Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: «Розробка програмного засобу захисту від атак користувача в системі Tor»

Виконав(ла): студент(ка) IV курсу, групи СБс 42
спеціальності 125 Кібербезпека
(шифр і назва спеціальності)

		Багрій О.Р.
	(підпис)	(прізвище та ініціали)
Керівник		Загородна Н.В
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Кареліна О.В
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Загородна Н.В
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2021

АНОТАЦІЯ

Розробка програмного засобу захисту від атак користувача в системі Tor // Кваліфікаційна робота ОР «Бакалавр» //Багрій Олександр Романович// Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБс-42 // Тернопіль, 2021 // С. 82, рис. 10 , табл. 2 , кресл. – , додат. 2 .

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

Кваліфікаційна робота присвячена дослідженню 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. 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.

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.

ANNOTATION

User security package development against cyberattacks in Tor system // Thesis of educational level "Bachelor" // Bahrii Oleksandr Romanovich // Ternopil National Technical University named after Ivan Pulyuy, Faculty of Computer Information Systems and software engineering, Department of Cybersecurity, СБс-42 group // Ternopil, 2021 // P. 82 , fig. 10, table.2 , chair. - , added. 2.

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

The qualification thesis is devoted 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. 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.

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.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

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

Проблема анонімності користувачів вирішується криптографічними мережами, які намагаються запобігти аналізу трафіку шляхом змішування повідомлень, що проходять через мережу. Ідея полягає в забезпеченні анонімності розмови через Інтернет. Одна система, яка пропонує анонімність, під назвою Tor, користується технологією, яка називається цибулева маршрутизація для шифрування повідомлень. В даний час ця технологія активно використовується для уникнення цензури в Інтернеті, сприяє свободі слова і прослуховуванню.

Тема актуальна і сьогодні, оскільки великі організації і країни вкладають великі суми грошей у розробку програмних засобів, щоб запобігати анонімності користувачів в Інтернеті.

Мета та завдання - вивчити принципи цибулевої маршрутизації, механізми мережі Tor, атаки на інкогніто користувачів в системах та способи протидії ним, а також розробити рішення, які зможуть допоможуть.

Об’єктом дослідження стала система Tor.

Предметом дослідження стали напади на систему Tor та методи їх протидії.

Методом досліду є аналізи джерел інформації, останні статті на тему дослідження, стандарту мережевих протоколів та створення досліджень із застосуванням сучасних програмних забезпечень.

Практична цінність роботи полягає у використанні програмного рішення для збільшення анонімності користувача за допомогою системи Tor та збільшення складності деанонімізації користувача.

РОЗДІЛ 1 ЦИБУЛЕВА МАРШРУТИЗАЦШЯ І 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

Цибулеву маршрутизацію назвали завдяки структурі даних, в межах якої інформація передається по мережі. Вона створюється з зашифрованих шарів відправником повідомлення. Всі маршрутизатори на комунікаційному шляху очищають усі рівні один за одним, під час проходження повідомлень через мережу. Вхідні повідомлення йдуть за тим самим шляхом, окрім маршрутизаторів які додають шари до цибулі, яка потім очищається приймаючою стороною.

Цибулева маршрутизація була заснована на утворенні криптографічного маршруту маршрутизаторів між двома комп'ютерами. Цей шлях називають цибулевим ланцюгом. Маршрутизаторів, які утворюють шлях, певним чином вибираються з усіх серверів, зареєстрованих у цибульній мережі, і утворюють схема так, щоб кожен вузол був з'єднаний між собою. Ініціатор шляху вирішує, які маршрутизатори і як вони будуть впровалжені в ланцюг.

Для запобіганні ситуації коли всі маршрутизатори, що мають участь у комунікаційному шляху, можуть отримувати повідомлення, які проходять через ланцюг, ініціатор повинен роздати маршрутизатору унікальне повідомлення. Причиною використання симетричних ключів є те, що шифрування використовує однаковий ключ для шифрування та дешифрування повідомлень, тому їх можна надсилати однаково в обидві сторони. Симетричне шифрування також набагато простіше в обчислювальному плані, ніж шифрування відкритим ключем.

Ключі симетричного шифрування поширюються за допомогою шифрування із відкритим ключем. Для шифрування відкритого ключа використовуються два ключі: закритий ключ та відкритий ключ. Повідомлення, які надіслані одержувачу, повинні шифруватися через відкритий ключ, який не буде використовуватися для дешифрування повідомлення. Отримувач може відкрити повідомлення своїм приватним ключем. Тобто кожен маршрутизатор повинен мати власний відкритий ключ, симетричні ключі, що використовуються для

створення носової частини, можна безпечно розділити з допомогою відкритих ключів без страху їх отримання.

Причиною використання цього шифрування для створення цибулин є те, що дане шифрування вимагає меншої потужності для обчислень, аніж шифрування з відомим ключем. У мережі є трохи повідомлень, які подорожують по мережі у всіх напрямках, і кожне повідомлення має оброблятися алгоритмом шифрування. Використовуючи симетричне шифрування замість шифрування з відкритим ключем, обчислювальне значення мережі виходить набагато им

Алгоритм формування ланцюга цибулі детально описаний у наступному розділі. Далі ми детальніше розглянемо еволюції технології маршрутизації цибулі.

1.2 Еволюція цибулевої маршрутизації

Як тільки прогнозування цибулевої маршрутизації було запущено в морську доклідницьку лабораторію Америки, в Холдшляда, і Ріда і Циверкона було тільки декілька основних припущень і принципів, що передбачають технологію, яка буде використовуватися в роботі. Лише коли було опубліковано перше покоління цибулевої маршрутизації вони отримали свою першу ілюстрацію того, наскільки функціональним буде насправді їхня робота. Функції були додані та вдосконалені в наступних поколіннях згідно з спостереженнями розробників, і відповідно до відгуків користувачів. Таким чином, вони представляють різні етапи розвитку цибулевої маршрутизації. Також вивчаючи ці покоління, можна зрозуміти, чому ця технологія реалізується саме так, як і зараз.

1.2.1 Перше покоління (1996 – 2004)

Основною ідеєю цибулевої маршрутизації є те, щоб зробити можливим створення анонімного з'єднання всередині цибулевої мережі із серверами. Було вирішено зробити можливим формування зв’язків між двома маршрутизаторами для підключеними до цибулевої мережі. В цих ситуаціях довжина ланцюга є

важливою з точки зору інформаційної безпеки.

Для того, щоб один із вузлів ланцюга порушив анонімність, йому потрібно підключити анонімні повідомлення відправника та одержувача. В основному, теоретичний зловмисник повинен отримати IP-адресу відправника та одержувача. Зазвичай це можна зробити, переглядаючи заголовки пакетів, але цибульна маршрутизація шифрує ці заголовки. Найважливіша атака на маршрутизацію цибулі - це встановити маршрутизатор як цибульний вузол і намагатися слідувати повідомленням, які проходять через цей маршрутизатор.

Якщо довжина ланцюга становить три вузли і середній вузол - це вузол зловмисника, який хоче дізнатися IP-адреси двох кінців ланцюга він одразу це дізнається. При використанні чотирьох вузлів зловмиснику необхідно порушити шифрування лише одного з вузлів, для того, щоб порушити анонімність ланцюга. Розробники вважали що це є дуже небезпечно і вирішили встановити довжина ланцюжка за замовчуванням, рівну п'яти вузлам.

Найбільшою різницею між першим поколінням і наступними є те, що у версії першого покоління використовується клієнтське програмне забезпечення та цибульневий маршрутизатор, які повністю інтегровані. Поки комп'ютер підключений до мережі для надсилання повідомлень, ви також можете буде додати як частина ланцюга цибулі. Ця функціональність суперечила одному з основних принципів цибулевої маршрутизації і була змінена після випуску наступного покоління, що дало можливість клієнту та цибулевому маршрутизатору працювати окремо.

1.2.2 Друге покоління(2004 – 2006)

Для підвищення інформаційної безпеки для цього покоління додані різні методи. Одним із методів є відправка повідомлень із випадковими даними, які називаються заповнювачами, разом із регулярним трафіком, щоб ускладнити зловмиснику аналіз трафіку. Даний спосіб полягає у обмеженні смуги пропускання постійної швидкості, що виключає ідентифікацію змін потоків руху. Вважалося, що додавання та обмеження пропускної здатності ускладнюють аналіз трафіку, але було зазначено, що ці методи є надто дорогими з точки зору обчислень порівняно з

безпекою. Наповнення та обмеження пропускної здатності були скасовані в наступному поколінні.

Певні криптомережі, які забезпечують анонімність, засновані на змішуванні компонентів, а не на непередбачуваному шляху. Ці мережі також відомі як мережі MIX [2]. Метою змішування є ускладнення з'єднання вхідних повідомлень із зашифрованими вихідними повідомленнями між собою.

Технологія MIX експериментально була включена до другого покоління цибулевої маршрутизації. Метою даного експерименту було вивчення переваг змішування для безпеки.

Змішування врешті-решт припинено у третьому поколінні з тиг причин, що і наповнення та обмеження ємності і це призвело до недостатнього підвищення інформаційної безпеки та суттєвого зрослостання вартості розрахунків.

Ще однією важливою поправкою стало створення ланцюгів довжиною більше п’яти вузлів. У цьому поколінні довжина ланцюгів змінювалася, і в одному ланцюжку могло бути десь десять вузлів. Проклавши тунелі ланцюгів або об’єднавши кілька ланцюгів, довжина шляху може стати довшою. Ця поправка була визнана непотрібною, і кількість вузлів було відновлено до п'яти в третьому поколінні.

1.2.3 Третє покоління(2006 -)

Третє покоління стає останньою версією цибулевої маршрутизації. Розвиток цього покоління все ще триває, тому в майбутньому можливості маршрутизації можуть відрізнятися від тих що представлені тут. Тому ми розглянемо особливості даного покоління таким, яким воно було опублікований у 2006 р.

Найбільшою корекцією порівняно з попередніми версіями стає протокол, який використовується для розподілу ключів шифрування вузлами при першому створенні каналу. Перша версія використовувала структуру onion-data для поступової побудови шляху [1]. Ця версія застосовує протокол Diffie-Hellmon для розподілу ключів по вузлу. Ця реалізація має так звану пряму секретність, що означає, що якщо зловмисник розшифрує та отримає один із ключів шифрування, він не зможе використовувати його для відкриття інших рівнів.

Ще однією зміною до цієї версії стало додавання серверів каталогів. До цієї поправки віднесено інформацію про адреси мережі які видаютьсь зовні від мережі. У міру збільшення розміру мережі такий спосіб видачі інформації став неможливим, а сервери каталогів пропонували гнучкий спосіб вирішення цієї проблеми. Це також підвищує безпеку, оскільки цю інформацію неможливо вивчити.

1.3 Tor

Перша версія цибулевої маршрутизації було опубліковано у 1996 році. Тоді Департамент морських досліджень фінансував проект. Він підтримував веб-сайт, де опубліковувалася інформація про хід проекту. Сайт містив технічну інформацію про різні покоління, а також наукову статтю про цибулеву маршрутизацію. Вони припинили своє фінансування у 2004 році і було припинено роботу сайту і він був закритий.

1.3.1 Браузер Tor

Браузер Tor створений для забезпечення анонімності в мережі Інтернет і створений на мові програмування C. Tor Browser Bundle, або TBB містить програму, яка дає можливість користувачеві під'єднуватися до мережі Tor як клієнт, а також надає спеціальний браузер Firefox за допомогою якого можна відвідувати веб-сайти.

Він має певне програмне забезпечення для запуску клієнта і реле. Попередньо налаштований для роботи в якості клієнта, і коріктувачу не потрібно вказувати додаткові параметри для вибору напрямків програмного забезпечення. Порівняно з деякими діями які коректувальник може зробити для підвищення конфіденційності: включення розширення NoScript, яке використання лише HTTPS для формування підключення.

