НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
Завідувач кафедри

_____ **Дмитро ЛАНДЕ**
«_____»_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»
спеціальності 125 «Кібербезпека»

на тему: Забезпечення безпеки інфраструктури органазації при впровадженні IP-телефонії.

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-02
Бойко Тетяна Ярославівна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент кафердри ІБ к.т.н. Ткач В. М. _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент доцент кафедри ММАД к.ф.-м.н. Южакова Г.О.. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2024 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Дмитро ЛАНДЕ
(підпис)

«___»_____2024 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Бойко Тетяна Ярославівна

1. Тема роботи: Забезпечення безпеки інфраструктури органазації при впровадженні IP-телефонії,

керівник роботи Ткач Володимир Миколайович, кандидат наук, доцент кафедри інформаційної безпеки,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2024 р. № 2251-с

2. Термін подання здобувачем вищої освіти роботи 14 червня 2024 р.
3. Вихідні дані до роботи: наукова література по системах VoIP, загрозах IP-телефонії, методах аутентифікації, аналізу трафіку.
4. Зміст роботи: огляд технології IP-телефонії та основних протоколів і програмних реалізацій, огляд основних загроз, опис реалізації програмного рішення для виявлення спроб вторгнення у протоколі SIP.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):

Презентація до захисту дипломної роботи.

6. Дата видачі завдання: 28.11.2023

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Отримання завдання та узгодження теми роботи	28.11.2023	Виконано
2	Огляд наукової літератури по темі	28.11.2023 - 10.01.2024	Виконано
3	Дослідження роботи систем зв’язку VoIP	11.01.2024 - 29.02.2024	Виконано
4	Аналіз загроз застосунків для VoIP	01.03.2024 - 31.03.2024	Виконано
5	Проектування системи і впровадження VoIP	01.04.2024 – 20.04.2024	Виконано
6	Розробка веб-застосунку для VoIP з убезпеченням від потенційних загроз	21.04.2024 - 05.05.2024	Виконано
7	Розробка засобу моніторингу та виявлення інцидентів безпеки	06.05.2024 - 19.05.2024	Виконано
8	Аналіз результатів	20.05.2024 - 10.06.2024	Виконано

Здобувач вищої освіти _____ Тетяна БОЙКО

Керівник роботи _____ Володимир ТКАЧ

РЕФЕРАТ

Обсяг дипломної роботи 43 сторінки, 13 ілюстрацій, 3 додатки і 9 джерел літератури.

Об’єкт дослідження: IT-інфраструктура організації.

Предмет дослідження: методи та механізми забезпечення безпеки VoIP-систем від основних загроз та атак.

Мета дослідження: розробити та впровадити комплексне рішення для забезпечення безпеки корпоративної VoIP-інфраструктури

Методи дослідження: аналіз літературних джерел, ознайомлення з існуючими рішеннями для організації безпеки VoIP, практичне налаштування серверу Asterisk та розробка програмного забезпечення для моніторингу і виявлення інцидентів на основі Python.

Отримані результати: визначено ключові загрози для VoIP-інфраструктури, налаштовано VoIP-сервер Asterisk, творено веб-застосунок на основі Flask з системою аутентифікації та авторизації користувачів, розроблено SIP IDS фреймворк для моніторингу SIP-трафіку в реальному часі, виявлення та блокування підозрілої активності та проведена інтеграція розробленої системи із системою моніторингу Elasticsearch.

Ключові слова: VoIP, IP-телефонія, Asterisk, кібербезпека, веб-застосунок, SIP IDS, Elasticsearch.

ABSTRACT

The volume of the thesis is 43 pages, 13 illustrations, 3 appendices, and 9 literature sources.

Object of research: IT infrastructure of an organization.

Subject of research: methods and mechanisms for ensuring the security of VoIP systems from major threats and attacks.

Purpose of research: to develop and implement a comprehensive solution for ensuring the security of corporate VoIP infrastructure.

Research methods: analysis of literary sources, familiarization with existing solutions for organizing VoIP security, practical configuration of the Asterisk server, and development of software for monitoring and incident detection based on Python.

Obtained results: key threats to VoIP infrastructure were identified, a VoIP server Asterisk was configured, a web application based on Flask with a user authentication and authorization system was created, a SIP IDS framework for real-time monitoring of SIP traffic, detection, and blocking of suspicious activity was developed, and the developed system was integrated with Elasticsearch.

Keywords: VoIP, IP telephony, Asterisk, cybersecurity, web application, SIP IDS, Elasticsearch.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DoS (Denial of Service) – відмова в обслуговуванні.

DDoS (Distributed Denial of Service) – розподілена відмова в обслуговуванні.

IP (Internet Protocol) - протокол передавання датаграм в мережах.

VoIP (Voice over Internet Protocol) – технологія передачі голосового трафіку через Internet Protocol.

SIP (Session Initiation Protocol) – протокол ініціації сеансів.

RTP (Real-time Transfer Protocol) – протокол передавання аудіо та відео через мережі IP.

IDS (Intrusion Detection System) – програмне забезпечення для виявлення несанкціонованого доступу до системи.

JWT (JSON Web Token) – веб-токен у форматі JSON для аутентифікації та авторизації.

Токен (Token) - цифровий ключ, маркер доступу користувача.

IP-телефонія – технологія здійснення голосових викликів через мережу IP.

Реєстрація SIP – процес, при якому користувач IP-телефонії реєструється на SIP-сервері для отримання послуг.

SIP-діалог – з’єднання між вузлами сеансу для обміну повідомленнями.

PSTN (Public Switched Telephone Network) – радиційна телефонна мережа, яка надає послуги стаціонарного телефонного зв'язку та використовується для голосових комунікацій.

GSM (Global System for Mobile Communications) – стандарт мобільного зв'язку.

PBX (Private Branch Exchange) – приватна телефонна мережа, що використовується в межах організації, яка дозволяє здійснювати внутрішні комунікації та управляти вхідними і вихідними дзвінками.

Сніфер (Sniffer) - програма або пристрій, що аналізує мережевий трафік.

Фрод (Fraud) - шахрайство або зловживання ресурсами системи.

ВСТУП

Для організації корпоративного зв’язку серед підприємств дедалі поширенішою стає технологія VoIP. VoIP дозволяє здійснювати голосові дзвінки через Інтернет, що забезпечує значні переваги: зниження витрат на зв'язок, інтеграцію з іншими бізнес-системами, гнучкість у налаштуваннях. Однак разом із цими перевагами з'являються й нові виклики, пов'язані з кібербезпекою. Зокрема, VoIP-системи стають об'єктом атак зловмисників, які намагаються отримати несанкціонований доступ, використовувати brute-force атаки, SQL-ін'єкції, DoS атаки та інші методи компрометації систем. Це може призвести до втрати конфіденційних даних, зниження продуктивності та значних фінансових збитків. Тому розробка ефективних методів захисту VoIP-інфраструктури є важливим завданням для забезпечення безпеки організацій.

Метою роботи є визначення основних напрямків загроз для VoIP системи, що розгортається в мережі організації, розробка комплексного рішення для протидії визначеним загрозам, розробка системи моніторингу та виявлення інцидентів безпеки в реальному часі для SIP-трафіку та інтеграція із системою моніторингу.

Об’єктом дослідження є IT-інфраструктура організації, яка включає VoIP-сервер, веб-сервер та бази даних.

Предметом дослідження є методи та механізми забезпечення безпеки VoIP-систем, зокрема захист від поширених загроз та атак, таких як несанкціонований доступ, brute-force атаки, SQL-ін'єкції, DoS атаки, а також інші методи компрометації системи.

Методи дослідження: аналіз літературних джерел, ознайомлення з існуючими рішеннями для організації безпеки VoIP, практичне налаштування

серверу Asterisk та розробка програмного забезпечення для моніторингу і виявлення інцидентів на основі Python.

Практичне застосування: розроблене рішення забезпечує комплексний захист корпоративної VoIP-інфраструктури від несанкціонованого доступу, brute-force атак, SQL-ін'єкцій, DoS атак та інших загроз. Веб-застосунок, розроблений у межах роботи, забезпечує зручний інтерфейс для користувачів, а SIP IDS фреймворк дозволяє виявляти та блокувати підозрілу активність в реальному часі. Розроблена система IDS може бути впроваджена в будь-якій організації, яка використовує IP-телефонію, для забезпечення комплексного захисту VoIP-інфраструктури від поширених загроз.

1 ТЕХНОЛОГІЇ VOIP ТА SIP

Voice over IP (VoIP) - це технологія, яка дозволяє передавати голосові та мультимедійні дані у вигляді пакетів через приватну або загальнодоступну IP-мережу. Вона стала популярною альтернативою традиційним телефонним мережам загального користування (PSTN) завдяки своїм численним перевагам. Серед них можна відзначити значно нижчу вартість дзвінків, особливо міжнародних, а також можливість інтеграції з різними сторонніми сервісами та додатками, що робить цю технологію надзвичайно гнучкою та функціональною.

Однією з ключових причин чому організації обирають системи VoIP, а не дзвінки через месенджери чи соціальні мережі, є їх здатність інтегруватися з іншими системами та додатками, такими як CRM-системи, системи управління контактами, електронна пошта та інші бізнес-інструменти. Це дозволяє забезпечити високий рівень взаємодії та автоматизації бізнес-процесів, що в свою чергу підвищує ефективність роботи організацій.

IP-телефонія – технологія здійснення телефонних дзвінків з допомогою систем VoIP. Організації обирають впровадження IP-телефонії у власну інфраструктуру, оскільки її легко масштабувати, вона дозволяє впроваджувати гнучкі налаштування маршрутизації дзвінків, здійснювати записи дзвінків, з допомогою IP-телефонії працівники організації можуть мати доступ до корпоративних дзвінків у будь-якій точці світу, де є доступ до інтернету.

IP-телефонія організаціями зазвичай розміщується на власних серверах, оскільки вона не створює значного навантаження на всю систему. Це надає значну економію коштів, проте вносить додаткові ризики для кібербезпеки інфраструктури мережі організації, тож вимагає ретельного підходу до забезпечення безпеки.

1.1 Основні принципи роботи VoIP

Здебільшого, VoIP дзвінки використовують і Інтернет, і PSTN (телефонну мережу загального користування) та GSM (мережу мобільного зв’язку) мережі, що вимагає медіашлюзів для конвертації аудіоданих. Для з’єднання між користувачами у VoIP мережі та абонентами GSM (звичними SIM-картками), на стороні провайдерів зв’язку здійснюється перетворення інтернет-трафіку в формат, сумісний для передачі кінцевому абоненту.

