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

«До захисту допущено»
В.о. завідувача кафедри

_____ М.В.Грайворонський
(підпис)
“_____”_____2019 р.

Дипломна робота
на здобуття ступеня бакалавра

з напряму підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

на тему: Атака на анонімність користувача в системі Tor та способи протидії ним

_____

_____

Виконав (-ла): студент (-ка) 4 курсу, групи ФБ-51
(шифр групи)

Руснак Дмитро Дмитрович
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент кафедри ІБ, к.т.н., Стьопочкіна І.В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент доцент кафедри ММЗІ, к.т.н., Яковлєв С.В.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

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

Київ - 2019 року

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

Рівень вищої освіти – перший (бакалаврський)
Напрям підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)
«___»_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту

Руснак Дмитро Дмитрович
(прізвище, ім'я, по батькові)

1. Тема роботи: Атака на анонімність користувача в системі Tor та способи протидії ним,
науковий керівник роботи: Стьопочкіна Ірина Валеріївна, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2019 р. № 1414-с

2. Термін подання студентом роботи 10 червня 2019 р.

3. Вихідні дані до роботи

1. Попередні дослідження.
2. Система Tor.

4. Зміст роботи

1. Вивчити принципи роботи цибулевої маршрутизації.
2. Вивчити механізми роботи практичної реалізації цибулевої маршрутизації під назвою Tor.
3. Проаналізувати існуючі атаки на систему Tor та способи протидії ним.
4. Вибрати категорію атак та запропонувати спосіб протидії.
5. Видібрати потрібні компоненти для реалізації програмного рішення.
6. Розробити відповідне програмне рішення та здійснити експериментальне дослідження.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

Атаки на анонімність користувача в системі Tor – презентація.

6. Дата видачі завдання 10.10.2018

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Отримання завдання	10.10.2018	
2	Збір інформації	01.02.2019	
3	Дослідження предметної області та існуючих рішень	01.04.2019	
4	Розробка плану роботи	15.04.2019	
5	Проведення експерименту з модифікацією пакетів	09.05.2019	
6	Проведення дослідження ефективності програмного рішення	16.05.2019	
7	Оцінка результатів	23.05.2019	
8	Оформлення дипломної роботи	26.05.2019	
9	Отримання допуску до захисту	28.05.2019	

Студента _____ Руснак Д.Д.
(підпис) (ініціали, прізвище)

Науковий керівник роботи _____ Стьопочкіна І.В.
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Дипломна робота має обсяг 90 сторінок, містить 4 таблиц та 10 рисунків, а також 45 бібліографічних джерел.

Актуальною науковою тенденцією є розробка та впровадження нових механізмів забезпечення анонімності в мережі Інтернет. Механізм, що використовується у даній роботі, дозволяє забезпечити збільшення анонімності в системі Tor. Тому це актуально для користувачів, які використовують систему Tor, щоб забезпечити збільшення анонімності проти атак типу аналізу трафіку й часу.

Об’єктом дослідження є система Tor.

Предметом дослідження є атаки на систему Tor і способи протидії ним: з теоретичної та технічної точки зору.

Метою роботи є дослідження принципів цибулевої маршрутизації, механізмів системи Tor, атак на анонімність користувача в системі та способів протидії ним та розробка рішень, які допоможуть запобігти найбільш популярним із цих атак.

Дана робота містить опис цибулевої маршрутизації та системи Tor, огляд існуючих атак та способи протидії ним. У ході роботи отримано програмне рішення для додавання затримок при посиланні та прийнятті пакетів, яке відрізняється використанням криптографічного генератора псевдовипадкових чисел при формуванні величин затримок. У подальшому, отриманий результат у вигляді програмного рішення можна використовувати забезпечення підвищення анонімності в мережі Tor.

Ключові слова: цибулева маршрутизація, Tor, атаки, аналіз трафіку, способи протидії, деанонімізація, затримки.

ABSTRACT

The thesis contains 90 pages, 4 tables and 10 figures as well as 45 names of bibliographic sources.

The actual scientific trend is the development and implementation of new mechanisms for ensuring anonymity on the Internet. The mechanism used in this work allows to increasing the anonymity in the Tor system. There is why it is important for users who use the Tor system to increase the anonymity against of attacks such as traffic analysis and time analysis.

The object of the study is the Tor system.

The subject of the study is the attacks on the Tor system and countermeasures: from a theoretical and technical point of view.

The aim of the work is to study the principles of onion routing, the mechanisms of the Tor system, and attacks on the anonymity of the user in the system and mathods to counteract it, and develop solutions that will help prevent the most popular of these attacks.

This work contains a description of onion routing and Tor systems, an overview of existing attacks, and methods to counteract it. In the course of work, a software solution obtained to add delays in the sending and acceptance of packets, what differs in use cryptographic pseudorandom number generator in the formation of delays values. In the future, the resulting result in the form of a software solution can be using to increase the anonymity of the Tor network.

Keywords: onion routing, tor, attacks, traffic analysis, methods of counteraction, deanonimization, delays.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

США – Сполучені Штати Америки

Tor – The Onion Router

ONR – Office of Naval Research

EFF – Electronic Frontier Foundation

TBB – Tor Browser Bundle

AS – Automation System

ПЗ – Програмне забезпечення

NRL - Naval Research Laboratory

AES – Advanced Encryption Standart

ВСТУП

Наразі проблему анонімності користувача вирішують криптографічні мережі, які намагаються запобігти аналізу трафіку, змішуючи повідомлення, що проходять через мережу, таким чином, щоб зовнішній спостерігач не зміг відстежити джерело і пункт призначення повідомлення. Мета полягає в тому, щоб забезпечити повну анонімність спілкування через мережу Інтернет. Одна з таких систем, що пропонують анонімність, маю назву Tor, що використовує технологію, звану цибулевою маршрутизацією, для шифрування повідомлень. Технологія, яка була спочатку розроблена для цілей армії США, в даний час розробляється ентузіастами-програмістами і вченими. В даний час цибулева маршрутизація активно використовується, щоб уникнути цензури в Інтернеті, запобігти підслуховування і сприяє свободі слова.

Актуальність роботи. Тема є актуальною і на сьогоднішній день, адже організації та великі країни кожний день вкладають великі кошти для розробки програмних засобів, щоб унеможливити та порушити анонімність користувачів в мережі Інтернет для відслідковування дій користувачів у особистих інтересах.

Мета і завдання дослідження. Метою і завданням роботи є дослідження принципів цибулевої маршрутизації, механізмів системи Tor, атак на анонімність користувача в системі та способів протидії ним та розробка рішень, які допоможуть запобігти найбільш популярним із цих атак.

Об’єкт дослідження. Об’єктом дослідження є система Tor.

Предмет дослідження. Предметом дослідження є атаки на систему Tor і способи протидії ним: з теоретичної та технічної точки зору.

Методи дослідження. Методами дослідження є аналіз інформаційних джерел, новітніх статей за темою дослідження, стандартів мережних протоколів, та здійснення експерименту із використанням сучасного програмного забезпечення.

Новизна одержаних результатів. Розроблено програмне рішення для додавання додаткових затримок при посиланні та прийнятті, яке відрізняється

використанням криптографічного генератора ПВЧ, що робить статистичний аналіз затримок та, відповідно, встановлення джерела пакету практично неможливим.

Практичне значення одержаних результатів. Практичною цінністю роботи є використання програмного рішення для забезпечення збільшення анонімності користувача використовуючи систему Tor та збільшення важкості деанонімізації користувача. Розроблене рішення може бути використано розробниками та дослідниками криптографічних мереж із забезпечення анонімності.

1 ЦИБУЛЕВА МАРШРУТИЗАЦІЯ ТА TOR

З моменту винаходу відправки повідомлень за допомогою листів між людьми також виникла необхідність запобіганню захопленню і прочитанню цих самих повідомлень іншими людьми. Коли поряд з фізичним відправленнь повідомлень повідомленнь прийшла ера електричного зв'язку й мережі Інтернет, проблема, звичайно, не зникла, а набула ще більшого характеру. Величезні зусилля докладаються для розробки ефективних способів боротьби та шифруванню повідомлень, щоб тільки ті сторони, які беруть участь в спілкуванні, знали, який зміст цих повідомлень. Зазвичай повідомлення, що проходять через мережу Інтернет, шифруються, і всім відомі тільки їх відправник і одержувач, але в деяких ситуаціях навіть цю інформацію бажано зберігати в секреті від інших людей.

Термін аналіз трафіку використовується для опису процесу, який намагається збирати та вивчати повідомлення, що проходять через інформаційну мережу, зі спробою зробити висновки про трафік на основі зібраних знань. Кожна дія, що здійснюється в мережі Інтернеті, відправляє серію повідомлень через безліч серверів, які можуть бути розташовані будь-якій точці світу, що робить дії досить публічними. Це означає, що для когось відносно легко відстежувати такий трафік. Підслуховувати трафік не складно. Оскільки дані проходять через мережу Інтернет, вони проходять через декілька місць, і хтось, хто намагається підслухати ці повідомлення, потрібно бути лише в одному з цих декількох місць на шлях зв’язку по якому передаються повідомлення, щоб спостерігати за трафіком.

IP-пакет, який передає дані через мережу Інтернет, складається з двох частин: по-перше, передані дані, які називаються корисним навантаженням, і, подруге, заголовок, що містить метадані про корисне навантаження. Корисне навантаження зазвичай шифрується перед відправленням, але частина заголовка - ні, і вона багато говорить про повідомлення, такі як відправник, отримувач, час відправки й розмір блоку даних. Це створює проблему інформаційної безпеки для окремих осіб і організацій, оскільки в загальному випадку будь-хто, хто перехоплює ці

повідомлення, може спостерігати за трафіком і формувати модель трафіку, грунтуючись тільки на заголовках мережевих пакетів.

З того, що описано вище, все більша частина користувачів мережі Інтернет не хоче, щоб посторонні могли подивитися, що вони роблять в мережі Інтернет, і через це відбувався попит на додатки, які можуть запропонувати повну анонімність. Користувачі та їх причини для використання таких додатків доволі розрізняються: звичайні люди хочуть захистити свою конфіденційність від маркетологів і безвідповідальних компаній, журналісти та їх аудиторія хочуть гарантувати свободу слова без політичної цензури, співробітники правоохоронних органів хочуть обережно вивчати інтернет-злочини, а військові хочуть захистити себе від шпигунства противника під час проведення операцій.

Історія цибулевою маршрутизації починається з 1995 року, коли Управління морських досліджень, або ONR, в США, почав фінансувати проект, метою якого було видалити ідентифікацію з маршрутизації. Іншими словами, їх мета полягала в тому, щоб створити спосіб створення зв'язків в мережі Інтернеті анонімно [1]. Спочатку три людини, Девід Голдшлад, Майкл Рід і Пол Сіверсон, почали розробку технології, яка згодом стала називатися цибулевою маршрутизацією. Початковим результатом проекту став перший дослідний зразок в 1996 році, і після цього було розроблено ще два покоління цієї технології.

Перш ніж Голдшлад, Рід і Сіверсон почали розробляти перші прототипи, у них було уявлення про те, які принципи буде мати технологія. По-перше, код повинен бути відкритий для всіх. Це зроблено для того, щоб користувачі, які використовують цю систему, могли бачити код, щоб вони довіряли системі. Другим основним принципом цибулевої маршрутизації є можливість вільного прямування. Це означає, що комп'ютер, який використовується для доступа до системи, не обов'язково повинен бути сервером. Цей принцип має деякі переваги безпеки, які будуть проаналізовані пізніше.

1.1 Огляд технологій

Цибулева маршрутизація отримала свою назву від структури даних, всередині якої інформація поширюється по мережі [2]. Цибуля створюється з криптографічно зашифрованих шарів відправником повідомлення. Кожен маршрутизатор на шлях зв'язку очищає ці рівні один за іншим, коли повідомлення проходить через мережу. Вхідні повідомлення проходять по тому ж шляху, за винятком того, що маршрутизатори додають шари до цибулі, який потім очищає приймаюча сторона.

Як зазначалося раніше, цибулева маршрутизація заснована на створенні криптографічного маршруту через кілька маршрутизаторів між двома комп'ютерами. Цей шлях називається цибулевим ланцюгом [1]. Маршрутизатори, які формують шлях, вибираються випадковим чином з усіх серверів, зареєстрованих в цибулевій мережі, і схема формується таким чином, щоб кожен вузол знав тільки наступний і попередній маршрутизатори в дорозі зв'язку. Ініціатор шляху приймає рішення про те, які маршрутизатори будуть включені в ланцюжок.

Щоб запобігти ситуації, коли всі маршрутизатори, які беруть участь в шлях зв'язку, можуть відкривати повідомлення, що проходять по ланцюжку, ініціатор повинен роздавати унікальні симетричні ключі шифрування індивідуально кожному задіяному маршрутизатора. Причина використання симетричних ключів полягає в тому, що симетричне шифрування використовує один і той же ключ для шифрування і дешифрування повідомлень, тому повідомлення можна відправляти в обидва напрямки з одним і тим же ключем. Симетричне шифрування також в обчислювальному відношенні набагато легше, ніж шифрування з відкритим ключем.

Симетричні ключі шифрування поширюються з використанням шифрування з відкритим ключем. Шифрування з відкритим ключем використовує два ключа: закритий ключ (який не відомий іншим) і відкритий ключ (який відомий всім). Повідомлення, відправлені одержувачу, шифруються за допомогою відкритого

ключа, який, в свою чергу, не може бути використаний для розшифрування повідомлень. Одержувач може відкрити повідомлення своїм закритим ключем. Оскільки кожен маршрутизатор має свій власний відкритий ключ, симетричні ключі, які використовуються для створення цибулі, можуть безпечно поширюватися за допомогою відкритих ключів без небезпеки їх виявлення іншими маршрутизаторами.

Причиною використання симетричного шифрування для створення дійсної цибулі є те, що симетричне шифрування вимагає набагато меншою обчислювальної потужності, ніж шифрування з відкритим ключем. Є безліч повідомлень, які подорожують по мережі в усіх напрямках, і кожне повідомлення має бути оброблено алгоритмом шифрування. Використовуючи симетричне шифрування замість шифрування з відкритим ключем, обчислювальна вартість мережі є значно легшою.

Алгоритм формування цибулевого ланцюжка докладно описаний в наступному розділі. Далі ми докладніше розглянемо еволюцію технології цибулевої маршрутизації. Як згадувалося раніше, цибулева маршрутизація розроблялася поетапно протягом декількох поколінь. Функції, згадані досі, не відповідають дійсності для всіх трьох поколінь, але були включені в технологію хоча б одну з них. Відмінності між поколіннями, що розвиваються будуть основним предметом наступного пункту.

1.2. Еволюція цибулевої маршрутизації

Коли проектування цибулевої маршрутизації було запущено в морській дослідницькій лабораторії США або в NRL, у Голдшлада, Ріда і Сіверсона було всього декілька основних принципів і деякі припущення про те, в яких ситуаціях буде використовуватися технологія, щоб керуватися ними у своїй роботі [1]. Лише коли перше покоління було опубліковано, тільки тоді вони отримали свою першу ілюстрацію того, наскільки функціональним насправді була їхня розробка. Функції були додані і покращенні в наступних поколіннях відповідно до спостережень,

зробленими розробниками, а також відповідно до відгуків користувачів. Таким чином, покоління представляють собою різні етапи розробки цибулевої маршрутизації, і, вивчаючи ці покоління, можна зрозуміти, чому цибулева технологія реалізована сама так, якою вона є сьогодні.

1.2.1 Перше покоління(1996 – 2004)

Основна ідея цибулевої маршрутизації полягає в тому, щоб зробити можливим створення анонімного з'єднання всередині цибулевої мережі з серверами, які є зовнішніми по відношенню до цибулевої мережі. На додаток до цього було вирішено зробити можливим формування з'єднань між двома маршрутизаторами, підключеними до цибулевої мережі. В обох цих ситуаціях довжина ланцюжка, тобто число маршрутизаторів в ланцюжку, включаючи відправника і одержувача, має велике значення з точки зору інформаційної безпеки.

Щоб один з вузлів в ланцюжку порушив анонімність, йому необхідно підключити анонімні повідомлення до відправника та одержувача. В основному теоретичний зловмисник повинен отримати IP-адресу відправника та одержувача. Зазвичай це можна зробити, переглядаючи заголовки пакетів даних, але цибулева маршрутизація шифрує ці заголовки. Найголовніша атака на цибулеву маршрутизацію - встановити маршрутизатор в якості цибулевого вузла і спробувати спостерігати за повідомленнями, що проходять через цей маршрутизатор.

Якщо довжина ланцюжка складає три вузла (відправник, цибулевий маршрутизатор і одержувач), і якщо середній вузол є вузел зловмисника, який хоче з'ясувати IP-адреси двох кінців ланцюжка, він негайно дізнається ці адреси. При використанні чотирьох вузлів (відправник, два маршрутизатора, одержувач) зловмисникові необхідно порушити шифрування тільки одного з вузлів, щоб порушити анонімність ланцюжка. Розробники вважали це досить небезпечним, тому вони вирішили встановити мінімальну довжину (довжину ланцюга за замовчуванням), рівну п'яти вузлам.