Клієнти використовують Tor для формування під’єднань через реле. Усі комп'ютери, які під’єднані один з одним за допомогою TBB, яка є мережею Tor. Для ефективної роботи мережі Tor необхідні і інші види компонентів, крім клієнтів і реле. Далі подивимося, які є інші види компонентів мережі Tor.

1.3.2 Мережа Tor

У своїй найпростішій реалізація мережі цибулева маршрутизація складається з клієнтів, які застосовують мережу для надсилання та отримання повідомлень, маршрутизаторів. Окрім клієнтів та ретрансляторів, мережа Tor має, також сервери імен, які відслідковуюьб сервери, зареєстровані в мережі, та приховані служби, які можна застосовувати для веб-публікацій. Приховані служби можна підключити лише через певні сервери Tor.

Клієнти та маршрутизатори є основною частиною Tor мережі. Клієнти можуть виступати в ролі маршрутизаторів в мережі, але більшість клієнтів застосовують мережу лише для встановлення зв’язків з клієнтами, прихованих служб та зовнішніх мереж.

Для того, щоб Tor став стійким до прослуховування, до мережі були додані охоронці входу разом із звичайними цибулевими маршрутизаторами. У початковій реалізації перший вузол був випадково вибраний з усіх маршрутизаторів, які були у мережі, що означало, що поки з'єднання формувалося знову і знову протягом певного часуб в будь який момент зловмисник зміг би отримати інформацію з першого вузла.

Іншим заходом підвищення безпеки є додавання мостів. З метою підвищення захисту певні маршрутизатори не розголошували свої IP адреси публічно.

Приховані послуги також згадувалися раніше. Вони не підвищують інформаційну безпеку, але розробники забажали включити приховані послуги. Ці послуги можна використовувати, для надсилання швидких повідомлень між клієнтами. Коли прихована служба хоче опублікувати свою адресу, вона створює кілька лампових каналів і повідомляє адресу певного маршрутизатора каналу серверу імен. Клієнт може підключатися до послуги через вузли зустрічі.

1.3.3 Створення цибулевого ланцюга

Поглянем, як створюється ланцюг. Користувач Tor встановлює з'єднання з межами Tor мережі, його програма завантажує список усіх зареєстрованих маршрутизаторів. З них перше встановлюється вхідний вузол. Він стає першим маршрутизатором у цьому ланцюгу тому він вибирається зі списку маршрутизаторів. Коли цей вузол є недосяжний, вибирається інший вузол.

Після цього вибору входу створюється наступна частина схеми. Щоб ланцюг став стабільним то потрібно обрати вузли, які є стабільні.

Після цього три маршрутизатори вибрані випадковим чином, формують ланцюг генерації ключів шифрування і використовується протокол Діффі-Хеллмана, тому нові пари ключів утворюються по черзі з кожним вузлом тоді коли ключі були встановлені з попереднім вузлом в один шлях. Це забезпечує пряму секретність, яка означає, що вузли не можуть відстежити ключі. Після цієї операції маршрутизатор має свій ключ шифрування для створення і очищення цибулі, а також інформацію кому потрібно відправляти повідомлення. На Рисунку показано під’єднання до служби імен та утворення цибулевого шляху.

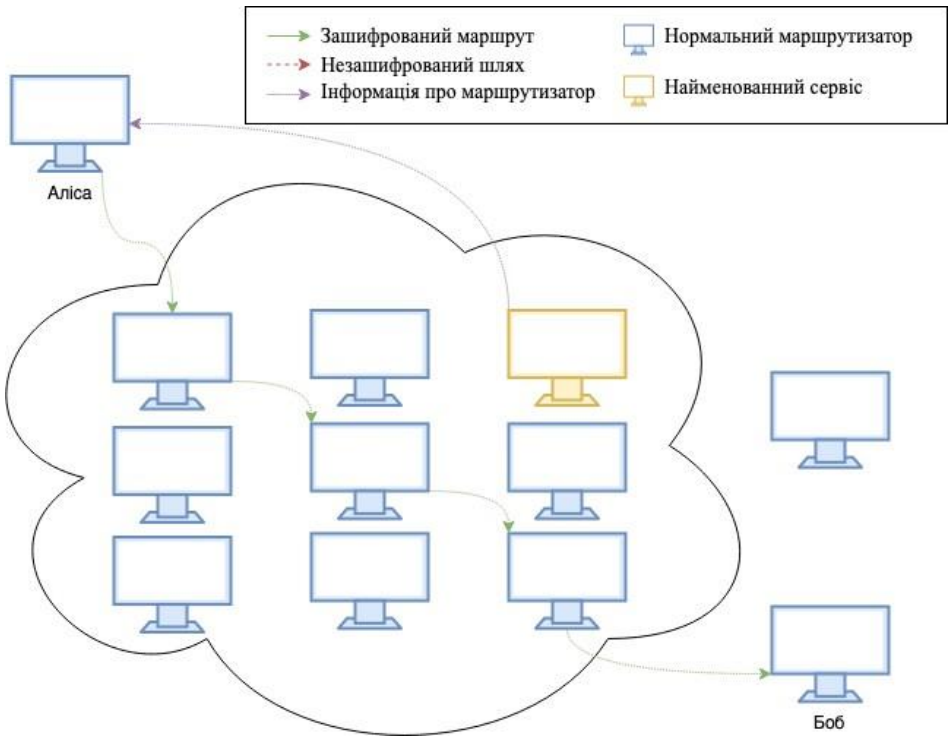

Рисунок 1.1 – Створення цибулевого шляху і його взаємодія з службою імен

У результаті формування шляху окремо з різними маршрутизаторами один вузол не знає адреси всіх вузлів в ланцюгу. Перший маршрутизатор бачить, чи клієнт підключений до мережі, але він не бачить пункт призначення повідомлення, який користувач перенаправляє через мережу Tor. Вузол входу знає, що споживач робить з мережею. Він отримує повідомлення і відправляє їх по ланцюгу.

Останній вузол бачить, куда відправляється інформація, але він не бачить, хто її відправляє. Цей вузол ще називають вузлом виходу. Він відправляє повідомлення до кінцевого користувача.

1.3.4 Комірки

Пеіні повідомлення, що передаються між вузлами всередині цибулевого ланцюга, мають назву комірки. Вони можуть використовуватися, наприклад, для створення, руйнування і підтримки живого з'єднання ланцюжків. Комірки мають певний розмір 512 байт, для того, щоб запобігати аналізу трафіку. Усі інші комірки, окрімре трансляційних комірок, мають назву контрольні комірми, також в них є заголовок, що описує їхній ідентифікатор що має назву circID, і призначення комірки називається командою контрольної комірки, або CMD.

Комірки, що утворюються для ретрансляції повідомлень, мають назву комірками ретрансляції.

На Рисунку 1.2 знаходиться структура комірки реле і комірки управління.

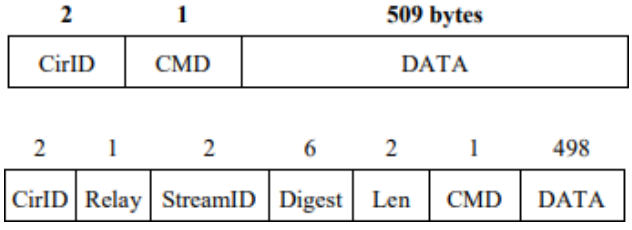

Рисунок 1.2 - Заголовок комірки управління (верхній) і заголовок комірки реле (нижній)

Висновки до розділу 1

У першому розділі розглянуто створення цибулевої маршрутизації і ідея її технологічного та еволюція технології впродовж трьох поколінь.

Показана практична реалізація використання цибулевої маршрутизації яка називається Tor. Переглянуто її особливості, алгоритм створення цибулевого ланцюга, направлення повідомлень в цибулевому ланцюгу і розглянено призначення комірок в системі.

Значіть, програмна реалізація цибулевої маршутизації яка називається Tor утворена для забезпечення повної анонімності користувача в мережі Інтернет. Користуючись програмним забезпечення Tor користувач забезпечив себе повною анонімністі в мережі Інтернет за допомогою цибулевої маршрутизації.

2 АТАКИ НА TOR ТА МЕТОДИ ПРОТИДІЇ НИМ

Мета цибулевої маршрутизації полягає в запобіганню локалізації та ідентифікації користувачів, тому що атаки на систему намагаються підключитися, тобто, журнали відвідувань веб-сайту і відправка повідомлень до користувача. В цьому розділі розглянемо, які саме атаки були розроблені проти системи Tor. Таким чином потенційні зловмисники можуть спробувати деанонімізувати користувачів. Певні атаки можуть мати на меті зменшення якості обслуговування, тобто, за допомогою атаки типу «відмова в обслуговуванні».

Ця маршрутизація захищається від ідентифікації користуючись заплутування. Отже вона може забезпечити анонімність, тільки тоді коли база користувачів дуже велика. Сила Tor полягає в тому, що зловмисники не простежують шлях повідомлення, і не можуть відстежити повідомлення, які виходять з мережі, тому, вони їх відправляють. Це є концепція системи.

Є багато атак на криптографічні системи, але більшість таких атак засновані на спробі захопити декілька маршрутизаторів в мережі. Переважно, один маршрутизатор вузла входить в ланцюг. Для цієї настройки можна виконати атаку аналіза трафіка, яка намагається зв'язати вхідні і вихідні повідомлення. Пасивні атаки відстежують потік повідомлень, а активні атаки модифікують трафік. В цьому розділі ми розглянемо, які типи атак призначені для системи Tor.

2.1 Категорії атак

Юха Сало подылив атаки на п'ять категорій в залежно від їх особливостей. Вын виділяє п'ять категорій атак: імовірнісна модель, атака вибору маршрутизатора входу і виходу, атака на рівні AS і на глобальному рівні, атака на аналізу трафіку і часу, і атака вразливості протоколів.

Моделі ймовірності застосовують математичні ймовірнісні моделі для аналізу мереж. Ці моделі були розроблені для мережі Tor. Моделі засновані на двох припущеннях:

- Тільки один користувач може бути пов'язаний з одним виходом.
- Вихід може бути пов'язаний з користувачем тільки тоді, коли можна

спостерігати за обидвами виходами.

Отже, вони насправді не є атаками, а швидше пропонують способи оцінки, наскільки можна зловмиснику деанонімізувати користувача.

Атака на вибір вхідного і вихідного маршрутизатора наводять на збільшення імовірнісного вибору маршрутизатора атакуючого як вузол входу і виходу. Він згадує про два типи атак з цьою категорією:

- Компрометація анонімності з використанням пакетного обертання, пробують створити вузли між маршрутизаторами, щоб можна було відмовитися від обслуговування інших клієнтів вбиваючи все в маршрутизатор.
- Низько ресурсні маршрутизації проти мережі Tor, які пробують повідомити брехливу інформацію про пропускну здатність і час роботи маршрутизатора в службі імен

Атаки аналізу трафіку і часу, займають велику кількість атак і категорій. Такі атаки намагаються послабити анонімність під час мережевого потоку. За допомогою цього шаблону зловмисники можуть зіставити вхідний і вихідний мережевий трафік. Пасивні атаки помічають, що пакети проходять через систему, а активні атаки пробують, помітити пакети водяним знаком, щоб їх було простіше аналізувати.

AS та атаки глобального рівня говорять про те, що зловмисник може виявити та маніпулювати великою частиною вхідного та вихідного трафіку. Цей тип зловмисників може використовувати, аналіз трафіку та часу для розпізнавання потоків даних. Tor не є призначеним для захисту користувача від AS рівня суперника.

Вразливості протоколів цеє остання категорії атаки, яка намагається знайти слабкі місця в комунікаційних протоколах. Ефективні атаки в протоколі автентифікації Tor свідчать про уразливість у розподілі ключів сеансу, що призводить до невідповідності ключів сеансу. Зловмисник намагається визначити можливих кандидатів, а потім намагається підтвердити результати активною атакою на схему.