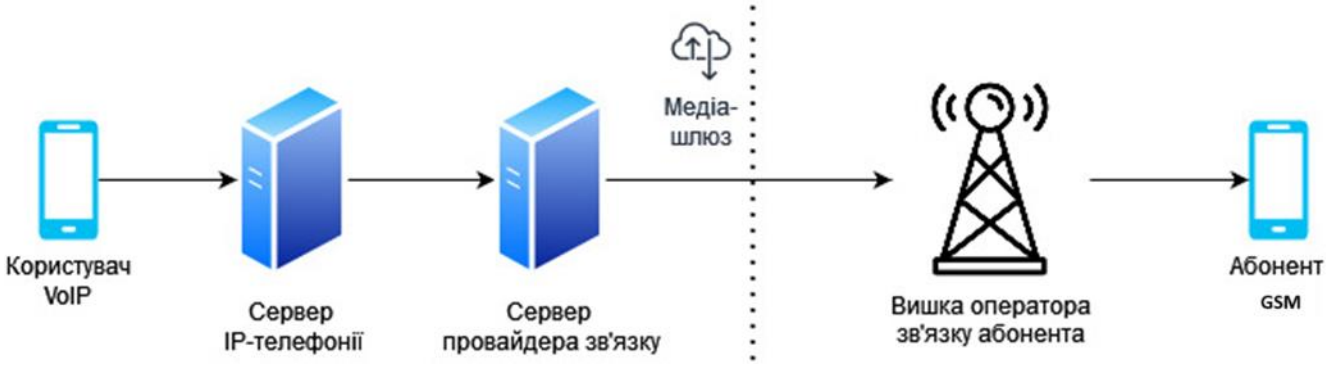

Рисунок 1.1 – Схема дзвінка із IP в GSM мережу

Для управління медіашлюзами між Інтернетом і телефонною мережею загального користування використовується протокол MGCP (Media Gateway Control Protocol).

У мережі VoIP голосовий трафік передається у цифровому форматі. Аналоговий сигнал перетворюється у цифровий з допомогою кодеків, які використовують у своїй роботі дискретизацію та квантування. Потім цифрові сигнали обробляються і стискаються для ефективної передачі через IP-мережу. Використання спеціальних кодеків, таких як G.711, G.729, G.722, забезпечує ефективну компресію без значної втрати якості звуку. Стиснуті дані розбиваються на невеликі пакети, та містять не тільки аудіоінформацію, а й контрольні дані, які потрібні для коректної передачі та відновлення інформації на стороні отримувача.

Пакети даних із аудіо, а також сигналізацією передаються через IP мережу

В IP-телефонії використовується ряд протоколів для встановлення, управління, забезпечення якості сеансів зв'язку, та передачі голосового трафіку.

До основних протоколів що використовуються відносяться: [getvoip]

- SIP (Session Initiation Protocol): протокол управління сигналізацією. Використовується для встановлення, управління і завершення голосових і відеодзвінків. SIP забезпечує ідентифікацію учасників дзвінка, визначає формат і порядок обміну повідомленнями між ними. Цей протокол наразі є найпоширенішим у IP-телефонії.

- H.323: був першочергово розроблений як стандарт для відеоконференцій та VoIP, проте наразі він стрімко витісняється протоколом SIP завдяки його простоті.

- IAX (Inter-Asterisk Exchange): протокол розроблений для VoIP системи Asterisk. Він ефективний при обході NAT та при обробці сервер-сервер зв'язків. IAX використовує один порт для передачі голосових та сеансових даних, що полегшує налаштування і знижує затримки.

- RTP (Real-time Transport Protocol): використовується для передачі аудіо та відео в реальному часі між кінцевими точками під час VoIP дзвінка.

- SDP (Session Description Protocol): використовується разом із SIP для обміну основною інформацією про сеанс, такою типи медіа, адреси кінцевих точок, використовувані кодеки в медіа.

- TCP та UDP: транспортні протоколи передачі даних в мережі IP. Здебільшого під час дзвінків використовується саме UDP, оскільки він є більш відповідним для потокової передачі інформації, проте використання TCP також можливе.

Оскільки основним та найпоширенішим протоколом є SIP, розглянемо його детальніше.

1.2 Принципи роботи протоколу SIP

Session Initiation Protocol є основним протоколом для встановлення, управління і завершення VoIP з'єднань.

SIP є протоколом прикладного рівня, який керує сеансом дзвінка. Учасниками дзвінка можуть бути як люди, так і «роботи», тобто при виклику абонента йому можна відтворити аудіофайл, або ж абонент може поговорити із штучним інтелектом.

SIP-запрошення, що використовуються для створення сеансів, містять опис сеансу, та дозволяє учасникам домовитися про набір сумісних типів медіа. SIP підтримує мобільність користувачів шляхом проксювання та перенаправлення запитів до поточного місцезнаходження користувача.

Ключові функції SIP:

- Ініціація та завершення сеансів: SIP повідомлення типу INVITE ініціюють дзвінки, а повідомлення типу BYE використовуються для завершення з'єднань. Ці повідомлення є основними для початку і завершення сеансів зв'язку, забезпечуючи зручний механізм для управління викликами.

- Ідентифікація учасників: SIP визначає та ідентифікує учасників дзвінка за допомогою SIP URI (Uniform Resource Identifier), що дозволяє визначити кінцеву точку дзвінка. Кожен учасник має унікальний SIP URI, що дозволяє точно ідентифікувати кожного користувача в мережі.

- Аутентифікація та безпека: SIP підтримує аутентифікацію користувачів і шифрування повідомлень для забезпечення безпеки сеансів зв'язку. Він може

використовувати механізми, такі як MD5, TLS і SRTP для забезпечення конфіденційності і захисту від несанкціонованого доступу.

- Сигналізація та управління дзвінками: SIP координує процес встановлення з'єднання між двома або більше сторонами, включаючи передачу повідомлень про стан з'єднання (наприклад, дзвінок у процесі, дзвінок на утриманні) і управління дзвінками (наприклад, перенаправлення дзвінків). Таким чином підтримується велика кількість сценаріїв взаємодії між користувачамм.

- Медіа-опис: використовується SDP для узгодження параметрів сеансу, таких як адреси медіа-серверів, медіа-кодеки, та інші параметри медіа, перед його початком.

- Контроль отримання повідомлень: на ключові повідомлення у сеансі сторона отримувача надсилає у відповідь повідомлення «200 ОК», сигналізуючи про отримання інформації.

- Підтримка мобільності: надає можливість користувачам отримувати дзвінки незалежно від їхнього фізичного розташування, використовуючи одну і ту ж SIP адресу. Для забезпечення цього користувачі можуть реєструвати свої SIP URI на сервері із будь-якого пристрою.

Реєстрація в SIP – це процес, за допомогою якого клієнт повідомляє сервер про своє місцезнаходження для можливості отримання вхідних дзвінків. У процесі реєстрації клієнт авторизується на сервері зі своїм логіном та паролем. Реєстрація здійснюється таким чином:

- Надсилання від клієнта на сервер повідомлення «REGISTER». Це повідомлення містить SIP URI користувача і поточну IP-адресу клієнта. Повідомлення «REGISTER» також може містити іншу інформацію, таку як порт, який використовує клієнт для прийому викликів.

- На це повідомлення сервер надсилає відповідь «401 Unauthorized», запитуючи аутентифікацію клієнта.
- Клієнт повторно надсилає «REGISTER», вказавши ім’я користувача і хеш пароля та підтверджуючи свою особу.
- В разі успішної реєстрації сервер надсилає відповідь «200 OK» та зберігає IP-адресу, на яку будуть маршрутизуватися дзвінки призначені цьому клієнту. При невдалій реєстрації клієнт отримає повторно відповідь «401 Unauthorized» та діалог між сервером та клієнтом буде завершений.
- З певною періодичністю буде відбуватися процес повторної реєстрації для збереження актуальної інформації про клієнта.

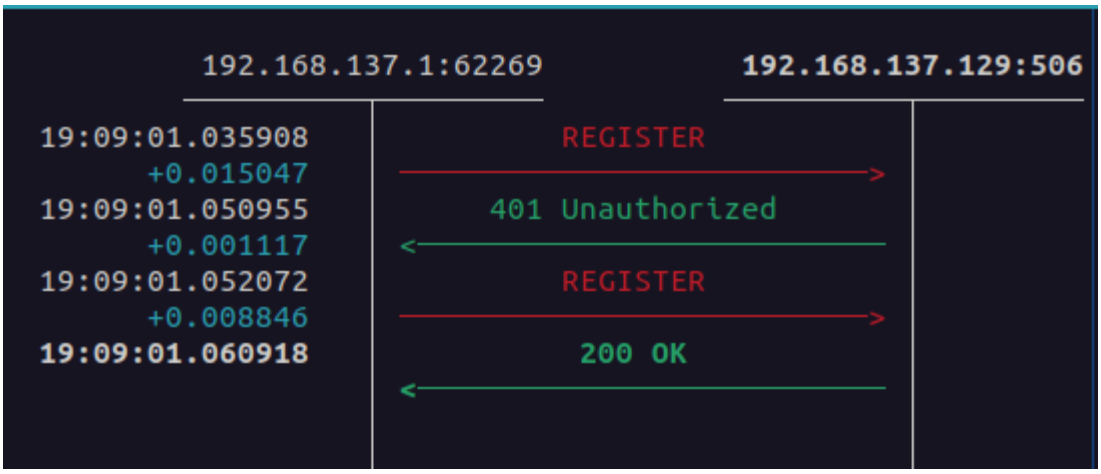

Рисунок 1.2 – Процес реєстрації клієнта на сервері.

Процес здійснення дзвінка та встановлення сесії між абонентами відбувається шляхом обміну сигналізаційними повідомленнями. Загалом, ініціація дзвінка здійснюється надсиланням SIP повідомлення «INVITE» клієнтом А (ініціатором) до абонента Б (отримувача). На це повідомлення в разі успішного з’єднання абонент А отримує відповідь «200 OK» від абонента Б. Абонент А підтверджує отримання відповіді надсилаючи «ACK» (Acknowledgement), після чого між користувачами розпочинається медіа-сеанс.

В процесі з’єднання сторона абонента А отримує проміжні відповіді від SIP-сервера, які описують процес обробки запиту, а саме: «100 Trying» - сигналізує

про початок обробки запиту на пошук абонента Б, «180 Ringing» - інформує про те, що абонента Б знайдено та він успішно отримав сигнал про виклик.

