НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

«На правах рукопису»
УДК 519.21

«До захисту допущено»
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
«Математичні методи криптографічного захисту інформації»

зі спеціальності: 113 Прикладна математика
на тему: **«Виявлення атак в потоці запитів, керованому прихованим марківським ланцюгом»**

Виконав:
студент IV курсу, групи ФІ-94
Костюк Кирило Миколайович _____

Керівник:
доцент ММЗІ ФТІ, к.ф.-м.н., доц.
Ніщенко Ірина Іванівна _____

Рецензент:
доцент ІБ ФТІ, к. т. н., доц.
Гальчинський Леонід Юрійович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ — 2023

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені Ігоря СІКОРСЬКОГО»
Навчально-науковий фізико-технічний інститут
Кафедра математичних методів захисту інформації

Рівень вищої освіти — перший (бакалаврський)
Спеціальність — 113 Прикладна математика,
ОПП «Математичні методи криптографічного захисту інформації»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Сергій ЯКОВЛЄВ
«__» _____ 2023 р.

ЗАВДАННЯ
на дипломну роботу

Студент: Костюк Кирило Миколайович

1. Тема роботи: «*Виявлення атак в потоці запитів, керованому прихованим марківським ланцюгом*», науковий керівник дипломної роботи: доцент ММЗІ ФТІ, к.ф.-м.н., доц. Ніщенко Ірина Іванівна,

затверджені наказом по університету №__ від «__» _____ 2023 р.

2. Термін подання студентом роботи: «__» _____ 2023 р.

3. Об'єкт дослідження: прихована марківська модель, що описує процес надходження запитів на сервер.

4. Предмет дослідження: спосіб виявлення атаки в потоці запитів, що керується прихованим ланцюгом Маркова.

5. Перелік завдань:

1) запропонувати приховану марківську модель для ймовірнісного опису процесу надходження та ідентифікації запитів на сервер;

2) в рамках запропонованої моделі побудувати оцінки параметрів ймовірнісних розподілів, якими моделюються спостережені дані;

3) провести чисельні експерименти для перевірки якості побудованих оцінок.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: Презентація доповіді

7. Орієнтовний перелік публікацій: доповідь «Виявлення атак в потоці запитів, керованому прихованим марківським ланцюгом» на XXI Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики».

8. Дата видачі завдання: 10 вересня 2022 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання	Примітка
1	Узгодження теми роботи із науковим керівником	01-15 вересня 2022 р.	Виконано
2	Огляд опублікованих джерел за тематикою дослідження	Вересень-листопад 2022 р.	Виконано
3	Вибір підходу до побудови прихованої марківської моделі для опису процесу надходження та ідентифікації запитів на сервер.	Листопад-грудень 2022	Виконано
4	Розробка математичних моделей для опису процесу надходження та ідентифікації запитів на сервер з використанням прихованої марківської моделі.	Січень-лютий 2023	Виконано
5	Реалізація генератора тестових даних	Лютий-березень 2023	Виконано
6	Реалізація розроблених математичних моделей та алгоритмів	Березень-квітень 2023	Виконано
7	Підготовка проміжного звіту та презентації результатів дослідження.	Квітень-травень 2023	Виконано
8	Оформлення та захист дипломної роботи	Травень-червень 2023	Виконано

Студент _____ Кирило Костюк

Керівник _____ Ірина Ніщенко

РЕФЕРАТ

Кваліфікаційна робота містить: 67 стор., 10 рисунків, 9 джерел.

У роботі розглянуто модель виявлення атак “грубої сили” на основі прихованої марківської моделі для підвищення захисту існуючих криптосистем. Ми вважаємо, що наявність чи відсутність атак на певному інтервалі часу залежить від стану прихованого ланцюга Маркова з двома станами, матриця перехідних імовірностей якого є невідомою. Спостережуваними даними є кількість запитів, які поступили на пристрій у комп’ютерній мережі впродовж фіксованого проміжку часу та кількість спроб, які використав кожен запит для ідентифікації у криптосистемі. За цим набором даних розв’язуємо задачі навчання прихованої марківської моделі та декодування – знаходимо найімовірнішу послідовність станів прихованого ланцюга, в такий спосіб відновлюючи інформацію про наявність чи відсутність атак на заданому інтервалі часу.

ЛАНЦЮГ МАРКОВА, ПРИХОВАНА МАРКІВСЬКА МОДЕЛЬ, ПУАССОНІВСЬКИЙ ПОТІК

ABSTRACT

Qualification work includes: 67 pages, 10 figures, 9 sources.

The work explores a model for detecting brute-force attacks based on a hidden Markov model to enhance the security of existing cryptographic systems. We believe that the presence or absence of attacks within a certain time interval depends on the state of the hidden Markov chain with two states, whose transition probability matrix is unknown. The observed data consists of the number of requests received by a device in a computer network during a fixed time interval and the number of attempts used by each request to identify itself in the cryptographic system. With this data set, we solve the tasks of training the hidden Markov model and decoding - finding the most probable sequence of states of the hidden chain, thus recovering information about the presence or absence of attacks within the specified time interval.

MARKOV CHAIN, HIDDEN MARKOV MODEL, POISSON PROCESS

ЗМІСТ

ВСТУП

Актуальність дослідження. Незважаючи на повільність та низьку ефективність для сучасних систем шифрування, атаки "грубої сили" залишаються загрозою для конфіденційності інформації в цифрових мережах. Наприклад, через вразливість POODLE [3], зловмисник може отримати додаткову інформацію, що дасть йому можливість застосувати "атаку грубої сили" до SSL [2] протоколу. Тому створення методів виявлення атак даного типу залишається актуальною задачею для реалізацій сучасних криптографічних протоколів та систем.

Метою дослідження є запропонувати метод виявлення атак "грубої сили", що базується на використанні прихованої марківської моделі. А саме, ми припускаємо, що процес запуску/закінчення атаки є автоматизованим і керується прихованим ланцюгом Маркова, який має два стани, і матриця перехідних імовірностей якого нам невідома. Дані, які нам доступні – це кількість спроб ідентифікації, що робить користувач, і момент часу, в який надходить запит від користувача. За цими даними потрібно оцінити параметри прихованого ланцюга, інтенсивності надходжень легітимних і зловмисних запитів та ймовірності вдалої ідентифікації легітимних користувачів та зловмисників.

Задачі дослідження

1) запропонувати приховану марківську модель для ймовірнісного опису процесу надходження та ідентифікації запитів на сервер;

2) в рамках запропонованої моделі побудувати оцінки параметрів ймовірнісних розподілів, якими моделюються спостережені дані;

3) провести чисельні експерименти для перевірки якості побудованих оцінок.

Об'єктом дослідження є прихована марківська модель, що описує процес надходження запитів на сервер.

Предметом дослідження є спосіб виявлення атаки в потоці запитів,

що керується прихованим ланцюгом Маркова.

При розв'язанні поставлених завдань використовувались такі *методи дослідження*: методи теорії імовірностей, математичної статистики, комбінаторного аналізу, теорії складності алгоритмів, методи комп'ютерного та статистичного моделювання

Наукова новизна отриманих результатів полягає в побудові ймовірнісної моделі, в якій надходження зловмисних запитів на сервер керується неспостережуваним ланцюгом Маркова, а кількість спроб, які робить кожен запит для ідентифікації вважається випадковою величиною з розподілом, який залежить від того, яким є запит – легітимним чи зловмисним.

Практичне значення результатів полягає у створенні методу атак грубої сили, а також оцінювання невідомих параметрів, якими характеризується процес надходження та ідентифікації запитів на сервер.

Апробація результатів та публікації. Результати даної роботи було представлено на XXI Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики». (11-12 травня 2023 року, м. Київ, Україна)

1 ПОБУДОВА МАРКІВСЬКОЇ МОДЕЛІ

У цьому розділі розглянуто поняття атаки, наведено приклади найпоширеніших атак і запропоновано ймовірнісну модель для виявлення атак грубої сили.

1.1 Поняття атаки, серверу, запиту

Як зазначається в деяких джерелах, першою кібератакою можна назвати випадок, що стався у Франції 1834 року. Зловмисники змогли отримати доступ до телеграфної мережі, що дозволило їм викрасти важливу фінансову інформацію. Джерел, які могли б остаточно підтвердити чи спростувати цю інформацію знайдено не було, але ми знаємо дату створення першого у світі вірусу – це був вірус під назвою Brain, створений 1986 року. Хоча і до першого вірусу багато теоретиків з кібербезпеки шукали способи захисту від шкідливого програмного забезпечення, але з реальною проблемою на момент виявлення вірусу ніхто не встиг ще зіштовхнутися, тому захист від такої атаки був не надто надійним, через що багато компаній зазнали збитків.

У 90х роках XX століття почався розвиток мережі Інтернет, доступ до неї почав з'являтися не тільки в університетських кампусах і лабораторіях, а й у звичайних користувачів. Через це гостро постало питання безпеки збереження та передачі інформації, як наслідок, розробка методів виявлення кібератак набирає актуальність.

Комп'ютерна мережа(далі мережа) - об'єднання комп'ютерних пристроїв каналами або лініями зв'язку, за допомогою яких є можливість обмінюватися інформацією між пристроями.

Інтернет - мережа мереж або об'єднання мереж в одну мережу.

Оскільки мережа почала розширюватися, то люди почали

створювати пристрої, які повинні обробляти, зберігати і надавати інформацію іншим користувачам мережі. Такий пристрів отримав назву **сервер**.

З появою серверів виникла потреба в обмеженні доступу до інформації, що знаходиться на сервері. Так виникли системи авторизації для мереж і протоколи підключення або запитів.