2.2 Нові атаки Tor

Напади на мережу Tor, є популярною темою дослідження. Після 2011 року на цю тему з’явилося багато різних статей, і в цьому розділі ми розглянемо деякі більш популярні та менш популярні.

2.2.1 The Bad Apple Attack

Використання програм P2P для відстеження та профілювання користувачів Tor

Tor - це популярна мережа анонімностей із низькою затримкою. Однак Tor не захищає від використання незахищеної програми для виявлення IP-адреси або відстеження потоку TCP. Крім того, через пов’язаність потоків Tor, що передаються разом по одній схемі, трасування одного потоку, відправленого по ланцюгу, відстежує їх усі. Дивно, але невідомо, чи дозволяє ця зв’язаність на практиці відстежувати значну кількість потоків, що походять із захищених (тобто проксі-серверів) програм. У цій роботі ми показуємо, що зв’язаність дозволяє нам відстежувати 193% додаткових потоків, включаючи 27% потоків HTTP, які, можливо, походять із «безпечних» браузерів. Зокрема, ми простежили 9% потоків Tor, що передаються нашими інструментальними вузлами виходу. Використовуючи BitTorrent як незахищений додаток, ми розробляємо дві атаки, що відстежують користувачів BitTorrent на Tor. Ми запускаємо ці атаки в природі протягом 23 днів і виявляємо 10000 IP-адрес користувачів Tor. Використовуючи ці IP-адреси, ми профілюємо не лише завантаження BitTorrent, але й відвідувані веб-сайти за країною походження користувачів Tor. Ми показуємо, що користувачі BitTorrent на Tor є надмірно представленими в деяких країнах порівняно з користувачами BitTorrent за межами Tor. Аналізуючи тип завантаженого контенту, ми пояснюємо спостережувану поведінку більшою концентрацією порнографічного контенту, завантаженого в масштабі країни. Нарешті, ми представляємо результати, що свідчать про існування підземної екосистеми BitTorrent на Tor.

У березні 2011 року дослідники з французького Інституту досліджень в галузі комп'ютерних наук та автоматизації задокументували атаку, здатну виявити IP-адреси користувачів BitTorrent у мережі Tor. "Атака поганого яблука" використовує дизайн Tor і використовує переваги небезпечного використання додатків, щоб пов'язати одночасне використання захищеного додатка з IP-адресою відповідного користувача Tor. Один із методів атаки залежить від контролю вихідного вузла або реакцій відстежувача викрадачів, тоді як вторинний метод атаки частково заснований на статистичній експлуатації відстеження розподіленої хеш-таблиці.

Результати, представлені в дослідницькій роботі про погані яблука, засновані на атаці, розпочатій авторами дослідження проти мережі Tor. Атака націлена на шість вихідних вузлів, тривала двадцять три дні і виявила загалом 10000 IP-адрес активних користувачів Tor. Це дослідження є важливим, оскільки це перша задокументована атака, призначена для націлювання програм обміну файлами P2P на Tor. [86] BitTorrent може генерувати до 40% всього трафіку на Tor. [87] Крім того, погана атака на яблуко ефективна проти небезпечного використання будь-якої програми над Tor, а не лише BitTorrent.

2.2.2 Application-level attack

Вони вивчають атаки на рівні додатків. Атака на рівні програми націлена на комп’ютери, навмисно спричиняючи несправність в операційній системі чи додатках комп’ютера. Це призводить до того, що зловмисник отримує можливість обійти звичайний контроль доступу. Зловмисник використовує цю ситуацію, отримуючи контроль над програмою, системою чи мережею. Атаки на рівні програми можуть виконуватися як на сервері, так і на клієнтському комп'ютері. Ключовою відмінністю від інших типів атак, таких як підслуховування / підробка мережевого трафіку, є здатність зловмисника бути активним (аж до повного контролю над скомпрометованою машиною), а не пасивно дивитись на мережевий трафік, що трапляється у будь-який момент часу.

Найпоширеніші причини атак на програми

Для надання необхідної підтримки споживачам, персоналу, постачальникам та іншим зацікавленим сторонам веб-сайти та відповідне програмне забезпечення

повинні бути доступні цілодобово та без вихідних.

Брандмауери та SSL не забезпечують захисту від атак веб-додатків виключно тому, що посилання на веб-сайт мають бути оприлюднені.

До всіх сучасних систем баз даних може бути легко отримати доступ через певні порти. Будь-яка людина може спробувати безпосередньо підключитися до баз даних, ефективно обходячи механізми безпеки операційної системи, і може отримати доступ до поточної бази даних через певні порти. Будь-хто може спробувати легко обійти протоколи захисту операційної системи за допомогою прямих посилань на бази даних. Це дозволяє контактувати з легальним перевезенням, і тому ці порти залишаються відкритими і становлять значну слабкість.

Веб-програми також мають прямий доступ до внутрішньої інформації, наприклад до баз даних клієнтів, які мають конфіденційну інформацію та яких захищати набагато складніше. Деякі сценарії полегшують збір та розповсюдження даних і будуть доступні тим, хто не має доступу. Вони легко перенаправлять нічого не підозрюючий трафік в інше місце і незаконно відмовляться від конфіденційної інформації, якщо зловмиснику стане відомо про такі вразливості при написанні.

Багато веб-додатків виготовляються на замовлення і тому потребують нижчого рівня перевірки, ніж готове програмне забезпечення. Однак користувацькі програми вразливіші до атак.

2.2.3 Probabilistic Analysis in a Black-box Model

Probabilistic analysis of onion routing in a black-box model

Проводимо імовірнісний аналіз маршруту цибулі. Аналіз представлений у моделі чорного ящика анонімного спілкування в рамках універсально складової (UC), яка абстрагує основні властивості цибулевої маршрутизації в присутності активного супротивника, який контролює частину мережі та знає всі апріорні розподіли на вибір користувачем місця призначення. Наші результати кількісно визначають, скільки може отримати противник при ідентифікації користувачів, використовуючи знання про їх імовірнісну поведінку. Зокрема, ми показуємо, що в межах, коли мережа стає великою, анонімність користувача є найгіршою або тоді,

коли інші користувачі завжди вибирають пункт призначення, і найменш вірогідно, що йому відвідають, або коли інші користувачі завжди обирають пункт призначення. Це є найгірша анонімність із супротивником, який контролює частку в маршрутизаторах, порівняно з найкращою анонімністю проти противника, який контролює дріб маршрутизатора.

Отже пропонуємо новий статистичний підхід до аналізу стохастичних систем на основі специфікацій, поданих у підлогіці неперервної стохастичної логіки (CSL). На відміну від минулих методів чисельного та статистичного аналізу, ми припускаємо, що досліджувана система - це невідома розгорнута чорна скринька, яку можна пасивно спостерігати для отримання слідів, зразків, але не можна контролювати. Враховуючи набір страт (отриманих за допомогою моделювання Монте-Карло) та властивість, наш алгоритм перевіряє, базуючись на тестуванні статистичних гіпотез, чи надає вибірка докази для висновку про задоволення чи порушення властивості, та обчислює кількісний показників впевненості у своїй відповіді; якщо вибірка не надає статистичних доказів для висновку про задоволення або порушення властивості, алгоритм може відповісти "не знаю".

Ця модель, зазвичай, є абстракцією, яка не враховує відомих атак на цибулеву маршрутизацію, але намагається продовжити постійний процес формалізації анонімності.

2.2.4 CellFlood

Атака CellFlood, щоразу, коли клієнт Tor розширює ланцюг, він генерує цибулеву маршрутизацію, використовуючи загальну цибулю ключ цільового маршрутизатора. Аналогічним чином цільовий маршрутизатор обробляє його, використовуючи свій приватний цибулевий ключ. Ця операційна модель робить обробку лушпиння з маршрутизаторів ще більш дорогою, ніж їх генерування.

Наприклад, як ми експериментально підтвердили, виконувати 1024-бітові операції із закритим ключем на сучасному високоякісному сервері в 20 разів повільніше ніж виконання 1024-розрядних операцій з відкритим ключем, що означає час для обробки клітинку CREATE в 4 рази більше, ніж її генерування.

Цей дисбаланс може бути використовуються зловмисними клієнтами, щоб з відносно невеликими зусиллями споживати всі обчислювальні ресурси АБО за допомогою безперервного потоку комірок CREATE. Що ще гірше, зловмисникові навіть не потрібно створювати різні цибульні шкурки для кожного, оскільки всі клітини можуть містити однакову цибульну шкіру. Через архітектуру програмного забезпечення Tor, заливаючи маршрутизатор Tor надмірною кількість запитів CREATE не обов'язково порушує можливість переадресації маршрутизатора.

Тор делегує обробку цибулевих шкурок одному пулу або більше потоків (процесів), які називаються CPU Workers; це дозволяє основний потік (процес) щоб не відставати від більш критичної роботи над клітинками RELAY DATA, тоді як працівники центрального процесора виконують дорогі та стійкі до затримок завдання у фоновому режимі. Тим не менше, а Маршрутизатор, який отримує клітинки CREATE зі швидкістю, вищою від того, що його колектив процесора може колективно обробити, врешті-решт почне їх відкидати, відповідаючи клітинками DESTROY. Як наслідок, АР, який зазнає атаки, збирається відкинути вироблену цибульну шкіру чесними клієнтами, які, в свою чергу, врешті-решт припинять вибирати або для своїх ланцюгів. Таким чином, якщо CellFlood виконується стратегічно, на вибраному наборі важливих АБО, це може призвести до перевантаження вцілілої частини Tor (наприклад, шляхом переповнення непошкоджених маршрутизаторів надмірною кількістю або ланцюгами), а також сприяння схемам, що проходять через певні маршрутизатори, які цілком можуть бути скомпрометовані або контрольовані зловмисниками, тим самим погіршуючи анонімність мережі Tor в цілому. Згідно з нашими експериментами, навіть маршрутизатори, що працюють на останньому обладнанні, можуть обробляти обмежену кількість клітин CREATE в секунду (тобто кілька Мбіт / с), що становить їх потенційно вразливі до CellFlood.

З іншого боку, маршрутизатори Tor можуть обробляти дані, що передаються на набагато більшій швидкості, близько десятків сотень Мбіт / с. Отже, зловмисник, який зацікавлений у виключенні маршрутизатора або набору маршрутизаторів із Мережі Tor буде краще використовувати потік CREATE комірок, а не простіший, але дорожче, DoS-атака в мережі рівень

2.2.5 EgotisticalGiraffe

Дана складається з двох етапів: перший етап полягає у ідентифікації користувачів Tor в Інтернеті, а другий етап полягає у спрямуванні ідентифікованих користувачів до користувачів на сайт. Перший етап ідентифікації користувачів Tor є простий. Це пакет, в якому користувачі завантажують програму Tor, яка називається Tor Browser Bundle і постачається з налаштованим браузером Firefox, який налаштовано для перегляду в режимі мережі Tor. Переважно браузери Firefox оголошують BuildID, які повідомляють веб-сайту про версію браузера, з якого користувач перебуває на сайт.

Другий етап атаки - атака "посередника" для зараження комп’ютера потенційного користувача Tor шкідливим кодом Firefox. Користувач повинен бути відправлений на спеціальний сервер, який заражає браузер користувача спеціально розробленим кодом. Певний код використовує розширення JavaScript, але повинні бути використані інші вразливості. Як тільки комп'ютер користувача заразиться цим зловмисним кодом, він буде надсилати всі види даних зворотного виклику користувачеві, які користувач може легко використовувати для ідентифікації.

Результат і захист

Тут також видно, що було проведено деякі тести та були деякі початкові проблеми з Tor Browser, але врешті-решт все спрацювало. Кріми цього, невідомо, скільки було здійснено атак на користувача. Незабаром після атаки Mozilla виправила уразливість JavaScript у Firefox, вимкнувши функцію E4X. Крім того, також усунуто уразливість у наборі Tor Browser. Tor Browser також використовується з опцією NoScript, яка відключає певний тип атаки, але вона не вмикається за замовчуванням та потрібно, щоб користувач вмикав її самостійно.