Для завершення сеансу будь-яка із сторін може надіслати повідомлення «BYE».

Для інформування про помилки під час встановлення з’єднання використовуються спеціальні інформаційні коди. Вони схожі до тих, що використовуються протоколом HTTP:

- 1xx - інформаційні відповіді:
  100 Trying: запит отримано, починається його обробка
  180 Ringing: кінцевий пристрій дзвонить
  181 Call is Being Forwarded: дзвінок перенаправлено (наприклад, на голосову пошту)
- 2xx - успішні відповіді:
  200 OK: запит успішно виконано
- 3xx - перенаправлення:
  301 Moved Permanently: запитуваний ресурс переміщений на постійну нову адресу.
  302 Moved Temporarily: запитуваний ресурс тимчасово переміщений на нову адресу.
- 4xx - клієнтські помилки:
  401 Unauthorized: необхідна аутентифікація
  404 Not Found: запитуваний ресурс не знайдено
- 5xx - серверні помилки:
  500 Internal Server Error: внутрішня помилка сервера
  503 Service Unavailable: сервіс недоступний
- 6xx - глобальні помилки:
  600 Busy Everywhere: усі кінцеві точки зайняті
  603 Decline: дзвінок відхилено.

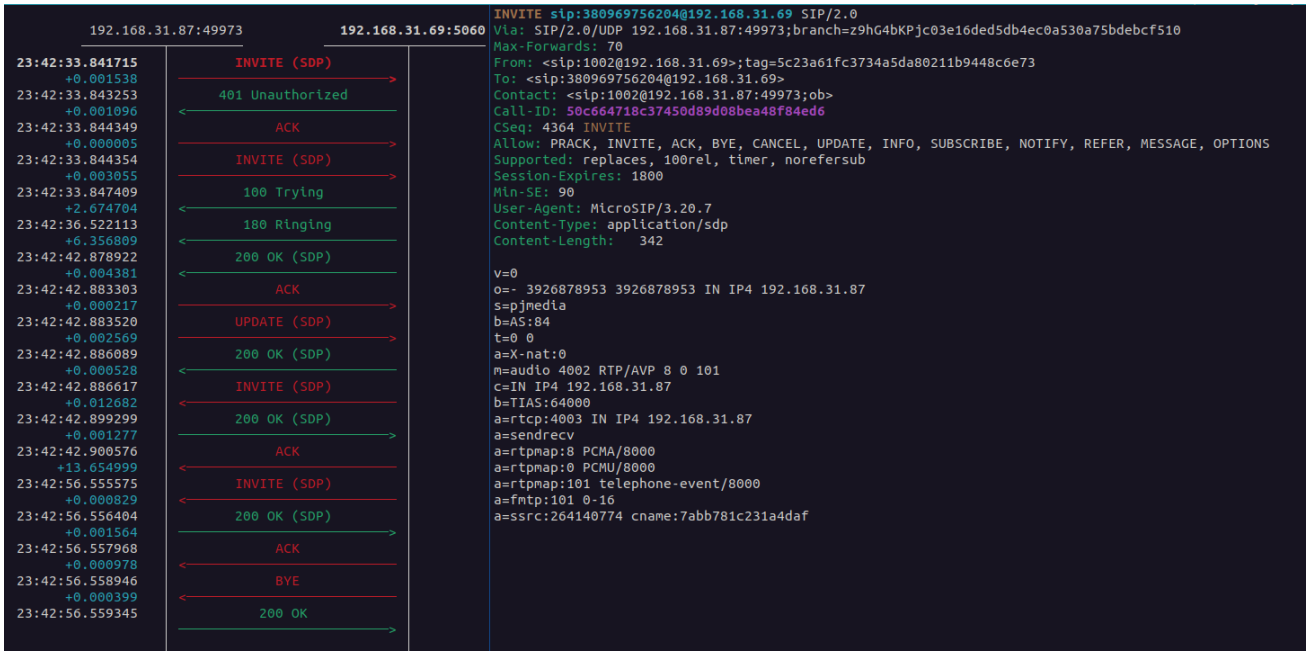

Рисунок 1.3 – Сеанс дзвінка між абонентами

Кожен сеанс має свій унікальний ідентифікатор – Call-ID. Він гарантує те, що повідомлення в рамках одного дзвінка будуть правильно співставлені. Call-ID включається у всі повідомлення, які надсилаються сторонами, та використовується сервером SIP для відстеження стану сеансу і маршрутизації повідомлень.

Отже, SIP є одним з ключових протоколів у сфері VoIP, забезпечуючи надійне встановлення, контроль та завершення сеансів зв'язку. Завдяки своїй гнучкості та розширюваності, SIP широко використовується для реалізації як простих телефонних дзвінків, так і складних мультимедійних конференцій. Його популярність обумовлена підтримкою різних типів медіа, інтеграцією з іншими інтернет-сервісами, можливістю забезпечення безпеки через TLS і SRTP, підтримкою мобільності користувачів. На основі цього протоколу були розроблені найпоширеніші рішення для серверів телефонії.

1.3 Рішення для серверів IP-телефонії (PBX)

Оскільки SIP є найпопулярнішим серед протоколів ініціації сеансів дзвінків, саме на його основі були розроблені найпоширеніші реалізації серверів телефонії. Розглянемо найпоширеніші з них.

Asterisk є одним з найбільш поширених і популярних SIP-серверів з відкритим вихідним кодом. Він використовується у всьому світі великими корпораціями та малими і середніми підприємствами. Завдяки своїй достусності, гнучкості, простоті та підтримці великої кількості функцій, Asterisk є основою для багатьох комерційних VoIP-продуктів. Він підтримує SIP через TLS для шифрування сигналізації та SRTP для шифрування медіа-трафіку. Використовує MD5 хешування для аутентифікації користувачів. Можливість інтеграції з fail2ban для запобігання атакам методом підбору паролів.

FreePBX - це надбудова над Asterisk, яка додатково надає веб-інтерфейс для управління PBX-системою. Він має популярність серед малих підприємств завдяки простоті використання та можливості розширення функціоналу за допомогою модулів.

Kamailio - потужний і універсальний SIP-проксі сервер, який має популярність у великих VoIP-інфраструктурах та серед провайдерів VoIP-послуг. Він може підтримувати стабільну роботу при великому навантаженні і може масштабуватися, тому його обирають для великих мереж.

Висновок до розділу 1

У цьому розділі ми детально розглянули основні принципи роботи VoIP, зокрема протокол SIP, який є основним для встановлення, управління і завершення VoIP-з'єднань. SIP, завдяки своїй гнучкості, масштабованості та безпеці, став ключовим елементом у сучасних системах IP-телефонії.

Основні технології для телефонії включають використання медіашлюзів для конвертації трафіку між Інтернетом і PSTN, а також застосування різних кодеків для ефективної передачі аудіо та відео. Протоколи, такі як SIP, H.323, IAX, RTP, SDP, TCP та UDP, забезпечують належну роботу VoIP-систем.

Розглянуті SIP-сервери, такі як Asterisk, FreePBX, Kamailio та OpenSIPS, пропонують різноманітні рішення для організацій різного масштабу. Asterisk та FreePBX відомі своєю зручністю та гнучкістю, що робить їх популярними серед малих та середніх підприємств. Kamailio та OpenSIPS, у свою чергу, використовуються у великих VoIP інфраструктурах завдяки своїй масштабованості та продуктивності. Загалом, SIP та пов'язані з ним технології є основою сучасних VoIP-систем, забезпечуючи ефективний та безпечний зв'язок.

У подальшому в цій роботі детально розглядатиметься система із впровадженням Asterisk, оскільки він є найпоширенішим SIP-сервером, який використовується не тільки як самостійний сервер в організаціях, а й на якому також базуються більшість розробок серед великих провайдерів IP-телефонії.

2 ВРАЗЛИВОСТІ І ЗАГРОЗИ ПРИ ВПРОВАДЖЕННІ VOIP

2.1 Загрози при взаємодії сервера телефонії з іншими сервісами в інфраструктурі.

При впровадженні VoIP систем, таких як Asterisk, існує ряд потенційних вразливостей і загроз, особливо коли сервер телефонії взаємодіє з іншими сервісами в інфраструктурі. Відсутність належного захисту може призвести до несанкціонованого доступу, перехоплення даних, зловживання ресурсами та інших атак.

Asterisk, підтримує кілька інтерфейсів для інтеграції та розширення функціональності. Два основних інтерфейси, які забезпечують широкі можливості для розробників і адміністраторів, це Asterisk Management Interface (AMI) та Asterisk Gateway Interface (AGI). AMI використовується для передачі зовнішнім застосункам інформації для контролю і моніторингу стану Asterisk. З допомогою AGI із зовнішніх додатків можна взаємодіяти із Asterisk, передавати йому запити та створювати події. AGI скрипти можуть бути написані на будь-якій мові програмування, що підтримує роботу зі стандартним вводом/виводом, таких як Perl, Python, PHP, Bash та інші. Зокрема, через доступ до цих ресурсів, зловмисник може отримати можливість підключитися до інших сервісів та серверів організації. Відсутність належного захисту може призвести до несанкціонованого доступу, перехоплення даних, зловживання ресурсами та інших атак.

Основні загрози що виникають:

- Несанкціонований доступ: якщо AMI інтерфейс не належним чином захищений, зловмисники можуть отримати доступ до управління сервером

Asterisk. Спричинити це може відсутність сильних паролів та відкритий доступ до AMI інтерфейсу з будь-якої IP-адреси

- Brute-force атаки: відсутність захисту від повторних спроб авторизації AMI-менеджера (системного користувача для AMI) може дозволити зловмиснику підібрати пароль та отримати доступ до AMI.
- Недостатня валідація команд: якщо AMI та AGI-скрипти не виконує належну валідацію команд, це може призвести до виконання небезпечних або некоректних команд, та дозволити проведення SQL-ін’єкцій. CVE-2018-7284 - вразливість дозволяє SQL-ін’єкції через некоректну обробку вхідних даних.
- Виконання довільного коду: зловмисники можуть використати вразливості AGI для виконання довільного коду на сервері. Зокрема, згідно із CVE-2017-1000383 зловмисник може виконувати довільний код через спеціально сформовані запити до AGI.

Сервери телефонії здебільшого інтегруються із іншими сервісами, такими як системи управління взаємодією з клієнтами (CRM), управління бізнес-процесами (ERP), платформами для командної взаємодії установи, а також месенджерами, електронною поштою, та веб-сервісами організації (рис. 2.1). Недостатня захищеність сервера телефонії може надати зловмиснику доступ до інших вузлів IT-інфраструктури організації.