Підключення або запит - певний алгоритм, який дозволяє отримувати інформацію з сервера і підтримувати зв'зок з сервером.

Також почали розділяти легальні підключення(запити) від зловмисних.

Легітимний запит - запланований запит, який використовується користувачем, щоб здобути інформацію на яку він має право або така дія, що не спричиняє серверу шкоди.

Зловмисний запит - запит, який використовується користувачем-зловмисником, щоб здобути інформацію, яка йому не належить, або дія, яка спричиняє шкоду серверу.

Знаючи поняття запитів, можна дати ще одне тлумачення поняттю сервер, саме ним ми і будемо користуватися в даній роботі.

Сервер - пристрій, який має можливість приймати запити, має критерії для розпізнавання легального і зловмисного запиту. Такі критерії можуть бути побудовані спеціалістом, який зможе вказати на факт зловмисності запиту.

Велику кількість одночасних зловмисних запитів на сервер будемо називати **атакою**.

Наразі, найбільш поширеними кібератаками є:

«Фішингові атаки» – полягають у здобутті доступу шляхом обману людини, тобто фактично людина сама віддає доступ зловмиснику.

« Атака "людина посередині" »("Man in the middle") – полягає в тому, що зловмисник перехоплює інформацію, "стаючи" між двома користувачами, що обмінюються інформацією. Ця атака в зв'язку з переходом до віддаленої роботи в більшості сфер стає все більш

небезпечною.

«DDOS-атака» – полягає в направленні зловмисником великих обсягів інформації з метою перенавантажити пристрій, що атакується, і вивести його з ладу.

А також, хоч і не дуже ефективна, але легка в реалізації через що й поширена – **« Атака "грубої сили" »(«Brute-force attack»)**. Дана атака полягає у переборі всіх можливих варіантів ідентифікації, щоб здобути доступ до даних. Саме такий тип атаки ми і будемо розглядати в цій роботі.

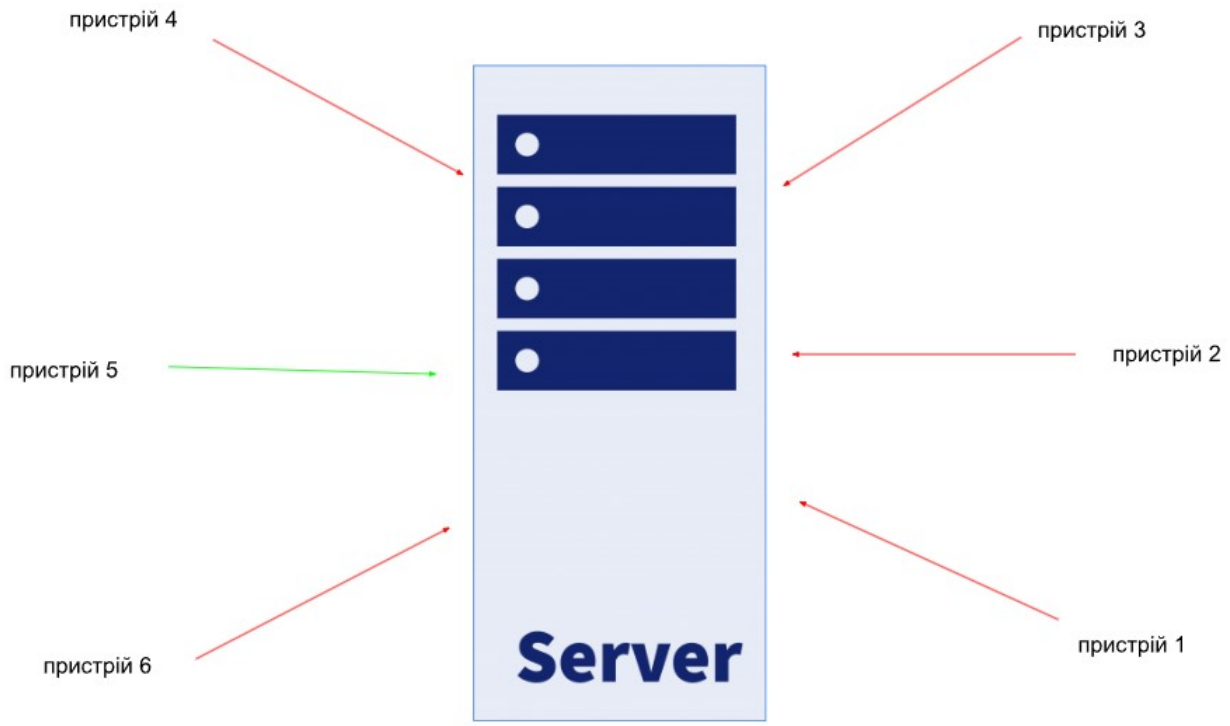

Рисунок 1.1 - Загальна схема підключень до уявного сервера

Існує чимало робіт присвячених застосуванню прихованих марківських моделей до виявлення атак різного роду. Зокрема, «Hidden Markov Model Modeling of SSH Brute-Force Attacks»[7], «A hidden markov model based framework for tracking and predicting of attack intention»[8], «A finite state hidden markov model for predicting multistage attacks in cloud systems»[1], «Real-time multistep attack prediction based on hidden markov models»[6]. У статті [7] розглядають застосування ПММ до атаки грубої

сили, проте в ній не передбачають оцінювання невідомих параметрів за набором спостережень, натомість параметри прихованого ланцюга задаються вручну.

1.2 Побудова прихованої марківської моделі

Ми припускаємо, що надходження легітимних запитів на сервер утворює стаціонарний пуассонівський потік [9] з інтенсивністю λ_0. Це означає, що кількість підключень N_t^0, які надходять в довільному інтервалі часу $[t, t+1)$ має розподіл Пуассона з параметром λ_0:

$$P(N_t^0 = K) = \frac{\lambda_0^K}{K!} \cdot e^{-\lambda_0}, K = 0, 1, 2, \dots \tag{1.1}$$

Крім того, якщо відомо, що на $[t, t+1)$ надійшло K підключень, то моменти їх надходження є незалежними, рівномірно розподіленими на $[t, t+1)$ випадковими величинами.

Кожен користувач, який надсилає запит, має щонайбільше M спроб для ідентифікації. Ми припускаємо, що кількість спроб, які знадобляться легітимному користувачу для ідентифікації має зрізаний геометричний розподіл з параметрами M, p_0, тобто

$$q_0(i) = (1-p_0)^{i-1} \cdot p_0^{\mathbb{1}(i \leqslant M-1)}, i = 1, 2, \dots, M, \tag{1.2}$$

де $q_0(i)$ – ймовірність того, що легітимному користувачу вдалося ідентифікуватися з i-ї спроби.

Далі припустимо, що в кожному інтервалі часу $[t, t+1)$ на сервер може бути скоєна атака, тобто надходять не лише легітимні, а й зловмисні запити. Наявність чи відсутність атаки визначається станом ланцюга Маркова $(X_t)_{t \geqslant 0}$ з двома станами $\{0, 1\}$: якщо $X_t = 0$, то на $[t, t+1)$ надходять лише легітимні запити, якщо ж $X_t = 1$, то на сервер здійснюється атака і крім легітимних запитів надходять ще й зловмисні.

Ми вважаємо, що при $X_t = 1$ кількість N_t^1 зловмисних підключень,

які надходять в проміжку $[t, t+1)$ має розподіл Пуассона з параметром λ_1:

$$P(N_t^1 = K) = \frac{\lambda_1^K}{K!} \cdot e^{-\lambda_1}, K = 0, 1, \ldots \tag{1.3}$$

і що моменти надходження цих запитів є незалежними рівномірно розподіленими на $[t, t+1)$ випадковими величинами. Кожному зловмисному запиту вдається ідентифікуватися з i-ї спроби з ймовірністю:

$$q_1(i) = (1-p_1)^{i-1} \cdot p_1^{\mathbb{1}(i \leqslant M-1)}, i = 1, 2, \ldots, M, \tag{1.4}$$

де $p_1 \ll p_0$.

Позначимо через N_t кількість запитів, що надходять в інтервалі часу $[t, t+1)$. Згідно зроблених вище припущень

$$N_t = N_t^0 + \mathbb{1}(X_t = 1) \cdot N_t^1 \tag{1.5}$$

З (1.5) випливає, що

$$P(N_t = K | X_t = 0) = P(N_t^0 = K) = \frac{\lambda_0^K}{K!} \cdot e^{-\lambda_0},$$
$$K = 0, 1, 2, \ldots$$
$$P(N_t = K | X_t = 1) = P(N_t^0 + N_t^1 = K) =$$
$$= \frac{(\lambda_0 + \lambda_1)^K}{K!} e^{-(\lambda_0+\lambda_1)}, K = 0, 1, \ldots \tag{1.6}$$

Позначимо далі через $Y^t = (Y_1^t, Y_2^t, \ldots)$ вектор, координата Y_r^t якого є кількістю спроб, яка знадобилася r-му запиту, що надійшов на $[t, t+1)$, для ідентифікації. Ці випадкові величини є незалежними. Оскільки при $X_t = 0$ всі запити є легітимними, то умовний розподіл вектора Y^t при

умові $X_t = 0, N_t = K$ є такими:

$$P(Y_1^t = y_1, \ldots, Y_K^t = y_K | X_t = 0, N_t = K) = \\ = \prod_{i=1}^{K} P(Y_i^t = y_i | X_t = 0, N_t = K) = \\ = \prod_{i=1}^{K} q_0(y_i) = [q_0(1)]^{K_1} \cdot \ldots \cdot [q_0(M)]^{K_M}, \quad (1.7)$$

де $1 \leqslant y_i \leqslant M$, $K_i = \sum_{r=1}^{K} \mathbb{1}(y_r = i)$ – кількість запитів, які використали i спроб для ідентифікації, а ймовірності $q_0(i)$ задано у (1.2).