2.2.5 Snipper Attack

У 2017 році дослідники виявили новий недолік алгоритмів управління потоком даних Tor. Вони назвали нову атаку, яка використовує слабкість снайперська атака. Атака вимагає малої кількості ресурсів і може бути

використана для паралізації певних вузлів в мережі Tor.

Схема атак

Sniper Attack використовує контроль потоку мережі Tor. Контроль потоку використовується в ситуації, коли велика кількість пакетів надсилається з одного місця у друге. Просто надсилання великої кількості пакетів даних одночасно може призвести до перевантаження приймаючої сторони, тому контроль потоку необхідний для більшої швидкості доставки певних пакетів. Контроль потоку дозволяє клієнту контролювати велику швидкість передачі даних. Доставка зберігає в своїй пам'яті буфер з 1100 комірок і відправляє 110 комірок для кожного виклику SENDME. Протокол такий:

- клієнт утворив ланцюг через цибульну мережу до зовнішнього сервера.

- клієнт надсилає запит на завантаження даних із сервера.

- клієнт надсилає запит на сервер і починає перенаправлення даних до 1100 комірок.

- клієнт зчитує дані та надсилає SENDME, як тільки будеи прочитано 110 комірок.

- результат, він додає ще 100 комірок до буфера обміну.

- якщо не всі дані були прочитані то потрібно перейти до попередніх кроків.

Одна з версій атаки показана на малюнку 2.2. Коли зловмисник хоче закрити вхідний вузол, він повинен створити схему так, щоб власний вузол противника використовувався як вихідний. Хитрість полягає в тому, що клієнт не читає з буфера дані які виводить, що може призвести до того, що пакет даних збирають в буфер вхідного вузла, що в кінці призводить до його небезпечного завершення.

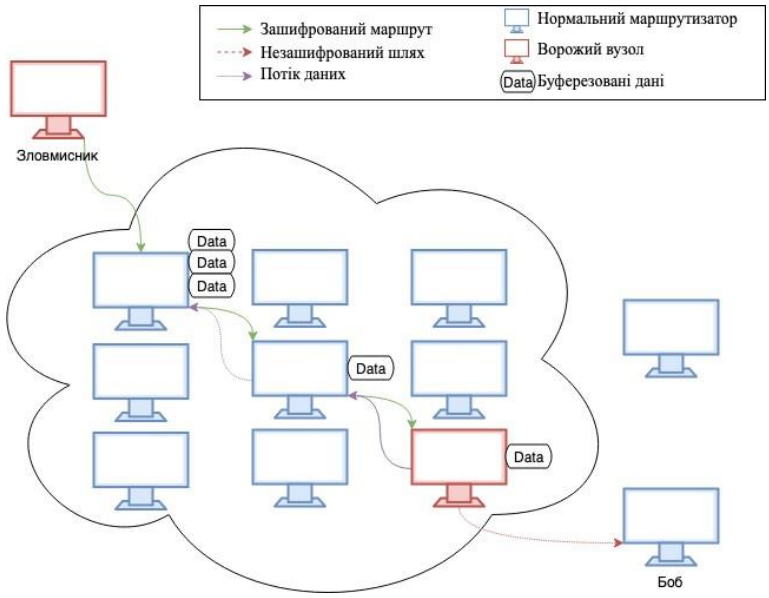

Рисунок 2.1 – Формування цибулевого шляху з зловмисником

Результат і захист

Snipper Attack може бути використана, щоб примусити приховану службу вибрати вузол зловмисника як реле захисту та використовувати це шкідливе реле захисту для деанонімізації сервісу. Дослідники провірили дану атаку і виявили, що з її допомогою можливо знеанонімізувати приховані служби, і підрахували, що зловмисник може використовувати її, щоб вимкнути 20 ліпших реле Tor.

Захистом, що був обраний для додавання до мережі Tor, є знищенням каналу, коли його пам’ять починає закінчуватися. В результаті ця атака стала неробочою (неефективною). Хоч ця атака не може більше використовувати для маршрутизатора DoS, вона все одно може бути використана для використання своєї пропускної здатності.

2.2.6 New traffic confirmation attacks

У 2017 року дослідники виявили групу ретрансляторів, які, як вони припускаємо, намагалися деанонімізувати користувачів. Схоже, вони націлені на людей, які працюють або мають доступ до прихованих служб мережі Tor. Атака включала модифікацію заголовків протоколу Tor для здійснення атак підтвердження трафіку.

Реле, що атакують, приєднали до мережі 30 січня 2017 року, а потім видалили їх із мережі 4 липня. Хоча вони не знали, коли вони почали робити атаку, користувачі, які працювали або мали доступ до прихованих служб з початку лютого по 4 липня, постраждали.

На жаль, дослідники й досі незрозуміли, що зробили користувачам. Вони знають, що атака шукала користувачів, які отримали приховані дескриптори служб, але зловмисники, ймовірно, не змогли побачити жодного трафіку на рівні програми. Можливо, через атаку намагалася дізнатись, хто публікував приховані дескриптори служб, що дозволило б зловмисникам дізнатися місцезнаходження цієї прихованої служби. Теоретично атака також може використовуватися для прив'язки користувачів до місця призначення на звичайних схемах Tor, але дослідники не знайшли доказів того, що зловмисники експлуатували будь-які вихідні реле, що робить цю атаку менш імовірною. І нарешті, вони не знають, скільки даних зберігали зловмисники, і завдяки способу розгортання атаки, їхня модифікація заголовка протоколу могла допомогти іншим зловмисникам у деанонімізації користувачів.

Реле слід оновити до недавнього випуску Tor, щоб закрити певну вразливість протоколу, яку використовували зловмисники, але пам'ятайте, що запобігання підтвердженню трафіку в цілому залишається відкритою проблемою дослідження. Клієнти, які оновляться (як тільки будуть випущені нові версії браузера Tor), зроблять ще один крок до обмеження кількості входу, які можуть бачити свій трафік, тим самим зменшуючи шкоду від майбутніх атак, подібних до цієї. Операторам прихованих служб слід розглянути можливість зміни розташування своєї прихованої послуги.

Технічні деталі

Дослідники вважають, що вони використовували комбінацію двох класів атак: атаки на підтвердження дорожнього руху та атаки на Сібіл.

Атака підтвердження трафіку можлива, коли зловмисник контролює або спостерігає за реле на обох кінцях ланцюга Tor, а потім порівнює терміни руху, обсяг або інші характеристики, щоб зробити висновок, що два реле дійсно знаходяться в одній схемі. Якщо перше реле в ланцюзі (яке називається "охороною входу") знає IP-адресу користувача, а останнє реле в ланцюзі знає ресурс або пункт

призначення, до якого вона отримує доступ, тоді разом вони можуть її деанонімізувати.

Особливою атакою підтвердження, яку вони використовували, була активна атака, коли реле на одному кінці вводить сигнал у заголовки протоколу Tor, а потім реле на іншому кінці зчитує сигнал. Ці атакуючі реле були достатньо стабільними, щоб отримати консенсус-прапори HSDir ("придатний для прихованого каталогу служб") та Guard ("придатний для охорони входу"). Потім вони вводили сигнал щоразу, коли їх використовували як приховану службову директорію, і шукали вводи сигналу, коли їх використовували як охорону входу.

Спосіб введення сигналу здійснювався шляхом надсилання послідовностей команд "реле" проти "ретрансляції", щоб кодувати повідомлення, яке вони хочуть відправити. Для фону Tor має два типи комірок: комірки зв'язку, які призначені для сусіднього реле в ланцюзі, і комірки реле, які передаються на інший кінець схеми. У 2016 році було додано новий тип релейної комірки, яка називається «ретрансляційна рання» клітина, яка використовується для запобігання людям будувати дуже довгі шляхи в мережі Tor. (Дуже довгі шляхи можуть бути використані, щоб викликати перевантаження та допомогти порушити анонімність). Але виправлення для шляхів нескінченної довжини спричинило проблему з доступом до прихованих служб, і одним із побічних ефектів цього виправлення для помилки 1038 було те, що, хоча й обмежується кількість вихідних (далеко від клієнта) клітин "ранньої ретрансляції" на ланцюгах, проте не обмежується кількість вхідних (до клієнта) ранніх комірок ретрансляції.

Отже, підсумовуючи, коли клієнти Tor звернулись до атакуючого ретранслятора в його ролі каталог прихованих служб, опублікували або отримували прихований дескриптор служби, це реле надішле приховану назву служби (закодована як зразок релейних і релейних ранніх комірок) назад по ланцюгу. Інші атакуючі реле, коли їх вибирають для першого стрибка схеми, шукали б вхідні ретрансляційні комірки (оскільки їх ніхто не надсилає) і, таким чином, дізнавались, які клієнти запитували інформацію про приховану послугу.

У цій атаці є три важливі моменти:

А) напад який кодував ім'я прихованої служби у введеному сигналі. Кодований сигнал зашифровується, оскільки він передається по каналу TLS між реле. Однак цей сигнал буде легко прочитати та інтерпретувати будь-хто, хто

керує ретрансляцією та отримує закодований трафік. А також вони можуть турбуватися про глобального супротивника (наприклад, велике розвідувальне агентство), яке реєструє Інтернет-трафік у вхідних охоронців, а потім намагається зламати шифрування посилань Tor. Те, як було здійснено цю атаку, послаблює анонімність Tor щодо інших потенційних зловмисників - як у той час, коли це відбувалося, так і після того, як у них є журнали трафіку. Отже, якщо атака була дослідницьким проектом (тобто не навмисно зловмисною), вона була розгорнута безвідповідально, оскільки піддає користувачам небезпеку на невизначений час у майбутньому.

Б) ця атака введення сигналу заголовка протоколу насправді досить акуратна з точки зору дослідження, оскільки вона дещо відрізняється від попередніх атак тегування, які націлювались на корисне навантаження на рівні програми. Попередні атаки тегування модифікували корисне навантаження на охороні входу, а потім шукали змінене навантаження на вихідному реле (яке може бачити розшифроване корисне навантаження). Ці атаки не працюють в іншому напрямку (від вихідного реле назад до клієнта), оскільки корисне навантаження все ще зашифровано на вході. Але оскільки цей новий підхід модифікує ("теги") заголовки комірок, а не корисне навантаження, кожне реле на шляху може бачити тег.

В) слід нагадати, що хоча цей конкретний варіант атаки підтвердження трафіку дозволяє мати високу впевненість та ефективну кореляцію, загальний клас пасивних (статистичних) атак підтвердження трафіку залишається невирішеним і, ймовірно, тут би спрацював чудово. Тож гарною новиною є те, що атаки на підтвердження дорожнього руху не є новими чи дивовижними, але поганими є те, що вони все ще працюють.

Тоді другим класом атак, який вони використовували разом із атакою на підтвердження дорожнього руху, була стандартна атака Sybil - вони підписали близько 115 швидких реле без виходу, всі працювали на 50.7.0.0/16 або 204.45.0.0/16. Разом ці реле склали приблизно 6,4% від пропускної здатності охорони в мережі. Потім, частково завдяки нашим поточним параметрам обертання захисних пристроїв, ці реле стали захисниками входу для значної частини користувачів протягом п'яти місяців роботи.

Вони фактично помітили ці реле, коли вони приєдналися до мережі, оскільки сканер DocTor повідомив про них. Тоді ми розглянули набір нових реле і прийняли рішення, що це не така велика частка мережі. Зрозуміло, що є місце для вдосконалення з точки зору того, як дозволити мережі Tor зростати, одночасно забезпечуючи підтримку соціальних зв'язків з операторами всіх великих груп ретрансляторів. (Загалом наявність дуже різноманітного набору місць розташування реле та операторів реле, але не допускаючи будь-яких пошкоджених реле, здається складною проблемою; з іншого боку, наші сценарії виявлення їх у цьому випадку помітили, тому є надія на краще рішення тут.)