Сервери телефонії також мають підвищену загрозу експлуатації зловмисником відомих вразливостей, оскільки дослідження показують, що величезна кількість організацій використовують застарілі версії Asterisk чи інших PBX. Така тенденція пов’язана з тим, що часто оновлення містить певні зміни в логіці роботи, які можуть призвести до збою у взаємодії з інтегрованими сервісами і вимагають значних витрат коштів та часу.

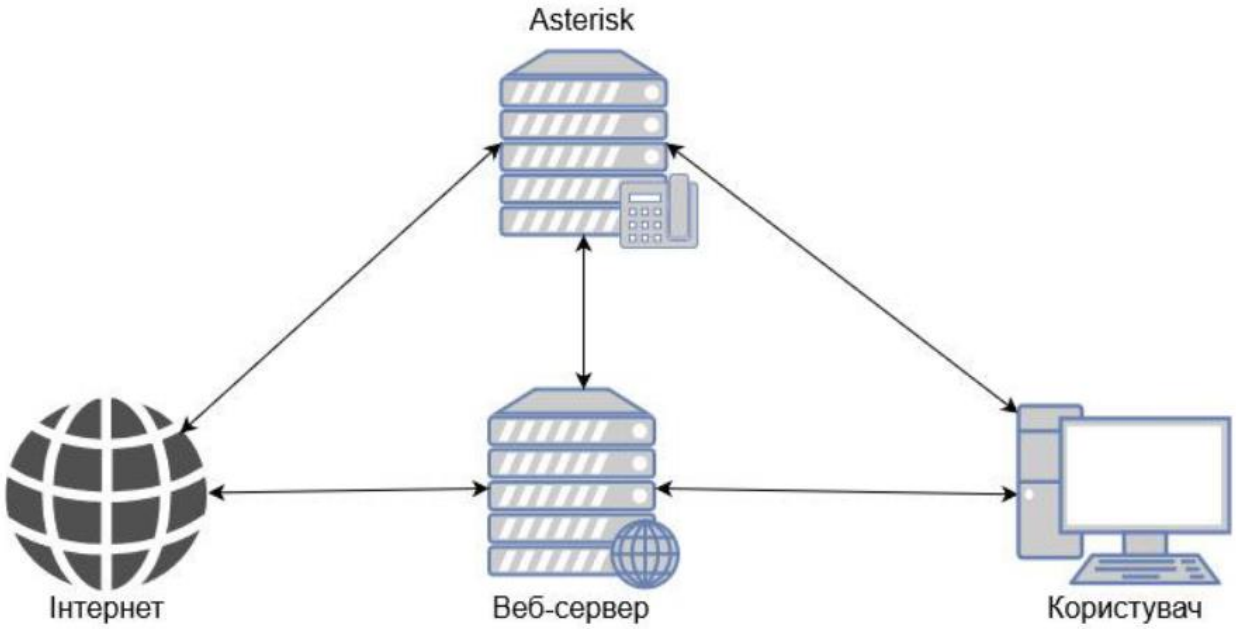

Рисунок 2.1 – Проста схема взаємодії Asterisk у внутрішній мережі

2.2 Загрози спричинені людським фактором

Людський фактор є однією з головних причин вразливостей у системах VoIP. Неправильні дії, недбалість або зловмисні наміри працівників можуть призвести до значних загроз безпеці та грошових втрат. Розглянемо основні загрози, пов'язані з людським фактором.

Шахрайство з тарифікацією (toll-fraud): зловживання корпоративними системами здійснення дзвінків на міжнародні або платні преміум-номери. Одним із найпоширеніших сценаріїв такого шахрайства є домовленість недобросовісного працівника із провайдером зв’язку компанії про організацію великої кількості трафіку на дорогі напрямки дзвінків. Таким чином компанія несе великі фінансові збитки, а провайдер та працівник отримують вигоду. Згідно з дослідженням Communications Fraud Control Association (CFCA), глобальні втрати від фроду у телекомунікаціях у 2021 році становили приблизно 39.89 мільярдів доларів США.

Одним із важливих активів для компаній є база контактів, із якими вона співпрацює, та часто вона стає ціллю для конкурентів. Недобросовісні працівники, маючи доступ до мобільних номерів клієнтів організації, отримують фінансову вигоду продаючи ці бази.

Фішинг через VoIP (vishing) — це різновид фішингових атак, де зловмисники використовують телефонні дзвінки для отримання конфіденційної інформації від жертв. Цей метод стає все більш популярним, оскільки він дозволяє обманювати людей, використовуючи голосову комунікацію, яка сприймається як більш достовірна порівняно з електронною поштою. Найпопулярнішими є випадки, коли зловмисник видає себе за представника банку чи ІТ-відділу, та виманює таким чином у жертви конфіденційну інформацію.

Причиною чому зловмисники використовують саме засоби IP-телефонії є можливість підробки Caller-ID, тобто шахраї представляються для абонента знайомим номером телефону, наприклад із його книги контактів. Отримавши дзвінок із знайомого номеру, жертва більш схильна до викриття конфіденційних даних, тому при підозрілому дзвінку зі знайомого номеру варто самостійно перетелефонувати на нього та переконатися в особі того, хто телефонує.

Таким чином можна відзначити, що людський фактор є однією з вагомих загроз, які виникають при використанні IP-телефонії, оскільки дуже складно технічно убезпечитися від такого виду шахрайства.

2.3 Вразливості протоколу SIP

Протокол SIP, будучи ключовим засобом у IP-телефонії, має низку вразливостей, які використовуються зловмисниками для атак на VoIP системи.

Види атак на SIP протокол:

- SQL-ін’єкції (CVE-2018-7284)
- Відмова в обслуговуванні (DoS) (CVE-2018-17764, CVE-2017-1000025)

- Віддалене виконання коду (RCE) (CVE-2019-7314)

Розглянемо детальніше кожну із вразливостей.

Ціллю здійснення SQL-ін’єкції є сервер SIP, який використовує базу даних для зберігання облікових записів користувачів SIP. Зловмисник може надіслати SIP-запит REGISTER або INVITE зі шкідливими SQL-командами, наприклад, у полі «From» або в іншому заголовку. Як результат, зловмисник може змінювати, видаляти або додавати дані до бази даних.

Відмова в обслуговуванні (DoS) – зловмисники можуть надіслати велику кількість SIP-запитів INVITE або REGISTER із великим обсягом даних або неправильними параметрами, що призводять до перевантаження або збою. Атаки здійснюються шляхом

Віддалене виконання коду (RCE) – вбудувавши виконуваний код або команди у заголовки чи тіло запитів SIP INVITE та MESSAGE, зловмисник може передати його для виконання на сервері.

Висновок до розділу 2

У цьому розділі були розглянуті основні загрози та вразливості, пов'язані з впровадженням VoIP систем, зокрема на прикладі сервера телефонії Asterisk. Сервери телефонії часто інтегруються з іншими сервісами в інфраструктурі організації, що створює додаткові ризики. Основними загрозами вважаємо несанкціонований доступ, brute-force атаки, недостатню валідацію команд, виконання довільного коду, а також можливість експлуатації відомих вразливостей через використання застарілих версій програмного забезпечення.

Вагомими є також загрози, спричинені людським фактором. Неправомірні дії або недбалість працівників можуть призвести до серйозних фінансових втрат, наприклад, через шахрайство з тарифікацією або фішингові атаки через VoIP. Такі

загрози є одними з найпоширеніших та найнепередбачуваніших, тому важливим є розмежування прав доступу користувачів до інформації, оскільки неможливо гарантовано убезпечитися від недобросовісних працівників.

Вразливості протоколу SIP показують, що зловмисники можуть використовувати SQL-ін'єкції, атаки на відмову в обслуговуванні (DoS) та віддалене виконання коду (RCE) для компрометації VoIP систем.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

У рамках даної роботи було проаналізовано найбільш поширені загрози для інфраструктури організації із імплементацією VoIP, у результаты чого розроблено комплексне рішення для забезпечення безпеки корпоративної VoIP-інфраструктури. Воно включає механізми захисту від поширених загроз та атак. Було реалізовано кілька ключових компонентів:

- Налаштування VoIP-сервера Asterisk - основою системи, що забезпечує функціональність VoIP у мережі. На сервері були налаштовані правила маршрутизації дзвінків, які мінімізують ризики фроду.
- Розробка веб-застосунку: для зручності користувачів було створено веб-застосунок, який дозволяє здійснювати та приймати дзвінки, а також управляти контактами. У застосунку реалізовано систему аутентифікації та авторизації на основі JWT (JSON Web Token), та забезпечено розмежування прав доступу для адміністратора та звичайних користувачів.
- Розробка SIP IDS (Intrusion Detection System) фреймворку: було створено спеціалізований для аналізу SIP-трафіку в реальному часі фреймворк. Розроблений фреймворк призначений для моніторингу трафіку з метою виявлення підозрілої активності та блокування IP-адрес, що порушують задані правила.

3.1 Встановлення і налаштування Asterisk

Першим етапом налаштування сервера Asterisk є підключення транкового з’єднання до зовнішнього провайдера зв’язку, що забезпечить можливість дзвінків за межами мережі. Також створимо внутрішніх користувачів своєї мережі. Це налаштування проводиться у конфігураційному файлі pjsip.conf.