Щоб знайти умовний розподіл Y^t при умові $X_t = 1, N_t = K$, введемо в розгляд такі випадкові величини:

$$Z_i^t = \mathbb{1}(\text{і-ий запит на } [t,t+1) \text{ є зловмисним}) = \\ = \begin{cases} 0, & \text{і-ий запит є легітимним,} \\ 1, & \text{і-ий запит є зловмисним.} \end{cases}$$

Зауважимо, що ці величини є неспостережуваними (прихованими, латентними). Величину Z_i^t назвемо статусом (легітимний/зловмисний) i-го запиту на $[t, t+1)$. Зрозуміло, що при відомому статусі $Z_i^t = j$ маємо:

$$P(Y_i^t = y | X_t = 1, N_t = K, Z_i^t = j) = [q_0(y)]^{\mathbb{1}(j=0)} \cdot [q_1(y)]^{\mathbb{1}(j=1)} = \\ = q_j(y), \quad (1.8)$$

і тоді

$$P(Y_i^t = y | X_t = 1, N_t = K) = \\ = P(Z_i^t = 0 | X_t = 1, N_t = K) \cdot q_0(y) + \\ + P(Z_i^t = 1 | X_t = 1, N_t = K) \cdot q_1(y) \quad (1.9)$$

Переконаймося, що для довільного $i = \overline{1, K}$

$$P(Z_i^t = 0 | X_t = 1, N_t = K) = \frac{\lambda_0}{\lambda_0 + \lambda_1},$$
$$P(Z_i^t = 1 | X_t = 1, N_t = K) = \frac{\lambda_1}{\lambda_0 + \lambda_1}. \quad (1.10)$$

Справді, зі зроблених вище припущень випливає, що кількість зловмисних запитів $\sum_{i=1}^{K} Z_i^t$ на $[t, t+1)$ має біноміальний розподіл $Bin(K, \frac{\lambda_1}{\lambda_0 + \lambda_1})$:

$$P(\sum_{i=1}^{K} Z_i^t = m | X_t = 1, N_t = K) = P(N_t^1 = m | X_t = 1, N_t = K) =$$
$$= \frac{P(N_t^1 = m, N_t^0 = k - m | X_t = 1)}{P(N_t = K | X_t = 1)} =$$

$$= \frac{\frac{(\lambda_1)^m}{m!} e^{-\lambda_1} \cdot \frac{(\lambda_0)^{K-m}}{(K-m)!} e^{-\lambda_0}}{\frac{(\lambda_0 + \lambda_1)^K}{K!} e^{-(\lambda_0 + \lambda_1)}} = C_K^m \left(\frac{\lambda_1}{\lambda_0 + \lambda_1} \right)^m \cdot \left(\frac{\lambda_0}{\lambda_0 + \lambda_1} \right)^{k-m}, m = \overline{0, K}$$

З того, що умовний розподіл моментів надходження запитів при заданій кількості є рівномірним випливає, що вектори $(Z_1^t, \ldots, Z_K^t)$ з однаковою сумою координат m мають мати однакову ймовірність, тобто

$$P(Z_1^t = j_1, \ldots, Z_K^t = j_K) = \frac{P(\sum_{i=1}^{K} Z_i^t = m)}{C_K^m} = \left(\frac{\lambda_1}{\lambda_0 + \lambda_1} \right)^m \cdot \left(\frac{\lambda_0}{\lambda_0 + \lambda_1} \right)^{k-m}.$$

Звідси отримуємо, що для довільного $r = \overline{1,K}$:

$$P(Z_r^t = 1) = \sum_{j_1,\dots,j_K, j_i \in \{0,1\}} P(Z_1^t = j_1, \dots, Z_r^t = 1, \dots, Z_K^t = j_K) =$$

$$= \frac{P\left(\sum_{i=1}^{K} Z_i^t = 1 + \sum_{i \neq r} j_i\right)}{\sum_{C_K^{i \neq r}} j_i + 1} = \sum_{m=0}^{K-1} C_{K-1}^m \cdot \frac{P\left(\sum_{i=1}^{K} Z_i^t = 1 + m\right)}{C_K^{m+1}} =$$

$$= \sum_{m=0}^{K-1} C_{K-1}^m \cdot \left(\frac{\lambda_1}{\lambda_0 + \lambda_1}\right)^{m+1} \left(\frac{\lambda_0}{\lambda_0 + \lambda_1}\right)^{K-m-1} =$$

$$= \frac{\lambda_1}{\lambda_0 + \lambda_1} \cdot \sum_{m=0}^{K-1} C_{K-1}^m \left(\frac{\lambda_1}{\lambda_0 + \lambda_1}\right)^{m} \left(\frac{\lambda_0}{\lambda_0 + \lambda_1}\right)^{K-1-m} = \frac{\lambda_1}{\lambda_0 + \lambda_1},$$

Оскільки $\sum_{m=0}^{K-1} C_{K-1}^m \left(\frac{\lambda_1}{\lambda_0 + \lambda_1}\right)^{m} \left(\frac{\lambda_0}{\lambda_0 + \lambda_1}\right)^{K-1-m} = 1.$

Таким чином при наявності атаки, умовний розподіл кількостей спроб ідентифікації запитів має вигляд

$$P(Y_1^t = y_1, \dots, Y_K^t = y_K | X_t = 1, N_t = K) = \prod_{r=1}^{K} P(Y_r^t = y_r | X_t = 1, N_t = k) =$$

$$= \prod_{r=1}^{K} \left[\frac{\lambda_0}{\lambda_0 + \lambda_1} \cdot q_0(y_r) + \frac{\lambda_1}{\lambda_0 + \lambda_1} \cdot q_1(y_r)\right] = \prod_{i=1}^{M} \left[\frac{\lambda_0}{\lambda_0 + \lambda_1} q_0(i) + \frac{\lambda_1}{\lambda_0 + \lambda_1} q_1(i)\right]^{K_i}, \tag{1.11}$$

де K_i – кількість запитів на $[t, t+1)$, які зробили i спроб для ідентифікації. Оскільки при $X_t = 0$ всі запити на $[t, t+1)$ є легітимними, то умовний розподіл вектора статусів запитів Z^t при умові $X_t = 0$ є таким:

$$P(Z_1^t = j_1, \dots, Z_K^t = j_K | X_t = 0, N_t = K) = \mathbb{1}(\sum_{r=1}^{K} j_r = 0), j_r \in \{0, 1\} \tag{1.12}$$

Тепер ми можемо записати умовний розподіл набору випадкових величин (N_t, Y^t) (N_t - кількість підключень на $[t, t+1)$); $Y^t = (Y_1^t, Y_2^t, \dots)$ –

кількість спроб ідентифікуватися, які використало кожне підключення) при заданому стані X_t ланцюга.

$$P\left((K; y_1, \ldots, y_K)|X_t = 0\right) = P(N_t = K|X_t = 0)\cdot$$
$$\cdot P(Y_1^t = y_1, \ldots, Y_K^t = y_K|X_t = 0, N_t = K) =$$
$$= \frac{\lambda_0^K}{K!} \cdot e^{-\lambda_0} \cdot \prod_{r=1}^{K} q_0(y_r) \equiv B_0(t),$$
$$K \geqslant 0; 1 \leqslant y_i \leqslant M \quad (1.13)$$

$$P\left((K; y_1, \ldots, y_K)|X_t = 1\right) =$$
$$= \frac{(\lambda_0 + \lambda_1)^K}{K!} e^{-(\lambda_0+\lambda_1)} \times$$
$$\times \prod_{r=1}^{K} \left[\frac{\lambda_0}{\lambda_0 + \lambda_1} \cdot q_0(y_r) + \frac{\lambda_1}{\lambda_0 + \lambda_1} \cdot q_1(y_r)\right] =$$
$$= \frac{e^{-(\lambda_0+\lambda_1)}}{K!} \cdot \prod_{r=1}^{K} \left[\lambda_0 \cdot q_0(y_r) + \lambda_1 \cdot q_1(y_r)\right] \equiv B_1(t),$$
$$K \geqslant 0; 1 \leqslant y_i \leqslant M \quad (1.14)$$

Підсумовуючи, ми побудували приховану марківську модель [4] $(X_t, (N_t, Y^t))_{t=0,1,2,\ldots}$, в якій:

1) $(X_t)_{t\geqslant 0}$ є ланцюгом Маркова на множині станів $E = \{0, 1\}$, з початковим розподілом $\mu = (\mu_0, \mu_1)$ та матрицею перехідних імовірностей $A = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix}$

2) Випадкові вектори $(N_0, Y^0), \ldots, (N_t, Y^t)$ є умовно незалежними при заданих $X_0, \ldots, X_t$ і $P(N_t, Y^t|X_0, \ldots, X_t) = P(N_t, Y^t|X_t)$,

3) Для кожного $t = 0, 1, 2, \ldots$ умовний розподіл $P(N_t, Y^t|X_t)$ задається формулами (1.13), (1.14). Цю модель надалі позначатимемо $\phi = (\mu, A; \lambda_0, \lambda_1; p_0, p_1)$.