Так звані сервери рандеву використовуються в тому випадку, якщо дві машини, підключені до цибулевої мережі, хочуть відправляти один одному повідомлення. Машини не можуть сформувати шлях з'єднання, грунтуючись на IP-адреси один одного, тому сервер рандеву використовується в якості посередника для поширення повідомлень. Оскільки обидва боки доводиться формувати свої власні цибулеві ланцюжки між серверами рандеву, довжина шляху з усіма включеними маршрутизаторами становитиме дев'ять вузлів. Ця функція залишилася і для наступних поколінь.

Найбільша відмінність між першим поколінням і наступними двома полягає в тому, що у версії першого покоління клієнтське програмне забезпечення і цибулевий маршрутизатор були повністю інтегровані. Оскільки комп'ютер приєднувався до мережі для відправки повідомлень, він також може бути доданий як частина цибулевого ланцюжка. Така функціональність суперечила одному з основних принципів цибулевої маршрутизації і була змінена після випуску другого покоління, що дозволило запускати клієнт і цибулевий маршрутизатор окремо.

1.2.2 Друге покоління(2004 – 2006)

Різні методи були додані до другого покоління з метою підвищення інформаційної безпеки. Одним з методів є відправлення повідомлень з випадковими даними, званими наповнювачами, разом зі звичайним трафіком, щоб зловмисникому було складніше аналізувати трафік. Другий метод полягає в обмеженні смуги пропускання постійною швидкістю, що виключає ідентифікацію змін в потоках трафіку. Вважалося, що доповнення і обмеження смуги пропускання ускладнюють аналіз трафіку, але було відзначено, що ці методи занадто дорогі в обчислювальному відношенні в порівнянні з безпекою, яку вони забезпечують. Заповнення і обмеження смуги пропускання були видалені в третьому поколінні.

Деякі кріпто-мережі, що пропонують анонімність, засновані на змішуванні компонентів, а не на непередбачуваному маршруті. Ці мережі також відомими як MIX мережі (розроблено Девідом Чаумом в 1981 році) [2]. Мета змішування -

ускладнити з'єднання вхідних повідомлень із зашифрованими вихідними повідомленнями один з одним.

MIX-технологія була експериментально включена в друге покоління цибулевої маршрутизації. Мета цього експерименту полягала в тому, щоб вивчити переваги безпеки, які забезпечить змішування. Передбачалося, що прийняття вже широко використовуваного методу зробить цибулеву маршрутизацію більш прийнятною для більш ширшої аудиторії.

Змішування в кінцевому підсумку було припинено в третьому поколінні з тих же причин, що і заповнення й обмеження пропускної здатності; це недостатньо підвищило інформаційну безпеку і значно збільшило вартість обчислень. Виконання фактичного змішування не вимагає великих обчислювальних витрат, але додавання змішування створює ризик того, що хтось зможе підслухати повідомлення, і до сих пір недорогий спосіб запобігти цьому не був знайдений.

Ще однією важливою поправкою, крім змішування, стало створення ланцюгів довжиною більше п'яти вузлів. У цій версії покоління довжина ланцюгів варіювалася і в одному ланцюжку могло бути до одинадцяти вузлів. Шляхом тунелювання ланцюжків або об'єднання декількох ланцюжків довжина шляху може стати ще більше. Як і в випадку схем змішування і заповнення, ця поправка також було визнано непотрібною, і число вузлів було відновлено до п'яти в наступному поколінні.

1.2.3 Третє покоління(2006 -)

Третє покоління - остання версія цибулевої маршрутизації. Розробка цибулевої маршрутизації ще триває, тому в майбутньому можливості цибулевої маршрутизації можуть відрізнятися від представлених тут. Однак ми розглянемо особливості третього покоління в тому вигляді, в якому він був опублікований в 2006 році. Відмінності між цим поколінням і попереднім менше, ніж відмінності між першим і другим поколіннями.

Найбільша поправка в порівнянні з попередніми версіями - це протокол, за допомогою якого ключі шифрування розподіляються по вузлах при першому створенні каналу. Перша версія використовувала структуру onion-data для побудови шляху [1]. Ця версія використовує протокол Diffie-Hellman для поступового розподілу ключів по вузлах. Це реалізує так звану пряму секретність, яка означає, що навіть якщо зловмисник може розшифрувати і отримати один з ключів шифрування, він не зможе використовувати його для відкриття інших цибулевих рівнів.

Ще однією поправкою в цій версії стало додавання серверів каталогів. Перед цією поправкою інформація про мережеві адреси видавалася ззовні з мережі. Коли розмір мережі збільшується, такий спосіб видачі інформації стає неможливим, і сервери каталогів пропонують гнучкий спосіб вирішення цієї проблеми. Це також підвищує безпеку, тому що розподілена інформація не може бути вивчена.

У наступному підрозділі більш детально розглядається фактична реалізація цибулевої технології третього покоління. Для підведення підсумку, Таблиця 1.1 показує невелике технічне порівняння між поколіннями.

Таблиця 1.1 – Відмінні риси різних поколінь

Покоління	**Рік публікації**	**Змішування даних**	**Довжина ланцюга**	**Клієнт і сервер**	**Протокол поширення ключів**
Перше покоління	1996[1]	-	5	Разом	Цибулевий протокол
Друге покоління	2004[3]	+	1-11	Окремо	Цибулевий протокол
Третє покоління	2006[3]	-	5	Окремо	Diffie-Hellman протокол

1.3 Tor

Перша версія цибулевої маршрутизації була опублікована в 1996 році [1]. Коли Управління морських досліджень фінансувало проект, вони також підтримували веб-сайт, на якому була опублікована інформація про хід реалізації проекту. Цей сайт містив технічну інформацію про різні покоління, а також наукові

статті, що стосуються цибулевої маршрутизації. Оскільки ONR припинив фінансування в 2004 році, також було припинено ведення веб-сайту і сайт був закритий [3]. Однак є копія сайту за адресою www.onion-router.net [3].

Коли ONR припинив фінансову підтримку, Фонд Electronic Frontier, або незабаром EFF, почав фінансувати проект. Пол Сіверсон і команда по розробці цибулевої маршрутизації встигли випустити три покоління цибулевої маршрутизації при фінансуванні ONR. Коли спонсор змінився, був зроблений перехід до відкритого вихідного коду, і разом з цим нова домашня сторінка перемістилася на адрес www.torproject.org [4].

З новим сайтом також було опубліковано додаток під назвою Tor. Мета Tor - створити мережу, яка захищає користувача від ідентифікації з використанням технології цибулевої маршрутизації. Іншими словами, Tor - це реалізація цибулевої маршрутизації на практиці. Починаючи з цього моменту, обговорюючи цибулеву маршрутизацію, ми говоримо про Tor, реалізації цибулевої маршрутизації третього покоління, і навпаки.

1.3.1 Браузер Tor

Додаток Tor написаний на мові програмування C, а його вихідний код зберігається в загальнодоступному репозиторії за адресою gitweb.torproject.org/tor.git. Щоб використовувати додаток, будь-який користувач може завантажити пакет під назвою Tor Browser Bundle, або TBB, з веб-сайту проекта [4]. TBB містить програму, яка дозволяє користувачеві підключатися до мережі Tor в якості клієнта, і спеціальний браузер Firefox, який можна використовувати для відвідування веб-сайтів.

TBB містить необхідне програмне забезпечення для запуска клієнта і реле. TBB попередньо налаштований для роботи в якості клієнта, і користувачеві не обов'язково вводити додаткові налаштування для використання програмного забезпечення [4]. Однак є деякі дії, які користувач може зробити для підвищення

конфіденційності, такі як включення розширення, званого NoScript, яке відключає JavaSript, і використання тільки HTTPS для формування з'єднання.

Щоб додати нове реле Tor в мережу, користувачеві необхідно відредагувати файл конфігурації з інформацією про порт і політикою виходу.
Клієнти використовують програмне забезпечення Tor для формування з'єднань через реле. Всі комп'ютери, з'єднані один з одним за допомогою TBB, утворюють мережу, яка називається мережею Tor. Для ефективної роботи мережі Tor потрібні й інші види компонентів, незважаючи на клієнтів і реле. Давайте подивимося, що є іншими видами компонентів мережі Tor.

1.3.2 Мережа Tor

У своїй найпростішій абстракції реалізація мережі цибулевої маршрутизації складається з клієнтів, які використовують мережу для відправки повідомлень, і маршрутизаторів або ретрансляторів, які отримують цибулю, очищають її і передають її наступному вузлу в ланцюзі. Крім клієнтів і ретрансляторів, мережа Tor має, наприклад, сервери імен, які відстежують сервери, зареєстровані в мережі, і приховані служби, які можна використовувати, наприклад, для веб-публікації. Приховані сервіси можуть бути підключені тільки через певні сервери Tor. [1]

Клієнти і маршрутизатори складають основну частину мережі Tor [5]. Клієнти також можуть виступати в якості маршрутизатора всередині мережі, але більшість клієнтів використовують мережу тільки для формування з'єднань з іншими клієнтами, прихованими службами і зовнішніми службами через інші маршрутизатори. Кількість клієнтів, які використовують мережу, в березні 2016 року становить близько 2 мільйонів, а кількість ретрансляторів в мережі в 2016 році становить 7000.

Щоб зробити Tor стійким до підслуховування, поряд зі звичайними цибульними маршрутизаторами, в мережу були додані так звані охоронці входу [6]. Мета охоронних пристроїв - запобігти атакам, коли зловмисна сторона намагається розмістити свій маршрутизатор в якості першого вузла ланцюга. У вихідній

реалізації перший вузол був обраний випадковим чином з усіх маршрутизаторів, які були підключені до мережі, що означало, що до тих пір, поки з'єднання формувалося знову і знову протягом достатнього часу, в якийсь момент зловмисний маршрутизатор був би обраний в якості першого вузол. Починаючи з версії 2006 років три захисних вузла замість одного вузла входу вибираються з невеликого числа надійних маршрутизаторів. Ці три захисних вузла будуть використовуватися від 30 до 60 днів для всіх каналів, які формує клієнт, перед тим як відмовитися й перейти до вибору нових вузлів входу [7]. Це значно збільшує здатність мережі протистояти атакам підслуховування.

Іншим заходом для підвищення безпеки є додавання так званих мостів [1]. З метою підвищення безпеки деякі маршрутизатори не оголошували свої адреси публічно. Мости існують тому, що звичайні загальнодоступні IP-адреси Tor блокуються деякими брандмауерами. Людина, яка не має доступу до мережі Tor через такі брандмауери, може використовувати мости для обхода цих фільтрів.

Також приховані сервіси були згадані раніше. Вони не підвищують інформаційну безпеку як таку, але розробники хотіли включити приховані сервіси. Ці послуги можуть використовуватися, наприклад, для відправки миттєвих повідомлень між клієнтами. Коли прихований сервіс хоче опублікувати свою адресу, він формує деякі цибулинні канали та повідомляє адресу останнього маршрутизатора каналу сервісів імен. Клієнт може сформувати з'єднання з сервісом через вузли зустрічей.

1.3.3 Створення цибулевого ланцюга

Давайте докладніше розглянемо, як формується ланцюг. Коли користувач Tor хоче встановити з'єднання з сервером за межами мережі Tor, його клієнтське програмне забезпечення спочатку завантажує список всіх зареєстрованих маршрутизаторів Tor [6]. З цих маршрутизаторів спочатку вибирається вхідний вузол. Вхідний вузол є першим маршрутизатором в цибулевому ланцюжку, і він вибирається зі списку маршрутизаторів, який служба імен оголосила як швидкою і

надійною. У разі, якщо обраний вхідний вузол недоступний, вибирається новий ихідний вузол.

Після вибору вузла входу формується інша частина схеми. Вузли, які перераховані з найбільшою пропускною спроможністю і часом безвідмовної роботи, з більшою ймовірністю будуть додані в канал. Щоб ланцюг був максимально стабільним, вибираються тільки ті вузли, які були ідентифіковані як стабільні. На Рисунок 1.1 показаний псевдоалгоритм для вибору вихідного маршрутизатора і самого середнього маршрутизатора в ланцюжку [6]. Алгоритм відрізняється від поточної реалізації тим, що маршрутизатор не може сам оголосити про пропускну здатність, але пропускна здатність вимірюється всередині мережі [8].

Алгоритм: Вибір вхідного маршрутизатор

Вхід: Список відомих Tor маршрутизаторів, router_list

Вихід: псевдовипадково обраний маршрутизатор, зважений по відношенню до маршрутизатора з максимальною пропускною спроможністю

```
B ← 0, T ← 0, C ← 0, i ← 0, router_bw ← 0,
bw_list ← Ø
foreach router r ϵ router_list do
  router_bw ← get_router_adv_bw(r)
  B ← B + router_bw
  bw_list ← bw_list ∪ router_bw
end
C ← random_int(1,B)
while T < C do
  T ← T + bw_list_i
  i ← i + 1
end
return router_list_i
```

Рисунок 1.1 - Псевдокод: Алгоритм вибору не вхідного маршрутизатора [6]

B = об'єднана смуга пропускання всіх маршрутизаторів, T = сума смуг пропускання в нижньому циклі, C = випадково вибране число з B, i = номер оброблюваної в даний момент смуги пропускання в нижньому циклі, router_bw = оголошена пропускна здатність, що обробляється в даний момент в верхньому циклі, bw_list = список з усіма смугами пропускання маршрутизатора.

Після того, як три маршрутизатори були вибрані випадковим чином, ланцюжок формується шляхом генерації ключів шифрування один за іншим з

кожним маршрутизатором з використанням протоколу Діффі-Хеллмана. Нові пари ключів узгоджуються поступово з кожним вузлом після того, як ключі були встановлені з попереднім вузлом в дорозі. Це забезпечує так звану пряму секретність, що означає, що жоден з вузлів не може відстежити ключі, які інші вузли використовують для відкриття повідомлень. Після цієї процедури кожен маршрутизатор має свій власний ключ шифрування для створення і очищення цибулеві, а також інформацію про те, кому відправляти повідомлення. На Рисунок 1.2 показано підключення до служби імен та цибульневий шлях після його формування.

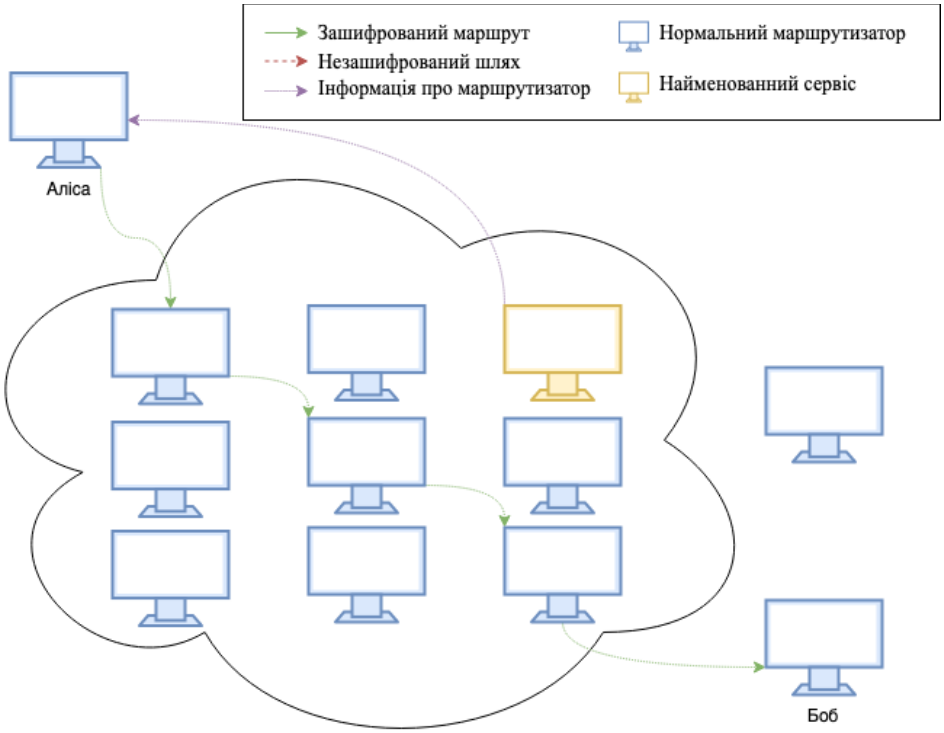

Рисунок 1.2 – Формування цибулевого шляху та взаємодія з службою імен

Клієнт Аліса завантажує список цибулевих маршрутизаторів зі служби імен та формує шлях. Вихідний вузол відправляє незашифроване повідомлення Боб.

