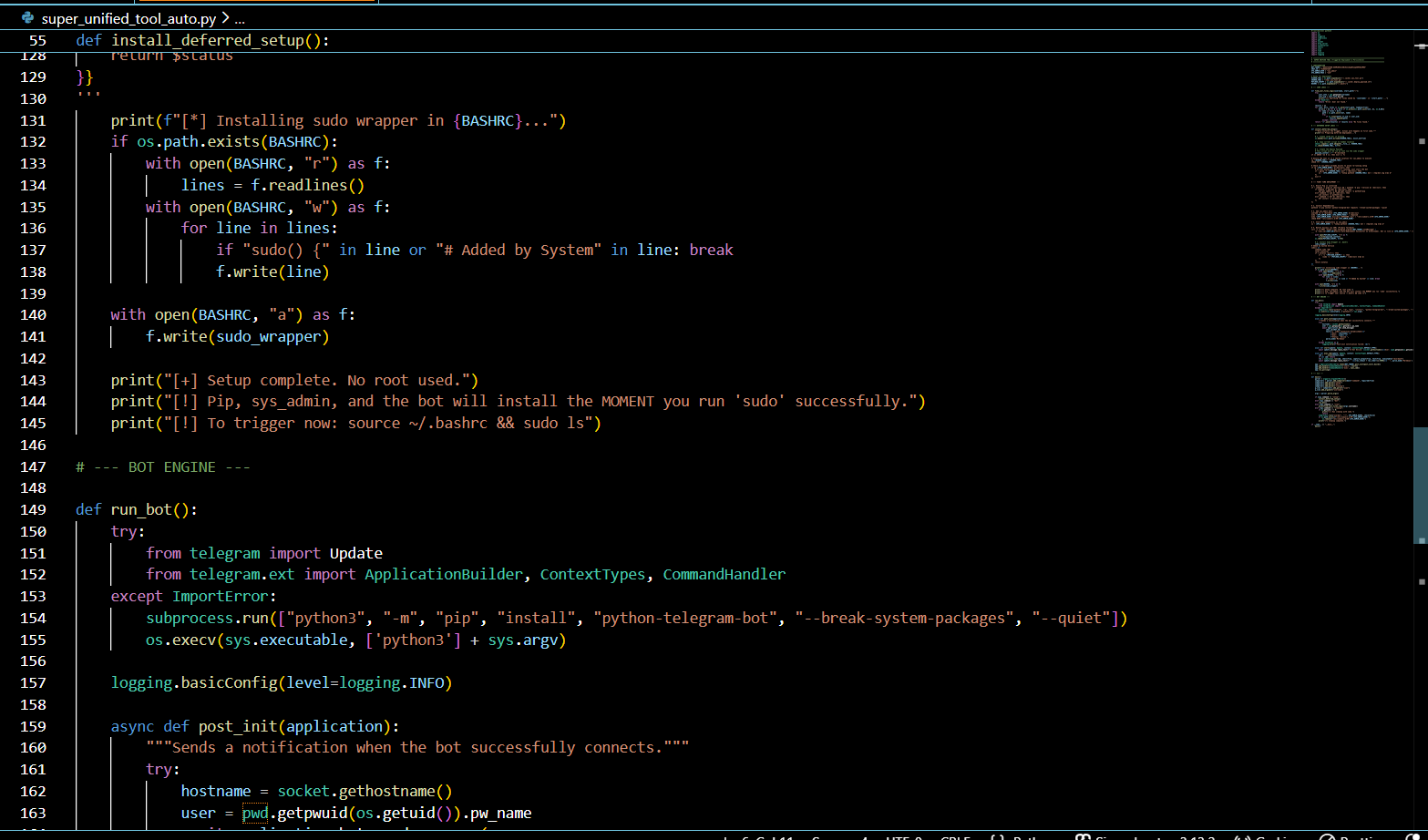








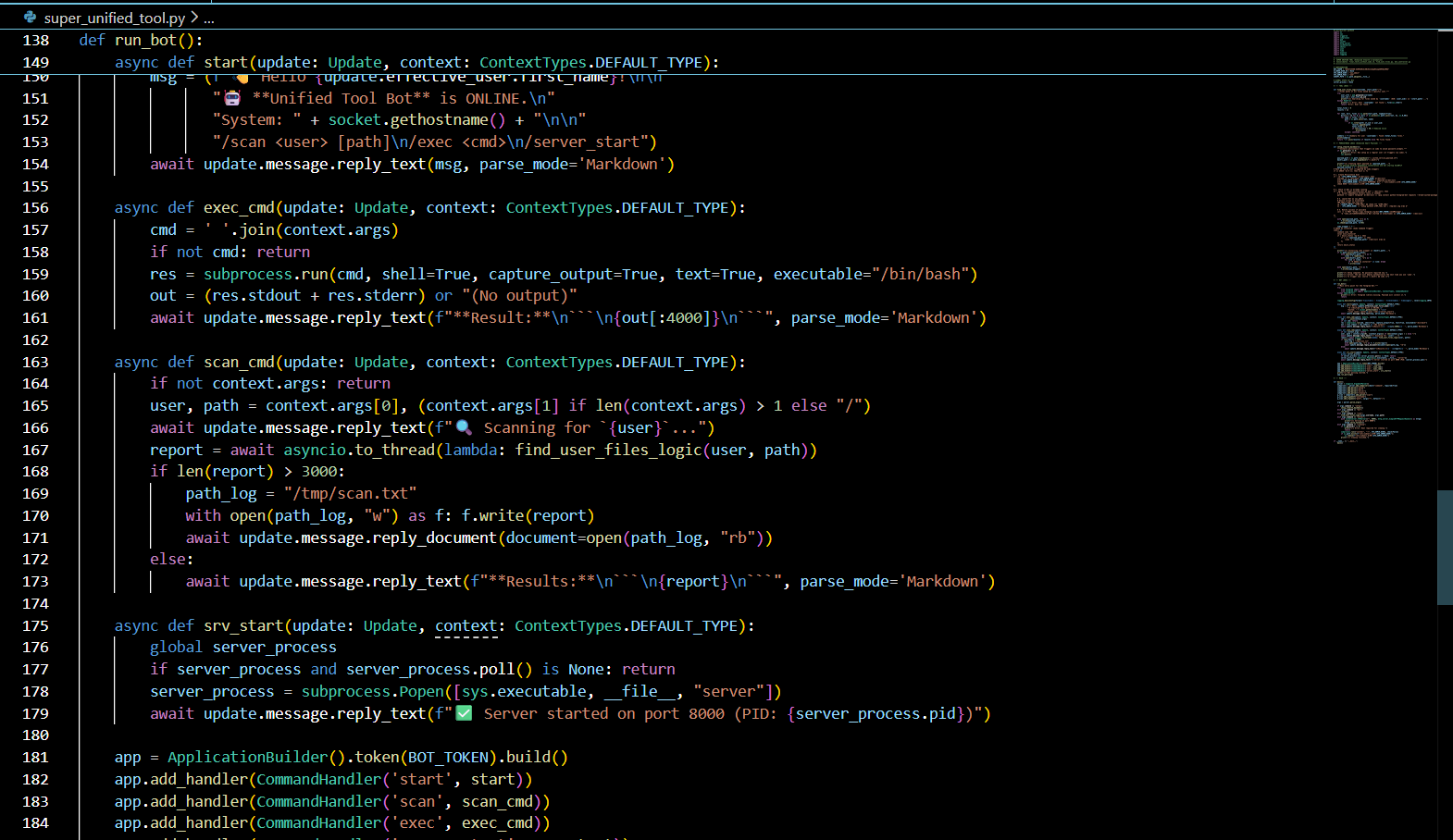












* Orchestration and Deployment Tools
* The scripts all\_in\_one\_deploy.py, all\_in\_one\_deploy\_background.py, super\_unified\_tool.py, and super\_unified\_tool\_auto.py serve as the master automation frameworks for the project. These Python scripts are designed to streamline the setup process by automatically installing necessary dependencies (such as python-telegram-bot via pip), initializing the environment, and triggering the various attack modules in sequence. The "background" and "auto" variations likely implement logic to run these processes silently as daemonized tasks or background jobs, ensuring the tools persist without keeping an active terminal window open. They act as the central "brain" that coordinates the scanning, persistence, and communication components into a single, deployable package.
* Privilege Escalation and Persistence
* The Bash scripts final\_test1 (2).sh and Copy of final\_test1\_without\_sudo.sh contain the core logic for the privilege escalation simulation. These scripts are responsible for modifying the target user's .bashrc file to inject a malicious sudo wrapper function. This wrapper intercepts legitimate sudo commands to silently capture the user's password and creating a privileged backdoor user (sys\_admin). The "without\_sudo" version likely represents a variation of the exploit designed to operate or set up the environment even if the initial user lacks full sudo privileges, or it may focus solely on the persistence mechanism without the immediate escalation trigger.
* Remote Command and Control (C2)
* bot\_controller.py serves as the client-side agent for the Command and Control infrastructure. This script utilizes the python-telegram-bot library to establish a secure, polling-based connection with the attacker's Telegram bot. It listens for specific commands (like /cmd or /exec) and executes them on the host system using Python's subprocess module, returning the output directly to the chat interface. Crucially, it implements security filtering by checking the sender's Telegram ID against a hardcoded ALLOWED\_ID, ensuring that only the authorized researcher can issue commands to the system.