Висновки до розділу 1

В цьому розділі було розглянуто основні поняття, які будуть використовуватися в роботі; зроблено огляд найпоширеніших атак та запропоновано імовірнісну модель , в якій надходження зловмисних запитів на сервер керується неспостережуваним ланцюгом Маркова, а кількість спроб, які робить кожен запит для ідентифікації, вважається випадковою величиною з розподілом, який залежить від того, яким є запит – легітимним чи зловмисним.

2 ОЦІНКА ПАРАМЕТРІВ ПРИХОВАНОЇ МАРКОВСЬКОЇ МОДЕЛІ

У даному розділі будуть виведені формули для оцінки невідомих параметрів марківської моделі, запропонованої в попередньому розділі.

2.1 Ітераційний алгоритм знаходження оцінки максимальної правдоподібності

Оптимальну модель $\phi^* = (\mu^*, A^*, \lambda_0^*, \lambda_1^*, p_0^*, p_1^*)$ шукатимемо як оцінку максимальної правдоподібності

$$\phi^* = \underset{\phi}{argmax} P_\phi(O_0, \dots, O_T) \tag{2.1}$$

тобто таку, що максимізує ймовірнсіть отримання спостережуваного набору даних $O_0, \dots O_T$. Функція правдоподібності $P_\phi(O_0, \dots, O_T)$ має наступний вигляд:

$$P_\phi(O_0, \dots, O_T) = \sum_{x=(x_0,\dots,x_T)} P_\phi(O_0, \dots, O_T, X_0 = x_0, \dots X_T = x_T)$$

і знаходження аргумента ϕ, при якому досягається її глобальний максимум, є складною задачею. Як правило для її вирішення користуються ітераційним EM-алгоритмом, суть якого полягає в наступному: задають деяке початкове наближення $\phi^{(0)}$, а кожне наступне наближення $\phi^{(n+1)}$, $n \geqslant 0$ знаходяться як аргумент, при якому досягається максимум так званої квазілогарифмічної функції правдоподібності:

$$\phi^{(n+1)} = \underset{\phi}{argmax} Q(\phi^{(n)}, \phi), \tag{2.2}$$

де

$$Q(\phi^{(n)}, \phi) = \sum_{x=(x_0,\ldots,x_T)} P_{\phi^{(n)}}(O_0, \ldots, O_T, x_0, \ldots, x_T) \cdot \ln P_\phi(O_0, \ldots, O_T, x_0, \ldots, x_T).$$

Цей метод базується на спостереженні, що якщо для деяких двох моделей ϕ_1, ϕ_2 виконується

$$P_{\phi_2}(O_0, \ldots, O_T) \geqslant P_{\phi_1}(O_0, \ldots, O_T),$$

то

$$\ln \frac{P_{\phi_2}(O_0, \ldots, O_T)}{P_{\phi_1}(O_0, \ldots, O_T)} \geqslant 0,$$

а для логарифма в лівій частині справедлива наступна нерівність[4]:

$$\ln \frac{P_{\phi_2}(O_0, \ldots, O_T)}{P_{\phi_1}(O_0, \ldots, O_T)} \geqslant \frac{Q(\phi_1, \phi_2) - Q(\phi_1, \phi_1)}{P_{\phi_1}(O_0, \ldots, O_T)}$$

Отже, максимізуючи функцію $Q(\phi_1, \phi)$ за другим аргументом при заданому першому, ми отримуємо покращену модель, для якої

$$P_\phi(O_0, \ldots, O_T) \geqslant P_{\phi_1}(O_0, \ldots, O_T)$$

Описаний ітераційний алгоритм є збіжним і дозволяє знайти аргумент, при якому досягається локальний максимум функції правдоподібності.

2.2 Функція правдоподібності спостережених даних

Припустимо, що в результаті спостережень за надходженням запитів на сервер упродовж часу $[0,T)$ отримано дані $(K_0, y^0), \ldots, (K_T, y^T)$, де для кожного $t = \overline{0,T}$ K_t – це спостережена в інтервалі часу $[t, t+1)$ кількість запитів; $y^t = (y_1^t, \ldots, y_{K_t}^t)$, y_r^t – номер спроби, з якої r-ому запиту вдалося ідентифікуватися. Для скорочення запису позначимо набір (K_t, y^t) символом O_t.

Запишемо функцію правдоподібності $L_{\phi,X,Z}$ спостережених даних,

доповнених ланцюжком $x = (x_0, \ldots, x_T)$ станів прихованого ланцюга Маркова та набором векторів $z = (z^0, \ldots, z^T)$ латентних статусів запитів.

$$L_{\phi,X,Z} \equiv P_\phi(O_0, \ldots, O_T; z^0, \ldots, z^T; x_0, \ldots, x_T) = \\ = P_\phi(x_0, \ldots, x_T) \cdot \prod_{t=0}^{T} P_\phi(O_t, Z^t | x_t)$$

Запишемо окремо множники в правій частині:

$$P_\phi(x_0, \ldots, x_T) = \mu_{x_0} A_{x_0 x_1} \cdot \ldots \cdot A_{x_{T-1} x_T} = \\ = \prod_{i=0,1} \mu_i^{\mathbb{1}(x_0=i)} \cdot \prod_{i,j=0}^{1} A_{ij}^{\sum_{t=0}^{T-1} \mathbb{1}(x_t = i, x_{t+1} = j)}.$$

Далі

$$P_\phi(O_t, z^t | x_t) = [P_M(O^t, z^t | X_t = 0)]^{\mathbb{1}(x_t=0)} \cdot [P_M(O^t, z^t | X_t = 1)]^{\mathbb{1}(x_t=1)}$$

. Використовуючи формули з попереднього пункту, і позначивши $J_t = \sum_{i=1}^{K_t} z_i^t$, знаходимо:

$$P_\phi(O^t, Z^t | X_t = 0) = \frac{(\lambda_0)^{K_t}}{K_t!} e^{-\lambda_0} \cdot \mathbb{1}(J_t = 0) \cdot \prod_{r=1}^{K_t} q_0(y_r^t) = \\ = \mathbb{1}(J_t = 0) \frac{(\lambda_0)^{K_t}}{K_t!} e^{-\lambda_0} \cdot (1 - p_0)^{\sum_{r=1}^{K_t} (y_r^t - 1)} \cdot p_0^{\sum_{r=1}^{K_t} \mathbb{1}(y_r^t \leqslant M-1)}.$$

$$P_\phi\left(O^t, Z^t | X_t = 1\right) = \frac{(\lambda_0+\lambda_1)^{K_t}}{K_t!} \cdot e^{-(\lambda_0+\lambda_1)} \cdot$$
$$\cdot \prod_{r=1}^{K_t} \left(\frac{\lambda_1}{\lambda_0+\lambda_1}\right)^{\mathbb{1}(z^t_{K_t}=1)} \cdot \left(\frac{\lambda_0}{\lambda_0+\lambda_1}\right)^{\mathbb{1}(z^t_{K_t}=0)} \cdot$$
$$\cdot \prod_{r=1}^{K_t} \left(q_0(y^t_r)\right)^{\mathbb{1}(z^t_r=0)} \left(q_1(y^t_r)\right)^{\mathbb{1}(z^t_r=1)} =$$
$$= \frac{e^{-(\lambda_0+\lambda_1)}}{K_t!} \cdot \prod_{r=1}^{K_t} \left(\lambda_0 \cdot q_0(y^t_r)\right)^{\mathbb{1}(z^t_r=0)} \left(\lambda_1 \cdot q_1(y^t_r)\right)^{\mathbb{1}(z^t_r=1)}.$$

Запишемо логарифм функції правдоподібності $L_{\phi,X,Z}$, позначивши

$$R_t = \sum_{r=1}^{K_t}(y^t_r - 1), F_t = \sum_{r=1}^{K_t} \mathbb{1}(y^t_r \leqslant M-1) :$$

$$\ln L_{\phi,X,Z} = \sum_{i=0,1} \ln \mu_i \cdot \mathbb{1}\left(x_0 = i\right) +$$
$$+ \sum_{i,j=0}^{1} \ln A_{ij} \cdot \sum_{t=0}^{T-1} \mathbb{1}\left(x_t = i; x_{t+1} = j\right) +$$
$$+ \sum_{t=0}^{T} \mathbb{1}\left(x_t = 0\right) \cdot \left[k_t \ln \lambda_0 - \lambda_0 + R_t \cdot \ln\left(1-p_0\right) + F_t \ln p_0\right] +$$
$$+ \sum_{t=0}^{T} \mathbb{1}\left(x_t = 1\right) \left[-\lambda_0 - \lambda_1 + \left(J_t - 1\right) \ln \lambda_1 + \left(k_t - J_t\right) \ln \lambda_0 +\right.$$
$$+ \sum_{r=1}^{k_t} \mathbb{1}\left(z^t_r = 0\right) \left(\left(y^t_r - 1\right) \ln\left(1-p_0\right) + \mathbb{1}\left(y^t_r \leqslant M-1\right) \ln p_0\right) +$$
$$\left. + \sum_{i=1}^{k_t} \mathbb{1}\left(z^t_r = 1\right) \cdot \left(\left(y^t_r - 1\right) \ln\left(1-p_1\right) + \mathbb{1}\left(y^t_r \leqslant M-1\right) \ln p_1\right)\right]$$

Зібравши коефіцієнти біля $\ln\lambda_0$, $\ln\lambda_1$, λ_0, λ_1, $\ln(1-p_0)$, $\ln p_0$, $\ln(1-p_1)$, $\ln p_1$,