В результаті формування шляху окремо з кожним маршрутизатором один єдиний вузол не може знати адреси всіх інших вузлів в ланцюзі. Перший маршрутизатор може бачити, що клієнт підключений до мережі, але він не може бачити кінцевий пункт призначення повідомлень, який користувач відправляє через Tor. Вузол входу не може знати, що користувач робить з мережею. Він отримує повідомлення і відправляє їх вперед по ланцюжку. Третій, останній вузол

може бачити, куди відправляються повідомлення, але він не може бачити, хто їх відправив або які дані у складі повідомлення. Третій вузол називається вузлом виходу. Вихідний вузол відправляє повідомлення до кінцевого одержувача.

Останній вузол в ланцюгу, який отримує незашифроване повідомлення від вихідного вузла, не обов'язково знає, що він є частиною цибулевого ланцюжку, тому що повідомлення залишає вихідний вузол незашифрованим. Тільки порівнюючи інформацію про відправника з заголовка зі списком відомих маршрутизаторів Tor (список зберігається на сайті torstatus.blutmagie.de [9]), одержувач може дізнатися, що повідомлення приходять зсередини цибулевої мережі. Багато інтернет-сервіси можуть блокувати трафік через адреси Tor.

1.3.4 Подорож повідомлення по цибулевому ланцюжку

DeFabbia-Kane [10] описує хід повідомлення через цибулеву мережу таким чином: клієнт формує цибулевий ланцюжок n-довжини (як описано раніше), після чого ланцюжок складається з маршрутизаторів R_1, …, R_n яким клієнт поширив симетричний ключ K_i з використанням протоколу Діффі-Хеллмана. Перед відправкою повідомлення клієнт спочатку шифрує його за допомогою ключа шифрування K_n, а після цього - за допомогою ключа K_{n-1} до рівня K_1.

Аліса хоче відправити повідомлення Бобу анонімно через Tor. Вона сформувала цибулевий ланцюг з трьома цибулинними вузлами: $R_1 \rightarrow R_2 \rightarrow R_3$. Позначення$[M]_{K_1}$ означає, що повідомлення M зашифровано симетричним ключем K_i, а позначення $[M]_{K_{i,j,k}}$означає, що повідомлення M зашифровано спочатку ключем K_i, потім ключем K_j і після це ключем K_k. Перш ніж Аліса відправить повідомлення через канал, її клієнт зашифрує його ключем K_3, потім ключем K_2 і, нарешті, ключем K_1. Повідомлення, яке в кінцевому підсумку відправить клієнт Аліси, буде $[M]_{K_{3,2,1}}$.

Кожен маршрутизатор відкриває або дешифрує повідомлення зі своїм власним ключем шифрування K_i, коли повідомлення переміщається по ланцюжку.

Після того, як маршрутизатор розшифрував повідомлення, він відправляє його наступному маршрутизатору в ланцюжку (або Бобу, якщо маршрутизатор є останнім вузлом в ланцюжку). Коли M проходить через ланцюжок, це буде виглядати так:

$$\text{Аліса} \xrightarrow{[M]_{K_{3,2,1}}} R_1 \xrightarrow{[M]_{K_{3,2}}} R_2 \xrightarrow{[M]_{K_3}} R_3 \xrightarrow{M} \text{Боб}.$$

Коли Боб відправляє у відповідь повідомлення M', він відправляє його на маршрутизатор R_3, який шифрує його ключем K_3і відправляє назад по ланцюжку. Кожний маршрутизатор R_i шифрує повідомлення ключем K_i. Шлях повідомлення назад по ланцюжку виглядає дуже схожим на шлях повідомлення M вперед по ланцюжку. Оскільки тільки Аліса знає всі три ключа K_1, K_2 і K_3, тільки вона може відкрити повідомлення M' і прочитати його.

$$\text{Аліса} \xleftarrow{[M]_{K_{3,2,1}}} R_1 \xleftarrow{[M]_{K_{3,2}}} R_2 \xleftarrow{[M]_{K_3}} R_3 \xleftarrow{M} \text{Боб}$$

1.3.5 Комірки

Повідомлення, які передаються між вузлами всередині цибулевого ланцюжка, називаються комірками [2]. На додаток до ретрансляції даних, комірки можуть також використовуватися, наприклад, для створення ланцюгів (так званих комірки CREATE), руйнування ланцюгів (комірки DESTROY) і підтримки живого з'єднання (комірки PADDING). Комірки мають фіксований розмір 512 байт, в основному через запобігання аналізу трафіку. Всі інші комірки, крім ретрансляційних комірок, називаються контрольними комірками, і у них є заголовок, який описує їх ідентифікатор каналу, який називається circID, і призначення комірки, зване командою контрольної комірки, або CMD.

Комірки, які використовуються для ретрансляції повідомлень, називаються комірками ретрансляції. Ці комірки мають трохи більше інформації в заголовок, ніж комірки управління: на додаток до circID і CMD заголовок комірки ретранслятора містить ідентифікатор потоку, контрольну суму і довжину корисного навантаження. Корисне навантаження даних має довжину 498 байт, і її

також можна використовувати для передачі керуючих комірок окремих вузлах в ланцюгах, наприклад, при виконанні рукостискання Diffie-Hellman. На Рисунок 1.3 показана структура комірки управління і комірки реле.

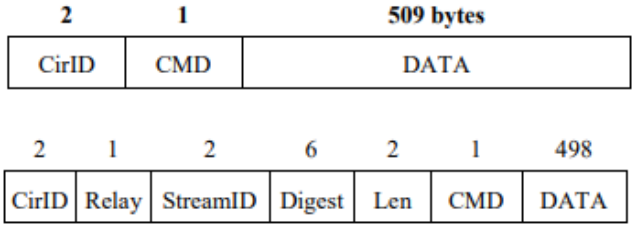

Рисунок 1.3 - Заголовок комірки управління (верхній) і заголовок комірки реле (нижній) [2]

Існують різні типи релейних комірок. Комірки RELAY DATA використовуються для передачі даних вперед або назад по ланцюгу. Комірки RELAY BEGIN використовуються для відкриття потоку, який закривається коміркою RELAY END. Комірки реле також використовуються для створення фактичної ланцюга через комірки RELAY EXTEND. Комірка «extend» повідомляє цибулевому маршрутизатору передати рукостискання вказаному маршрутизатору, і маршрутизатор відповідає коміркою CREATED в разі, якщо рукостискання пройшло успішно. На Рисунок 1.4 показана схема використання комірок при створенні схеми з двома стрибками.

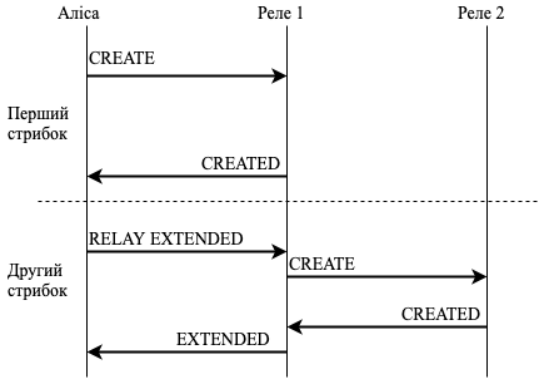

Рисунок 1.4 - Створення перших двох стрибків ланцюга

Висновки до розділу 1

В цьому розділі було розглянуте виникнення цибулевої маршрутизації та її ідея, технологічний огляд цибулевої маршрутизації та еволюція технології протягом трьох поколінь.

Наведено практична реалізація використання цибулевої маршрутизації під назвою Tor. Проаналізовано її особливості, алгоритм створення цибулевого ланцюжка, подорож повідомлення в цибулевому ланцюжку та розгляду значення комірок в системі.

Отже, програмна реалізація цибулевої маршутизації під назвою Tor створенна для забезпечення повної анонімності користувача та його дій в мережі Інтернет. Використовуючи програмне забезпечення Tor користувач забезпечує себе повною анонімності дій в мережі Інтернет завдяки цибулевої маршрутизації.

2 АТАКИ НА TOR ТА МЕТОДИ ПРОТИДІЇ НИМ

Метою цибулевої маршрутизації є запобігання локалізації та ідентифікації користувача. Це означає, що атаки на систему намагаються підключитися, наприклад, до журналів відвідування веб-сайту або відправки повідомлень назад користувачеві. У цьому розділі ми розглянемо, які атаки були розроблені проти Tor. Яким чином потенційний зловмисник може спробувати деанонімізувати користувачів і як мережа призначена для захисту від подібних атак? Деякі атаки можуть мати на меті зниження якості обслуговування, наприклад, за допомогою атак типу «відмова в обслуговуванні», але в данній тезі ми виключаємо подібні атаки і концентруємося на деанонімізація атак.

Цибулева маршрутизація захищає від ідентифікації за допомогою заплутування. На практиці цибулева маршрутизація може забезпечити анонімність, тільки якщо її користувацька база досить велика. Чим більше користувачів в мережі, тим більше цілей для об'єднання певної активності. Це означає, що один користувач не може створити власну цибулеву мережу і створити собі анонімний захист. Сила Tor в тому, що зловмисники не можуть простежити шлях повідомлень, а також відстежити повідомлення, які виходять з мережі, тому, хто їх відправляє. Це як мінімум концепція системи.

Існує безліч різних атак на криптографічні системи. Більшість атак на Tor засновані на спробі захопити один або два маршрутизатора в мережі. Зазвичай, принаймні, один маршрутизатор вузла входу в ланцюзі і, можливо, також вихідний вузол в тому же ланцюзі. За допомогою цієї настройки можна виконати атаку аналізу трафіку, яка намагається зв'язати вхідні і вихідні повідомлення. Пасивні атаки відстежують потік повідомлень, не втручаючись в нього, а активні атаки якимось чином модифікують трафік, щоб спростити аналіз. У цьому розділі ми розглянемо, які інші типи атак призначені для системи.

2.1 Категорії атак

Юха Сало ділить атаки на п'ять категорій залежно від їх особливостей. Сало виділяє п'ять категорій атак: імовірнісні моделі, атаки вибору маршрутизатора входу і виходу, атаки на рівні AS і на глобальному рівні, атаки на основі аналізу трафіку і часу, а також вразливості протоколів. У своїй статті Сало перераховує ці категорії і описує раніше опубліковані атаки, які відносяться до цих категорій. Давайте коротко розглянемо їх.

Імовірнісні моделі використовують математичні імовірнісні моделі для аналізу мережі. Ці моделі спочатку були призначені для MIX-мереж і використовують Байєсовські імовірнісні моделі, щоб виміряти, наскільки безпечна мережа. Такі ж моделі були розроблені для Tor. Моделі розглядають систему як чорний ящик, намагаючись створити математичні моделі, засновані на двох припущеннях:

- По-перше, що тільки один користувач пов'язаний з одним виходом.
- По-друге, що вихід може бути пов'язаний з користувачем у тому випадку, якщо можна спостерігати обидва виходи.

Ці моделі насправді не є атаками, а скоріше пропонують спосіб оцінити, наскільки можливо для зловмисника деанонімізувати користувача.

Атаки на вибір вхідного і вихідного маршрутизатора націлені на підвищення ймовірності вибору маршрутизатора атакуючого як вузол входу і/або виходу. Сало згадує про два типи атака з цієї категорії:

- По-перше, компрометація анонімності з використанням пакетного обертання, намагається створити петлі між маршрутизаторами, щоб вони могли відмовитися обслуговувати інших клієнтів, на практиці вбиваючи маршрутизатор. Мета полягає в тому, щоб збільшити ймовірність того, що зловмисні маршрутизатори будуть обрані в якості частини певного каналу.
- По-друге, низько ресурсною маршрутизацією проти Tor, які намагаються повідомити неправдиву інформацію про пропускну здатність і часу

роботи маршрутизатора службі імен, що дає маршрутизаторам більший пріоритет при виборі вузлів входу.

Атаки, засновані на аналізі трафіку і часу, охоплюють найбільшу кількість атак в категорії. Атаки такого роду намагаються послабити анонімність, спостерігаючи закономірності в трафіку, наприклад, під час мережевого потоку. За допомогою цих шаблонів зловмисник може зіставляти вхідний і вихідний мережевий трафік. Пасивні атаки виявляють, що пакети проходять через систему, а активні атаки намагаються, наприклад, помітити пакети водяними знаками, щоб їх було легше аналізувати.

У цій категорії сім дослідницьких робіт: Low-Cost Traffic Analysis of Tor представляє спосіб відстеження вузлів, які використовуються в ланцюзі, а також спосіб зв'язати незв'язані потоки з їх ініціатором на основі різних затримок в потоках. A cell counter based attack against Tor представляє техніку аналізу трафіку шляхом додавання водяних знаків для лічильника комірця. Browser-Based Attacks on Tor використовують атаку «людина-посередник» і модифікують HTTP-трафік для поліпшення аналізу. A Practical Congestion Attack on Tor Using Long Paths використовує атаку перевантаження з модифікацією HTTP потоку для вивчення шляхів каналів. How Much Anonymity does Network Latency Leak? вимірює затримки в мережі для розпізнавання користувача. Passive Logging Attacks Against Anonymous Communications Systems спостерігає за поведінкою користувача і намагається передбачити ініціатора певного шляху.

AS і атаки глобального рівня припускають, що зловмисник може спостерігати або маніпулювати значною частиною трафіку, який входить і виходить з мережі. Цей вид зловмисника може використовувати, наприклад, аналіз трафіку і часу для розпізнавання потоків даних. Tor не призначений для захисту користувача від AS (мається на увазі автономна система) рівня супротивника. Сало згадує дві статті про AS - усвідомлення в Tor Path Selection, які намагаються оцінити, наскільки ймовірний насправді зловмисник на рівні AS, і Large Scale Simulation of Tor, яке оцінює кілька атак аналізу трафіку в модельованій мережі.

Вразливості протоколу - остання категорія. Ці атаки намагаються знайти слабкі сторони в протоколах зв'язку. Ефективні атаки в протоколі аутентифікації Tor припускають вразливість в способі розподілу сеансових ключів, яка призведе до невідповідностей в сеансових ключах. Про ризик обслуговування, коли ви займаєтеся серфінгом намагається розпізнати мости в мережі Tor. Спочатку зловмисник намагається розпізнати можливих міст-кандидатів, а потім намагається підтвердити результати активною атакою на засмічення ланцюга.

2.2 Нові атаки на Tor

Атаки на Tor, здається, є популярним предметом дослідження, принаймні, якщо подивитися на кількість статей, які можна знайти за допомогою простого пошуку в Google. Після 2010 року з'явилося багато нових статей на цю тему, і ми розглянемо деякі більш популярні і менш популярні в цьому розділі.

2.2.1 The Bad Apple Attack

У 2010 році дослідники Le Blond і співавтор Французький інститут досліджень в області комп'ютерних наук і автоматизації виявив спосіб відстеження потоків TCP peer-to-peer (P2P) додатків, що працюють над Tor. Завдяки цій атаці дослідники змогли відстежити дев'ять відсотків всіх потоків, що проходять через їх вихідні вузли, виявивши, серед іншого, 10 000 користувачів BitTorrent, який являється протоколом для обміну файлами в мережі P2P. За допомогою цієї інформації вони змогли профілювати використання BitTorrent.

2.2.1.1 Схема атаки

Дослідники провели експеримент протягом 23 днів, спостерігаючи за шістьма вихідними вузлами, які вони контролювали. З етичних причин вони не

зберігали ніякої інформації, яка могла б бути використана пізніше для певних цілей. Атака заснована на спостереженні, хоча користувачі BitTorrent можуть шукати однорангові вузли усередині Tor, фактично більшість з них як і раніше здійснюють фактичний обмін контентом поза мережею Tor. Для цього може бути кілька причин:

- По-перше, спільне використання контенту поза Tor набагато ефективніше через більш високу пропускну здатність.
- По-друге, Tor не підтримує UDP, який є протоколом, що використовую BitTorrent для обміну інформацією про користувачів в мережі P2P, так звані трекери DHT.

Дослідники виявили, що 72 відсотки користувачів BitTorrent або підписують свої загальнодоступні IP-адреси на трекер DHT, і/або створюють P2P-з'єднання зі своїми загальнодоступними IP-адресами. Зловмисник може легко додати свій IP-адресу в список однорангових вузлів та чекати, поки до нього не підключиться інший вузол. Порівнюючи підключений IP-адреса зі списком відомих вихідних вузлів Tor, можна дізнатися, чи йде з'єднання від Tor або поза ним. Інший спосіб ідентифікації користувачів - моніторинг підписок на DHT через вихідний вузол.

Підписаний IP-адреса повинен знаходитися за межами Tor, оскільки Tor не підтримує UDP, який використовує DHT. Порівнюючи підписані комбінації IP/порт в списку підписок і рукостискань з цільовим користувачем, можна ідентифікувати користувача Tor.

Після виявлення вихідного IP одного потоку в вузлі виходу, всі потоки в межах цієї схеми можна простежити до одного і того ж користувача. Трасування потоків у різних схемах може бути виконано за допомогою ідентифікатора вузла або порівняння IP/порт. З цією інформацією дослідники змогли профілювати використання BitTorrent всередині Tor, наприклад кількість потоків, країни походження та кількість користувачів. Аналіз також виявив величезну кількість інших потоків, таких як HTTP-потоки з деяких браузерів. За допомогою інформації з браузера дослідники змогли побачити, які сайти відвідують користувачі Tor.