У відповідь було зроблено такі кроки:

1) Вилучення атакуючих реле з мережі.

2) Випускання оновлення програмного забезпечення для реле, щоб запобігти використанню клітин "ранніх ретрансляцій".

3) Випускання оновлення програмного забезпечення, яке (як тільки оновиться достатня кількість клієнтів) дасть змогу сказати клієнтам перейти на використання одного охоронного входу, а не трьох, щоб зменшити вплив реле з часом.

4) Клієнти можуть сказати, чи отримали вони реле або релейну комірку. Для досвідчених користувачів нова версія Tor попереджає вас у своїх журналах, якщо реле на вашому шляху вводить будь-які ретрансляційні клітини.

5) Подальше зростання мережі Tor і різноманітність операторів реле, що зменшить вплив супротивника заданого розміру.

6) Вивчення кращих механізмів, направлення соціальних зв'язків, щоб обмежити вплив зловмисного набору реле.

7) Подальше зменшення впливу огороджень з часом, можливо, шляхом продовження терміну служби обертання.

8) Краще розуміння статистичних атак кореляції трафіку та того, чи можуть заповнення або інші підходи пом'якшити їх.

9) Покращення дизайну прихованих служб, включаючи ускладнення реле, що слугують прихованими точками каталогів служб, щоб дізнатися, з якою прихованою адресою служби вони працюють.

2.3. Модель загроз для системи Tor

Отже з існуючі атаки можна зробити висновок, що більшість атак направлені на те, щоб користувач при створенні зв’язку в системі зробив свій вибір у бік маршрутизаротів зловмисника для подальшого аналізу трафіку користувачів зловмисника з ціллю деанонімізації користувача. Висновок цих атак наведено в Таблиці 2.1.

Дивлячись на Рисунок 2.3, ми побачимо, що у зловмисника буде певний набір дій, з яких можна почати деанонімізацію користувачів абоприховати сервіси. Тобто, зловмисник може використовувати атаку кореляції трафіка, так, щоб ідентифікувати певну кількість потоків, щоб їм не довелося нічого робити, щоб отримати певну інформацію про трафік.

Рисунок 2.2 – Схема здійснення загроз

Проте, вони тоже можуть використовувати певну атаку вибору входу і виходу маршрутизаторів, щоб покращити своє становище для аналізу певного трафіку.

Під час перебування зловмисника не на рівні AS основний клас атаки, який зловмисник може використати для погіршення анонімності, є аналіз трафіку. Проте, перед тим як проводити аналіз трафіку, необхідно виконати певні попередні кроки, щоб отримати можливість використання атаки для підтвердження трафіку. Зловмисник може користовуватися ще декількома різними способами, щоб туди дістатися: атака вибору маршрутизатора входу і виходу і вразливості на програмному рівні.

Висновки до розділу 2

Тут розділі було описано основні категорії атак на систему мережі Tor. Детальноописано кожний тип атак: схеми атак, мета атак та результати відбування цих атак.

Кроме цього було описано способи протидії кожній атаці, які були добавлені в програмне забезпечення мережі Tor для унеможливлення проведення таких атак в майбутньому.

На основі 2 розділу розроблено програмне рішення для захисту від атак типу аналіза трафіку і часу.

3 ПРОГРАМНЕ РІШЕННЯ ДОДАВАННЯ ВИПАДКОВИХ ЗАТРИМОК В СИСТЕМІ TOR

Основним завданням розділу є утворення програмованого вирішення для створення анонімності користувача в системі Tor завдяки додавання випадкових затримок при проходженні пакета по мережі відклієнта до сервера та навпаки.

Це рішення має завдання протистояти атакам аналізу трафіку і часуметою якої є визначення відправника і одержувача пакетів в системі Tor для деанонімізації користувача.

3.1 Теоретичні основи

Tor система побудована з використанням протоколу TCP для надсилання та отримання пакетів. Через цей механізму можна перехоплювати та модифікувати пакети за допомогою спеціального написаного коду.

Код програми написаний мовою програмування Go за допомогою бібліотеки з відкритим кодом Trudy, яка працює на віртуальній машині Vogront, реалізація якої відбувається мовою програмування Ruby.

Також необхідна віртуальна машина яка буде використовуватися для перехоплення трафіку, який раніше перенаправлявся із хост-машини. Завдяки цьому виділяється програмне рішення, в якому додаються затримки, коли пакети проходять через віртуальну машину з подальшим відправленням в мережу Tor.

3.1.1 Генератор псевдовипадкових чисел

Під час втілення програмного рішення необхідно забезпечити генерацію випадкового числа, яке виконує роль затримки пакета в секундах. Ми розглянемо генератори псевдовипадкових чисел.

Генератор псевдовипадкових чисел це алгоритм, який генерує послідовність чисел, елементи яких не залежать один від одного і підкоряються певному розподілу.

Якісні вимоги до ГПСЧ:

- досить тривалий період, який гарантує відсутність циклу послідовності в межах вирішеної задачі. Тривалість періоду повинна бути математично доведена.
- ефективність це швидкість роботи алгоритму та низьке споживання пам'яті.
- відтворення - Ви можете відтворити раніше створену послідовність чисел в будь-яку кількість разів.
- Портативність це однакова операція на різному обладнанні та операційних системах.
- Швидкість отримання елемента X_{n+1} послідовності чисел, при завданні елемента X_n, для i деякої величини, що дозволить розділити послідовність на декілька потоків.

Для провірки послідовності потрібно мати критерії для провірки якості різних двійкових послідовностей. Вони провіряють статичні властивості цих послідовностей: рівно ймовірність знаків, незалежність різних сусідніх знаків, однорідність послідовності. Усі дані критерії є критеріями XI-квадрата Пірсона для перевірки даної гіпотези.

Далі переглянемо наступні послідовності $\{Y_j\}$, j = 1, …, m, де кожна Y_j є певною величиною, що приймає вибір значень із алфавіту A. Всі величини Y_j повинні мати однаковий розподіл та розглядатися як вихідні значення певного генератора.

Ця послідовність $\{Y_j\}$ повинна задовольняти умови рівноімовірності знаків, коли кожна Y_j розподілена рівноймовірно на A. Отже, будь яке значення із A повиннобути у довільній реалізації даної послідовності на однакову кількість разів.

Тест на виконання умов рівноймовірності не відрізняється великою чутливістю, проте він є досить швидким. Отже в якості підсилення можна розглянути умову рівноймовірності знаків, коли рівноймовірними в послідовності повинні бути пари, трійки або четвірки знаків.

Також послідовність $\{Y_j\}$ може задовольняти умову незалежності знаків, якщо за ймовірність прийняти якесь певне значення для Y, яке не залежить від того, які значення можуть прийняти Y_1, Y_2 , … , $Y_{j-1.}$ Проте провірка певної умови зазвичай вкрай важка, тому інколи розглядають більш слабкі вимоги – такі як, значення Y_j неповинно залежати від значення Y_{j-1}.

Послідовність $\{Y_j\}$ повинна задовольняти умову однорідності, коли для довільної реалізації певний вибірковий розподіл, одержаний на усій послідовності, повинен співпадати із вибірковим розподілом, отриманим на певній довільній підпослідовності певної довжини. Потрібно зазначити, що для виконання умови однорідності не є важливо, який розподіл повинні мати Y_j. Цей розподіл не обов'язково має бути рівноймовірним.

Вважається, що періодичність, яка представлена у вигляді байтової послідовності, це область значень певного випадкового знначення величини Y_j і лежить в межах між 0 та 255.

3.1.2 Архітектура взаємодії компонентів

Архітектура остаточної системи складається з таких компонентів:

- Tor клієнт.
- Хост-машина.
- Віртуальна машина VirtualBox з оболонкою Vagrant.
- Програмне вирішення Trudy на мові Go.

Tor клієнт знаходиться на хост-машині і створюється віртуальна машина VirtualBox з оболонкою Vagrant. На ній можна встановлювати та запускати програмні рішення Trudy для провірки і утримки пакета. Взаємодію компонентів даної архітекрути можна побачити на Рисунку 3.1.

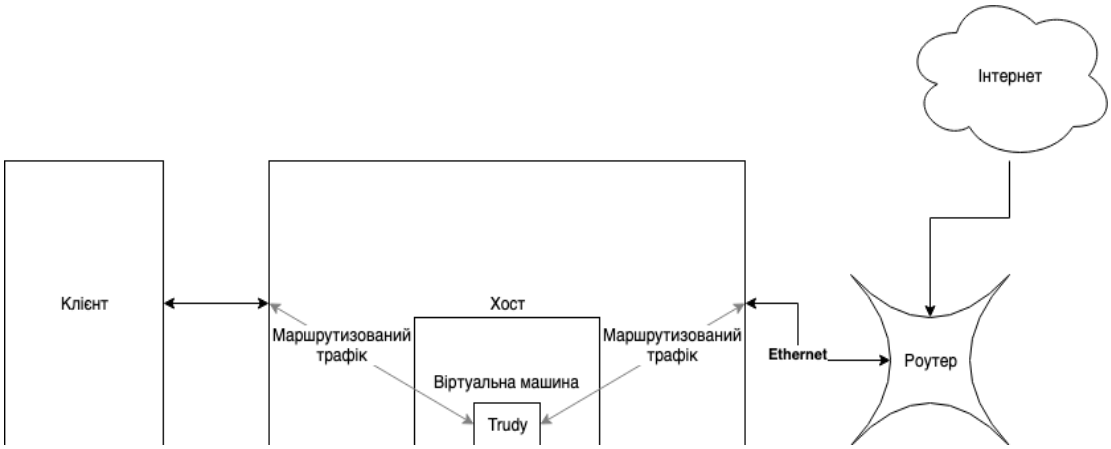

Рисунок 3.1 - Побудова взаємодій з мережею Tor

Послідовність пакетів по побудові утворюється так:

1. Формуються пакети на стороні клієнта за допомогою системи Tor.
2. Відправляються пакети клієнтам.
3. Перехоплюються пакети віртуальних машин.
4. Проходять пакети через якесь програмне рішення.
5. Затримуються усі пакети.
6. Проходять пакети через роутери.
7. Отримуються пакети в сервері.
8. Формуються пакети з боку сервера.
9. Відправляються пакети до сервера.
10. Проходять пакети через певні роутери.
11. Перехоплюються пакети віртуальних машин.
12. Проходять через пакети певні програмні рішення.
13. Затримуються певні пакети.
14. Отримують пакети клієнти.

3.1.3 Бібліотека Trudy

Бібліотека Trudy - це програмне вирішення, яке діє як проксі-сервер, який може змінювати та скидати трафік для довільних TCP-з'єднань. Trudy можна використовувати для програмної зміни TCP-трафіку для клієнтів, які не знають свій проксі-сервер. Trudy створює двосторонній "міст" для кожного з'єднання, яке є проксі-сервером. Пристрій, який ви міксуєте, під'єднується до Trudy, а Trudy - до місця призначення, призначеного клієнту.

Модуль Trudy - це запрограмований код для виконання певних модифікацій, які потрібують користувачі. Trudy наводять приклад заглушки певного модуля і користувачі застосовують його. Оскільки Trudy пишеться в Go, всі модулі теж потрібно писати в Go. Для маршрутизації трафіку та правильного налаштування середовища потрібна мати підготувати віртуальну машину.

Створення модуля для Trudy є дуже простою. Він користується якимись методи у певному порядку. Кожен з них призначений для якоїсь дії. Бібліотека Trudy пропонує кілька методів роботи з її пакетами:

- Drop () - якщо повернути значення true, весь пакет відкидається.
- DoMangle () - якщо повернути true, викличеться Mongle ().
- Mongle () - змінює завантаженні дані.
- DoIntercept () - якщо true повернеться, дані відправляться перехоплювачеві Trudy.
- DoPrint () - якщо true повернеться, викличеться PrettyPrint ().
- Deserialize() – перетворює послідовність бітів в певну потрібну структуру даних.
- PrettyPrint () - друкує дані в зрозумілому і зручному для людини форматі.
- Serialize () – перетворює структуру даних у послідовність бітів.
- BeforeWriteTo () - дії, які використовуються перед записом даних в будь яку сторону.
- AfterWriteTo () - дії, які використовуються після запису даних в будь яку сторону.