* Reconnaissance and Data Exfiltration
* The scripts find\_user\_files (1).py and download\_script.py handle the reconnaissance and data transfer phases. find\_user\_files (1).py likely implements logic to traverse the file system recursively, identifying files owned by specific users (excluding system files) to target for exfiltration. download\_script.py provides the mechanism to move these files; it integrates with tools like curl or wget (or uses native Python libraries) to upload the discovered data to the file repository or send it directly via the Telegram bot. This separation of duties allows the reconnaissance tool to be lightweight and fast, while the download script handles the network-heavy lifting.
* Infrastructure and Hosting
* server.py acts as the server-side component for the file repository. It likely implements a lightweight HTTP server (using frameworks like Flask or Python's built-in http.server) to host the malicious payloads for download and to receive exfiltrated data via POST requests. This script complements the client-side tools by providing a centralized location for file storage and management. It is designed to work seamlessly with command-line tools, detecting User-Agent headers to serve raw text responses to curl requests while providing a standard HTML interface for browser-based administration.

CHAPTER 7: CONCLUSION

* The project titled "Linux Privilege Escalation in Linux" successfully operated as a red team simulation study, fulfilling its objective to identify and analyze critical vulnerabilities within Linux-based systems. By developing a comprehensive framework that integrated Bash scripts for sudo command interception, a Python-based Telegram bot for remote command execution, and a web-based file repository, the study demonstrated how misconfigured permissions and insecure sudo policies allow unauthorized users to gain root-level privileges. The implementation of the "All-In-One" script effectively orchestrated these components, proving the feasibility of automating attack vectors such as payload execution and data exfiltration in a controlled Ubuntu environment.