Оригінальна стаття доповнює інші дослідження, спрямовані на профілювання користувачів Tor, але також вводить новий метод атаки, який можна використовувати для виявлення IP-адрес користувачів. Це також може загрожувати іншим сервісам анонімності. Захист від атаки в основному залежить від того, наскільки обережний користувач при використанні системи. Дослідники також відзначають, що Peer Exchange (PEX) та централізоване відстеження UDP все ще можуть використовуватися таким же чином, що може стати потенційним предметом дослідження в майбутньому.

2.2.2 Application-level attack

У статті, опублікованій в 2011 році дослідниками з Південно-східного університету Китаю і Чанчжоуского коледжу інформаційних технологій в Китаї, описується атака типу «людина-посередник» на додатки з малою затримкою на основі TCP. Існує дві схеми атаки:

- При атаці з використанням підроблених веб-сторінок вузол шкідливого виходу відповідає підробленими веб-сторінками на веб-запити клієнта, створюючи характерний трафік для аналізу трафіку.
- Цільова атака модифікації веб-сторінки аналогічна, але замість того, щоб відповідати абсолютно новою веб-сторінкою, вона передає вихідну сторінку, як першу, щоб мінімізувати затримку, а замість цього впроваджує невидимі посилання на сторінку.

Обидві ці атаки можуть потенційно використовуватися для ідентифікації клієнта.

2.2.2.1 Схема атаки

Перша схема, атака з використанням підроблених веб-сторінок, створює більш відмітний шаблон, ніж друга, тому розпізнавання потоків більш ефективно. Атака вимагає, щоб один вхідний маршрутизатор і один вихідний маршрутизатор

були підключені до мережі Tor. Вихідний маршрутизатор відповідає на веб-запит з підробленою веб-сторінкою, що містить теги img, які веб-браузер одержувача потім обманює при отримані. Приклад підробленої веб-сторінки з оригінальної статті показаний на Рисунок 2.1 запит на вибірку цих зображень створює схему трафіку, що складається з ретрансляційних комірців, в п'ять разів перевищує кількість вставлених тегів img за певний проміжок часу. Маршрутизатор входу може розпізнавати комірці за ідентифікатором каналу (CircID) і по команді комірця (CMD або Relay). Після цього потік даних може бути підтверджений шляхом порівняння даних спостережень між маршрутизатором входу і виходу.

```
<html>
<head>
<meta http-equiv="refresh" content="w;url= URL_Bob">
</head>
<body>
<img src=" URL_Bob/img1.gif" style="display:none"/>
<img src=" URL_Bob/img2.gif" style="display:none"/>
<img src=" URL_Bob/img3.gif" style="display:none"/>
... ...
<img src=" URL_Bob/imgm.gif" style="display:none"/>
</body>
</html>
```

Рисунок 2.1 – Введення img тегів у підроблену сторінку

Перша версія атаки може бути виявлена, тому що це створює затримку в комунікації. Друга схема, атака модифікації цільової веб-сторінки, працює аналогічно першій схемі, але відмінність полягає в тому, що вихідний вузол відповідає справжній веб-сторінці в яку він вводить деякі теги img. У цій ситуації шаблон трафіку складніше виявити, оскільки веб-сторінка може викликати додатковий трафік. Запуск атаки в кілька разів підвищує точність.

2.2.2.2 Результат та захист

Дослідники проаналізували атаку, а також провели кілька експериментів з невеликою вибіркової мережею. Згідно зі статтею, аналіз показує, що обидві схеми атаки придатні для профілювання клієнтів і прихованих серверів. Частота

виявлення першої схеми не страждає від посилань всередині цільової веб-сторінки, але частота виявлення другої схеми зменшується, якщо цільова веб-сторінка містить посилання. Проте, обидві схеми залишалися ефективними для кореляції зв'язку.

Три типу захисту пропонується проти цієї атаки. Перший тип захисту полягає в тому, щоб зменшити ймовірність вибору шкідливого маршрутизатора для каналу шляхом розгортання в мережі більшої кількості безпечних маршрутизаторів і поліпшення алгоритму вибору маршрутизатора, щоб зробити його більш точним. Другий тип захисту полягає в тому, щоб спробувати виявити ненормальну активність, таку як отримання невидимих зображень перед показом веб-сторінки. Третій тип захису - використовувати HTTPS для з'єднання, щоб запобігти атаці «людина-посередник».

2.2.3 Probabilistic Analysis in a Black-box Model

Джоан Фейгенбаум з Єльського університету, а також Аарон Джонсон і Пол Сіверсон з Військово-морської дослідницької лабораторії США в 2012 році опублікували статтю про імовірнісний аналізі цибулевої маршрутизації. Цей метод насправді не є атакою на систему в тому сенсі, що він намагається позбавити користувача анонімності, але він підпадає під категорії атак, які ми ввели на початку цього розділу. Мета цієї атаки - надати математичне доказ анонімності, яку забезпечує цибулева маршрутизація. Доказ змодельоване проти обмеженого в обчислювальному відношенні активного противника, який ставить під загрозу деяку невідому частину системи.

Модель, яку вони використовують для аналізу, являє собою абстракцію системи, що забезпечує анонімність, тому її також можна використовувати для аналізу інших анонімних систем. Існує три припущення про систему в моделі, яку вони описують:

- По-перше, зловмисник може знати, чи спостерігає він за користувачем чи ні.

- По-друге, один користувач може бути пов'язаний тільки з одним підключенням в системі.
- По-третє, припущення полягає в тому, що зловмисник не може дізнатися, чи є користувач і з'єднання частиною одного і того ж потоку, просто спостерігаючи за системою.

Анонімність, описана цією моделлю, - це анонімність відносин між користувачами, що визначає, наскільки добре зловмисник може визначити, чи спілкувалися дві сторони один з одним. По суті, анонімність системи вимірюється ймовірністю вибору користувачем певного пункту призначення. У своїй 29-сторінковій статті дослідники оцінюють спрощені нижні (мінімальні) значення границь і найгірші підвищені (максимальні) значення границь для анонімності користувача в мережі Tor.

У документі показано, що анонімність користувача є найгіршою, коли, принаймні, у кожного другого користувача є те саме призначення, що і у нього, але це, звичайно, дуже малоймовірна ситуація, і саме тому оцінюється більш реалістичний сценарій, заснований на поширенні Zipfian. (Закон Zipf говорить нам, що з'єднання з пунктами призначення розподіляються нерівномірно, але найпопулярніші з'єднання матимуть експоненціально більше з'єднань, ніж менш популярні.) Використовуючи розподіл Zipfian, показано, що анонімність системи зазвичай дуже сильна, близько до нижньої границі.

Ця модель, звичайно, є абстракцією, яка не враховує багато відомих атаки на цибулеву маршрутизацію, але вона намагається продовжити постійний процес формалізації анонімності, що охоплює більшість з цих ситуацій.

2.2.4 CellFlood

CellFlood - це атака, яку можна використовувати для того, щоб зробити маршрутизатори Tor недоступними для користувачів, безперервно відправляючи їм запити, поки маршрутизатор не відмовиться обслуговувати інших клієнтів. Ця атака типу «відмова в обслуговуванні» була виявлена в 2013 році чотирма

дослідниками з університету Сапієнца в Римі і Колумбійського університету в Нью-Йорку. Дослідники також розробили захист, що включає так звані клієнтські головоломки проти цієї атаки, яка також була додана в програмне забезпечення Tor.

2.2.4.1 Схема атаки

Атака використовує той факт, що шифрування повідомлень за допомогою відкритих ключів приблизно в двадцять разів легше, ніж відкриття зашифрованих повідомлень за допомогою закритого ключа. Коли цибулева схема створюється вперше, сеансові ключі для кожного маршрутизатора розподіляються з використанням відкритого ключа кожного маршрутизатора з використанням так званих комірців CREATE. Вартість відправки великої кількості комірців CREATE на цибулевий маршрутизатор для зловмисника набагато дешевше, ніж для атакованого маршрутизатора.

Маршрутизатор обробляє комірця CREATE в окремому процесі від основного потоку ретрансляції, тому отримання великої кількості комірців CREATE не порушує процес ретрансляції повідомлень. Однак фоновий процес створення нових з'єднань може виконувати тільки обмежений обсяг роботи, а коли він не може обробити будь-які нові запити, він починає їх відкидати. Було виявлено, що маршрутизатору-противнику навіть не потрібно створювати нові комірця для кожного запиту, що ним відправляється, але він може відправляти одну і ту ж комірку знову та знову. Це дозволяє виконати дешеву DoS-атаку, щоб не дати певному маршрутизатору створювати нові канали і обслуговувати нових клієнтів.

2.2.4.2 Результат та захист

Дослідники експериментували з атакою в контрольованому середовищі і з реальною мережею Tor, і у всіх їх тестах атака залишалася дуже ефективною в порушенні здатності цілей обробляти нові запити. Вони також оцінили, як може

вплинути CellFlood, і оцінили, що зловмисникові потрібно невелика смуга пропускання, близько 116-232 Мбіт/с, для засмічення одного маршрутизатора.

Запропонований і реалізований захист полягає в тому, щоб змусити клієнта виконувати деяку обчислювальну роботу кожен раз, коли він хоче відправити запит на обробку маршрутизатора. Ця робота зазвичай включає в себе рішення криптографічної головоломки, яку маршрутизатор входу перевіряє на правильність. Дослідники виміряли, що такого роду клієнтська головоломка в більшості випадків ефективно блокує DoS-атаки типу CellFlood.

2.2.5 EgotisticalGiraffe

У 2013 році Едвард Сноуден отримав відомий витік у вигляді тисячі надсекретних документів Агентства національної безпеки США. Одним з цих тисячі документів була презентація PowerPoint, в якій описувалося, як агентство змогло ідентифікувати деяких користувачів Tor. Хоча для цього методу атаки немає офіційної документації, легко отримати уявлення про те, як це було зроблено, тому ми коротко розглянемо метод тут.

2.2.5.1 Схема атаки

Атака складається з двох етапів: перший етап полягає в тому, щоб ідентифікувати користувачів Tor в мережі Інтернет, і після цього другий етап полягає в тому, щоб направити ідентифікованих користувачів на сайт, який заразив би комп'ютер користувача деяким призначеним для користувача шкідливим ПЗ для подальшої ідентифікації. Перший етап ідентифікації користувача Tor досить простий. Пакет, в який користувачі завантажують прикладну програму Tor, називається Tor Browser Bundle і поставляється з налаштованим браузером Firefox, який налаштованй для перегляду в режимі Tor. Зазвичай браузери Firefox оголошують BuildID, який повідомляє веб-сайту про версії випуску браузера, з якої

користувач переглядає сайт. На слайдах оголошується, що BuildID Tor Browser Bundle за замовчуванням був встановлений рівним нулю замість стандартного номера збірки, а це означає, що користувачі, які звертаються до сайтів з браузерами Firefox з нульовим BuildID, є потенційними користувачами Tor.

Другим етапом атаки є атака «людина-посередник», щоб заразити комп'ютер потенційного користувача Tor шкідливим експлойтом Firefox. Цільовий користувач був спрямований на спеціальний сервер, який потім заражав б браузер користувача спеціально розробленим експлойтом. Особливий експлойт, згаданий в слайдах, використовує розширення для JavaScript, але ходять чутки, що були використані й інші вразливості. Після того, як комп'ютер користувача був заражений цим зловмисним експлойтом, він відправляв би всі види даних зворотного виклику про користувача, які можна було б легко використовувати для їх ідентифікації.

2.2.5.2 Результат та захист

На слайдах також повідомляється, що були проведені деякі тести і були деякі початкові проблеми з Tor Browser Bundle, але в підсумку все запрацювало. Крім цього, невідомо, скільки призначених для користувацьких данних ця конкретна атака була проведена. Незабаром після опублікування цієї атаки Mozilla виправила уразливість JavaScript в Firefox, відключивши експлуатований функцію E4X. Наряду з цим також була виправлена вразливість в комплекті Tor Browser. Tor Browser Bundle також поставляється з опцією NoScript, яка відключає цей вид атак, але він не включений за замовчуванням і вимагає, щоб користувач включив його самостійно.

2.2.6 Snipper Attack

У 2014 році чотири дослідника з NRL і Берлінського університету ім. Гумбольдта виявили новий недолік алгоритмів управління потоком даних Tor. Вони назвали нову атаку, що використовує цю слабкість Sniper Attack. Атака вимагає мінімальної кількості ресурсів і може використовуватися для паралізації довільних вузлів з мережі Tor. На думку дослідників, Sniper Attack потенційно може бути використаний для порушення анонімності прихованих сервісів. Вони розробили три різні типи захисту від цієї атаки, і один з цих типів захисту був доданий в програмне забезпечення Tor.

2.2.6.1 Схема атаки

Sniper Attack використовує так зване управління потоком Tor. Управління потоком використовується в ситуації, коли велика кількість пакетів відправляється з одного місця в інше, наприклад, з сервера на клієнт. Проста відправка великої кількості пакетів даних за один раз може привести до перевантаження приймаючої сторони, тому управління потоком необхідно для збільшення швидкості доставки пакетів. Управління потоком дозволяє клієнту контролювати швидкість передачі, змушуючи вихідний вузол схеми буферизувати дані і дозволяючи клієнту відправляти так звані комірці SENDME кожен раз, коли він готовий до читання з цього буфера. Кінець доставки зберігає буфер в 1000 комірців у своїй пам'яті і відправляє 100 комірців для кожного отриманого виклику SENDME (додаючи ще 100 комірців в буфер). Протокол йде наступним чином:

1. Клієнт сформував ланцюг через цибулеву мережу до зовнішнього сервера.
2. Клієнт: Посилає запит на завантаження даних з сервера.
3. Вихід: Відправляє запит на сервер і починає буферизацію даних до 1000 комірців.
4. Клієнт: Читає дані і відправляє SENDME, коли прочитано 100 комірців.

5. Вихід: Додає ще 100 осередків в буфер.
6. Якщо не всі дані були прочитані, перейдіть до кроку №4.

Цей протокол можна використовувати для атаки на вузли входу або виходу двома різними способами. Перша версія атаки показана на Рисунок 2.2. Якщо зловмисник хоче закрити вхідний вузол, він формує схему таким чином, що власний вузол противника використовується в якості виходу. Протокол передбачає, що вихідний вузол виконує фактичне управління потоком, але оскільки в цій ситуації вихід контролюється зловмисником, він може відправляти стільки даних, скільки він хоче, ігноруючи межі управління потоком пакетів. Хитрість в тому, що клієнт не читає з буфера, а вихід відправляє безліч даних, що призводить до того, що пакети даних збираються в буфері вузла входу, що в кінцевому підсумку призводить до його аварійного завершення.

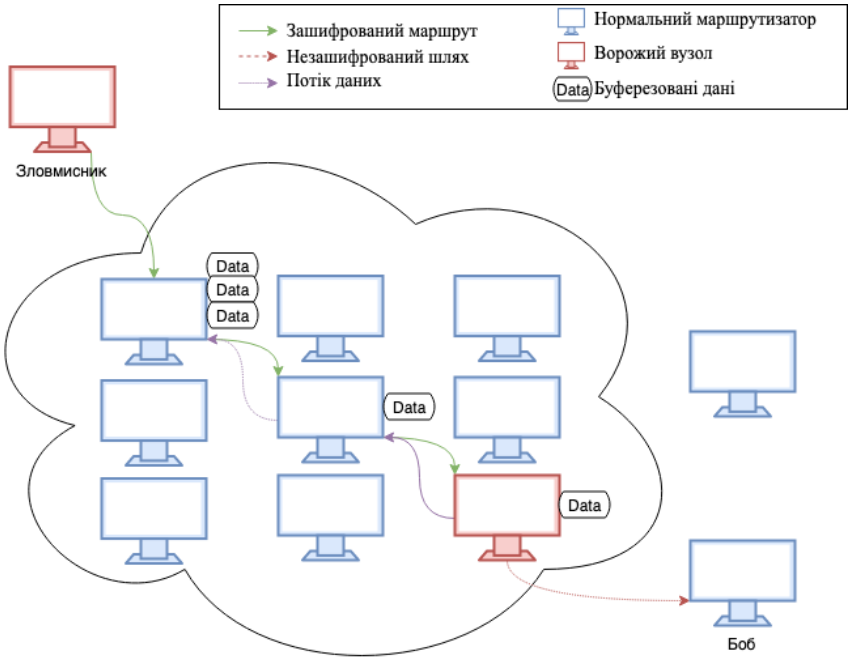

Рисунок 2.2 – Формування цибулевого шляху з зловмисником

Зловмисник контролює клієнта та вихідний вузол, орієнтуючись на вхідний вузол. Вихідний вузол відправляє обсяг даних через схему, і одночасно клієнт припиняє читання з вхідного вузла, що викликає заповнення буфера вхідних вузлів і, в кінцевому підсумку, збій.