Реєстрація з’єднання здійснюється шляхом створення сутності registration, яка зберігає інформацію для маршрутизації даних від Asterisk до сервера провайдера, та auth із даними для аутентифікації нашого користувача.

```
[mytrunk]
type=registration
transport=transport-udp
outbound_auth=mytrunk_auth
server_uri=sip:185.168.129.136
client_uri=sip:7642@185.168.129.136
contact_user=7642
retry_interval=60
expiration=3600
line=yes
endpoint=mytrunk

[mytrunk_auth]
type=auth
auth_type=userpass
password=****************
username=7642
realm=185.168.129.136
```

Рисунок 3.1 – Конфігурація з’єднання з провайдером

Внутрішні користувачі Asterisk створюються як сутності типу endpoint, де ми визначаємо яким чином буде працювати користувач (задаємо кодеки для медіа, визначаємо поведінку його внутрішнього номеру). Для внутрішніх користувачів також задається сутність auth.

```
[1002]
type=endpoint
transport=transport-udp
context=from-internal
disallow=all
allow=ulaw
allow=alaw
auth=1002
aors=1002
device_state_busy_at=1
allow_subscribe=yes
sub_min_expiry=30

[1002]
type=auth
auth_type=userpass
password==========
username=1002
```

Рисунок 3.2 – Конфігурація внутрішнього користувача

Також на сервері Asterisk налаштовуються контексти для маршрутизації дзвінків. Це правила, згідно яких сервер буде направляти дзвінок залежно від шаблону набраного номеру. Для убезпечення від шахрайства з тарифікацією, правила були налаштовані таким чином, щоб користувач не міг напряму із програмного телефону здійснити жоден виклик – дзвінки будуть доступні тільки через веб-інтерфейс та виключно на номери, які є у базі даних контактів. При цьому в нашому контексті додатково встановлено обмеження на дзвінки на будь-які номери, окрім українських (за виключенням платних номерів, що починаються на 380900). Цей функціонал налаштований у файлі extensions.conf.

```
[default]

; Заборона дзвінків на 380900
exten => _380900XXXXXX,1,NoOp(Calls to numbers 380900 are forbidden)
same => n,Hangup()

; Дозволені виклики на інші номери 380
exten => _380XXXXXXXXX,1,NoOp()
same => n,SET(CALLERID(num)=7642)
same => n,Dial(PJSIP/${EXTEN}@mytrunk,20)
same => n,Hangup()
```

Рисунок 3.3 – Конфігурація контексту викликів

3.2 Розробка веб-застосунку

Оскільки доступ для прямих дзвінків користувачам заборонений, надаємо їм таку можливість через веб-інтерфейс. Це слугуватиме одним із рівнів захисту як від внутрішніх загроз, так і від зовнішніх атак. Для бекенду застосунку використовується фреймворк Flask для Python (Додаток А).

В застосунку реалізоване розмежування рівня доступу користувачів. Для звичайних користувачів – виключена можливість отримати інформацію про номер телефону абонента, до якого вони телефонують.

При додаванні користувачів задається їхнє ім’я, пароль, а також вказується SIP-номер користувача. Пароль хешується з використанням алгоритму bcrypt та разом разом з іншими даними зберігається у базі даних.

Для аутентифікації запитів користувачів використовується технологія JSON Web Token (JWT). При успішній авторизації сервер надає користувачу його токен, який містить інформацію про користувача: його ID, роль, та внутрішній SIP-номер. Токени зберігаються в cookies клієнта і автоматично додаються в заголовки кожного запиту, що дозволяє безпечно передавати та перевіряти аутентифікаційну інформацію.

Одним з найважливіших аспектів захисту веб-додатків є захист від SQL-ін'єкцій. В застосунку це реалізовується шляхом використання підготованих запитів до бази даних, вони дозволяють розділити SQL-код і дані, що передаються користувачем. Це запобігає вставці шкідливого коду у SQL-запит. Таким чином аутентифікація користувача та його запити захищені від ін’єкцій.

Для розробленого застосунку налаштований зворотній проксі-сервер Nginx. У нашому застосунку Nginx використовується для проксування HTTP трафіку до VoIP- застосунку та забезпечення додаткового рівня безпеки, а також для встановлення ліміту на кількість запитів від одного клієнта для забезпечення захисту від DoS та DDoS атак.

При запиті користувача на здійснення дзвінка, сам запит валідується на аутентичність, контактний номер абонента не надається користувачу, а сам дзвінок здійснюється безпосередньо системою з допомогою AMI. Asterisk генерує два виклики – до внутрішнього номеру користувача та до абонента, якого ми викликаємо, і з’єднує їх. У запиті до AMI із cookie користувача передається інформація про внутрішній номер співробітника, який ініціює дзвінок, номер абонента зчитується з бази даних контактів безпосередньо перед дзвінком, додатково у програмний телефон користувача передається ім’я абонента.

```
# Send Originate action
action = {
    'Action': 'Originate',
    'Channel': f'PJSIP/{internal_number}',
    'Context': 'default',
    'Exten': num_to_call,
    'Priority': '1',
    'CallerID': f"{contact_name}",
    'Async': 'true'
}
```

Рисунок 3.4 – Програмна генерація виклику

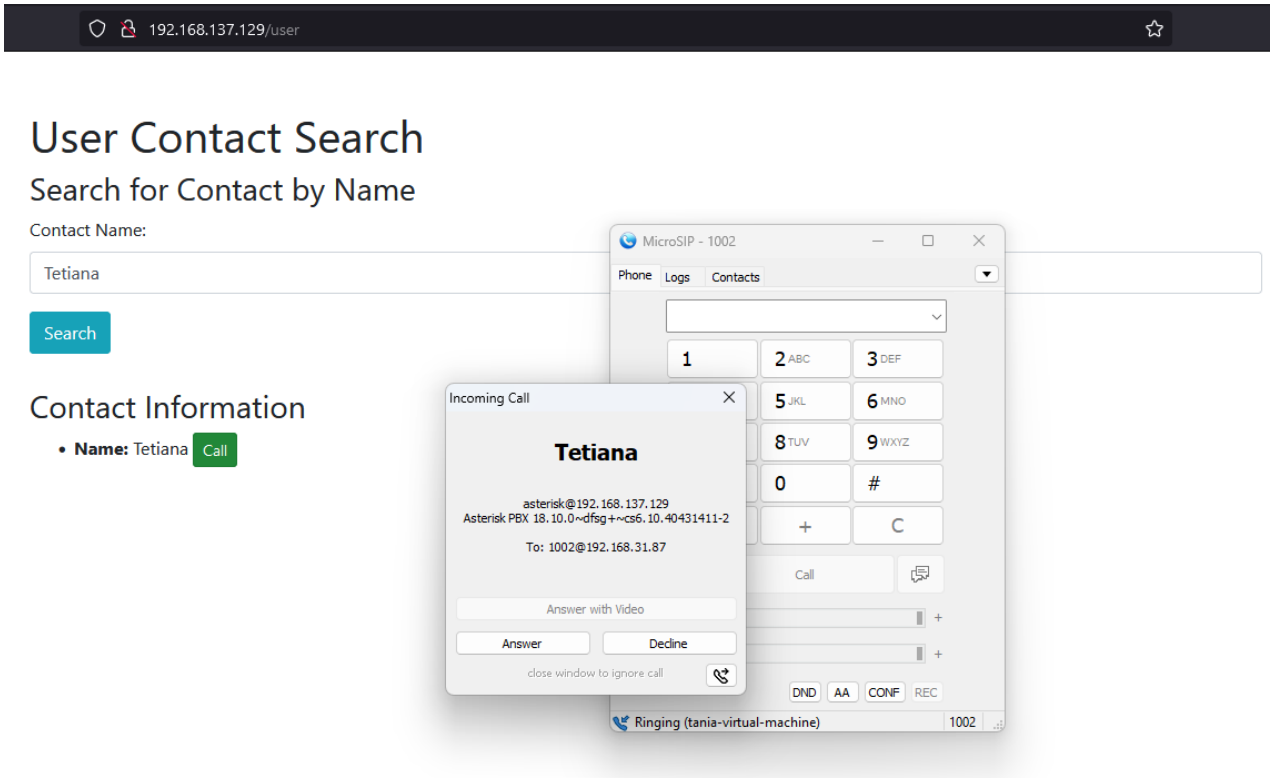

Рисунок 3.5 – Виклик контакту з веб-інтерфейсу

3.3 Розробка фреймворку для моніторингу SIP

Для забезпечення безпеки IP-телефонії зазвичай використовуються SIP-проксі, проте їх налаштування зазвичай є складним та часозатратним завданням, тому ним часто нехтують. Вирішенням цієї проблеми є використання системи виявлення інцидентів (Intrusion Detection System, IDS).

Для реалізації аналізу трафіку використовується бібліотека scapy для Python. Спеціально для коректної обробки SIP тарфіку розроблений модуль, який зчитує дані з шару SIP у пакетах і допомагає визначити їх структуру, та передає до подальшої обробки (Додаток Б).

Основні функції розробленого фреймфорку:

- Моніторинг в реальному часі
- Конфігурація правил через YAML - дозволяє задавати правила моніторингу через конфігураційний файл у форматі YAML, тож є можливість легко налаштувати його для будь-яких потреб.
- Підтримка різних типів правил - підтримуються типи правил, такі як перевірка заголовків, обмеження кількості запитів та обробка неавторизованих запитів.
- Інтеграція з системою моніторингу Elasticsearch: логи подій передаються в Elasticsearch для подальшого аналізу та візуалізації в Kibana.

Структура фреймворку:

- Ядро (core.py) – ініціалізує моніторинг SIP-трафіку, обробляє кожен пакет, застосовуючи налаштовані правила та забезпечує блокування підозрілих IP-адрес і логування подій.
- Модуль для інтеграції з Elasticsearch (log_handler.py) – форматує логи для зручного аналізу та передає до Elasticsearch.
- Конфігураційний файл (config.yaml) – містить налаштування правил для моніторингу SIP-трафіку та дозволяє гнучке додавання нових правил без необхідності змінювати основний код. Конфігурація правил передбачає можливість валідації заголовків (header validation), обробку неавторизованих запитів (unauthorized requests) і контроль за кількістю запитів (rate limit).

Конфігураційний файл дозволяє встановлювати правила до заголовка чи методу, вміст якого перевіряється відповідно до заданого шаблону, який

вважається коректним, а також надає можливість встановити пороги та часові інтервали моніторингу подій. Таким чином розроблений фреймворк здатний адаптуватися до будь-яких потреб моніторингу трафіку і сценаріїв обробки SIP інформації на стороні сервера Asterisk.

```yaml
rules:
  - type: header_validation
    header: 'To'
    valid_pattern: '^[0-9@.]+$'
    threshold: 3
    interval: 60
  - type: rate_limit
    method: 'INVITE'
    threshold: 20
    interval: 5
  - type: unauthorized
    threshold: 3
    interval: 10
```

Рисунок 3.6 – Структура конфігурації правил YAML

Для розробленого фреймворку SIP-моніторингу були обрані три основні типи правил: header_validation, rate_limit, та unauthorized.

- header_validation: це правило напрямлене на протидію спуфінгу, оскільки підроблені SIP-заголовки можуть свідчити про спроби маніпуляції SIP-пакетами або про атаки, спрямовані на експлуатацію вразливостей у SIP-реалізаціях.
- rate_limit: впроваджене з метою обмеження кількості запитів від однієї IP-адреси за певний інтервал часу для запобігання DoS-атакам. Тому відстежується кількість запитів від кожної IP-адреси за визначений інтервал часу та здійснюється блокування, якщо кількість запитів перевищує допустимий поріг.