отримуємо:

$$\ln L_{\phi,X,Z} = \sum_{i=0}^{1} \ln \mu_i \cdot \mathbb{1}\left(x_0 = i\right) + \sum_{i,j=0}^{1} \ln A_{ij} \cdot$$
$$\cdot \sum_{t=0}^{T-1} \mathbb{1}\left(x_t = i, x_{t+1} = j\right) + \ln \lambda_0 \left[\sum_{t=0}^{T} k_t - \sum_{t=0}^{T} \mathbb{1}\left(x_t = 1\right) \cdot J_t\right] - \lambda_0 \cdot (T+1) +$$
$$+ \ln \lambda_1 \cdot \sum_{t=0}^{T} \mathbb{1}\left(x_t = 1\right)\left(J_t - 1\right) - \lambda_1 \cdot \sum_{t=0}^{T} \mathbb{1}\left(x_t = 1\right) +$$
$$+ \ln\left(1 - p_0\right) \cdot \sum_{t=0}^{T} \left(\mathbb{1}\left(x_t = 0\right) \cdot R_t + \mathbb{1}\left(x_t = 1\right) \sum_{r=1}^{K_t} \mathbb{1}\left(z_r^t = 0\right)\left(y_r^t - 1\right) \right) +$$
$$+ \ln p_0 \cdot \sum_{t=0}^{T} \left(\mathbb{1}\left(x_t = 0\right) \cdot F_t + \mathbb{1}\left(x_t = 1\right) \sum_{r=1}^{K_t} \mathbb{1}\left(z_r^t = 0\right) \mathbb{1}\left(y_r^t \leqslant M - 1\right) \right) +$$
$$+ \ln\left(1 - p_1\right) \cdot \sum_{t=0}^{T} \mathbb{1}\left(x_t = 1\right) \sum_{z=1}^{K_t} \mathbb{1}\left(z_r^t = 1\right)\left(y_z^t - 1\right) +$$
$$+ \ln p_1 \cdot \sum_{t=0}^{T} \mathbb{1}\left(x_t = 1\right) \sum_{z=1}^{K_t} \mathbb{1}\left(z_z^t = 1\right) \mathbb{1}\left(y_2^t \leqslant M - 1\right).$$

Запишемо тепер квазілогарифмічну функцію правдоподібності

$$Q(\phi, \phi^*) = \sum_{X,Z} L_{\phi,X,Z} \cdot \ln L_{\phi^*,X,Z},$$

де ϕ – поточна модель.

$$Q\left(\phi, \phi^*\right) = \sum_{i=0,1} \ln \mu_i^* \cdot P_\phi\left(O_0, \ldots, O_T, X_0 = i\right) +$$
$$+ \sum_{i,j=0,1} \ln A_{ij}^* \cdot \sum_{t=0}^{T-1} P_\phi\left(O_0, \ldots, O_T, X_t = i, X_{t+1} = j\right) +$$
$$+ \ln \lambda_0^* \cdot \left(P_\phi\left(O_0, \ldots, O_T\right) \cdot \sum_{t=0}^{T} K_t - \sum_{t=0}^{T} \sum_{r=1}^{K_t} P_\phi\left(O_0, \ldots, O_T, X_t = 1, Z_r^t\right) \right) -$$
$$- \lambda_0^* \cdot (T+1) \cdot P_\phi\left(O_0, \ldots, O_T\right) - \lambda_1^* \cdot \sum_{t=0}^{T} P_\phi\left(O_0, \ldots, O_T, X_t = 1\right) +$$
$$+ \ln \lambda_1^* \cdot \sum_{t=0}^{T} \sum_{r=1}^{K_t} P_\phi\left(O_0, \ldots, O_T, X_t = 1, z_r^t = 1\right) +$$

$$+\ln(1-p_0^*)\left[P_\phi(O_0,\ldots,O_T)\cdot\sum_{t=0}^{T}R_t-\right.$$
$$\left.-\sum_{t=0}^{T}\sum_{r=1}^{K_t}P_\phi\left(O_0,\ldots,O_T,X_t=1,Z_r^t=1\right)\left(y_r^t-1\right)\right]+$$
$$+\ln p_0^*\left[P_\phi(O_0,\ldots,O_T)\sum_{t=0}^{T}F_t-\right.$$
$$\left.-\sum_{t=0}^{T}\sum_{r=1}^{k_t}P_\phi\left(O_0,\ldots,O_T,X_t=1,Z_r^t=1\right)\cdot\mathbb{1}\left(y_r^t\leqslant M-1\right)\right]+$$
$$+\ln(1-p_1^*)\cdot\sum_{t=0}^{T}\sum_{r=1}^{K_t}P_\phi\big(O_0,\ldots,O_T,X_t=1,Z_r^t=1\big)\left(y_r^t-1\right)+$$
$$+\ln p_1^*\cdot\sum_{t=0}^{T}\sum_{r=1}^{K_t}P_\phi\big(O_0,\ldots,O_T,X_t=1,z_r^t=1\big)\mathbb{1}\left(y_r^t\leqslant M-1\right).$$

Диференціюючи $Q(\phi,\phi^*)$ за μ^*, A^*, λ_0^*, λ_1^*, p_0^*, p_1^* і прирівнюючи похідні до нуля, отримуємо покращені параметри:

$$\mu_0^*=\frac{P_\phi(O_0,\ldots,O_T,X_0=0)}{P_\phi(O_0,\ldots,O_T)},\qquad \mu_1^*=\frac{P_\phi(O_0,\ldots,O_T,X_0=1)}{P_\phi(O_0,\ldots,O_T)};$$

$$A_{ij}^*=\frac{\sum_{t=0}^{T-1}P_\phi(O_0,\ldots,O_T,X_t=i,X_{t+1}=j)}{\sum_{t=0}^{T-1}P_\phi(O_0,\ldots,O_T,X_t=i)}$$

$$\lambda_0^*=\frac{\sum_{t=0}^{T}K_t}{T+1}-\frac{\sum_{t=0}^{T}\sum_{r=1}^{K_t}P_\phi\left(O_0,\ldots,O_T,X_t=1,Z_r^t=1\right)}{(T+1)P_\phi(O_0,\ldots,O_T)}$$

$$\lambda_1^*=\frac{\sum_{t=0}^{T}\sum_{r=1}^{K_t}P_\phi\left(O_0,\ldots,O_T,X_t=1,Z_r^t=1\right)}{\sum_{t=0}^{T}P_\phi(O_0,\ldots,O_T,X_t=1)}$$

$$p_0^*=\frac{P_\phi(O_0,\ldots,O_T)\cdot\sum_{t=0}^{T}F_t-\sum_{t=0}^{T}\sum_{r=1}^{K_t}P_\phi\left(O_0,\ldots,O_T,X_t=1,Z_r^t=1\right)\cdot\mathbb{1}\left(y_r^t\leqslant M-1\right)}{P_\phi(O_0,\ldots,O_T)\sum_{t=0}^{T}(R_t+F_t)-\sum_{t=0}^{T}\sum_{r=1}^{K_t}P_\phi\left(O_0,\ldots,O_T,X_t=1,Z_r^t=1\right)\left(y_r^t-1+\mathbb{1}\left(y_r^t\leqslant M-1\right)\right)}$$

$$p_1^* = \frac{\sum\limits_{t=0}^{T}\sum\limits_{r=1}^{K_t} P_\phi\left(O_0,\ldots,O_T, x_t = 1, Z_r^t = 1\right)\mathbb{1}\left(y_r^t \leqslant M-1\right)}{\sum\limits_{t=0}^{T}\sum\limits_{r=1}^{K_t} P_\phi\left(O_0,\ldots,O_T, x_t = 1, Z_r^t = 1\right)\left(y_r^t - 1 + \mathbb{1}\left(y_r^t \leqslant M-1\right)\right)}$$

Дані оцінки можна обчислити за допомогою змінних прямого і зворотного ходу.

2.3 Змінні прямого і зворотного ходу

Позначимо

$$\alpha_t(i) = P_\phi(O_0,\ldots,O_t, X_t = i), t = \overline{0,T}; i = 0,1$$

$$\beta_t(i) = P_\phi(O_{t+1},\ldots,O_T | X_t = i), t = \overline{0,T-1}; i = 0,1$$

Ці змінні обчислюються за рекурентними співвідношеннями

$$\alpha_{t+1}(i) = \sum_{j=1}^{2} \alpha_t(j) A_{ji} B_i(t+1)$$

$$\beta_t(i) = \sum_{j=1}^{2} A_{ij} \cdot B_j(t+1) \cdot \beta_{t+1}(j)$$

В термінах цих змінних можна записати

$$P_\phi(O_0,\dots,O_T) = \alpha_T(0) + \alpha_T(1)$$

$$P_\phi(O_0,\dots,O_T, X_t = i) = \alpha_t(i)\cdot\beta_t(i), t = \overline{0,T}$$

$$P_\phi(O_0,\dots,O_T, X_t = i, X_{t+1} = j) = \alpha_t(i)\cdot A_{ij}\cdot B_j(t)\cdot\beta_{t+1}(j)$$

$$P_\phi(O_0,\dots,O_T, X_t = 1, Z_r^t = 1) =$$

$$= \sum_{i=0,1} P_\phi(O_0,\dots,O_{t-1}, X_{t-1} = i, X_t = 1, Z_r^t = 1, O_t, O_{t+1},\dots,O_T) =$$

$$= \sum_{i=0,1} P_\phi(O_0,\dots,O_{t-1}, X_{t-1} = i)\cdot$$

$$\cdot P(X_t = 1|X_{t-1} = i)\cdot P(O_t, Z_r^t = 1|X_t = 1)\cdot$$

$$\cdot P(O_{t+1},\dots,O_T|X_t = 1) =$$

$$= \sum_{i=0,1} \alpha_{t-1}(i)\cdot A_{i1}\cdot P(O^t, Z_r^t = 1|X_t = 1)\cdot\beta_t(1),$$