Інший спосіб націлений на вихідні вузли, що дуже схоже на минулий спосіб атаки, але дані відправляються в іншому напрямку в ланцюзі. Клієнт, керований

зловмисником, відправляє дані на сервер, знову ігноруючи обмеження управління потоком. Приймаючий сервер, що належить також зловмисникові, припиняє читання з з’єднання, в результаті чого дані заповнюють буфер вихідних вузлів. Це також призводить до збою цільового вузла.

Існує також ефективна версія другої атаки, яка вимагає, щоб противник контролював тільки клієнта. Вихідний вузол має обмеження в 1000 комірців, після чого він припиняє буферизацію даних і закриває канал, але у проміжних і вхідних вузлів таких обмежень немає, і вони будуть продовжувати буферизацію до тих пір, поки клієнт не дасть іншу команду. Зберігаючи відправку пакетів відправляючим вузлом, відправляючи повідомлення SENDME, зловмисник може продовжувати вводити дані в канал, не зчитуючи їх. Об'єднуючи декілька потоків, атака може бути дуже ефективною.

2.2.6.2 Результати та захист

Snipper Attack можна використовувати, щоб змусити прихований сервіс вибрати вузол атакуючого в якості захисного реле і використовувати це шкідливе захисне реле для деанонімізація прихованого сервісу. Дослідники перевірили цю атаку і з'ясували, що деанонімізація прихованих сервісів за допомогою цієї атаки можлива, і оцінили, що зловмисник може використовувати її для відключення 20 кращих вихідних реле Tor.

Захист, який був обраний для додавання в Tor, полягає в тому, щоб убити канал, якщо його пам'ять починає вичерпуватися. Після цього дана атака стала неефективною. Однак автори нагадують, що, хоча ця атака більше не може використовуватися для DoS маршрутизатора, вона все одно може використовуватися для використання її пропускної здатності.

2.2.7 New traffic confirmation attacks

В останні роки було опубліковано кілька атак з підтвердженням трафіку (або кореляцією трафіку) з використанням різних методів, наприклад, записів потоку від програмного забезпечення під назвою Netflow, введення сигналу для передачі інформації між вузлами і зняття відбитків каналу шляхом маркування трафіку для забезпечення кореляції. Атаки з підтвердженням трафіку в групі безумовно мають найбільшу кількість розроблених атак. Команда розробників Tor з самого початку прокоментувала атаки з підтвердженням трафіку, заявивши, що це тривіально зосередитися на атаках з підтвердженням трафіку, оскільки цибулева маршрутизація не намагається захиститися від них. Замість цього Tor намагається захиститися від атак аналізу трафіку, які зловмисник може використовувати для визначення точок атаки в мережі.

У 2014 році розробники прокоментували в своєму блозі нову атаку для підтвердження трафіку з використанням журналів Netflow, яка:

«It’s great to see more research on traffic correlation attacks, especially on attacks that don’t need to see the whole flow on each side. But it’s also important to realize that traffic correlation attacks are not a new area».

Що в перекладі означає:

«Приємно бачити більше досліджень атак з кореляцією трафіку, особливо атак, яким не потрібно бачити весь потік з кожного боку. Але також важливо розуміти, що атаки з кореляцією трафіку не є новою областю».

Розглянемо деякі з таких атак.

2.2.7.1 Потік записів

У статті під назвою «Ефективність аналізу трафіку в анонімних мережах з використанням Flow Records» (2014 року) оцінюється, чи можна використовувати програмне забезпечення для аналізу мереж NetFlow, упаковане в маршрутизатори Cisco, для аналізу анонімних мереж, таких як Tor. Саме програмне забезпечення

NetFlow не є досить точним для аналізу трафіку, але записи, які воно надає, потенційно можна проаналізувати, щоб знайти джерело з'єднання, що надходить на сервер. Автори представляють атаку, яка передбачає великого противника на рівні AS, але кажуть, що атака також може бути здійснена зловмисником, які не є AS, якщо може бути ідентифікований цільовий вхідний вузол.

При атаці використовуються дані NetFlow між сервером і виходом, а також входом і клієнтом, щоб знайти підходящі шаблони з коефіцієнтом кореляції Пірсона. Він також вводить деякі шаблони трафіку в TCP-з'єднання. Результати повідомляють про 81,4% успіху з 12,2% хибно негативні спрацювання і 6,4% хибно позитивні спрацювання в реальному мережі Tor.

2.2.7.2 «Relay early». Атака підтвердження трафіку

Це ще одна атака, про яку не написано жодної статті, але яка набула широкого розголосу. Подробиці були повідомлені в блозі Tor в липні 2014 року, коли було виявлено, що якийсь невідомий зловмисник здійснює активну атаку підтвердження трафіку проти мережі Tor. При атаці використовувалися так звані «ранні ретрансляції» - комірці, які були додані в Tor для запобігання побудови дуже довгих шляхів. Зловмисники використовували ці ранні комірці ретрансляції проти своєї призначеної мети, щоб відправити інформацію про приховані адреси служб від вузлів виходу до вузлів зловмисного входу, які зловмисники раніше помістили в мережу.

Невідомо, яку інформацію зловмисники змогли отримати від цієї атаки, або навіть якщо метою зловмисників було дослідження, шкода або щось ще. Швидше за все, зловмисники не отримали адреса користувачів або прихованих сервісів або чого-небудь ще, що могло б поставити під загрозу безпеку користувачів. Як завжди, програмне забезпечення Tor було доповнено засобами захисту, основною з яких це було не дозволяти використовувати «ранні» комірці для атак подібно цієї. Щоб запобігти подібним атаки в майбутньому, існує також план зі зміни клієнтів

Tor на використання тільки одного засобу захисту входу замість трьох, щоб зменшити вразливість клієнтів в мережі, але це ще не реалізовано.

2.2.7.3 Схема зняття відбитків пальців

У статті «Атаки з використанням відбитків пальців: пасивна деанонімізація прихованих сервісів Tor», опублікованій в 2015 році, розглядаються деякі можливі недоліки в комунікації прихованих сервісів і пропонуються дві нові атаки з деякими можливими схемами захисту. Перша атака починається з пошуку трафіку, який виглядає як створення прихованого сервісу, і після цього намагається ідентифікувати сервіс з атакою за відбитками пальців сайту. Друга атака націлена на сценарій, коли клієнт використовує тільки один елемент захисту входу, і в цьому випадку зловмисник може побачити всі канали клієнта.

Ефективність обох атак оцінювалася в реальному мережі Tor. Дослідники повідомляють, що 98% та 99% представляють істинно позитивні показники з атаками і 0,1% та 0,07% хибно позитивних показників. Вони також пропонують деякі засоби захисту від атак: скорочення часу життя каналу і відправка випадкових комірців заповнення. Щодо другої атаки пропонується випереджаюча побудова каналів, щоб виключити ідентифікацію встановлення прихованого каналу обслуговування.

2.2.8 A Stealthy Attack Against Tor Guard Selection

У 2015 році три дослідника з Університету електронної науки і технологій Китаю і Китайської академії наук виявили спосіб прискорити вибір нових захисних вузлів для цибулевого ланцюжка в ситуації, коли зловмисник може контролювати частину користувача, а саме точку входу трафіку. Зазвичай клієнт використовує один захисний вузол протягом одного-двох місяців, перш ніж вибрати інший. Можливість вибору вузла шкідливого входу описується наступним чином:

«…if an attacker controls C out N relays (ignoring bandwidth), then the attacker will control both the entry and exit nodes of any given circuit with probability $\left(\frac{C}{N}\right)^2$… While using entry guards, we would have probability $\frac{N-C}{N}$ - to choose a good entry and not be compromised until the next round of entry guards ».

Що в перекладі означає:

« ... Якщо зловмисник контролює C з N ретрансляторів (ігноруючи смугу пропускання), то зловмисник буде контролювати і вузли входу та виходу будь-якого заданого каналу з імовірністю $\left(\frac{C}{N}\right)^2$... При використанні засобів захисту входу у нас буде можливістю $\frac{N-C}{N}$ вибрати хороший вхід і не бути скомпрометованим до наступного раунду входу захисту ».

2.2.8.1 Схема атаки та результати

Перша частина атаки спрямована на підвищення ймовірності того, що цільовий користувач (званий Алісою) вибере маршрутизатор атакуючого як вузол захисту. Захисні вузли вибираються випадковим чином, але більшу вагу віддається вузлів з більш високою пропускною здатністю. Ось чому зловмисникові необхідно розгорнути маршрутизатори з високою пропускною здатністю в якості захисних вузлів. Крім того, чим більше захисних вузлів у атакуючого, тим більша ймовірність того, що Аліса вибере один з них в якості маршрутизатора. Таким чином, збільшення зміни Аліси при виборі шкідливого захисного вузла здійснюється шляхом розгортання максимально можливої кількості захисних вузлів з високою пропускною здатністю.

Друга частина атаки намагається скоротити час, за яке Аліса вибирає захисні вузли. Цей інтервал часу зазвичай становить 30-60 днів, але його можна змінити приблизно до однієї-півтори хвилини, заблокувавши два з трьох обраних захисних вузлів, обраних Алісою. Оскільки ця атака вимагає, щоб зловмисник міг

контролювати частину трафіку цільового користувача, він може заблокувати трафік для всіх, крім одного, захисних вузлів.

Дослідники оцінюють атаку як дуже складну для виявлення і дуже ефективно підриває анонімність цільового користувача. Атака не впливає на використання мережі цільовим користувачем, і результати їх експериментів показують, що понад 80% користувачів можуть бути скомпрометовані приблизно через 30 хвилин за допомогою цієї техніки.

2.3. Модель загроз для системи Tor

Підводячи підсумок існуючих атак можна зробити висновок, що більшість існуючих атак направлені на те, щоб користувач при формуванні зв’язку в системі зробив вибір у бік маршрутизаротів зловмисника для подальшого аналізу трафіку користувача зловмисником з ціллю деанонімізації користувача. Підсумок існуючих атак наведено в Таблиця 2.1.

Таблиця 2.1 – Модель загроз існуючих атак

Атака	**Принцип дії**	**Порушення (К, Ц, Д чи С)**	**Імовірність + Наслідки**	**Спосіб протидії**
The Bad Apple Attack	Атака на спостереження	К	Середня + Критична	Обережність користувача
Application-level attack	Атака типу «людина-посередник»	К, Ц	Низька + Критична	Поліпшення алгоритму вибору маршрутизатора, виявлення ненормальної активності, використання HTTPS-з’єднання
Black-box Model	Імовірнісний аналіз	К	Низька + Критична	-

Продовження Таблиця 2.1.

CellFlood	Атака типу «відмова в обслуговуванні»	Ц, Д	Середня + Критична	Виконання головоломки на стороні клієнта
Egotistical Giraffe	Атака типу «людина-посередник»	К	Низька + Критична	Використання опції NoScript
Snipper Attack	Атака типу «людина-посередник»	К, Д	Низька + Середня	Убити канал, якщо його пам’ять починає вичерпувати-ся
New traffic confirmation attacks	Атаки з підтвердженням трафіку	К	Середня + Критична	Не давати використовувати «ранні» комірці, використання одного засобу захисту входу, скорочення часу життя каналу і відправка випадкових комірців заповнення, випереджаюча побудова каналів, щоб виключити ідентифікацію встановлення прихованого каналу обслуговування
A Stealthy Attack Against Tor Guard Selection	Атака типу «людина-посередник»	К	Низька + Критична	-

Грунтуючись на Рисунок 2.3, ми бачимо, що у теоретичного зловмисника буде певний набір опцій, з яких можна починати деанонімізацію користувачів або прихованих сервісів. Так званий глобальний зловмисник може використовувати

атаку кореляції трафіку, щоб ідентифікувати велику кількість потоків, щоб їм не довелося нічого робити додатково, щоб отримати інформацію про трафік.

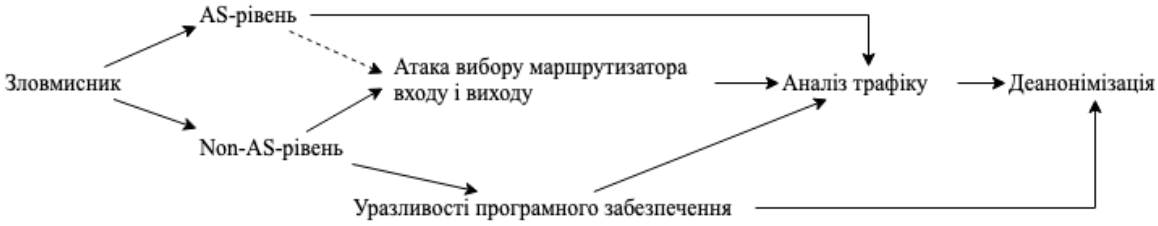

Рисунок 2.3 – Схема здійснення загроз

Однак, вони також можуть використовувати якусь атаку вибору входу і виходу маршрутизатора, щоб поліпшити своє становище для аналізу трафіку. Однак це може не бути необхідним.

У разі зловмисника не на рівні AS основним класом атаки, який зловмисник буде використовувати для порушення анонімності, є знову аналіз трафіку. Однак, перш ніж проводити аналіз трафіку, їм потрібно буде виконати деякі попередні кроки, щоб отримати можливість використовувати атаку для підтвердження трафіку. Ми бачили, що зловмисник може використовувати це двома різними способами, щоб дістатися до цієї позиції: атаки вибору маршрутизатора входу і виходу або уразливості на рівні програмного забезпечення.

Атаки на вибір цибулинних маршрутизаторів входу і виходу самі по собі не можуть використовуватися для деанонімізація користувачів, тому за ними завжди повинен слідувати якийсь пасивний або активний метод аналізу. Однак уразливості в програмному забезпеченні можуть розкрити деяку інформацію про користувачів, тому вони можуть також призвести до деанонімізація безпосередньо.

Висновки до розділу 2

В цьому розділі були описані основні категорії атак на систему Tor. Детально описаний кожний тип атаки: схема атаки, мета атаки та результати проведення цих атак.

Крім цього описані способи протидії кожної з атаки, які були добавлені в програмне забезпечення Tor для унемножування проведення таких атак в майбутньому.

На основі розділу 2 необхідно розробити програмне рішення для захисту від атак типу аналізу трафіку і часу, чому буде присвячений наступний розділ.

3 ПРОГРАМНЕ РІШЕННЯ ДОДАВАННЯ ВИПАДКОВИХ ЗАТРИМОК В СИСТЕМІ TOR

Основною метою цього розділу є створення програмного рішення для забезпечення збільшення анонімності користувачів в системі Tor завдяки додаванням випадкових затримок під час проходження пакетів по мережі від клієнта до сервера й навпаки.

Дане рішення несе в собі мету протидіяти атакам виду аналізу трафіку й часу метою яких є визначення відправника або прийомника пакетів в системі Tor для подальшої деанонімізації користувача.

3.1 Теоретичні основи

Система Tor побудована використовуючи TCP прокотол для відправки й прийняті пакетів. Завдяки даному механізму є можливість перехоплювати та модифікувати пакети за допомогою спеціально написаного програмного коду.

Програмний код написаний на мові програмування Go використовуючи бібліотеку з відкритим кодом під назвою Trudy, який запускається на віртуальній машині Vagrant реалізація якої написана на мові програмування Ruby.

Віртуальна машина використовується для перехоплення трафіку, який напередодні перенаправляється з хост-машини. Завдяки цьому відбувається ізоляція програмного рішення в якому відбувається додавання затримок при проходженні пакетів через віртуальну машину з подальшим відправленням у мережу Tor.

3.1.1 Генератор псевдовипадкових чисел

Під час реалізації програмного рішення потрібно забезпечити генерування випадкового числа, який виконує роль затримки пакетів у секундах. Розглядати будемо генератори псевдовипадкових чисел.

Генератор псевдовипадкових чисел - алгоритм, який породжує послідовність чисел, елементи якої майже незалежні один від одного і підкоряються заданому розподілу (зазвичай рівномірному).

Якісні вимоги, що пред'являються до ГПСЧ:

- Досить довгий період, який гарантує відсутність зациклення послідовності в межах розв'язуваної задачі. Довжина періоду повинна бути математично доведена.
- Ефективність - швидкість роботи алгоритму і малі витрати пам'яті.
- Відтворюваність - можливість заново відтворити раніше згенерувану послідовність чисел будь-яку кількість разів.
- Портативність - однакове функціонування на різному обладненні та операційних системах.
- Швидкість отримання X_{n+i} елемента послідовності чисел, при завданні X_n елемента, для i будь-якої величини; це дозволяє розділяти послідовність на кілька потоків (послідовностей чисел).

Для перевірки послідовностей використовуються критерії для перевірки якості випадкових двійкових послідовностей, що перевіряють статистичні властивості псевдовипадкових послідовностей: рівноімовірність знаків, незалежність сусідніх знаків, однорідність послідовності. Всі ці критерії є критеріями хі-квадрат Пірсона для перевірки відповідним чином сформульованих гіпотез.