3.1.4 Віртуальна машина

VirtualBox - це потужний продукт для віртуалізації x86 та AMD64 / Intel64 як для корпоративного, так і для домашнього використання. VirtualBox не тільки надзвичайно багатий на функціональні можливості, високопродуктивний продукт для корпоративних клієнтів, це також єдине професійне рішення, яке є у вільному доступі як програмне забезпечення з відкритим кодом на умовах Загальної публічної ліцензії GNU (GPL) версії 2.

В даний час VirtualBox працює на Windows, Linux, Macintosh і Solaris і підтримує велику кількість гостьових операційних систем, включаючи, але не обмежуючись ними, Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10), DOS / Windows 3.x, Linux (2.4, 2.6, 3.x та 4.x), Solaris та OpenSolaris, OS / 2 та OpenBSD.

VirtualBox активно розробляється з частими випусками і має постійно зростаючий перелік функцій, підтримуваних гостьових операційних систем та платформ, на яких він працює. VirtualBox - це зусилля спільноти, що підтримуються цілеспрямованою компанією: кожному пропонується зробити свій внесок, тоді як Oracle забезпечує, щоб продукт завжди відповідав професійним критеріям якості.

3.2 Практичні основи

Початком створення програмного рішення додавання випадкових затримок в пакети, буде відбуватися вибмрання генераторів псевдо випадкових чисел для генерування псевдо випадкового числа з неуможливленням здійснення обчислень певних наступних чисел зловмисником. Наступни етап це є вибір оптимальних діапазонів границь затримки експериментальним способом. Останнім етапом стає реалізація програмного рішення з використанням вибраного генератора випадкових чисел та діапазон границь затримок.

3.2.1 Генератор псевдовипадкових чисел

Під час вибору генератора псевдовипадкового числа вибір попав на один з генераторів, який називається Fortuna.

Fortuna це є сімейство криптографічно стабільних генераторів псевдовипадковог числа. Система Фортуни складається з трьох частин:

• Власний генератор, який ініціалізується носінням фіксованої довжини і видає певну довільну кількість псевдовипадкових бітів.

• Акумулятор ентропії, який збирає випадкові дані з різних джерел і змінює початковий номер генераторів кожен раз, коли набирається певна кількість ентропії.

• Система управління файлами початкових номерів, що забезпечує можливість генерувати випадкові числа безпосередньо після перезагрузки комп'ютера.

Генератор приймає випадкове початкове число фіксованого розміру з приводу ентропії і генерує досить довгу послідовність псевдовипадкових даних. Генератор складається з блокового шифру який знаходиться в режимі перехресного шифрування, як зоюражено на рисунку 3.2. У цій реалізації в якості блочного шифру було обрано AES 256. Введення певного відкритого тексту в блок-шифр - це просто лічильник, а 256-бітні ключі повинні надходити від акумулятора. Через те, що AES є 128-розрядним блочним шифром, вона генерує 16 байт даних одночасно. Як тільки вона генерує ці дані, вона генерує два додаткові блоки, які повинні використовуватися як новий ключ наступного разу. Це задовільняє вимогу про те, що знання поточного стану генератора не розкриває минулі генеровані дані.

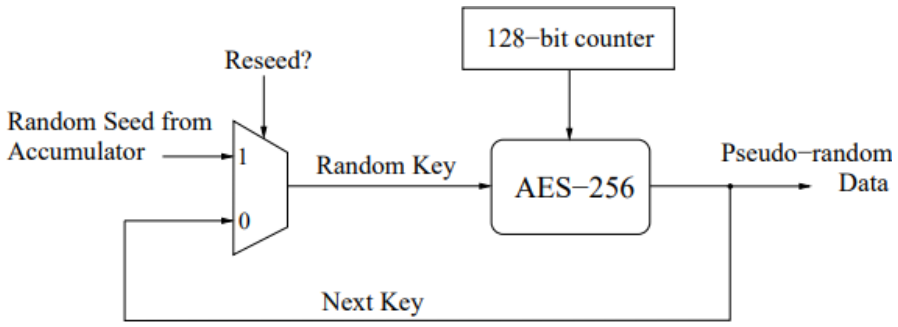

Рисунок 3.2 – Ядро генератора Fortuno

Використовуючи генератор псевдовипадкового числа Fortuna без файла з послідовністю, перше число ґрунтується на певному часі доби, імені користувача або списку, який встановлюється на мережевих інтерфейсах. Апаратний стан показано на Рисунку 3.3.

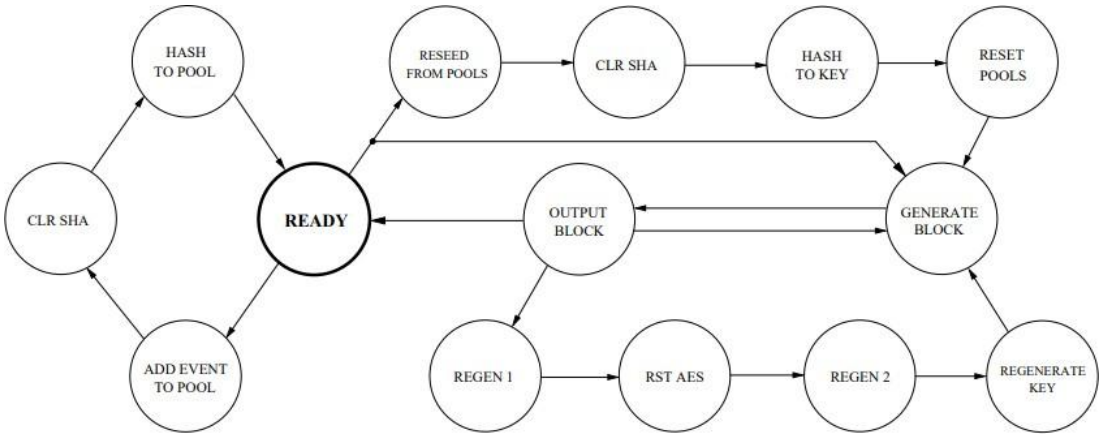

Рисунок 3.3 – Апаратний стан машини Fortuna

Батарея для роботи використовує 32 ентропійних пули для збору випадковостей з навколишнього середовища. Використання цієї ентропії допомагає виходити з ситуацій, коли зловмисник отримує інформацію про стан генератора.. Схема пулів ентропії показано на Рисунку 3.4.

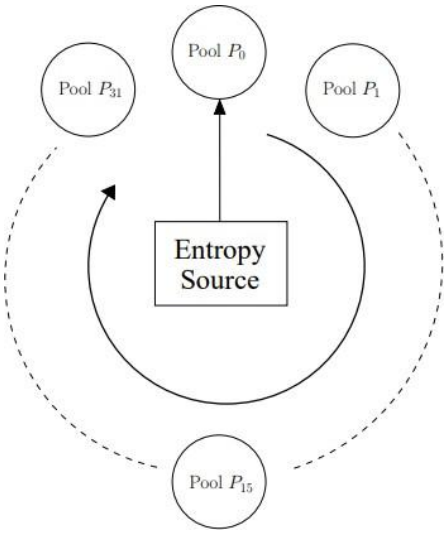

Рисунок 3.4 – 32 ентропійний пул ентропії Fortuno

Результати перевірки генератора

При перевірці тестів на критерії для проведення якості певної випадкової двійкової послідовності генерування послідовності довжиною в 100000 біт. Отже результ проведення тестів, для перевірки якості випадкових двійкових послідовностей показано в Таблиці 3.1.

Таблиця 3.1 – Результати проведення тестів на критерії

Значення				
Критерій	**Теоретичні**	**Практині**		
Рівноймірність знака	268.54	288.56	556.00	281.10
Незалежність знака	65288.14	64889.65	64440.24	65671.25
Однорідність двійкових послідовностей	2606.47	2257.38	2257.44	2407.25

Отже виходячи з ц результатів генератора Fortuna кожний тест повинен проходити на критеріїпроте в різних послідовностях. Нерівномірність певних згенерованих послідовностей утворених нерівномірним часом проведення перевірки.

3.2.2 Діапазон границь затримки

Діапазон обмежень затримки використовується для обмеження генерації випадкового числа для затримки пакета. Вибір діапазону меж затримки здійснюється експериментально. Експеримент проводився за допомогою клієнта Tor, завантажуючи 50 сторінок www.google.com під час запуску. Мета експерименту - визначати певні сторінки, які не будуть завантажуватися в різних діапазонах затримки. Відомо, що є ймовірність збою зв'язку між клієнтом і сервером у разі великої затримки, оскільки сторона сервера вважатиме, що на стороні клієнта виникають проблеми.

Встановити мінімальну межу затримки у вигляді 0 секунд. Це пов’язано з найменшим невід’ємним числом. Максимальна межа затримки встановлюється у вигляді 5 секунд з подальшим зменшенням до 0,5 та 1 секунди до мінімальної межі затримки. Результати експериментів наведені в таблиці 3.2.

Таблиця 3.2 – Результати проведення експеримента вибору діапазону границі затримки

Діапазон границі затримки, сек.	**Кількість завантажувальних сторінок, шт.**	**Кількість не завантажувальних сторінок, шт.**
o – 5	o	5o
o – 4	o	5o
o – 3	o	5o
o – 2.5	o	5o
o – 2	5o	o
o – 1.5	5o	o
o – 1	5o	o
o – o.5	5o	0

Експериментально виявлено оптимальний діапазон генерації випадкових чисел від 0 до 1 секунди.

Максимальне значення мінімального обмеження встановлено на 0 секунд. Утворює той факт, що використання більшості значення мінімальної межі призводить до того, що змінюється діапазон меж затримки, якщо це не

застосовується, то воно не змінює величину макцимальної межі через що змінює загальний діапазог додавання затримок маніпуляції або мінімальний ризик затримка.

Мінімальне значення максимального обмеження встановлюється на 1 секунду. Це пов'язано з тим, що використання більш високого значення максимального ліміту призвело до того, що час від часу сервер перестає реагувати на запити, вважаючи, що у клієнта проблема з мережею.

3.2.3 Програмна реалізація

Реалізація методів з самого початку є порожніми, тому нам потрібно реалізувати метод, який називається Mangle. За допомогою цього методу затримка пакетів буде реалізована для випадково сформованого числа в певних діапазонах межі. Реалізація цього методу виглядає так:

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


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

// Закриття блоку оброблення помилкиdefer rng.Close()


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

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

        // Перевірка чи входить значення в границіif timeout > min && timeout
        < max {
            break
        }
    }
    // Змінна зберігання згенерованого випадкового числа в наносекунда
    // Затримка пакету на випадкове певне число в наносекундахtime.Sleep(pause)
    var pause = time.Duration(timeout) * time.Nanosecond
```

Через це програмний код здійснює генерацію випадкового числа за допомогою Fortun в діапазоні границь 0 – 1 секунди із затримкою пакетів на випадково згенеровані числа в певному діапазоні.

3.2.4 Результати

На даному рисунку видно відкриття та закриття TCP-з'єднання, що відбуваються у асинхронному режимі. Тому програмне рішення здатне працювати паралельно з кількома потоками даних, які проходять через цю машину, так що не виникає затримки всіх пакетів при відвідуванні, наприклад, кілька мторінок нараз.