- unauthorized: Численні невдалі спроби авторизації можуть свідчити про брутфорс-атаку. Це правило допомагає запобіганню неавторизованим спробам доступу до облікових записів користувачів.

Процес обробки вхідного трафіку включає такі ключові етапи:

- Отримання IP пакета на мережевий інтерфейс
- Зчитування трафіку на порту 5060 (використовується для SIP сигналізації)
- Модуль scapy_sip зчитує структуру пакету та передає заголовки шару SIP до обробника правил моніторингу
- Обробник правил згідно сконфігурованих опцій перевіряє вміст заголовків SIP
- При виявленні підозрілої активності інформація логується до Elasticsearch, при досягненні заданих порогів IP-адреса походження пакету блокується. За відсутності підозрілої активності пакет пропускається моніторингом.
- Інформація, передана у Elasticsearch структурована таким чином, щоб вона була зручною та інформативною для візуалізації з допомогою Kibana.

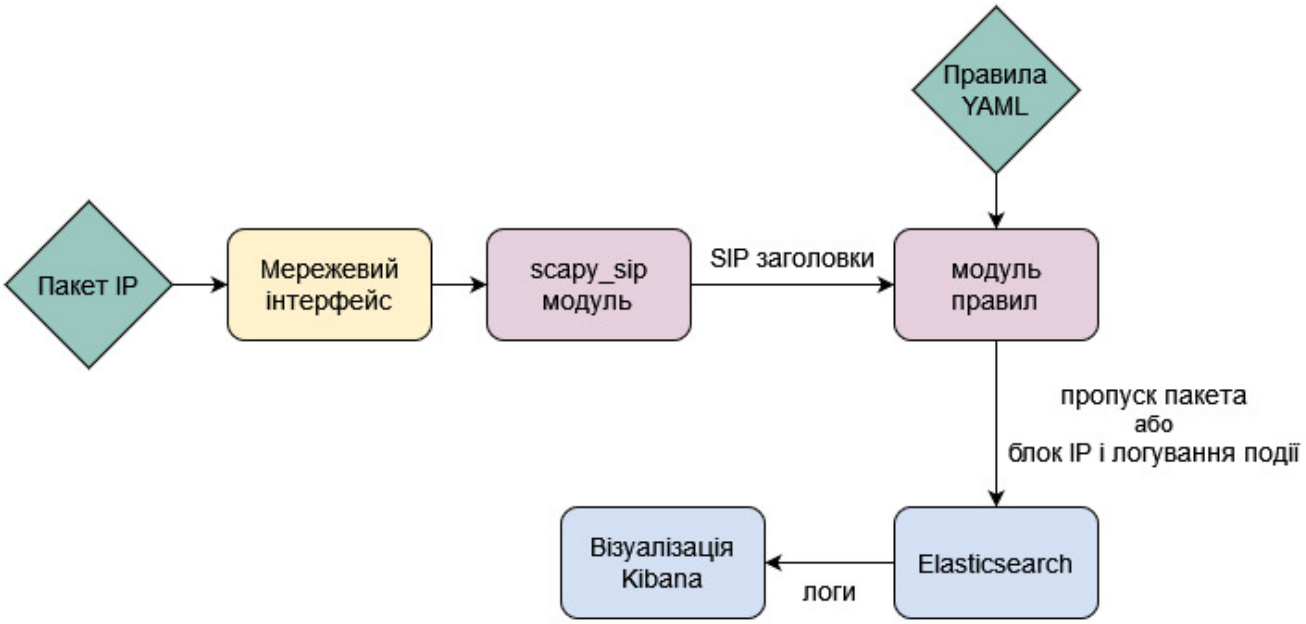

Рисунок 3.7 – Процес обробки трафіку

В результаті аналізу інформація про підозрілі дії збирається у Elasticsearch для можливості зручно відстежувати адміністратором системи аномальну активність.

event	ip	message	call_id	rule.type
block_ip	192.168.31.87	Blocked IP: 192.168.31.87	-	header_validation
invalid_field	192.168.31.87	Invalid field detected in To: thisisfield@192.168.31.69	d450ec276a064851bcd6756cbf5d0aa1	header_validation
invalid_field	192.168.31.87	Invalid field detected in To: thisisfield@192.168.31.69	d450ec276a064851bcd6756cbf5d0aa1	header_validation
invalid_field	192.168.31.87	Invalid field detected in To: admin@192.168.31.69	926b7f1a751e44de8c9bab52b7090c28	header_validation
block_ip	192.168.31.87	Blocked IP: 192.168.31.87	-	header_validation
invalid_field	192.168.31.87	Invalid field detected in To: admin@192.168.31.69	926b7f1a751e44de8c9bab52b7090c28	header_validation
block_ip	192.168.31.87	Blocked IP: 192.168.31.87	-	unauthorized

Рисунок 3.8 – Результат роботи IDS

В результаті спрацювання системи IDS, отримуємо в Elasticsearch інформацію про подію, підозрілу IP-адресу, яке саме правило відпрацювало та що стало тригером правила, а також call-ID підозрілого SIP-діалогу.

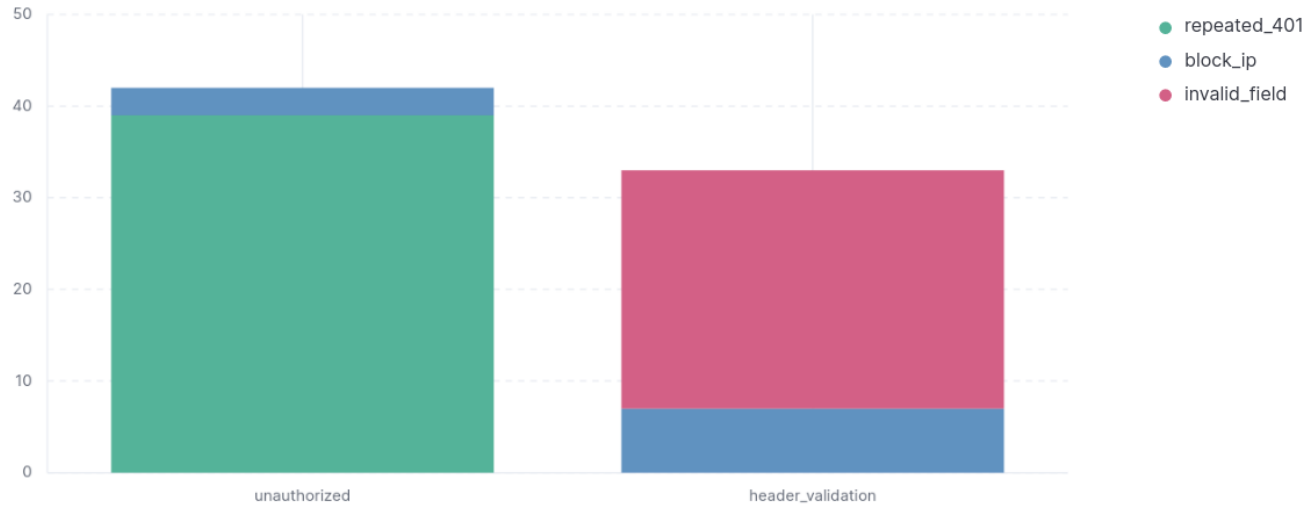

Рисунок 3.9 – Приклад візуалізації даних у Kibana

Висновок до розділу 3

У цій програмній реалізації ми комплексно підійшли до забезпечення безпеки VoIP-інфраструктури організації, розробивши програмне рішення з урахуванням найкритичніших загроз.

Основою системи є VoIP-сервер Asterisk. Сервер інтегрується із зовнішніми провайдерами та внутрішніми користувачами. За допомогою контекстів були мінімізовані ризики шахрайства з тарифікацією.

Другим етапом було створення веб-застосунку на базі Flask, що забезпечує інтерфейс для користувачів. Веб-застосунок реалізує систему аутентифікації та авторизації на основі JWT, що дозволяє безпечно управляти контактами та здійснювати дзвінки. Завдяки налаштуванню зворотного проксі-сервера Nginx, вдалося додатково підвищити безпеку, запобігаючи DoS та DDoS атакам, а також забезпечити захист від SQL-ін'єкцій через використання підготованих запитів до бази даних.

Найважливішою частиною нашої роботи стало розроблення SIP IDS фреймворку для моніторингу SIP-трафіку в реальному часі. Завдяки використанню бібліотеки scapy для Python, ми створили модуль для коректної обробки SIP-пакетів, що дозволяє виявляти підозрілу активність. Налаштування правил через YAML-файл надає гнучкість у моніторингу трафіку та дозволяє легко додавати нові правила. Інтеграція з системою моніторингу Elasticsearch забезпечує зручний аналіз та візуалізацію подій у Kibana, отримувати графічно в реальному часі дані про аномальні події у системі.

ВИСНОВКИ

У ході роботи було визначено ключові загрози для мережі організації, у якій впроваджена технологія VoIP:

- Brute-force атаки: відсутність захисту від повторних спроб авторизації може дозволити зловмисникам підібрати паролі.
- Недостатня валідація команд: якщо скрипти не валідують команди належним чином, це може призвести до SQL-ін'єкцій.
- Виконання довільного коду: зловмисники можуть використати вразливості для виконання довільного коду на сервері.
- Загрози пов’язані із людським фактором: є найскладнішими для технічного убезпечення.

У цій роботі було розроблено комплексне рішення для забезпечення безпеки VoIP-інфраструктури організації. Наш підхід охоплює всі важливі аспекти кібербезпеки систем VoIP і забезпечує захист від широкого спектру загроз.

Налаштування VoIP-сервера Asterisk: основою нашої системи став VoIP-сервер Asterisk. Ми налаштували правила маршрутизації дзвінків, що мінімізують ризики шахрайства з тарифікацією. На цьому етапі важливим моментом є налаштування конфігураційних файлів, які визначають роботу внутрішніх користувачів, методи аутентифікації та правила для зовнішніх з'єднань.

Розробка веб-застосунку: для забезпечення зручності користувачів було створено веб-застосунок на базі фреймфорку Flask, який дозволяє здійснювати дзвінки і управляти контактами. Веб-застосунок включає систему аутентифікації та авторизації користувачів на основі JWT, що забезпечує безпеку передачі даних та розмежування прав доступу. Впровадження зворотного проксі-сервера Nginx

дозволило додатково підвищити безпеку, запобігаючи DoS та DDoS атакам, а також захистити систему від SQL-ін'єкцій через використання підготованих запитів до бази даних.