Розглянемо послідовність $\{Y_j\}$, j = 1, …, m, де кожна Y_j є випадковою величиною, що приймає набір значень із алфавіту A. Всі величини Y_j мають однаковий розподіл та розглядаються як вихідні значення деякого генератору.

Послідовність $\{Y_j\}$ задовольняє умові рівноімовірності знаків, якщо кожна Y_j розподілена рівноімовірно на A. Таким чином, кожне значення із A повинно зустрічатись у довільній реалізації даної послідовності однакову кількість разів.

Тест на виконання умови рівноімовірності не відрізняється великою чутливістю, однак він доволі швидкий. В практичних задачах його рекомендується

застосовувати в першу чергу, оскільки якщо послідовність не пройде цей тест, то немає рації застосовувати до неї інші. Також в якості підсилення можна розглядати умову рівноімовірності серій знаків, коли рівноімовірними в послідовності повинні бути пари, трійки, четвірки знаків тощо.

Послідовність $\{Y_j\}$ задовольняє умові незалежності знаків, якщо імовірність прийняти деяке значення для Y не залежить від того, які значення прийняли $Y_1, Y_2, \dots, Y_{j-1}$. Однак перевірка такої умови зазвичай вкрай важка, тому часто розглядають більш послаблені вимоги – наприклад, значення Y_j не повинно залежати від значення Y_{j-1} (незалежність від попереднього знаку).

Послідовність $\{Y_j\}$ задовольняє умові однорідності, якщо для довільної реалізації вибірковий розподіл, одержаний на всій послідовності, буде співпадати із вибірковим розподілом, одержаним на довільній її підпослідовності достатньої довжини; іншими словами, на довільному фрагменті послідовність веде себе однаково. Зауважимо, що для виконання умови однорідності не важливо, який саме розподіл будуть мати Y_j. Зокрема, цей розподіл не обов'язково повинен бути рівноімовірним.

Сформулюємо критерії Пірсона для кожної з наведених умов. Надалі ми вважаємо, що послідовність, яка перевіряється, представлена у вигляді байтової послідовності, тобто область значень кожної випадкової величини Y_j лежить між 0 і 255.

3.1.2 Архітектура взаємодія компонентів

Архітектура кінцевої системи складається з наступних компонентів:

- Клієнт Tor.
- Хост-машина.
- Віртуальна машина VirtualBox з оболонкою Vagrant.
- Програмне рішення Trudy на мові Go.

Клієнт Tor встановлюється на хост-машині або окремий машині з підключенням до хост-машини. На хост-машині встановлюється віртуальна машина VirtualBox з оболонкою Vagrant. На віртуальній машині встановлюється та запускається програмне рішення Trudy для перехвату та затримки пакетів. Взаємодію компонентів заданої архітекрути показано на Рисунок 3.1.

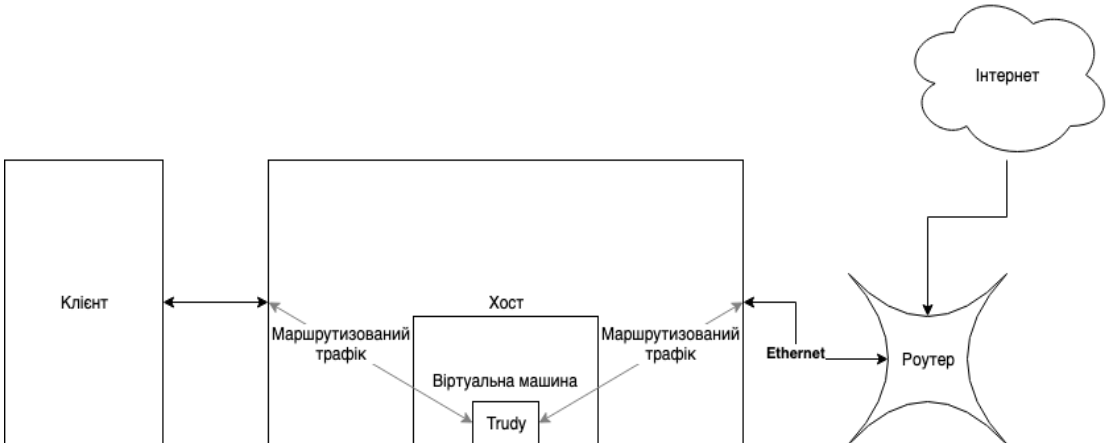

Рисунок 3.1 - Архітектура взаємодії з Tor

Послідовність пакетів по архітектурі відбувається наступним чином:

1. Формування пакетів на стороні клієнта за допомогою Tor.
2. Відправка пакетів клієнтом.
3. Перехват пакетів віртуальної машиною.
4. Проходження пакетів через програмне рішення.
5. Затримка пакетів.
6. Проходження пакетів через роутер.
7. Отримання пакетів сервером.
8. Формування пакетів на стороні сервера.
9. Відправка пакетів сервера.
10. Проходження пакетів через роутер.
11. Перехват пакетів віртуальної машиною.
12. Проходження пакетів через програмне рішення.
13. Затримка пакетів.
14. Отримання пакетів клієнтом.

3.1.3 Програмне рішення

Бібліотека Trudy - це програмне рішення, який виконує роль проксі-сервера, який може змінювати і скидувати трафік для довільних TCP-з'єднань. Trudy може використовуватися для програмної зміни трафіку TCP для клієнтів, які не знають проксі-сервера. Trudy створює 2-смуговий "міст" для кожного з'єднання, яке є проксі-серверами. Пристрій, який ви проксімуєте («клієнт»), підключається до Trudy (але не знаючи цього), а Trudy підключається до призначеного для клієнта місця призначення («сервер»). Потім між цими містами проходить весь трафік.

Модулі Trudy являють собою попередньо запрограмований код для виконання модифікацій, бажаних користувачем. Trudy надає приклад заглушки такого модуля, і користувачам рекомендується його реалізувати. Оскільки Trudy написаний на Go, всі модулі також повинні бути написані на Go. Для маршрутизації трафіку і правильного налаштування середовища потрібна «підготовлена віртуальна машина».

Реалізація модуля для Trudy досить проста. Trudy викликає певні методи в фіксованому порядку, кожен з яких призначений для однієї конкретної дії. Бібліотека Trudy пропонує нам декілька методів для роботи з пакетами:

- Deserialize() – перетворює послідовність бітів в потрібну структуру даних.
- Drop () - якщо повернуто значення true, весь пакет відкидається.
- DoMangle () - якщо повертається true, викликається Mangle ().
- Mangle () - змінює завантаженні дані.
- DoIntercept () - якщо true повертається, дані відправляються перехоплювачі Trudy.
- DoPrint () - якщо true повертається, викликається PrettyPrint ().
- PrettyPrint () - друкує дані в зручному для людини форматі.
- Serialize () – перетворює структура даних у послідовність бітів, якщо вони були раніше десеріалізовані.
- BeforeWriteTo () - дії, вжиті перед записом даних в одну або іншу сторону.
- AfterWriteTo () - дії, вжиті після запису даних в ту чи іншу сторону.

3.1.4 Віртуальна машина

Віртуальна машина використовується для розміщення і запуску програмного коду та одночасно виконує роль моста через який і проходить весь трафік. В даному випадку використовується віртуальна машина VirtualBox та оболонка Vagrant.

VirtualBox - це потужний продукт для віртуалізації x86 і AMD64/Intel64 як для підприємств, так і для домашнього використання. VirtualBox є не тільки надзвичайно багатофункціональним і високопродуктивним продуктом для корпоративних клієнтів, але і єдиним професійним рішенням, яке вільно доступно у вигляді програмного забезпечення з відкритим вихідним кодом на умовах GNU General Public License (GPL) версії 2.

Vagrant є інструментом з відкритим вихідним кодом (MIT) для створення та управління середовищами для віртуалізованого розробки, розробленими Мітчеллом Хашімото і Джоном Бендером. Vagrant керує віртуальними машинами, розміщеними в Oracle VirtualBox - повноцінному віртуалізаторі x86, який також є відкритим на умовах GNU General Public License (GPL) версії 2.

Для того, щоб Vagrant працював у режимі мосту або «людина-посередник» потрібно встановити попередньо налаштований проект під назвою MITV-VM. MITM-VM - це проста в розгортанні віртуальна машина, яка може надати гнучкі можливості «людина-посередник». Цей проект потребує невеликого налаштування, вимагає невеликих додаткових апаратних засобів, а також надає безліч утиліт і інструментів для виконання загальних (і не дуже поширених) сценаріїв «людина-посередник».

3.2 Практичні основи

Початковим етапом реалізації програмного рішення додавання випадкових затримок в пакети є вибір генератора псевдовипадкових чисел для генерування випадкового числа з унемножуванням здійснення обчислення наступних чисел зловмисником при можливому захопленні хост-машини. Наступним етапом є вибір

оптимального діапазона границь затримки експериментальним способом. Останнім етапом є сама реалізація програмного рішення з використанням вибраного генератора псевдовипадкових чисел й діапазон границь затримки.

3.2.1 Генератора псевдовипадкових чисел

При виборі генератора псевдовипадкових чисел вибір впав на один з генераторів під назвою Fortuna.

Fortuna — це сімейство криптографічно стійких генераторів псевдовипадкових чисел. Система Fortuna складається з трьох частин:

- Власне генератор, який ініціалізується початковим числом (англ. seed) фіксованої довжини і видає довільну кількість псевдовипадкових бітів.
- Акумулятор ентропії, що збирає випадкові дані з різних джерел і змінює початкове число генератора кожного разу, коли накопичено достатню кількість ентропії.
- Система управління файлом початкового числа, що забезпечує можливість генерації псевдовипадкових чисел безпосередньо після перезавантаження комп'ютера.

Генератор бере випадкове початкове число фіксованого розміру з накопичувача ентропії і генерує довільно довгі послідовності псевдовипадкових даних. Генератор складається з блочного шифру в режимі зустрічного шифрування, як показано на Рисунок 3.2. У цій реалізації AES 256 був обраний в якості блочного шифру. Введення відкритого тексту в блоковий шифр - це просто лічильник, а 256-бітові ключі надходять від акумулятора. Оскільки AES є 128-бітовим блоковим шифром, він генерує 16 байтів даних одночасно. Як тільки він згенерує ці дані, він генерує два додаткових блока, які будуть використовуватися в якості нового ключа в наступний раз, коли він буде потрібний для генерації даних. Це задовольняє вимогу, щоб знання поточного стану генератора не розкривавав минуло згенеровані дані.

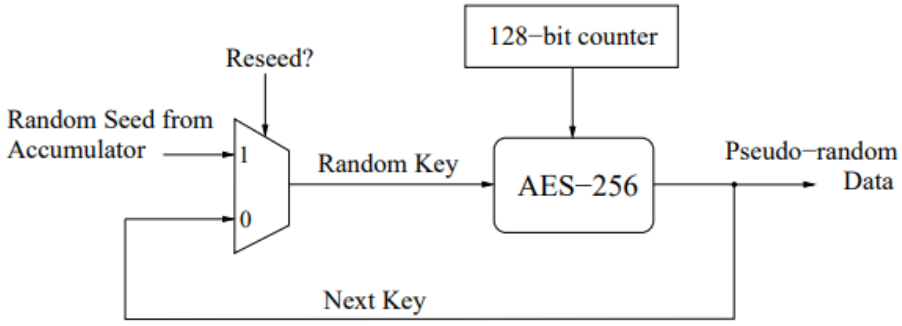

Рисунок 3.2 – Ядро генератора Fortuna[37]

Використовуючи криптографічно стійкий генератор псевдовипадкових чисел Fortuna без використання файла з послідовностями, початкове число ґрунтується тільки на поточному часу доби, поточному імені користувача, списку встановлених на даний момент мережевих інтерфейсів і вихідних даних генератора випадкових чисел системи. Апаратний стан машини відображено на Рисунок 3.3.

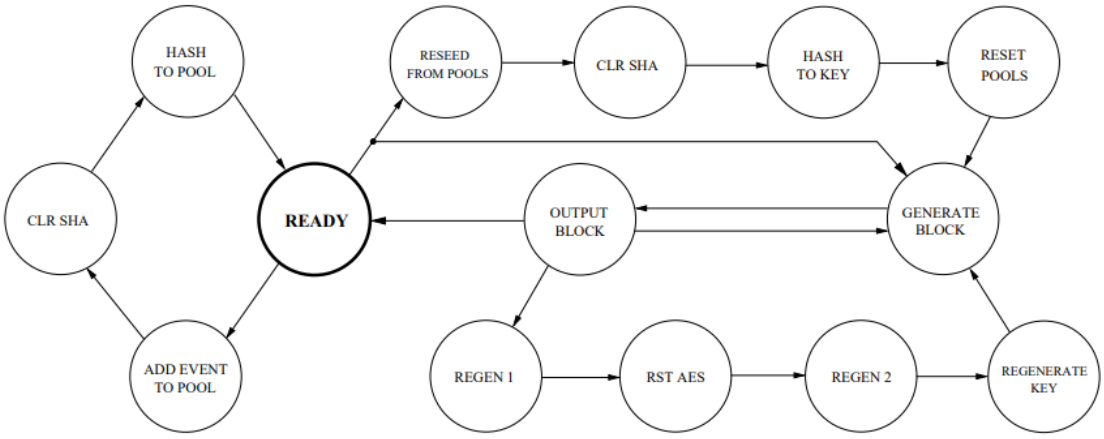

Рисунок 3.3 – Апаратний стан машини для Fortuna[37]

Акумулятор використовує 32 ентропійних пулу для збору випадковості з навколишнього середовища. Використання зовнішньої ентропії допомагає відновлюватися з ситуацій, коли зловмисник отримав (часткове) знання про стан генератора. Схема пулов ентропії відображена на Рисунок 3.4.

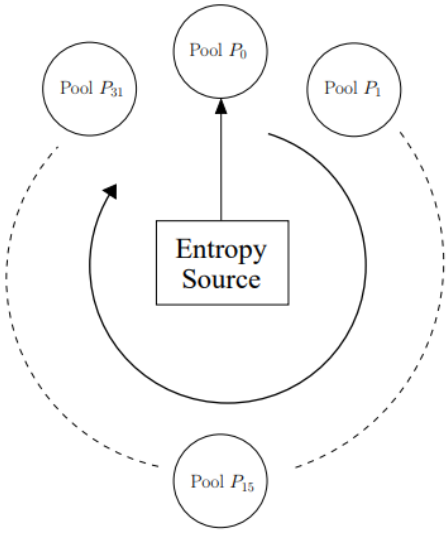

Рисунок 3.4 – 32 пул ентропії Fortuna[37]

3.2.1.1 Результати перевірки генератора

При проведенні тестів на критерії для перевірки якості випадкових двійкових послідовностей генерувалися послідовності довжиною в 100000 байтів. Результом проведення тестів на критерії для перевірки якості випадкових двійкових послідовностей наведені в Таблиця 3.1.

Таблиця 3.1 – Результати проведення тестів на критерії

Значення				
Критерій	**Теоретичні**	**Практичні**		
Рівномірність знаків	258.96	248.46	256.00	241.10
Незалежність знаків	65088.15	64589.67	64340.24	65371.25
Однорідність двійкової послідовності	2306.87	2287.37	2217.44	2107.25

Виходячи з результатів генератор Fortuna проходить кожний тест на критерії, але у різних послідовностях. Нерівномірність згенерованих послідовностей зумовлено нерівномірним часом проведення перевірки й не використання файла з послідовностями.

3.2.2 Діапазон границь затримки

Діапазон границь затримки використовується для обмеження генерування випадкового числа для затримки пакетів. Вибір діапазон границь затримки проводився експериментальним шляхом. Експеримент проводився використовуючи клієнт Tor, під час запуску завантажував 50 сторінок www.google.com. Сенс експеримента стоїть в тому, щоб визначити кількість сторінок, які не завантажаться в різних діапазонах границь затримки. Теоретично відомо, що є ймовірність відмови спілкування між клієнтом і сервером у випадку великої затримки, так як сторона сервера буде вважати, що на стороні клієнта виникли неполадки з мережею, якщо пакети проходять в обидві сторони доволі довго.

Встановлено мінімальною границею затримки у вигляді 0 секунд. Зумовлено це найменшим невід'ємним числом. Максимальна границя затримки встановлена у вигляді 5 секунд з подальшим зменшенням її по 0.5 та 1 секунд до мінімальної границі затримки. Результати експерименти наведені в Таблиця 3.2.

Таблиця 3.2 – Результати проведення експеримента вибір діапазон границь затримки

Діапазон границь затримки, сек.	**Кількість завантажуваних сторінок, шт.**	**Кількість не завантажуваних сторінок, шт.**
0 – 5	0	50
0 – 4	0	50
0 – 3	0	50
0 – 2.5	0	50
0 – 2	50	0
0 – 1.5	50	0
0 – 1	50	0
0 – 0.5	50	0

Експериментальним шляхом було виявлено оптимальний діапазон границь генерування випадкового числа від 0 до 1 секунди.