```
2019/05/27 15:41:14 [INFO] ( 27 ) Closing TCP connection.
2019/05/27 15:41:14 [INFO] ( 40 ) Closing TCP connection.
2019/05/27 15:41:14 [INFO] ( 45 ) TCP Connection accepted!
```

Рисунок 3.5 – Інформація про відкриття та закриття TCP з’єднання

На рисунку 3.6 показані результати програмного рішення, такі як відображення інформація про пакет: дата та час, вихідна та вхідна IP-адреса та порт.

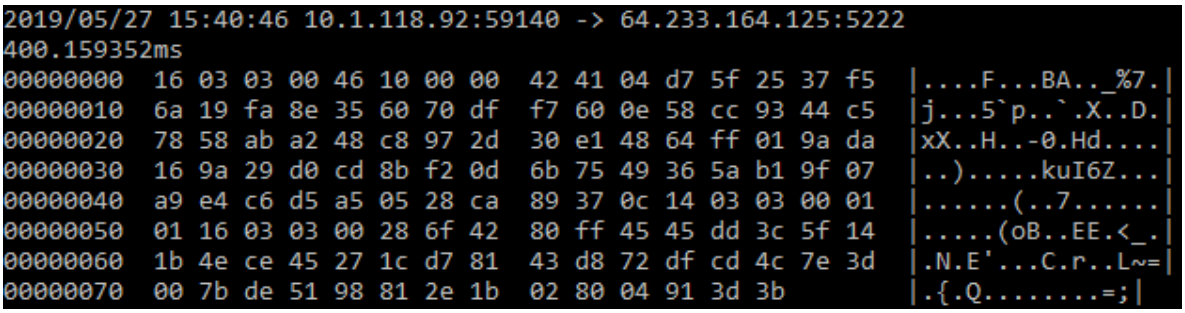

Рисунок 3.6 – Інформація про пакети

Висновки до розділу 3

В даному розділі було розглянуто основи поняття вибору генератора псевдо-випадкових чисел, використовувалися бібліотеки Trudy на проектах MITM-VM для віртуальної машини Vagrant. Через комбінацію даних компонентів було реалізовано систему додавання різних випадкових затримок при проходженні пакетівчерез віртуальної машини для системи мережі Tor.

Далі було розброблено певне програмне рішення, яке було зреалізовано на мові програмування Go, яка надає гарантію швидкої реалізації програмного коду та паралельно можна виконувати обчислювальні робіти.

Отже результатами цього програмного рішення стало додавання випадкових затримок при проходженні пакетів через віртуально машину Vagrant використовуючи систему мережі Tor.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Надзвичайні ситуації: визначення причини, класифікація.

Щодня у світі фіксуються тисячі подій, при яких відбувається порушення нормальних умов життя і діяльності людей і які можуть призвести або призводять до загибелі людей та/або до значних матеріальних втрат. Такі події називаються надзвичайними ситуаціями.

Засоби масової інформації, як правило, привертають увагу громадськості до надзвичайних ситуацій, особливо коли вони пов'язані з життям відомих особистостей, призвели або можуть призвести до великої кількості жертв, становлять загрозу нормальному життю і діяльності груп людей, цілих регіонів чи навіть країн. Майже жодне газетне видання, жоден випуск радіо або телевізійних новин не виходить без таких повідомлень.

Основними причинами виникнення надзвичайних ситуацій в Україні є:

- надзвичайне техногенне навантаження території;

- значний моральний та фізичний знос основних виробничих фондів більшості підприємств України;

- погіршення матеріально-технічного забезпечення, зниження виробничої і технологічної дисципліни;

- незадовільний стан збереження, утилізації та захоронення високотоксичних, радіоактивних та побутових відходів;

- ігнорування економічних факторів, вимог, стандартів;

- недостатня увага керівників відповідних органів державного управління до проведення комплексу заходів, спрямованих на запобігання надзвичайним ситуаціям природного і техногенного характеру та зниження їх наслідків;

- відсутність сучасних систем управління небезпечними процесами;

- низька професійна підготовка персоналу та населення до дій в екстремальних умовах;

- дефіцит кваліфікованих кадрів;

- низький рівень застосування прогресивних ресурсозберігаючих і екологобезпечних технологій [46].

Класифікація надзвичайних ситуацій:

1) Надзвичайні ситуації класифікуються за характером походження, ступенем поширення, розміром людських втрат та матеріальних збитків.

2) Залежно від характеру походження подій, що можуть зумовити виникнення надзвичайних ситуацій на території України, визначаються такі види надзвичайних ситуацій:

- техногенного характеру;
- природного характеру;
- соціальні;
- воєнні.

3) Залежно від обсягів заподіяних надзвичайною ситуацією наслідків, обсягів технічних і матеріальних ресурсів, необхідних для їх ліквідації, визначаються такі рівні надзвичайних ситуацій:

- державний;
- регіональний;
- місцевий;
- об’єктовий [47].

Надзвичайна ситуація - порушення нормальних умов життя і діяльності людей на об'єктах або територіях, спричинене аварією, катастрофою, епідемією, стихійним лихом, епізоотією, епіфітотією, великою пожежею, застосуванням засобів ураження, що призвели або можуть призвести до людських і матеріальних втрат, а також велике зараження людей і тварин [48].

4.2 Контроль за станом охорони праці.

Контроль за станом охорони праці може бути :

- відомчим, що здійснюється посадовими особами, повноважними представниками та службами міністерства або іншого центрального органу виконавчої влади, а також асоціації, корпорації, концерну або іншого об'єднання підприємств;

- регіональним, що здійснюється посадовими особами, повноважними представниками та службами місцевих органів виконавчої влади та органів місцевого самоврядування;

- громадським, що здійснюється виборними органами та представниками професійних спілок, інших громадських організацій;

- страховим, що здійснюється страховими експертами з охорони праці Фонду соціального страхування від нещасних випадків на виробництві та професійних захворювань;

- внутрішнім, що здійснюється в межах підприємства (установи, організації) відповідними службами, посадовими особами та громадськими інспекторами (уповноваженими трудових колективів) цього підприємства.

Відповідно до Закону України "Про охорону праці" [49] державне управління охороною праці в Україні здійснюють:

- Кабінет Міністрів України;

- міністерства та інші центральні органи виконавчої влади;

- Рада міністрів Автономної Республіки Крим, місцеві державні адміністрації та органи місцевого самоврядування.

Державний нагляд за додержанням законодавчих та інших нормативно правових актів з охорони праці здійснюють:

- органи державного нагляду за охороною праці Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду (спеціально уповноважений центральний орган виконавчої влади з нагляду за охороною праці);

- органи Головної державної інспекції з нагляду за ядерною та радіаційною безпекою Міністерства екології та природних ресурсів України;

- органи державного пожежного нагляду Державного департаменту пожежної безпеки Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи;

- органи та заклади санітарно-епідеміологічної служби Міністерства охорони здоров'я України.

Кожен із перерахованих органів виконує функції в межах своїх повноважень, визначених положеннями про ці органи. Вищий нагляд за додержанням і правильним застосуванням законодавства з охорони праці здійснюється Генеральним прокурором України і підпорядкованими йому прокурорами [50].

Ступеневий контроль за станом охорони праці

Основним заходом профілактики виробничого травматизму є запровадження ступеневого контролю за станом охорони праці.

Ступеневий контроль за станом умов безпеки праці на робочих місцях, у бригадах, цехах, структурних підрозділах здійснює адміністрація (роботодавець) разом з профспілковою організацією або уповноваженими трудового колективу з питань охорони праці. Метою ступеневого контролю є своєчасне виявлення недоліків і вжиття оперативних заходів щодо їх усунення та попередження виробничого травматизму. Контроль здійснюється від робочого місця до підрозділу в цілому.

Об'єктами контролю на кожному ступені, як правило, повинні бути:

- на першому ступені: робочі місця, обладнання, засоби захисту, виконавці робіт;
- на другому ступені: ланки, бригади, групи чи інші аналогічні підрозділи у складі виробничої дільниці, цеху тощо;
- на третьому ступені: виробнича дільниця, цех та інші підрозділи у складі структурного підрозділу.

Кількість ступенів контролю у кожному структурному підрозділі встановлюється адміністрацією та профспілковою організацією, залежно від особливостей його організаційної структури.

Порядок проведення ступеневого контролю:

1) Перший ступінь контролю повинні здійснювати майстри.

Перший ступінь контролю здійснюється перед початком і протягом робочого дня. На першому ступені контролю перевіряють підготовленість і стан робочих місць, обладнання, засобів захисту, готовність до роботи виконавців та інші питання. Виявлені недоліки записуються у журналах ступеневого контролю, які ведуться безпосередніми керівниками робіт.

2) Другий ступінь контролю здійснює комісія під головуванням головного спеціаліста або головного механіка, за участю члена профспілки, уповноваженого трудового колективу з охорони праці.

Другий ступінь контролю повинен здійснюватися один раз на декаду. На другому ступені контролю перевіряють організацію і наслідки проведення першого ступеня контролю, виконання заходів з охорони праці в підрозділах, а також стан робочих місць, утримання обладнання, засобів захисту, дотримання працюючими безпечних прийомів праці та інші питання.

Результати перевірки голова комісії записує у журнал ступеневого контролю. За наявності недоліків, які створюють загрозу здоров'ю і життю людей, роботи припиняються, про що повідомляється керівництво структурного підрозділу.

3) Третій ступінь контролю здійснює комісія під головуванням, як правило, керівника підприємства. До складу комісії повинні входити: головний інженер та інші заступники керівника підприємства, інженер з охорони праці, голова профспілкового комітету.

Третій ступінь контролю повинен проводитися один раз на місяць.

На третьому ступені контролю перевіряють організацію і наслідки перевірки першого та другого ступенів контролю, виконання заходів з охорони праці в цехах, а також стан умов праці, утримання обладнання, засобів захисту та дотримування працівниками безпечних умов праці на робочих місцях, проведення медичних оглядів, перевірки знань з охорони праці та інші питання.

Наслідки перевірки на третьому ступені контролю оформлюються актом або записом в журналі ступеневого контролю. Акти перевірок і журнали третього ступеня контролю повинні зберігатися у інженера з охорони праці протягом одного року з дня закінчення перевірки.

Наслідки перевірок на третьому ступені контролю розглядаються на нараді у керівника підприємства за участю майстрів цехів, головного механіка та голови первинної організації профспілки на «Дні охорони праці». Проведення наради оформляється протоколом із заходами по усуненню виявлених недоліків та порушень, переліком термінів та відповідальних осіб. За підсумками перевірки на третьому ступені контролю керівник підприємства при необхідності видає наказ із зазначенням заходів щодо поліпшення стану охорони праці, притягнення до відповідальності винних посадових осіб.

Контроль за виконанням цього порядку на підприємстві здійснює служба охорони праці.

Запровадження ступеневого контролю за станом охорони праці не потребує матеріальних витрат і в той же час є дієвим заходом щодо підвищення відповідальності не тільки посадових осіб, а й працівників за дотриманням вимог чинного законодавства з охорони праці, попередження виробничого травматизму [51].

Контроль за станом охорони праці є найбільш відповідальною та

трудомісткою функцією процесу управління, від якої залежить система управління охороною праці підприємства в цілому. Оперативно виявити можливі відхилення від норм безпеки праці, перевірити виконання запланованих заходів та управлінських рішень можливо лише на підставі регулярного та об’єктивного контролю на підприємстві. Контроль має здійснюватися керівниками всіх рівнів управління виробництвом. При створенні безпечних умов праці на підприємстві значну роль також відіграє громадський контроль, що провадиться громадськими інспекторами (представниками профспілок) або уповноваженими особами з питань охорони праці (у разі відсутності профспілки) [52].