де $B_j(t+1) = P(O_{t+1}|X_{t+1} = j)$ задається в (1.13), (1.14). Знайдемо $P(O_t, Z_r^t = 1|X_t = 1)$. Використовуючи знайдені умовні розподіли в п. 1, можемо записати:

$$P(O_t, Z_r^t = 1|X_t = 1) = \frac{(\lambda_0+\lambda_1)^{K_t}}{K_t!}e^{-(\lambda_0+\lambda_1)}\cdot\frac{\lambda_1}{\lambda_0+\lambda_1}q_1(y_r^t)\cdot$$

$$\cdot\prod_{m\neq r}\left[\frac{\lambda_0}{\lambda_0+\lambda_1}\cdot q_0(y_m^t) + \frac{\lambda_1}{\lambda_0+\lambda_1}\cdot q_1(y_m^t)\right] =$$

$$= \frac{e^{-(\lambda_0+\lambda_1)}}{K_t!}\cdot[\lambda_1 q_1(y_r)]\prod_{m\neq r}[\lambda_0 q_0(y_m) + \lambda_1 q_1(y_m)].$$

Таким чином ми побудували оцінки для невідомих параметрів прихованого ланцюга, а також оцінки інтенсивності надходження запитів та ймовірностей вдалої ідентифікації запитів на сервер.

2.4 Алгоритм Вітербі. Декодування ланцюга

Після отримання оцінки параметрів прихованої марковської моделі перейдемо до задачі декодування. Нашою метою є дізнатися, на яких проміжках з велкиою ймовірністю була атака. Для цього застосуємо алгоритм Вітербі [5]. Результатом виконання даного алгоритму буде розв'язок такої задачі: $(\widehat{X_0}, \ldots, \widehat{X_T}) = \underset{x}{argmax} P_\phi(X_0, \ldots, X_T | O_0, \ldots, O_T)$. За отриманою оцінкою ми і зможемо зробити висновки про те, коли з великою ймовірністю відбувалися атаки: якщо $\widehat{X_t} = 1$, то на $[t, t+1)$ була атака, якщо $\widehat{X_t} = 0$, то на $[t, t+1)$ не було атаки.

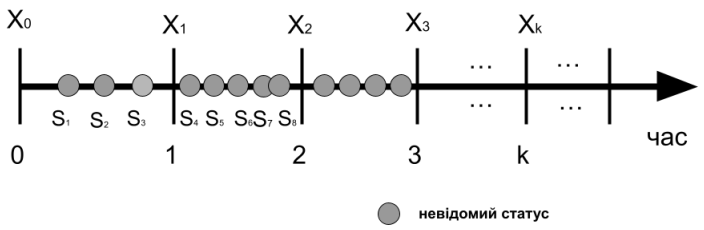

Рисунок 2.1 – Вигляд прямої часу з нанесеними подіями - запитами до сервера, статус яких невідомий

Висновки до розділу 2

У даному розділі було виведено всі необхідні оцінки для параметрів моделі. Описано ітераційний алгоритм знаходження оцінки максимальної правдоподібності і як наслідок декодували ланцюг. За оціненою моделлю

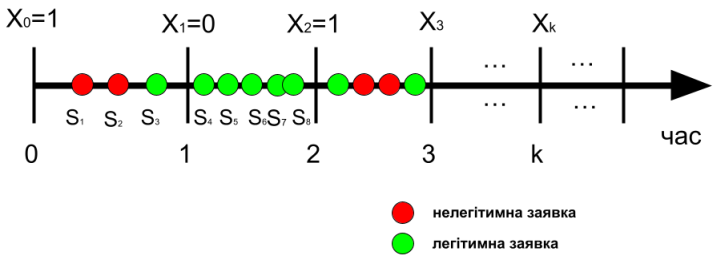

Рисунок 2.2 – Вигляд прямої часу з нанесеними подіями - запитами до сервера, статус яких відомий

ϕ^* знайдено оцінку станів прихованого ланцюга, що дозволяє виявити інтервали часу, в яких на сервер здійснювалася атака.

3 ЕКСПЕРИМЕНТ

У даному розділі буде описано процеси генерації даних, оцінки моделі і декодування ланцюжка. Для даної роботи було використано мову програмування C++ через її швидкодію і вже готові бібліотеки, які можуть допомогти при розробці(наприклад, «OpenCV»).

3.1 Генерація даних

Для роботи з моделлю перш за все треба створити дані для перевірки правильності виконання.

Було створено генератор, який працює за даним алгоритмом:

1) генерує ланцюг Маркова за допомогою Рівномірного розподілу на $(0, 1)$ (рис. 3.1)

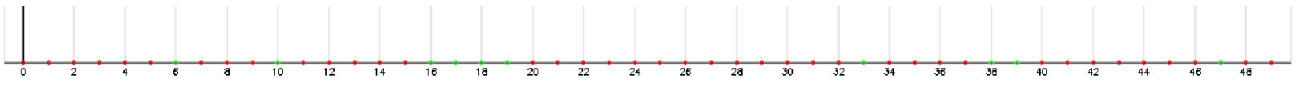

Рисунок 3.1 – Генерація ланцюга Маркова

2) генерує загальну кількість легальних підключень, які будуть на прямій часу за допомогою розподілу Пуссона

3) генерує на прямій часу легальні підключення, тобто розставляє на прямій часу точки за допомогою Рівномірного розподілу на всьому інтервалі часу, який ми розглядаємо

4) генерується кількість спроб ідентифікації для кожного легального запиту (рис. 3.2)

5) генерує загальну кількість зловмисних підключень, які будуть на прямій часу за допомогою розподілу Пуссона

6) генерує на прямій часу зловмисні підключення і розставляє її тільки в ті інтервали, де ланцюг Маркова має індикацію атаки

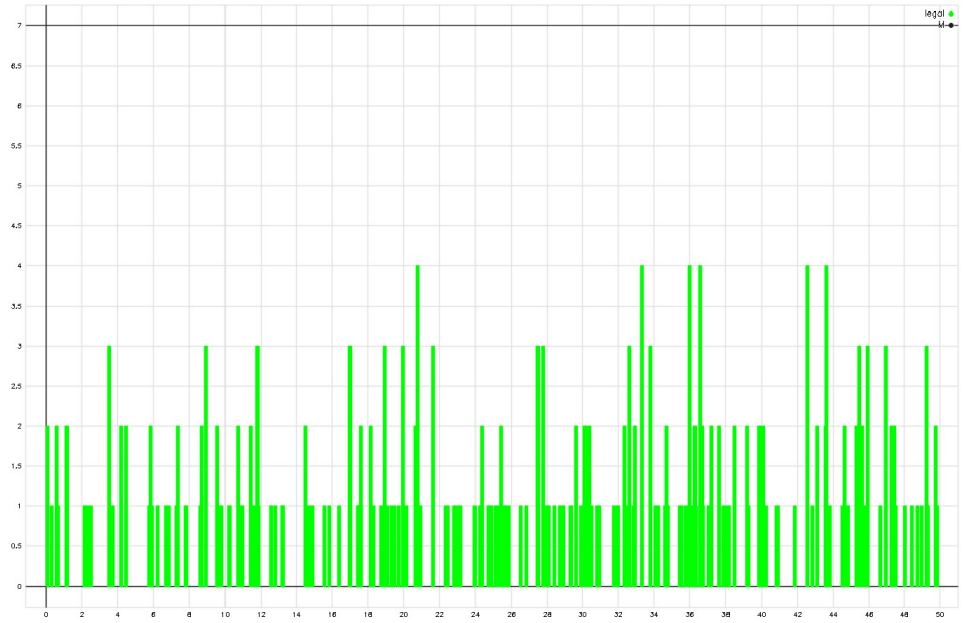

Рисунок 3.2 – Генерація легальних підключень

7) генерується кількість спроб ідентифікації для кожного зловмисного запиту (рис. 3.3)

За допомогою даного генератора було згенеровано тестові дані для моделі $\phi = (\mu, A; \lambda_0, \lambda_1; p_0, p_1)$ з такими параметрами: $\mu = (0.43, 0.57)$, $A = \begin{pmatrix} 0.6 & 0.4 \\ 0.3 & 0.7 \end{pmatrix}$, $\lambda_0 = 2$, $\lambda_1 = 3$, $p_0 = 0.7$, $p_1 = 0.08$.

Інтервал часу: $T = 50$.

Максимально дозволена кількість спроб ідентифікації: $M = 7$.