Максимальне значення мінімальної границі встановлен на 0 секунді. Зумовлено це тим, що використання більшого значення мінімальної границі

призводить до того, що зменшується діапазон границь затримки, якщо не змінювати значення максимальної границі, через що зменшується загальний сенс додавання затримок, або при зміщенні мінімальної та максимальної границі є ризик отримати непрацездатний діапазон границь затримки.

Мінімальне значення максимальної границі встановлено на 1 секунді. Зумовлено це тим, що використання більшого значення максимальної границі призводило до того, що час від часу сервер переставав відповідати на запити, вважаючи, що в клієнта виникли неполадки з мережою, в кінцевому підсумку якого веб-сторінка не завантажувалася.

3.2.3 Програмна реалізація

Реалізація методів від самого початку є пустою, так що нам потрібно реалізувати один з методів під назвою Mangle. Використовуючи цей метод буде реалізована затримка пакетів на випадково згенерована число в певних діапазонах границі. Реалізації даного метода виглядає наступним чином:

```
// Змінна для зберігання згенерованого числа
timeout := int64(0)
// Мінімальне значення границі генерування числа
min := int64(500000000)
// Максимальне значення границі генерування числа
max := int64(1500000000)

// Створення rand реалізації Fortune
rng, err := fortuna.NewRNG("") // fortune.NewRNG(seedFileName)
// Перевірка на помилку
if err != nil {
        // Виведення помилки
        panic("cannot initialise the RNG: " + err.Error())
}
```

```
// Закриття блоку оброблення помилки
defer rng.Close()

for { // Запуск цикла для генерування числа в заданих границях
        // Змінна зберігання згенеруваних випадкових байтів довжиною 8
        data := rng.RandomData(8)
        // Змінна зберігання  отримання випадкового числа з змінни data
        binaryData := binary.BigEndian.Uint32(data)

        // Перетворення формат даних з uint32 в int64
        timeout = int64(binaryData)

        // Перевірка чи входить значення в границі
        if timeout > min && timeout < max {
                break
        }
}
// Змінна зберігання згенерованого випадкового числа в наносекунда
var pause = time.Duration(timeout) * time.Nanosecond

// Зупинка пакету на випадково згенероване число в наносекундах
time.Sleep(pause)
```

Завдяки вище наведеному програмному коду здійснюється генерація випадкового числа за допомогою Fortune в діапазоні границь 0 – 1 секунда і затримання пакета на випадково згенероване число в заданому діапазоні границь.

3.2.4 Результати

На Рисунок 3.2 відображено відкриття й закриття TCP з’єднання, який

відбувається в асинхронному режимі. Тобто, програмне рішення вміє працювати паралельно з декількаами потоками даних, які проходять через віртуальну машину, завдяки чому не здійснюється затримка всіх пакетів при відвідуванні, наприклад, відразу декілька сторінок.

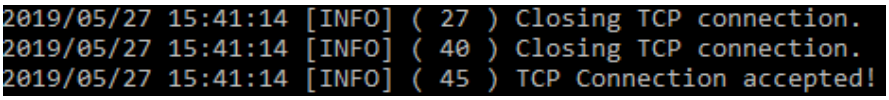

Рисунок 3.2 – Інформація про відкриття й закриття TCP з’єднання

На Рисунок 3.3 результати роботи програмного рішення, а саме, відображено інформацію про пакет: дату та час, вихідний і вхідний IP-адреса та порт, час затримки пакету й дані пакета в структурованому вигляді.

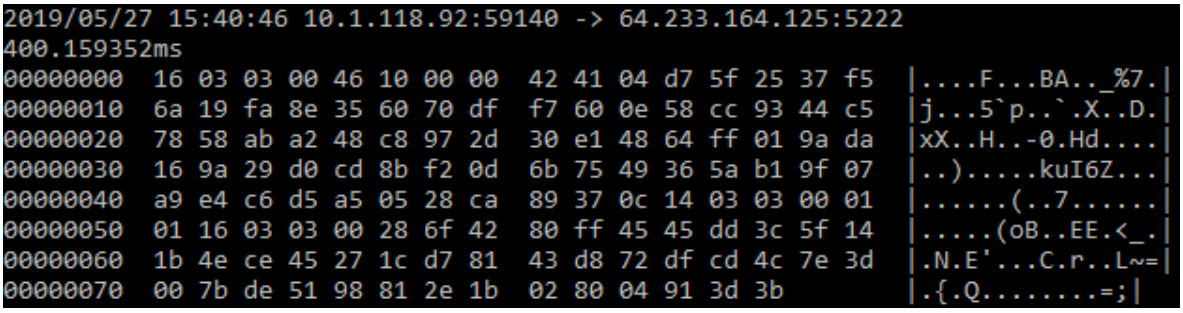

Рисунок 3.3 – Інформація про пакет

Висновки до розділу 3

У цьому розділі було описано теоретичні основи поняття вибору генератора псевдовипадкових чисел, використання бібліотеки Trudy та проекта MITM-VM для віртуальної машини Vagrant. Завдяки комбінації даних компонентів була реалізована система додавання випадкових затримок при проходженні пакетів через віртуальну машину для системи Tor.

Окремо було розброблено програмне рішення, яке було реалізовано на мові програмування Go, що надає гарантію швидкодії реалізованого програмного коду й паралельного виконання обчислювальних робіт.

Результатами даного програмного рішення є додавання випадкових затримок при проходженні пакетів через віртуально машину використовуючи систему Tor.

ВИСНОВКИ

Дана робота присвячена створенню програмного рішення для додавання випадкових затримок у пакетів даних. Було отримано ряд результатів:

- Проаналізовано роботу цибулевої маршрутизація та її розвиток поколіннями.
- Проаналізована практична реалізації цибулевої маршрутизації Tor та її технологічні особливості. В результаті аналізу встановлено, що собою представляє собою Tor.
- Описано атаки на систему Tor та способи протидії ним.
- Описано модель загроз системи Tor.
- Проаналізовано компоненти для реалізації додавання випадкових затримок. В результаті, обрано програмне рішення у вигляді бібліотеки Trudy реалізованої на мові Golang, віртуальну машину VirtualBox з оболонкою Vagrant й встановлений на неї проект MITM-VM реалізації «людина-поседерник».
- Розроблено архітектуру взаємодії компонентів. Експериментально вибрано генератор псевдовипадкових чисел, а саме Fortune.
- Експериментально вибрано працездатний діапазон границь генерування затримки.
- Реалізовано програмне рішення додавання випадкових затримок на мові Golang, для платформ UNIX-подібних та Windows операційних систем, працездатність якого протестована на операційній системі Windows 10.

Оскільки програмна реалізація додавання випадкових затримок реалізована поза Tor, то можна сподіватися, що зловмиснику буде важче аналізувати трафік. Програмна реалізація може бути доповнена механізмами перехвата та модифікації пакетів в залежності від вимог користувача або організації, що використовує дане рішення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Tor: The Second-Generation Onion Router [Електронний ресурс] / R. Dingledine, N. Mathewson, P. Syverson. – 2004. – Режим доступу до ресурсу: https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf.
2. A Peel of Onion [Електронний ресурс] / Paul Syverson. – 2011. – Режим доступу до ресурсу: https://www.acsac.org/2011/program/keynotes/syverson.pdf.
3. Onion routing [Електронний ресурс] – Режим доступу до ресурсу: http://www.onion-router.net/.
4. Tor [Електронний ресурс] – Режим доступу до ресурсу: http://www.torproject.org/.
5. Tor Metrics [Електронний ресурс] – Режим доступу до ресурсу: https://metrics.torproject.org/.
6. Low-Resource Routing Attacks Against Tor [Електронний ресурс] / [K. Bauer, D. McCoy, D. Grunwald та ін.]. – 2007. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/bauer:wpes2007.pdf.
7. A Stealthy Attack Against Tor Guard Selection [Електронний ресурс] / Q. Li, P. Liu, Z. Qin. – 2015. – Режим доступу до ресурсу: https://pdfs.semanticscholar.org/973a/3ad736c743cb50eacccbfb03e275f8ed2e10.pdf.
8. Recent Attacks On Tor [Електронний ресурс] / Juha Salo. – 2010. – Режим доступу до ресурсу: http://www.cse.hut.fi/en/publications/B/11/papers/salo.pdf.
9. Tor Network Status: TorStatus [Електронний ресурс] – Режим доступу до ресурсу: https://torstatus.blutmagie.de/.
10. Analyzing the Effectiveness of Passive Correlation Attacks on the Tor Anonymity Network [Електронний ресурс] / Sam DeFabbia-Kane. – 2011. – Режим доступу до ресурсу:

https://pdfs.semanticscholar.org/17a6/736b5b8d9a2e516adbebb338e3265681b1c9.pdf.

11. Probabilistic Analysis of Onion Routing in a Black-box Model [Електронний ресурс] / J. Feigenbaum, A. Johnson, P. Syverson. – 2012. – Режим доступу до ресурсу: https://www.ohmygodel.com/publications/wpes08-feigenbaum.pdf.
12. Compromising Anonymity Using Packet Spinning [Електронний ресурс] / [V. Pappas, E. Athanasopoulos, S. Ioannidis та ін.]. – 2008. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/torspinISC08.pdf.
13. On the Effectiveness of Traffic Analysis Against Anonymity Networks Using Flow Records [Електронний ресурс] / [S. Chakravarty, M. V. Barbera, G. Portokalidis та ін.]. – 2014. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/nfattackpam14.pdf.
14. Low-Cost Traffic Analysis of Tor [Електронний ресурс] / S. J. Murdoch, G. Danezis. – 2005. – Режим доступу до ресурсу: https://www.cs.ucy.ac.cy/courses/EPL682/papers/anon-2.pdf.
15. A New Cell Counter Based Attack Against Tor [Електронний ресурс] / [Z. Ling, J. Luo, W. Yu та ін.]. – 2009. – Режим доступу до ресурсу: http://web.cse.ohio-state.edu/~xuan.3/papers/09_ccs_llyfxj.pdf.
16. Browser-Based Attacks on Tor [Електронний ресурс] / T.Abbott, K. Lai, M. Lieberman, E. Price. – 2007. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/abbott-pet2007.pdf.
17. A Practical Con-gestion Attack on Tor Using Long Paths [Електронний ресурс] / N. S. Evans, R. Dingledine, C. Grothoff. – 2009. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/congestion-longpaths.pdf.
18. How Much Anonymity does Network Latency Leak? [Електронний ресурс] / N. Hopper, E. Y. Vasserman, E. Chan-Tin. – 2010. – Режим доступу до ресурсу: https://www-users.cs.umn.edu/~hoppernj/ccs-latency-leak.pdf.

19. Passive-Logging Attacks Against Anonymous Communications Systems [Електронний ресурс] / M.Wright, M. Adler, B. N. Levine, C. Shields. – 2008. – Режим доступу до ресурсу: http://people.cs.georgetown.edu/~clay/research/pubs/wright-tissec-2008.pdf.
20. AS-awareness in Tor Path Selection [Електронний ресурс] / M. Edman, P. Syverson. – 2009. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/DBLP:conf/ccs/EdmanS09.pdf.
21. Large Scale Simulation of Tor: Modelling a Global Passive Adversary [Електронний ресурс] / G. Gorman, S. Blott. – 2007. – Режим доступу до ресурсу: https://pdfs.semanticscholar.org/b3e8/7f8d290ac9f38e9321fac7e94c1e74b62e6a.pdf.
22. On the risks of serving whenever you surf: Vulnerabilities in Tor's blocking resistance design [Електронний ресурс] / J. McLachlan, N. Hopper. – 2009. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/wpes09-bridge-attack.pdf.
23. One Bad Apple Spoils the Bunch: Exploiting P2P Applications to Trace and Profile Tor Users [Електронний ресурс] / [S. Le Blond, P. Manils, A. Chaabane та ін.]. – 2011. – Режим доступу до ресурсу: https://arxiv.org/pdf/1103.1518.pdf.
24. A potential HTTP-based application-level attack against Tor [Електронний ресурс] / X.Wang, J. Luo, M. Yang, Z. Ling. – 2011. – Режим доступу до ресурсу: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.710.6952&rep=rep1&type=pdf.
25. CellFlood: Attacking Tor Onion Routers on the Cheap [Електронний ресурс] / [M. Barbera, V. Kemerlis, V. Pappas та ін.]. – 2013. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/esorics13-cellflood.pdf.

26. Peeling back the layers of Tor with EgotisticalGiraffe [Електронний ресурс]. – 2013. – Режим доступу до ресурсу: http://media.encrypted.cc/files/nsa/egotisticalgiraffe-guardian.pdf.
27. Attacking Tor: how the NSA targets users' online anonymity [Електронний ресурс] / Paul Schneier. – 2013. – Режим доступу до ресурсу: https://www.theguardian.com/world/2013/oct/04/tor-attacks-nsa-users-online-anonymity.
28. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network [Електронний ресурс] / R.Jansen, F. Tschorsch, A. Johnson, B. Scheuermann. – 2014. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/sniper14.pdf.
29. New Tor Denial of Service Attacks and Defenses [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://blog.torproject.org/new-tor-denial-service-attacks-and-defenses.
30. On the Effectiveness of Traffic Analysis Against Anonymity Networks Using Flow Records [Електронний ресурс] / [S. Chakravarty, M. Barbera, G. Portokalidis та ін.]. – 2014. – Режим доступу до ресурсу: https://www.freehaven.net/anonbib/cache/nfattackpam14.pdf.
31. Tor security advisory: "relay early" traffic confirmation attack [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://blog.torproject.org/tor-security-advisory-relay-early-traffic-confirmation-attack.
32. Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services [Електронний ресурс] / [A. Kwon, M. AlSabah, D. Lazar та ін.]. – 2015. – Режим доступу до ресурсу: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-kwon.pdf.
33. Traffic correlation using netflows [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://blog.torproject.org/traffic-correlation-using-netflows.

34. Tor suffers traffic confirmation attacks. Say goodbye to anonymity on the Web [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://www.techtimes.com/articles/11711/20140802/tor-suffers-traffic-confirmation-attacks-say-goodbye-to-anonymity-on-the-web.htm.
35. Oracle VM VirtualBox Overview [Електронний ресурс] – Режим доступу до ресурсу: http://www.oracle.com/us/technologies/virtualization/oracle-vm-virtualbox-overview-2981353.pdf.
36. Ferguson N. Cryptography Engineering [Електронний ресурс] / N. Ferguson, B. Schneier, T. Kohno. – 2010. – Режим доступу до ресурсу: http://the-eye.eu/public/Books/HumbleBundle/cryptography_engineering_design_principles_and_practical_applications.pdf.
37. Fortuna: Cryptographically Secure Pseudo-Random Number Generation In Software And Hardware [Електронний ресурс] / R.McEvoy, J. Curran, P. Cotter, C. Murphy. – 2006. – Режим доступу до ресурсу: https://www.researchgate.net/publication/215858122_Fortuna_Cryptographically_Secure_Pseudo-Random_Number_Generation_In_Software_And_Hardware
38. Testing Random Number Generators [Електронний ресурс] / Dan Biebighauser. – 2000. – Режим доступу до ресурсу: http://www-users.math.umn.edu/~garrett/students/reu/pRNGs.pdf.
39. STATISTICAL PROPERTIES OF PSEUDORANDOM SEQUENCES [Електронний ресурс] / Ting Gu. – 2016. – Режим доступу до ресурсу: https://uknowledge.uky.edu/cs_etds/44/.
40. Statistical dependence: Beyond Pearson's ρ [Електронний ресурс] / D. Tjøstheim, H. Otneim, B. Støve. – 2018. – Режим доступу до ресурсу: https://arxiv.org/pdf/1809.10455.pdf.
41. How to modify general TCP/IP traffic on the fly with Trudy [Електронний ресурс] / Tomas Susanka. – 2017. – Режим доступу до ресурсу: https://blog.susanka.eu/how-to-modify-general-tcp-ip-traffic-on-the-fly-with-trudy/.

42. Trudy [Електронний ресурс] / Kelby Ludwig. – 2017. – Режим доступу до ресурсу: https://github.com/praetorian-code/trudy.
43. Palat J. Introducing Vagrant [Електронний ресурс] / Jay Palat. – 2012. – Режим доступу до ресурсу: https://www.linuxjournal.com/content/introducing-vagrant.
44. MITM-VM [Електронний ресурс] / Kelby Ludwig. – 2016. – Режим доступу до ресурсу: https://github.com/praetorian-code/mitm-vm.
45. Fortuna [Електронний ресурс] / Jochen Voss. – 2013. – Режим доступу до ресурсу: https://github.com/seehuhn/fortuna.