Розробка SIP IDS фреймворку: ключовою розробкою в цій роботі є створений SIP IDS фреймворк для моніторингу SIP-трафіку в реальному часі. Використовуючи бібліотеку scapy для Python, ми розробили модуль для аналізу SIP-пакетів та виявлення підозрілої активності. Налаштування правил через YAML-файл забезпечує гнучкість моніторингу трафіку та дозволяє легко додавати нові правила для обробки SIP-інформації. Це дозволяє швидко реагувати на будь-які підозрілі дії та забезпечувати захист від можливих загроз.

Інтеграція з системою моніторингу Elasticsearch: наш фреймворк інтегрується з системою Elasticsearch для аналізу та візуалізації подій у Kibana. Це дозволяє адміністраторам системи оперативно виявляти та реагувати на аномальну активність, підвищуючи загальний рівень безпеки інфраструктури. Завдяки цьому рішенню можна отримувати детальну інформацію про події в реальному часі, що значно полегшує моніторинг та управління безпекою.

Запропоноване рішення забезпечує комплексний захист мережі із впровадженням VoIP-інфраструктури від несанкціонованого доступу, brute-force атак, SQL-ін'єкцій, DoS атак та інших загроз. Розроблені рішення може бути впроваджені в різних організаціях, що використовують VoIP технології.

ПЕРЕЛІК ДЖЕРЕЛ ТА ПОСИЛАНЬ

1. RFC 3261 - SIP: Session Initiation Protocol [Електронний ресурс] – https://datatracker.ietf.org/doc/html/rfc3261
2. SURVEY OF SECURITY VULNERABILITIES IN SESSION INITIATION PROTOCOL IEEE Communications Surveys & Tutorials 3rd Quarter 2006 [Електронний ресурс] – http://solomon.ipv6.club.tw/Course/VoIP/1112/Ref/sip_vulnerability.pdf
3. Securing VoIP Networks [Електронний ресурс] / Peter Thermos, Ari Takanen – https://books.google.com.ua/books?id=5F76E72oIR0C&lpg=PT33&ots=t4Czs-8w1P&dq=voip%20architecture&lr&hl=uk&pg=PT5#v=onepage&q=voip%20architecture&f=false
4. Asterisk Documentation [Електронний ресурс] – https://docs.asterisk.org/
5. NIST SP 800-58: Security Considerations for Voice Over IP Systems [Електронний ресурс] – https://doi.org/10.6028/NIST.SP.800-58
6. Security Measures for VoIP: Analysis and Recommendations [Електронний ресурс] – https://www.unitedworldtelecom.com/learn/voip-security-guide/
7. VoIP Security: Vulnerabilities, Exploits, and Defenses [Електронний ресурс] –
8. Security of Voip networks [Електронний ресурс] – https://ieeexplore.ieee.org/abstract/document/5485790
9. Scapy Documentation [Електронний ресурс] – https://scapy.readthedocs.io/en/latest/

ДОДАТОК А

```
from flask import Flask, request, jsonify, make_response, render_template, redirect,
url_for, g
from flask_httpauth import HTTPTokenAuth
import pymysql
import jwt
import datetime
from functools import wraps
import asyncio
from panoramisk.manager import Manager
import bcrypt
import logging
from logging.handlers import RotatingFileHandler

app = Flask(__name__)
app.config['SECRET_KEY'] = 'my_secret_key'  # test key

log_file_path = '/home/tania/flask_login_attempts.log'
handler = RotatingFileHandler(log_file_path, maxBytes=10000, backupCount=1)
handler.setLevel(logging.INFO)
formatter = logging.Formatter('%(asctime)s %(levelname)s: %(message)s [in
%(pathname)s:%(lineno)d]')
handler.setFormatter(formatter)
if not app.logger.handlers:
    app.logger.addHandler(handler)
app.logger.setLevel(logging.INFO)
auth = HTTPTokenAuth(scheme='Bearer')
global num_to_call
num_to_call = None

# middleware for token from cookies
@app.before_request
def before_request():
    token = request.cookies.get('token')
    if token:
        headers = dict(request.headers)
        headers['Authorization'] = 'Bearer ' + token
        request.headers = headers

# connection to MySQL
def create_connection():
    connection = None
    try:
        connection = pymysql.connect(
```

```
            host='localhost',
            user='dbuser',
            password='qweqwe',
            database='mydb',
            cursorclass=pymysql.cursors.DictCursor  # Returns results as dictionaries
        )
    except pymysql.MySQLError as e:
        print(f"Error occurred MySQL: {e}")
    return connection

# hash password
def hash_password(password):
    return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

# check password
def check_password(hashed_password, plain_password):
    return bcrypt.checkpw(plain_password.encode('utf-8'), hashed_password.encode('utf-
8'))

# authentication function
def authenticate(username, password):
    connection = create_connection()
    if connection is not None:
        cursor = connection.cursor()
        cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
        user = cursor.fetchone()
        cursor.close()
        connection.close()
        if user and check_password(user['password'], password):
            return user
    return None

# generate token
def generate_token(user):
    token = jwt.encode({
        'user_id': user['id'],
        'role': user['role'],
        'internal_number': user['internal_number'],
        'exp': datetime.datetime.utcnow() + datetime.timedelta(hours=1)
    }, app.config['SECRET_KEY'], algorithm='HS256')
    return token

# verify token
@auth.verify_token
```

```
def verify_token(token):
    try:
        data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
        return data
    except jwt.ExpiredSignatureError:
        return None
    except jwt.InvalidTokenError:
        return None


# check role
def requires_role(role):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            token_data = auth.current_user()
            if token_data and token_data['role'] == role:
                return f(*args, **kwargs)
            else:
                return jsonify({'message': 'Access forbidden: requires {} role'.format(role)}),
403
        return decorated_function
    return decorator

@app.route('/login', methods=['POST'])
def login():
    data = request.json
    username = data.get('username')
    password = data.get('password')
    user = authenticate(username, password)
    if user:
        token = generate_token(user)
        response = jsonify({'token': token})
        response.set_cookie('token', token)
        return response
    else:
        app.logger.info(f'Failed login attempt for user: {username}')
        return jsonify({'message': 'Invalid credentials'}), 401

@app.route('/register', methods=['POST'])
def register():
    data = request.json
    username = data.get('username')
    password = data.get('password')
```

```
    internal_number = data.get('internal_number')
    hashed_password = hash_password(password).decode('utf-8')

    connection = create_connection()
    if connection is not None:
        cursor = connection.cursor()
        cursor.execute("INSERT INTO users (username, password, role, internal_number)
VALUES (%s, %s, %s, %s)",
                       (username, hashed_password, 'user', internal_number))
        connection.commit()
        cursor.close()
        connection.close()
        return jsonify({'message': 'User registered successfully'}), 201
    else:
        return make_response('Server error', 500)

@app.route('/contacts', methods=['POST'])
@auth.login_required
@requires_role('admin')
def add_contact():
    data = request.json
    name = data.get('name')
    phone = data.get('phone')
    connection = create_connection()
    if connection is not None:
        cursor = connection.cursor()
        cursor.execute("INSERT INTO contacts (phone, name) VALUES (%s, %s)",
(phone, name))
        connection.commit()
        cursor.close()
        connection.close()
        return jsonify({'message': 'Contact added successfully'}), 201
    else:
        return make_response('Server error', 500)

@app.route('/admin', methods=['GET'])
@auth.login_required
@requires_role('admin')
def admin():
    return render_template('admin.html')

@app.route('/search_contacts', methods=['POST'])
@auth.login_required
@requires_role('user')
```

```
def search_contacts():
    data = request.json
    name = data.get('name')
    connection = create_connection()
    if connection is not None:
        cursor = connection.cursor()
        cursor.execute("SELECT id, name FROM contacts WHERE name LIKE %s", ('%'
+ name + '%',))
        contacts = cursor.fetchall()
        cursor.close()
        connection.close()
        return jsonify({'contacts': contacts}), 200
    else:
        return make_response('Server error', 500)

@app.route('/user', methods=['GET'])
@auth.login_required
@requires_role('user')
def user():
    return render_template('user.html')

@app.route('/exec_initiate-call', methods=['POST'])
@auth.login_required
@requires_role('user')
def exec_initiate_call():
    contact_id = request.form['contactId']
    token_data = auth.current_user()
    internal_number = token_data['internal_number']
    connection = create_connection()
    if connection is not None:
        cursor = connection.cursor()
        cursor.execute("SELECT phone FROM contacts WHERE id = %s", (contact_id,))
        contact = cursor.fetchone()
        cursor.close()
        connection.close()
        if contact:
            phone_number = contact['phone']
            loop = asyncio.new_event_loop()
            asyncio.set_event_loop(loop)
            loop.run_until_complete(initiate_call(phone_number, internal_number))
            return redirect(url_for('user'))
        else:
            return jsonify({'message': 'Contact not found'}), 404
    else:
```

```
        return make_response('Server error', 500)

async def initiate_call(num_to_call, internal_number, contact_name):
    if num_to_call and internal_number:
        manager = Manager(loop=asyncio.get_event_loop(), host='127.0.0.1', port=5038,
username='tania', secret='m@naG3R_sEcr3t')
        await manager.connect()
        action = {
            'Action': 'Originate',
            'Channel': f'PJSIP/{internal_number}',
            'Context': 'default',
            'Exten': num_to_call,
            'Priority': '1',
            'CallerID': f"{contact_name}",
            'Async': 'true'
        }
        try:
            response = await manager.send_action(action)
            if response is None:
                logging.error("Failed to send action: response is None")
            else:
                logging.info(f"AMI Response: {response}")
        except Exception as e:
            logging.error(f"Exception occurred while sending action: {e}")
        await manager.close()

@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')

@app.route('/redirect', methods=['GET'])
@auth.login_required
def redirect_user():
    token_data = auth.current_user()
    if token_data['role'] == 'admin':
        return redirect(url_for('admin'))
    else:
        return redirect(url_for('user'))

if __name__ == '__main__':
    app.run(debug=True)
```

ДОДАТОК Б

```
from scapy.packet import Packet, bind_layers
from scapy.fields import *
from scapy.layers.inet import UDP, IP
class SIP(Packet):
    name = "SIP"
    fields_desc = [
        StrStopField("Method", None, ':'),
        StrField("Request_URI", None),
        StrStopField("SIP_Version", None, '\r\n'),
        StrLenField("Headers", "", length_from=lambda pkt: pkt.underlayer.len -
len(pkt.Method) - len(pkt.Request_URI) - len(pkt.SIP_Version)),
        StrLenField("Body", "", length_from=lambda pkt: pkt.underlayer.len -
len(pkt.Method) - len(pkt.Request_URI) - len(pkt.SIP_Version) - len(pkt.Headers))
    ]
    def guess_payload_class(self, payload):
        return Packet.guess_payload_class(self, payload)
bind_layers(UDP, SIP, dport=5060)
bind_layers(UDP, SIP, sport=5060)
```