Отриманий ланцюг (рис. 3.4): $X_0 = 1$, $X_1 = 0$, $X_2 = 1$, $X_3 = 1$, $X_4 = 1$, $X_5 = 0$, $X_6 = 0$, $X_7 = 1$, $X_8 = 1$, $X_9 = 1$, $X_{10} = 0$, $X_{11} = 0$, $X_{12} = 1$, $X_{13} = 0$, $X_{14} = 1$, $X_{15} = 0$, $X_{16} = 1$, $X_{17} = 1$, $X_{18} = 1$, $X_{19} = 1$, $X_{20} = 0$, $X_{21} = 0$, $X_{22} = 0$, $X_{23} = 0$, $X_{24} = 1$, $X_{25} = 1$, $X_{26} = 1$, $X_{27} = 1$, $X_{28} = 1$, $X_{29} = 1$, $X_{30} = 1$, $X_{31} = 1$, $X_{32} = 0$, $X_{33} = 0$, $X_{34} = 1$, $X_{35} = 0$, $X_{36} = 0$, $X_{37} = 0$, $X_{38} = 1$, $X_{39} = 1$, $X_{40} = 1$, $X_{41} = 1$, $X_{42} = 0$, $X_{43} = 1$, $X_{44} = 1$, $X_{45} = 1$, $X_{46} = 0$, $X_{47} = 1$, $X_{48} = 1$, $X_{49} = 1$

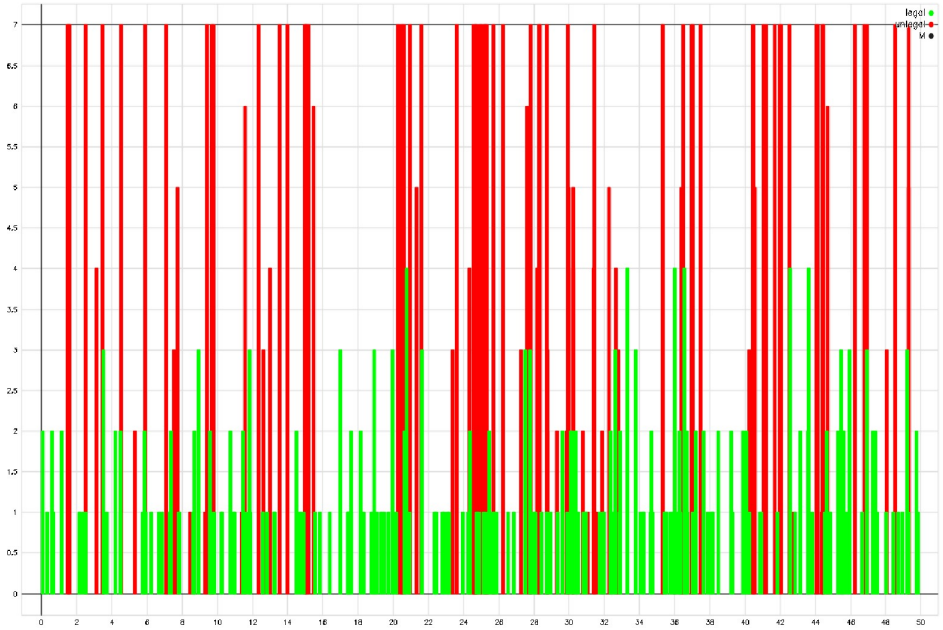

Рисунок 3.3 – Додавання зловмисних підключень

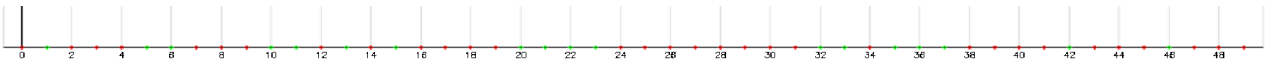

Рисунок 3.4 – Згенерований ланцюг Маркова

3.2 Алгоритм Баума-Велша

Алгоритм Баума-Велша [5] є важливою складовою нашого експерименту. Цей алгоритм використовується для оцінки прихованої Марковської моделі, що дає змогу встановити найбільш імовірні послідовності станів у моделі, враховуючи спостереження.

Алгоритм Баума-Велша показав високу точність і ефективність у визначенні послідовності станів у моделі, навіть в умовах обмежених даних. Він демонструє здатність до адаптації до зміни умов та розпізнавання складних закономірностей в послідовностях спостережень.

Крім того, експерименти показали, що алгоритм Баума-Велша добре справляється з проблемою недостовірних даних. Він може врахувати незрівнянність спостережень і моделі та забезпечити надійну імовірнісну оцінку послідовності станів.

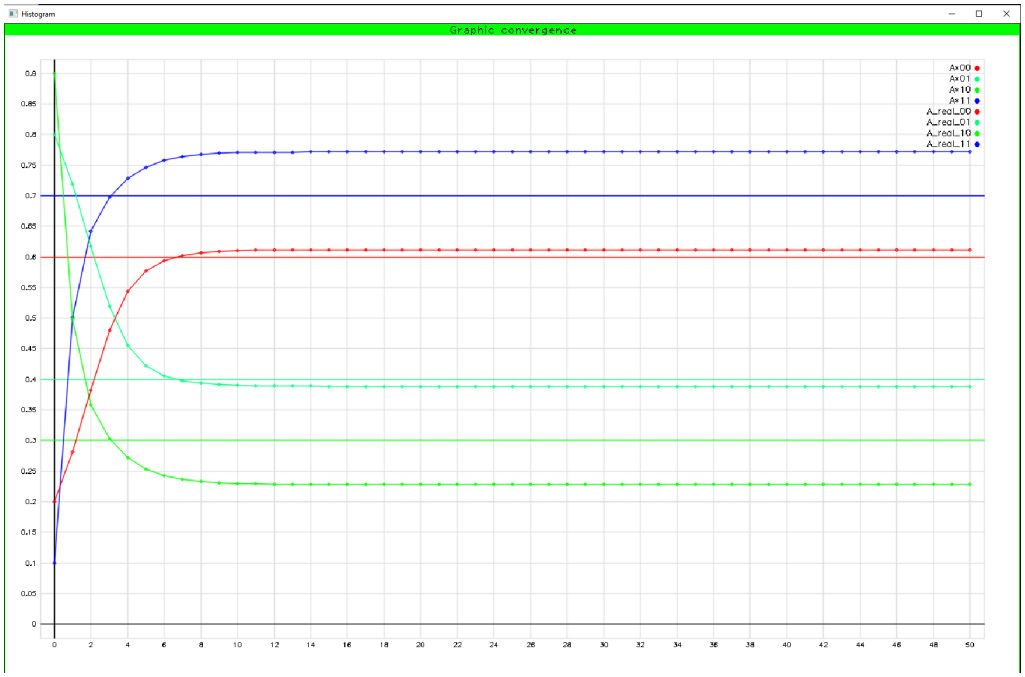

Рисунок 3.5 – Графік збіжності матриці A (прямі – істині значення матриці A)

На даному етапі експерименту, як початкову точку для пошуку матриці A, було обрано: $A^* = \begin{pmatrix} 0.2 & 0.8 \\ 0.9 & 0.1 \end{pmatrix}$,

В результаті експерименту була оцінена матриця A (рис. 3.5): $A^* = \begin{pmatrix} 0.611559 & 0.388442 \\ 0.228563 & 0.771437 \end{pmatrix}$,

3.3 Алгоритм Вітербі

Метою нашої роботи є отримання інтервалів, на яких найімовірніше була проведена атака, саме з цим нам і допоміг алгоритм Вітербі. За отриманою вище матрицею A^* ми декодуємо(відновлюємо) прихований ланцюг і як наслідок можемо говорити, на яких проміжках з великою ймовірністю була проведена атака зловмисником.

Як буде видно, алгоритм Вітербі показує гарні результати, навіть на

невеликих вибірках, і що найголовніше майже не пропускає атаки.

В результаті було декодовано прихований ланцюг Маркова (рис. 3.6): $X_0 = 0$, $X_1 = 0$, $X_2 = 1$, $X_3 = 1$, $X_4 = 0$, $X_5 = 0$, $X_6 = 0$, $X_7 = 1$, $X_8 = 1$, $X_9 = 1$, $X_{10} = 0$, $X_{11} = 1$, $X_{12} = 1$, $X_{13} = 0$, $X_{14} = 0$, $X_{15} = 1$, $X_{16} = 1$, $X_{17} = 1$, $X_{18} = 0$, $X_{19} = 1$, $X_{20} = 0$, $X_{21} = 0$, $X_{22} = 0$, $X_{23} = 0$, $X_{24} = 1$, $X_{25} = 1$, $X_{26} = 1$, $X_{27} = 1$, $X_{28} = 0$, $X_{29} = 1$, $X_{30} = 1$, $X_{31} = 1$, $X_{32} = 0$, $X_{33} = 0$, $X_{34} = 1$, $X_{35} = 0$, $X_{36} = 0$, $X_{37} = 0$, $X_{38} = 1$, $X_{39} = 1$, $X_{40} = 1$, $X_{41} = 1$, $X_{42} = 0$, $X_{43} = 1$, $X_{44} = 0$, $X_{45} = 1$, $X_{46} = 0$, $X_{47} = 1$, $X_{48} = 0$, $X_{49} = 1$

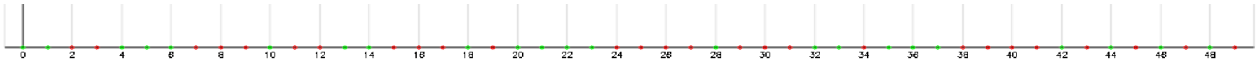

Рисунок 3.6 – Декодований ланцюг Маркова

Даний ланцюг від істиного відрізняється в параметрах(рис. 3.7): X_0, X_4, X_{11}, X_{14}, X_{15}, X_{15}, X_{18}, X_{28}, X_{44}, X_{48}, що для невеликої вибірки є гарним результатом. Тут варто зауважити, що модель пропустила тільки одну атаку, а всі інші – хибні атаки. Даний результат можна покращити, застосувавши більш складну модель, яка враховує додаткові умови, наприклад, що при атаці хоча б одне підключення є зловмисним. Також можна збільшити спостережуваний інтервал, але для кращого опису, в даній роботі був використаний проміжок часу $[0, 50)$

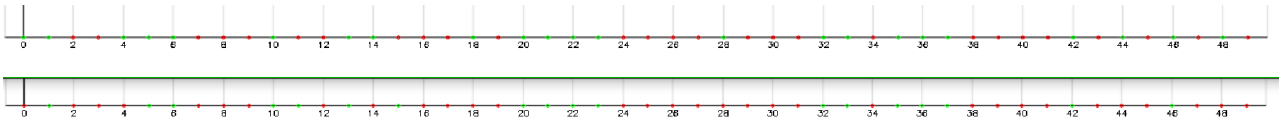

Рисунок 3.7 – Різниця декодованого ланцюга від істиного.