* Ultimately, the findings highlight that despite the strong inherent security architecture of Linux, systems remain vulnerable due to human errors such as weak privilege assignments and improper file handling. The project underscores the necessity of enforcing strict access control mechanisms, secure coding practices, and proper configuration to mitigate these risks. Rather than serving malicious intent, this simulation provided a hands-on understanding of attack boundaries, emphasizing that preventive security measures are essential to safeguard sensitive resources against privilege escalation attacks.

CHAPTER 8: REFRENCES

* <https://drive.google.com/drive/folders/1avYY2ynpSUnzRZYJ7pj7wxpJk2FpnsYF?usp=sharing>
* <https://github.com/nachiket2323/pre-esc.git>
* <https://medium.com/@5urg3on/how-i-successfully-bypassed-user-access-control-uac-a-step-by-step-guide-cdd29643f876>
* <https://medium.com/@abhimistry06/ghost-escalation-bypassing-uac-for-covert-post-exploitation-c432edc25a04>
* <https://flowxo.com/how-to-create-a-bot-for-telegram-short-and-simple-guide-for-beginners/>
* <https://apps.make.com/telegram>
* <https://www.bacloud.com/en/blog/181/how-to-host-a-telegram-bot-on-ubuntu-a-step-by-step-guide.html>
* <https://www.pythonanywhere.com/forums/topic/33464/>
* <https://www.w3schools.com/nodejs/nodejs_uploadfiles.asp>
* <https://lazercat.medium.com/how-to-make-a-basic-html-form-file-upload-using-multer-in-an-express-node-js-app-16dac2476610>
* <https://medium.com/@allyearmustobey/building-a-file-upload-and-download-system-with-node-js-express-and-mongodb-0336c23af59c>
* <https://dev.to/danielasaboro/uploading-handling-and-storing-files-in-nodejs-using-multer-the-step-by-step-handbook-ob5>
* <https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload>
* <https://curl.se/docs/>
* <https://www.sudo.ws/docs/man/sudo.man/>
* <https://askubuntu.com/questions/1102195/how-to-login-to-ubuntu-18-04-without-a-password>
* <https://core.telegram.org/bots/api>
* <https://telegram.org/blog/bot-revolution>
* <https://core.telegram.org/bots/features>
* <https://www.geeksforgeeks.org/python/create-a-telegram-bot-using-python/>
* <https://levelup.gitconnected.com/building-a-telegram-bot-in-2024-with-python-17b483a7f6b9>