ВИСНОВКИ

Ця робота була створена для програмного рішення та додавання випадкових затримок у пакетах даних. Було отримано такі результати:

- Досліджено роботу цибулевої маршрутизації і розвиток її поколінь.
- Досліджено практичну реалізацію цибулевої маршрутизації Tor і її технологічні особливості. У результаті аналізу було встановлено, що собою являє мережа Tor.
- Було описано атаки на систему Tor і способи їх протидії.
- Було описано моделі загрози систем мережі Tor.
- Досліджено компоненти для програмної реалізації додавання випадкових затримок. У результаті, було вибрано програмне рішення у вигляді бібліотеки Trudy, яка була реалізована мовою Go, віртуальної машини VirtualBox з оболонкою Vagrant та встановлено на ній проект MITM-VM реалізації «людина-поседерник».
- Досліджено архітектуру побудови компонентів. Вибрано генератори псевдовипадкових чисел.
- За допомогою експериментів розроблено певний діапазон границь генерування затримок.
- Досліджено програмні рішення додавання випадкових затримок на мові Go, для платформ UNIX-подібних платформ і Windows операційних систем, робота якого досліджена на операційній системі Windows 10.

Отже програмна реалізація додавання випадкових затримок створена поза мережею Tor, тому зловмиснику буде важко аналізувати трафік. Програмна реалізація може бути доповнена певними механізмами перехоплення і перетворення пакетів у відповідності до вимог користувача та організації, які використовують це рішення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Tor: The Second-Generation Onion Router / R. Dingledine, N. Mathewson, P. Syverson. – 2004. URL: https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf (дата звернення 15.05.2021).
2. A Peel of Onion / Paul Syverson. – 2011. URL: https://www.acsac.org/2011/program/keynotes/syverson.pdf (дата звернення 15.05.2021).
3. Onion routing. URL: http://www.onion-router.net/ (дата звернення 15.05.2021).
4. Tor. URL: http://www.torproject.org/ (дата звернення 15.05.2021).
5. Tor Metrics. URL: https://metrics.torproject.org/ (дата звернення 15.05.2021).
6. Low-Resource Routing Attacks Against Tor/ [K. Bauer, D. McCoy, D. Grunwald та ін.]. – 2007. URL: https://www.freehaven.net/anonbib/cache/bauer:wpes2007.pdf (дата звернення 15.05.2021).
7. A Stealthy Attack Against Tor Guard Selection / Q. Li, P. Liu, Z. Qin. – 2015. URL:https://pdfs.semanticscholar.org/973a/3ad736c743cb50eacccbfb03e275f8ed2e 10.pdf (дата звернення 16.05.2021).
8. Recent Attacks On Tor / Juha Salo. – 2010. URL: http://www.cse.hut.fi/en/publications/B/11/papers/salo.pdf (дата звернення 17.05.2021).
9. Tor Network Status: TorStatus. URL: https://torstatus.blutmagie.de/ (дата звернення 16.05.2021).
10. Analyzing the Effectiveness of Passive Correlation Attacks on the Tor Anonymity Network / Sam DeFabbia-Kane. – 2011. URL: https://pdfs.semanticscholar.org/17a6/736b5b8d9a2e516adbebb338e3265681b1c9.pdf (дата звернення 20.05.2021).

11. Probabilistic Analysis of Onion Routing in a Black-box Model / J. Feigenbaum, A. Johnson, P. Syverson. – 2012. URL:https://www.ohmygodel.com/publications/wpes08- feigenbaum.pdf (дата звернення 19.05.2021).
12. Compromising Anonymity Using Packet Spinning / [V. Pappas, E. Athanasopoulos, S. Ioannidis та ін.]. – 2008. URL: https://www.freehaven.net/anonbib/cache/torspinISC08.pdf (дата звернення 20.05.2021).
13. On the Effectiveness of Traffic Analysis Against Anonymity Networks Using Flow Records / [S. Chakravarty, M. V. Barbera, G. Portokalidis та ін.]. – 2014. URL: https://www.freehaven.net/anonbib/cache/nfattackpam14.pdf (дата звернення 20.05.2021).
14. Low-Cost Traffic Analysis of Tor / S. J. Murdoch, G. Danezis. – 2005. – URL: https://www.cs.ucy.ac.cy/courses/EPL682/papers/anon-2.pdf (дата звернення 20.05.2021).
15. A New Cell Counter Based Attack Against Tor / [Z. Ling, J. Luo, W. Yu та ін.]. – 2009. URL: http://web.cse.ohio-state.edu/~xuan.3/papers/09_ccs_llyfxj.pdf (дата звернення 20.05.2021).
16. Browser-Based Attacks on Tor / T.Abbott, K. Lai, M. Lieberman, E. Price. – 2007. URL: https://www.freehaven.net/anonbib/cache/abbott-pet2007.pdf (дата звернення 22.05.2021).
17. A Practical Con-gestion Attack on Tor Using Long Paths / N. S. Evans, R. Dingledine, C. Grothoff. – 2009. URL: https://www.freehaven.net/anonbib/cache/congestion- longpaths.pdf (дата звернення 22.05.2021).
18. How Much Anonymity does Network Latency Leak? / N. Hopper, E. Y. Vasserman, E. Chan-Tin. – 2010. URL: https://www-users.cs.umn.edu/~hoppernj/ccs-latency-leak.pdf (дата звернення 22.05.2021).

19. Passive-Logging Attacks Against Anonymous Communications Systems / M.Wright, M. Adler, B. N. Levine, C. Shields. – 2008. URL: http://people.cs.georgetown.edu/~clay/research/pubs/wright-tissec-2008.pdf (дата звернення 25.05.2021).
20. AS-awareness in Tor Path Selection / M. Edman, P. Syverson. – 2009. URL: https://www.freehaven.net/anonbib/cache/DBLP:conf/ccs/EdmanS09.pdf (дата звернення 25.05.2021).
21. Large Scale Simulation of Tor: Modelling a Global Passive Adversary/ G. Gorman, S. Blott. – 2007. URL: https://pdfs.semanticscholar.org/b3e8/7f8d290ac9f38e9321fac7e94c1e74b62e6a.pdf (дата звернення 25.05.2021).
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 / [S. Le Blond, P. Manils, A. Chaabane та ін.]. – 2011. URL: https://arxiv.org/pdf/1103.1518.pdf (дата звернення 28.05.2021).
24. A potential HTTP-based application-level attack against Tor / X.Wang, J. Luo, M. Yang, Z. Ling. – 2011. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.710.6952&rep=rep1&type=pdf (дата звернення 28.05.2021).
25. CellFlood: Attacking Tor Onion Routers on the Cheap / [M. Barbera, V. Kemerlis, V. Pappas та ін.]. – 2013. URL: https://www.freehaven.net/anonbib/cache/esorics13-cellflood.pdf (дата звернення 28.05.2021).

26. Peeling back the layers of Tor with EgotisticalGiraffe. - 2013. URL: http://media.encrypted.cc/files/nsa/egotisticalgiraffe-guardian.pdf (дата звернення 01.06.2021).
27. Attacking Tor: how the NSA targets users' online anonymity / Paul Schneier. – 2013. URL: https://www.theguardian.com/world/2013/oct/04/tor-attacks-nsa-users-online-anonymity (дата звернення 01.06.2021).
28. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network / R.Jansen, F. Tschorsch, A. Johnson, B. Scheuermann. – 2014. URL: https://www.freehaven.net/anonbib/cache/sniper14.pdf (дата звернення 01.06.2021).
29. New Tor Denial of Service Attacks and Defenses. – 2014. URL: https://blog.torproject.org/new-tor- denial-service-attacks-and-defenses (дата звернення 01.06.2021).
30. On the Effectiveness of Traffic Analysis Against Anonymity Networks Using Flow Records / [S. Chakravarty, M. Barbera, G. Portokalidis та ін.]. – 2014. URL: https://www.freehaven.net/anonbib/cache/nfattackpam14.pdf (дата звернення 01.06.2021).
31. Tor security advisory: "relay early" traffic confirmation attack. – 2014. URL: https://blog.torproject.org/tor-security-advisory-relay-early-traffic-confirmation-attack (дата звернення 01.06.2021).
32. Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services / [A. Kwon, M. AlSabah, D. Lazar та ін.]. –2015. URL: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15paper-kwon.pdf (дата звернення 01.06.2021).
33. Traffic correlation using netflows. – 2014. URL: https://blog.torproject.org/traffic-correlation-using- netflows (дата звернення 01.06.2021).

34. Oracle VM VirtualBox Overview. URL: http://www.oracle.com/us/technologies/virtualization/oracle-vm- virtualbox-overview-2981353.pdf (дата звернення 01.06.2021).
35. Ferguson N. Cryptography Engineering / N. Ferguson, B. Schneier, T. Kohno. – 2010. URL: http://the-eye.eu/public/Books/HumbleBundle/cryptography_engineering_design_principles_and_practical_applications.pdf (дата звернення 01.06.2021).
36. Testing Random Number Generators / Dan Biebighauser. – 2000. URL: http://www- users.math.umn.edu/~garrett/students/reu/pRNGs.pdf (дата звернення 01.06.2021).
37. Fortuna: Cryptographically Secure Pseudo-Random Number Generation In Software And Hardware / R.McEvoy, J. Curran, P.Cotter, C. Murphy. 2006.URL:https://www.researchgate.net/publication/215858122_Fortuna_Cryptographic (дата звернення 06.06.2021).
38. STATISTICAL PROPERTIES OF PSEUDORANDOM SEQUENCES / Ting Gu. – 2016. URL: https://uknowledge.uky.edu/cs_etds/44/ (дата звернення 06.06.2021).
39. Statistical dependence: Beyond Pearson's ρ / D. Tjøstheim, H. Otneim, B. Støve. – 2018. URL: https://arxiv.org/pdf/1809.10455.pdf (дата звернення 06.06.2021).
40. How to modify general TCP/IP traffic on the fly with Trudy / Tomas Susanka. – 2017. URL:https://blog.susanka.eu/how-to-modify-general-tcp-ip-traffic-on-the-fly-with- trudy/ (дата звернення 06.06.2021).
41. Trudy / Kelby Ludwig. – 2017. URL: https://github.com/praetorian-code/trudy (дата звернення 06.06.2021).
42. Palat J. Introducing Vagrant / Jay Palat. – 2012. URL: https://www.linuxjournal.com/content/introducing-vagrant (дата звернення 06.06.2021).

43. MITM-VM / Kelby Ludwig. – 2016. URL: https://github.com/praetorian-code/mitm-vm (дата звернення 06.06.2021).
44. Fortuna / Jochen Voss. – 2013. URL: https://github.com/seehuhn/fortuna (дата звернення 06.06.2021).
45. Zhyrovetsky, V., Kovalyuk, B., Mocharskyi, V., Nikiforov, Y., Onisimchuk, V., Popovych, D., Serednytski, A. **Modification of structure and luminescence of ZnO nanopowder by the laser shock-wave treatment** (2013) Physica Status Solidi (C) Current Topics in Solid State Physics, 10 (10), pp. 1288-1291. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84885839967&doi=10.1002%2fpssc.201200889&partnerID=40&md5=287d09fff92386b92195d1d04be83ac2 (дата звернення 06.06.2021).
46. Надзвичайні ситуації та їх класифікація. URL: https://ru.osvita.ua/vnz/reports/bjd/22895/ (дата звернення 12.06.2021).
47. Класифікація надзвичайних ситуацій. URL: http://dktb.if.ua/media/com_lazypdf/pdf/Klasyfikacija%20nadzvychajnyh%20sytuacij.pdf (дата звернення 12.06.2021).
48. Надзвичайна ситуація. URL: https://uk.wikipedia.org/wiki/Надзвичайна_ситуація (дата звернення 12.06.2021).
49. Закон України “Про охорону праці”. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення 12.06.2021).
50. Державне управління охорони праці. URL: http://www.ztec.com.ua/ztec/e-lib/Охорона%20праці/Тема%203%20Держ%20управління%20ОП.pdf (дата звернення 12.06.2021).
51. Ступеневий контроль за станом охорони праці. URL: https://oppb.com.ua/news/stupenevyy-kontrol-za-stanom-ohorony-praci (дата звернення 12.06.2021).
52. Контроль за станом охорони праці на підприємстві. URL: https://ubr.ua/labor-market/ukrainian-labor-market/kontrol-za-stanom-ohoroni-praci-na-pidpriemstvi-69656 (дата звернення 12.06.2021).

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) {

}
```