ДОДАТКИ

ДОДАТОК А

Конфігураційні файли віртуальної машини

```
Vagrant.configure("2") do |config|
  config.vm.box = "debian/jessie64"
  config.vm.provider "virtualbox" do |v|
    v.memory = 2048
    v.cpus = 2
  end

  config.ssh.shell = "bash -c 'BASH_ENV=/etc/profile exec bash'"
  config.vm.network "public_network", ip: "10.1.118.200"

  config.vm.provision :shell, path: "bootstrap.sh"
  config.vm.provision :shell, path: "route.sh", run: "always"

  config.vm.provider :virtualbox do |vb|
    vb.customize ["modifyvm", :id, "--usb", "on", "--usbehci", "on"]
  end
end
```

```
#!/usr/bin/env bash

export INTERNET_ROUTER_IP="10.1.118.1"

echo "nameserver $INTERNET_ROUTER_IP" > /etc/resolv.conf

export DEBIAN_FRONTEND=noninteractive

/sbin/iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 8888 -m tcp -j REDIRECT --to-ports 8080

/sbin/iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 443 -m tcp -j REDIRECT --to-ports 6443

/sbin/iptables -t nat -A PREROUTING -i eth1 -p tcp -m tcp -j REDIRECT --to-ports 6666

/bin/ip route del 0/0
/sbin/route add default gw $INTERNET_ROUTER_IP dev eth1
sysctl -w net.ipv4.ip_forward=1
```

```
#!/usr/bin/env bash

export DEBIAN_FRONTEND=noninteractive

echo "nameserver 8.8.8.8" > /etc/resolv.conf

apt-get update

apt-get install -y curl git netsed nmap build-essential make build-essential libssl-dev zlib1g-dev libbz2-dev \
   libreadline-dev libsqlite3-dev wget curl llvm libncurses5-dev python-pip python python-dev python-setuptools tcpdump iptables iptables-dev vim

apt-get install -y libffi-dev libssl-dev libxml2-dev libxslt1-dev  zlib1g-dev \
       libfreetype6-dev liblcms2-dev libwebp-dev tcl8.5-dev tk8.5-dev python-tk
pip install --upgrade cffi
pip install --upgrade pyasn1
pip install mitmproxy

apt-get install -y sslstrip

apt-get install -y sslsniff

apt-get install -y socat

apt-get install -y bluez bluez-cups bluez-dbg bluez-hcidump bluez-tools python-bluez libbluetooth-dev libbluetooth3 python-gobject python-dbus
git clone https://github.com/conorpp/btproxy.git
cd btproxy
python setup.py install
```

```
pip uninstall pyyaml
apt-get install -y python-gtk2 python-cairo python-usb python-crypto python-serial python-dev libgcrypt-dev mercurial libyaml-dev libgcrypt11-dev libpython2.7-dev usbutils
pip install pyyaml
hg clone https://bitbucket.org/secdev/scapy-com
cd scapy-com
python setup.py install
cd ..
git clone https://github.com/riverloopsec/killerbee.git
cd killerbee
python setup.py install
chmod +x ./tools/*
echo 'export="$PATH:$HOME/killerbee/tools"' >> ~/.bashrc

echo 'export GOPATH="/root/go"' >> /root/.bashrc
echo 'export PATH=$PATH:/usr/local/go/bin' >> /root/.bashrc
mkdir -p /root/go
mkdir -p /root/go/src
mkdir -p /root/go/pkg
mkdir -p /root/go/bin
mkdir -p /root/go-src
wget -q -O /root/go-src/go.tar.gz https://storage.googleapis.com/golang/go1.6.2.linux-amd64.tar.gz
tar -C /usr/local -xzf /root/go-src/go.tar.gz
export GOPATH="/root/go"
/usr/local/go/bin/go get "github.com/praetorian-inc/trudy"
```

```
#!/bin/bash
sudo launchctl unload /System/Library/LaunchDaemons/com.apple.blued.plist
while read -r line ; do
    echo $line >> disabled_mods
    sudo kextunload -b $line
done < <(kextstat -l | grep -i bluet | tr -s ' ' | cut -d' ' -f7)
```

```
#!/bin/bash
sudo launchctl load /System/Library/LaunchDaemons/com.apple.blued.plist
while read -r line ; do
    sudo kextload -b $line
 done < <(cat disabled_mods)
```

ДОДАТОК Б

Програмний код

```
package main

import (
	"crypto/tls"
	"encoding/hex"
	"flag"
	"fmt"
	"github.com/gorilla/websocket"
	"github.com/praetorian-inc/trudy/listener"
	"github.com/praetorian-inc/trudy/module"
	"github.com/praetorian-inc/trudy/pipe"
	"io"
	"log"
	"net"
	"net/http"
	"strings"
	"sync"
	"time"
)

var connectionCount uint
var websocketConn *websocket.Conn
var websocketMutex *sync.Mutex
var tlsConfig *tls.Config

func main() {
	var tcpport string
```

```
	var tlsport string

	var x509 string
	var key string

	var showConnectionAttempts bool

	flag.StringVar(&tcpport, "tcp", "6666", "Listening port for non-TLS connections.")
	flag.StringVar(&tlsport, "tls", "6443", "Listening port for TLS connections.")
	flag.StringVar(&x509, "x509", "./certificate/trudy.cer", "Path to x509 certificate that will be presented for TLS connection.")
	flag.StringVar(&key, "key", "./certificate/trudy.key", "Path to the corresponding private key for the specified x509 certificate")
	flag.BoolVar(&showConnectionAttempts, "show", true, "Show connection open and close messages")

	flag.Parse()

	tcpport = ":" + tcpport
	tlsport = ":" + tlsport
	setup(tcpport, tlsport, x509, key, showConnectionAttempts)
}

func setup(tcpport, tlsport, x509, key string, show bool) {

	//Setup non-TLS TCP listener!
	tcpAddr, err := net.ResolveTCPAddr("tcp", tcpport)
	if err != nil {
```

```
		log.Printf("There appears to be an error with the TCP port you specified. See error below.\n%v\n", err.Error())
		return
	}
	tcpListener := new(listener.TCPListener)

	//Setup TLS listener!
	trdy, err := tls.LoadX509KeyPair(x509, key)
	if err != nil {
		log.Printf("There appears to be an error with the x509 or key values specified. See error below.\n%v\n", err.Error())
		return
	}
	tlsConfig = &tls.Config{
		Certificates:       []tls.Certificate{trdy},
		InsecureSkipVerify: true,
	}
	tlsAddr, err := net.ResolveTCPAddr("tcp", tlsport)
	if err != nil {
		log.Printf("There appears to be an error with the TLS port specified. See error below.\n%v\n", err.Error())
		return
	}
	tlsListener := new(listener.TLSListener)

	//All good. Start listening.
	tcpListener.Listen("tcp", tcpAddr, &tls.Config{})
	tlsListener.Listen("tcp", tlsAddr, tlsConfig)

	log.Println("[INFO] Trudy lives!")
```

```
	log.Printf("[INFO] Listening for TLS connections on port %s\n", tlsport)
	log.Printf("[INFO] Listening for all other TCP connections on port %s\n", tcpport)

	go websocketHandler()
	go connectionDispatcher(tlsListener, "TLS", show)
	connectionDispatcher(tcpListener, "TCP", show)

}

func connectionDispatcher(listener listener.TrudyListener, name string, show bool) {
	defer listener.Close()
	for {
		fd, conn, err := listener.Accept()
		if err != nil {
			continue
		}

		p := new(pipe.TrudyPipe)
		if name == "TLS" {
			err = p.New(connectionCount, fd, conn, true)
		} else {
			err = p.New(connectionCount, fd, conn, false)
		}

		if err != nil {
			log.Println("[ERR] Error creating new pipe.")
			continue
		}
		if show {
```

```
                log.Printf("[INFO] ( %v ) %v Connection accepted!\n",
connectionCount, name)
            }
            go clientHandler(p, show)
            go serverHandler(p)
            connectionCount++
        }
}

func errHandler(err error) {
        if err != nil {
            panic(err)
        }
}

//clientHandler manages data that is sent from the client to the server.
func clientHandler(pipe pipe.Pipe, show bool) {
        if show {
            defer log.Printf("[INFO] ( %v ) Closing TCP connection.\n", pipe.Id())
        }
        defer pipe.Close()

        buffer := make([]byte, 65535)

        for {
            bytesRead, clientReadErr := pipe.ReadFromClient(buffer)

            if clientReadErr != io.EOF && clientReadErr != nil {
                break
            }
```

```
if clientReadErr != io.EOF && bytesRead == 0 {
        continue
}

data := module.Data {
   Timeout: time.Duration(0) * time.Second,
   FromClient: true,
   Bytes:      buffer[:bytesRead],
   TLSConfig:  tlsConfig,
   ServerAddr: pipe.ServerInfo(),
   ClientAddr: pipe.ClientInfo()}

data.Deserialize()

if data.Drop() {
        continue
}

if data.DoMangle() {
        data.Mangle()
        bytesRead = len(data.Bytes)
}

if data.DoIntercept() {
        if websocketConn == nil {
                log.Printf("[ERR] Websocket Connection has not been setup yet! Cannot intercept.")
                continue
        }
```

```
                websocketMutex.Lock()
                bs := fmt.Sprintf("% x", data.Bytes)
                if err := websocketConn.WriteMessage(websocket.TextMessage,
[]byte(bs)); err != nil {
                        log.Printf("[ERR] Failed to write to websocket: %v\n", err)
                        websocketMutex.Unlock()
                        continue
                }
                _, moddedBytes, err := websocketConn.ReadMessage()
                websocketMutex.Unlock()
                if err != nil {
                        log.Printf("[ERR] Failed to read from websocket: %v\n", err)
                        continue
                }
                str := string(moddedBytes)
                str = strings.Replace(str, " ", "", -1)
                moddedBytes, err = hex.DecodeString(str)
                if err != nil {
                        log.Printf("[ERR] Failed to decode hexedited data.")
                        continue
                }
                data.Bytes = moddedBytes
                bytesRead = len(moddedBytes)
        }

        if data.DoPrint() {
                        log.Printf("%v -> %v\n%v\n%v\n", data.ClientAddr.String(),
data.ServerAddr.String(), data.Timeout.String(), data.PrettyPrint())
        }
```

```
                data.Serialize()

                data.BeforeWriteToServer(pipe)
                bytesRead = len(data.Bytes)

                _, serverWriteErr := pipe.WriteToServer(data.Bytes[:bytesRead])
                if serverWriteErr != nil || clientReadErr == io.EOF {
                        break
                }

                data.AfterWriteToServer(pipe)
        }
}

//serverHandler manages data that is sent from the server to the client.
func serverHandler(pipe pipe.Pipe) {
        buffer := make([]byte, 65535)

        defer pipe.Close()

        for {
                bytesRead, serverReadErr := pipe.ReadFromServer(buffer)

                if serverReadErr != io.EOF && serverReadErr != nil {
                        break
                }

                if serverReadErr != io.EOF && bytesRead == 0 {
                        continue
                }
```

```
data := module.Data {
   Timeout: time.Duration(0) * time.Second,
   FromClient: true,
   Bytes:      buffer[:bytesRead],
   TLSConfig:  tlsConfig,
   ServerAddr: pipe.ServerInfo(),
   ClientAddr: pipe.ClientInfo()}

data.Deserialize()

if data.Drop() {
        continue
}

if data.DoMangle() {
        data.Mangle()
        bytesRead = len(data.Bytes)
}

if data.DoIntercept() {
        if websocketConn == nil {
                log.Printf("[ERR] Websocket Connection has not been setup yet! Cannot intercept.")
                continue
        }
        websocketMutex.Lock()
        bs := fmt.Sprintf("% x", data.Bytes)
        if err := websocketConn.WriteMessage(websocket.TextMessage, []byte(bs)); err != nil {
```

```
                log.Printf("[ERR] Failed to write to websocket: %v\n", err)
                websocketMutex.Unlock()
                continue
            }
            _, moddedBytes, err := websocketConn.ReadMessage()
            websocketMutex.Unlock()
            if err != nil {
                log.Printf("[ERR] Failed to read from websocket: %v\n", err)
                continue
            }
            str := string(moddedBytes)
            str = strings.Replace(str, " ", "", -1)
            moddedBytes, err = hex.DecodeString(str)
            if err != nil {
                log.Printf("[ERR] Failed to decode hexedited data.")
                continue
            }
            data.Bytes = moddedBytes
            bytesRead = len(moddedBytes)
        }

        if data.DoPrint() {
            log.Printf("%v -> %v\n%v\n%v\n", data.ClientAddr.String(),
data.ServerAddr.String(), data.Timeout.String(), data.PrettyPrint())
        }

        data.Serialize()

        data.BeforeWriteToClient(pipe)
        bytesRead = len(data.Bytes)
```

```
            _, clientWriteErr := pipe.WriteToClient(data.Bytes[:bytesRead])
            if clientWriteErr != nil || serverReadErr == io.EOF {
                break
            }

            data.AfterWriteToClient(pipe)
        }
    }

    func websocketHandler() {
        websocketMutex = &sync.Mutex{}
        upgrader := websocket.Upgrader{ReadBufferSize: 65535, WriteBufferSize:
    65535}
        http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
            io.WriteString(w, editor)
        })
        http.HandleFunc("/ws", func(w http.ResponseWriter, r *http.Request) {
            var err error
            websocketConn, err = upgrader.Upgrade(w, r, nil)
            if err != nil {
                log.Printf("[ERR] Could not upgrade websocket connection.")
                return
            }
        })
        err := http.ListenAndServe(":8080", nil)
        if err != nil {
            panic(err)
        }
    }
```

```
package module

import (
	"crypto/tls"
	"encoding/hex"
	"github.com/praetorian-inc/trudy/pipe"
	"github.com/seehuhn/fortuna"
	"encoding/binary"
	"net"
	"time"
)

//Data is a thin wrapper that provides metadata that may be useful when mangling bytes on the network.
type Data struct {
	Timeout time.Duration  //Timeout to pause send packet from/to client/server
	FromClient bool	//FromClient is true is the data sent is coming from the client (the device you are proxying)
	Bytes	[]byte	//Bytes is a byte slice that contians the TCP data
	TLSConfig  *tls.Config //TLSConfig is a TLS server config that contains Trudy's TLS server certficiate.
	ServerAddr net.Addr	//ServerAddr is net.Addr of the server
	ClientAddr net.Addr	//ClientAddr is the net.Addr of the client (the device you are proxying)
}

//DoMangle will return true if Data needs to be sent to the Mangle function.
func (input Data) DoMangle() bool {
	return true
}

//Mangle can modify/replace the Bytes values within the Data struct. This can
//be empty if no programmatic mangling needs to be done.
func (input *Data) Mangle() {
		timeout := int64(0)
	// 100000000: 0.1
	// 1000000000: 1.0
	min := int64(000000000) // Мінімальне значення границі
	max := int64(1000000000) // Максимальне значення границі

	// Створення rand реалізації Fortune
	rng, err := fortuna.NewRNG("")
	if err != nil {
		panic("cannot initialise the RNG: " + err.Error())
	}
```

```
	defer rng.Close()

	for {
		data := rng.RandomData(8) // Генерування випадкових байтів довжиною 8
		binaryData := binary.BigEndian.Uint32(data) // Отримання випадкового
числа з data

		timeout = int64(binaryData) // Перетворення формат даних з uint32 в int64

		if timeout > min && timeout < max { // Перевірка чи входить значення в
границі
					break
		}
	}
	//
	input.Timeout = time.Duration(timeout) * time.Nanosecond

	// Зупинка пакету на випадково згенероване число в наносекундах
	time.Sleep(input.Timeout)
}

//Drop will return true if the Data needs to be dropped before going through
//the pipe.
func (input Data) Drop() bool {
	return false
}

//PrettyPrint returns the string representation of the data. This string will
//be the value that is logged to the console.
func (input Data) PrettyPrint() string {
	return hex.Dump(input.Bytes)
}

//DoPrint will return true if the PrettyPrinted version of the Data struct
//needs to be logged to the console.
func (input Data) DoPrint() bool {
	return true
}

//DoIntercept returns true if data should be sent to the Trudy interceptor.
func (input Data) DoIntercept() bool {
	return false
}

//Deserialize should replace the Data struct's Bytes with a deserialized bytes.
```

```
//For example, unpacking a HTTP/2 frame would be deserialization.
func (input *Data) Deserialize() {

}

//Serialize should replace the Data struct's Bytes with the serialized form of
//the bytes. The serialized bytes will be sent over the wire.
func (input *Data) Serialize() {

}

//BeforeWriteToClient is a function that will be called before data is sent to
//a client.
func (input *Data) BeforeWriteToClient(p pipe.Pipe) {

}

//AfterWriteToClient is a function that will be called after data is sent to
//a client.
func (input *Data) AfterWriteToClient(p pipe.Pipe) {

}

//BeforeWriteToServer is a function that will be called before data is sent to
//a server.
func (input *Data) BeforeWriteToServer(p pipe.Pipe) {

}

//AfterWriteToServer is a function that will be called after data is sent to
//a server.
func (input *Data) AfterWriteToServer(p pipe.Pipe) {

}
```