ДОДАТОК В

Файл main.py

```
from sip_monitoring.core import SIPMonitor
if __name__ == "__main__":
    interface = 'ens37'
    whitelist = {'127.0.0.1', '185.168.129.136'}
    config_file = 'config.yaml'
    monitor = SIPMonitor(interface, whitelist, config_file)
    monitor.start()
```

Файл sip_monitoring/core.py

```
import logging
import re
from collections import defaultdict, deque
from datetime import datetime
import pyshark
import yaml
from .log_handler import ElasticLogHandler

class SIPMonitor:
    def __init__(self, interface, whitelist, config_file, es_host='http://localhost:9200'):
        self.interface = interface
        self.whitelist = whitelist
        self.rules = []
        self.load_config(config_file)
        self.ip_activity = defaultdict(lambda: deque(maxlen=20))
        self.invalid_fields_count = defaultdict(lambda: deque(maxlen=3))
        self.failed_dialogs = defaultdict(lambda: deque(maxlen=3))
        self.dialog_attempts = defaultdict(lambda: defaultdict(int))
        self.es_handler = ElasticLogHandler(es_host)

    def load_config(self, config_file):
        with open(config_file, 'r') as file:
            config = yaml.safe_load(file)
            self.rules = config['rules']

    def block_ip(self, ip, rule):
        message = f'Blocked IP: {ip}'
        self.es_handler.log('block_ip', message, ip, rule=rule)
        logging.info(message)

    def process_packet(self, packet):
        try:
            if hasattr(packet, 'ip'):
```

```
            src_ip = packet.ip.src
            dst_ip = packet.ip.dst
            logging.info(f"Processing packet from {src_ip} to {dst_ip}")
            if src_ip not in self.whitelist and dst_ip not in self.whitelist:
                current_time = datetime.now()

                if hasattr(packet, 'sip'):
                    logging.info(f"SIP packet detected from {src_ip} to {dst_ip}")
                    for rule in self.rules:
                        logging.info(f"Applying rule: {rule}")
                        if rule['type'] == 'header_validation':
                            self.apply_header_validation_rule(packet, rule, src_ip,
current_time)
                        elif rule['type'] == 'rate_limit':
                            self.apply_rate_limit_rule(packet, rule, src_ip, current_time)
                        elif rule['type'] == 'unauthorized':
                            self.apply_unauthorized_rule(packet, rule, dst_ip, current_time)
                else:
                    logging.info(f"No SIP layer found in packet from {src_ip} to {dst_ip}")
        except AttributeError as e:
            logging.error(f"AttributeError during processing packet: {e}")

    def apply_header_validation_rule(self, packet, rule, src_ip, current_time):
        try:
            if hasattr(packet.sip, 'Method') and packet.sip.Method in ('INVITE',
'REGISTER'):
                header = packet.sip.get_field_by_showname(rule['header'])
                call_id = packet.sip.call_id if hasattr(packet.sip, 'call_id') else 'UNKNOWN'
                if header:
                    sip_uri = self.extract_sip_uri(header.showname_value)
                    if sip_uri and not re.match(rule['valid_pattern'], sip_uri):
                        self.invalid_fields_count[src_ip].append(current_time)
                        message = f'Invalid field detected in {rule["header"]}: {sip_uri}'
                        self.es_handler.log('invalid_field', message, src_ip, call_id, rule=rule)
                        logging.info(message)
                        if len(self.invalid_fields_count[src_ip]) >= rule['threshold']:
                            first_invalid_time = self.invalid_fields_count[src_ip][0]
                            if (current_time - first_invalid_time).seconds <= rule['interval']:
                                self.block_ip(src_ip, rule)
                                self.invalid_fields_count[src_ip].clear()
        except AttributeError as e:
            logging.error(f"AttributeError during header validation rule: {e}")

    def apply_rate_limit_rule(self, packet, rule, src_ip, current_time):
```

```
        try:
            if hasattr(packet.sip, 'Method') and packet.sip.Method == rule['method']:
                self.ip_activity[src_ip].append(current_time)
                if len(self.ip_activity[src_ip]) >= rule['threshold']:
                    first_request_time = self.ip_activity[src_ip][0]
                    if (current_time - first_request_time).seconds <= rule['interval']:
                        self.block_ip(src_ip, rule)
                        self.ip_activity[src_ip].clear()
                        message = f'Rate limit exceeded for {rule["method"]} from {src_ip}'
                        self.es_handler.log('rate_limit', message, src_ip, rule=rule)
                        logging.info(message)
        except AttributeError as e:
            logging.error(f"AttributeError during rate limit rule: {e}")

    def apply_unauthorized_rule(self, packet, rule, dst_ip, current_time):
        try:
            if 'SIP/2.0 401 Unauthorized' in str(packet.sip):
                if hasattr(packet.sip, 'call_id'):
                    call_id = packet.sip.call_id
                    self.dialog_attempts[dst_ip][call_id] += 1
                    if self.dialog_attempts[dst_ip][call_id] > 1:
                        self.failed_dialogs[dst_ip].append(current_time)
                        message = 'Repeated 401 Unauthorized'
                        self.es_handler.log('repeated_401', message, dst_ip, call_id, rule=rule)
                        logging.info(message)

                        while self.failed_dialogs[dst_ip] and (
                                current_time - self.failed_dialogs[dst_ip][0]).seconds >
rule['interval']:
                            self.failed_dialogs[dst_ip].popleft()

                        if len(self.failed_dialogs[dst_ip]) >= rule['threshold']:
                            self.block_ip(dst_ip, rule)
                            self.failed_dialogs[dst_ip].clear()
        except AttributeError as e:
            logging.error(f"AttributeError during unauthorized rule: {e}")

    @staticmethod
    def extract_sip_uri(field):
        sip_uri_regex = re.compile(r'<sip:(.*?)>')
        match = sip_uri_regex.search(field)
        if match:
            return match.group(1)
        return None
```

```
    def start(self):
        capture = pyshark.LiveCapture(interface=self.interface, bpf_filter='udp port 5060')
        for packet in capture.sniff_continuously():
            self.process_packet(packet)
```

Файл sip_monitoring/log_handler.py

```
from elasticsearch import Elasticsearch
from datetime import datetime

class ElasticLogHandler:
    def __init__(self, es_host='http://localhost:9200'):
        self.es = Elasticsearch([es_host])

    def log(self, event, message, ip, call_id=None, method=None, rule=None):
        log_entry = {
            "timestamp": datetime.now(),
            "event": event,
            "message": message,
            "ip": ip,
            "call_id": call_id,
            "method": method,
            "rule": rule  # Ensure rule is always a string
        }
        self.es.index(index="sip-ids-logs", body=log_entry)
```

Файл sip_monitoring/rules.py

```
import re
from datetime import datetime

valid_field_regex = re.compile(r'^[0-9@.]+$')
sip_uri_regex = re.compile(r'<sip:(.*?)>')
def extract_sip_uri(field):
    match = sip_uri_regex.search(field)
    if match:
        return match.group(1)
    return None

def call_flood_rule(packet, monitor):
    try:
        if hasattr(packet.sip, 'Method') and packet.sip.Method == 'INVITE':
            src_ip = packet.ip.src
            monitor.ip_activity[src_ip].append(datetime.now())
            if len(monitor.ip_activity[src_ip]) >= monitor.thresholds['requests']:
```

```
            first_request_time = monitor.ip_activity[src_ip][0]
            if (datetime.now() - first_request_time).seconds <=
monitor.intervals['request']:
                monitor.block_ip(src_ip)
                monitor.ip_activity[src_ip].clear()
                message = f'Call flood detected from {src_ip}'
                monitor.es_handler.log('call_flood', message, src_ip)
                return
    except AttributeError as e:
        monitor.es_handler.log('error', f"AttributeError during call flood rule: {e}")

def invalid_field_rule(packet, monitor):
    try:
        if hasattr(packet.sip, 'Method') and packet.sip.Method in ('INVITE', 'REGISTER'):
            from_field = packet.sip.get_field_by_showname('From')
            to_field = packet.sip.get_field_by_showname('To')
            call_id = packet.sip.call_id if hasattr(packet.sip, 'call_id') else 'UNKNOWN'
            fields_to_check = [from_field, to_field]
            for field in fields_to_check:
                if field:
                    sip_uri = extract_sip_uri(field.showname_value)
                    if sip_uri and not valid_field_regex.match(sip_uri):
                        monitor.invalid_fields_count[packet.ip.src].append(datetime.now())
                        message = f'Invalid field detected: {sip_uri}'
                        monitor.es_handler.log('invalid_field', message, packet.ip.src, call_id)
                        if len(monitor.invalid_fields_count[packet.ip.src]) >=
monitor.thresholds['invalid_fields']:
                            first_invalid_time = monitor.invalid_fields_count[packet.ip.src][0]
                            if (datetime.now() - first_invalid_time).seconds <=
monitor.intervals['fields']:
                                monitor.block_ip(packet.ip.src)
                                monitor.invalid_fields_count[packet.ip.src].clear()
                                return
    except AttributeError as e:
        monitor.es_handler.log('error', f"AttributeError during invalid field rule: {e}")

def unauthorized_rule(packet, monitor):
    try:
        if 'SIP/2.0 401 Unauthorized' in str(packet.sip):
            if hasattr(packet.sip, 'call_id'):
                call_id = packet.sip.call_id
                dst_ip = packet.ip.dst
                monitor.dialog_attempts[dst_ip][call_id] += 1
                if monitor.dialog_attempts[dst_ip][call_id] > 1:
```

```
                monitor.failed_dialogs[dst_ip].append(datetime.now())
                message = 'Repeated 401 Unauthorized'
                monitor.es_handler.log('repeated_401', message, dst_ip, call_id)

                while monitor.failed_dialogs[dst_ip] and (datetime.now() -
monitor.failed_dialogs[dst_ip][0]).seconds > monitor.intervals['dialog']:
                    monitor.failed_dialogs[dst_ip].popleft()

                if len(monitor.failed_dialogs[dst_ip]) >=
monitor.thresholds['failed_dialogs']:
                    monitor.block_ip(dst_ip)
                    monitor.failed_dialogs[dst_ip].clear()
                    return
    except AttributeError as e:
        monitor.es_handler.log('error', f"AttributeError during unauthorized rule: {e}")
```