Висновки до розділу 3

Отже, у цьому розділі було побудовано генератор тестових даних для нашої моделі, реалізовані алгоритми Баума-Велша і Вітербі, а також проведений експеримент.

Експеримент показав, що дану модель можна використовувати для аналізу деякого проміжку часу на наявність атаки, що значно спрощує роботу фахівцям з кібербезпеки.

ВИСНОВКИ

В даній дипломній роботі було поставлено завдання розробити приховану марківську модель для ймовірнісного опису процесу надходження та ідентифікації запитів на сервер. Для досягнення цієї мети було використано методи статистики та ймовірнісного моделювання.

У першій частині роботи була запропонована прихована марківська модель, яка дозволяє описувати процес надходження та ідентифікації запитів на сервер. Модель складається з прихованої послідовності станів, що представляють наявність/відустність атак, та спостережень, які є результатами цього процесу. Використання прихованої марковської моделі дозволяє отримати інформацію про приховані стани системи на основі спостережень.

У другій частині роботи було побудовано оцінки параметрів ймовірнісних розподілів, якими моделюються спостережені дані. Ці оцінки параметрів дозволяють точніше визначити ймовірнісні характеристики процесу надходження та ідентифікації запитів, ніж якщо параметри прихованого ланцюга задавати вручну.

В останній частині роботи були проведені чисельні експерименти для перевірки якості побудованих оцінок. Ці експерименти демонструють, наскільки точно запропонована модель описує реальний процес надходження та ідентифікації запитів. Ці експерименти підтвердили гіпотези, які ставилися у роботі щодо процесу надходження та ідентифікації запитів на сервер.

Отже, в результаті проведення дослідження було розроблено приховану марківську модель для ймовірнісного опису процесу надходження та ідентифікації запитів на сервер. Було побудовано оцінки параметрів ймовірнісних розподілів, які моделюють спостережені дані. Чисельні експерименти підтвердили високу якість побудованих оцінок і відповідність моделі реальному процесу. Дані результати вказують на

ефективність запропонованої моделі для аналізу процесу надходження та ідентифікації запитів на сервер.

Щоб покращити запропоновану модель – її можна розширити, тобто вдосконалити, додавши додаткові стани моделі. Також можна більш точно оцінити параметри додавши додаткові умови, наприклад, що при атаці обов'язково є хоча б одне зловмисне підключення.

ПЕРЕЛІК ПОСИЛАНЬ

[1] Kholidy H. A. та ін. «A finite state hidden markov model for predicting multistage attacks in cloud systems». Англ. В: *IEEE International Conference on Dependable, Autonomic and Secure Computing* (2014), с. 14—19.

[2] A. Freier, P. Karlton та P. Kocher. *The Secure Sockets Layer (SSL) Protocol Version 3.0*. Англ. Серп. 2011.

[3] Megan Kaczanowski. *Common Attacks on SSL/TLS – and How to Protect Your System*. Англ.

[4] T. Koski. *Hidden Markov models for bioinformatics. T. 2*. Англ. Springer Science & Buisiness Media, 2001.

[5] M. Nilsson. *First Order Hidden Markov Model Theory and Implementation Issues*. Англ. Department of Signal Processing Blekinge Institute of Technology, Sweden, 2005.

[6] Holgado P., Vilagra V. A. та Vazquez L. «Real-time multistep attack prediction based on hidden markov models». Англ. В: *IEEE Transactions on Dependable and Secure Computing* 1 (2020), с. 17.

[7] Anna Sperotto та ін. «Hidden Markov Model Modeling of SSH Brute-Force Attacks». Англ. В: *Integrated Management of Systems, Services, Processe and People in IT* (2009), с. 164—176.

[8] Zan X. та ін. «A hidden markov model based on framework for tracking and predicting of attack intention». Англ. В: *International COnference on Multimedia Information Networking and Security* (2009), с. 498—501.

[9] Е. С. Вентцель та Л. А. Овчаров. *Теория случайных процессов и ее инженерные приложения*. Рос. Высшая школа, 2000.

ДОДАТОК А ТЕКСТИ ПРОГРАМ

А.1 HelpersTypes.h(Допоміжні типи даних)

```
#include <map>
enum ConnectionStatus
{
  LEGAL = 0,
  UNLEGAL = 1,
  UNKNOWN
};

using time_int = unsigned long long int;

using prob_mat = std::vector<std::vector<float>>;

using time_space = std::map<double, std::pair<ConnectionStatus,
    int>>;
```

А.2 Generating.h(Генератор)

```
#pragma once
#include "Printing.h"
#include "HelpersTypes.h"
#include <random>
#include <iostream>
#include "algs.h"

int truncated_geometric_distribution(double p, int M)
{
```

```
  std::vector<double> p_m;
  p_m.reserve(M);
  for (size_t i = 0; i < M - 1; i++)
  {
    p_m.push_back(std::pow((1 - p), i) * p);
  }
  p_m.push_back(std::pow((1 - p), M - 1));

  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_real_distribution<> u(0, 1);
  auto point_on_D = u(gen);

  double sum = 0;
  for (size_t i = 1; i <= M; i++)
  {
    sum += p_m[i - 1];
    if (point_on_D < sum)
    {
      return i;
    }
  }

  return M;
}

//choose where will be connection
std::map<double, std::pair<ConnectionStatus, int>>
   generate_legal_connections_points(int k, time_int T)//good
   connections
{
```

```
  //adding connections on time line
  std::map<double, std::pair<ConnectionStatus, int>> time_line;
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_real_distribution<> s_i(0, T);
  for (size_t i = 0; i < k; i++)
  {
    time_line[s_i(gen)].first = ConnectionStatus::LEGAL;
    time_line[s_i(gen)].second++;
  }

  return time_line;
}

//generate count of connections on choosen points
void generate_connections_counts(std::map<double, std::pair<
   ConnectionStatus, int>>& time_line, double p, unsigned long
   long M, ConnectionStatus status)
{
  for (auto& time_point : time_line)
  {
    if (time_point.second.first == status)
    {
      time_point.second.second =
   truncated_geometric_distribution(p, M);
    }

  }
}

//data_count - T_max
```

```
std::vector<int> generate_Markov(const std::array<std::array<
    double, 2>, 2>& probability_matrix, /*const std::array<
    double, 2>& initial_state,*/ double criteria, size_t
    data_count)
{
  std::vector<int> x_vec(data_count, 0);
  long double null_to_null = 0;
  long double null_count = 0;
  long double one_to_one = 0;
  long double one_count = 0;

  {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> u(0, 1);
    //X_0 = U > mu[0] => 1, else 0
    x_vec[0] = (u(gen) > criteria) ? 1 : 0;
  }
  for (size_t i = 1; i < data_count; i++)
  {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> u(0, 1);
    //U > Aj0 => X_i = 1, else X_i = 0
    x_vec[i] = (u(gen) > probability_matrix[x_vec[i - 1]][0]) ?
    1 : 0;
    switch (x_vec[i])
    {
    case 0:
    {
      if (x_vec[i - 1] == 0)
```

```
      {
        ++null_to_null;
      }
      ++null_count;
      break;
    }
    case 1:
    {
      if (x_vec[i - 1] == 1)
      {
        ++one_to_one;
      }
      ++one_count;
      break;
    }
    }
  }

  std::cout << "NULL TO NULL: " << null_to_null << std::endl;
  std::cout << "NULLS COUNT: " << null_count << std::endl;
  std::cout << "NULLS PROBS: " << null_to_null / null_count <<
   std::endl;
  std::cout << "ONE TO ONE: " << one_to_one << std::endl;
  std::cout << "ONES COUNT: " << one_count << std::endl;
  std::cout << "ONES PROBS: " << one_to_one / one_count << std
   ::endl;
  return x_vec;
}

void add_not_legal_connections(const std::vector<int>&
   markovModel, double lambda_1, std::map<double, std::pair<
```

```
   ConnectionStatus, int>>& time_line)
{
  for (size_t i = 0; i < markovModel.size(); i++)
  {
    std::random_device rd_ksi;
    std::mt19937 gen_ksi(rd_ksi());

    std::poisson_distribution<> ksi(lambda_1);
    //int k = 1 + ksi(gen_ksi); //FOR more accurate model
    int k = ksi(gen_ksi);

    //X_i = 1
    if (markovModel[i] == 1)
    {
      for (size_t j = 0; j < k; j++)
      {
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_real_distribution<> s_i(i, i + 1);

        auto point = s_i(gen);
        if (time_line.find(point) != time_line.end())
        {
          point += 0.0001;
        }
        time_line[point].first = ConnectionStatus::UNLEGAL;
        time_line[point].second++;
      }
    }
  }
}
```

```
//generate
std::map<double, std::pair<ConnectionStatus, int>>
   generate_data(double lambda_0, double lambda_1, time_int T,
   double p_0, double p_1, unsigned long long M, const std::
   array<std::array<double, 2>, 2>& probability_matrix, double
   criteria)
{
  auto x_vec = generate_Markov(probability_matrix, criteria, T)
   ;
  for (auto x_i : x_vec)
  {
    std::cout << x_i << " ";
  }
  print_Markov_chain(x_vec);
  std::cout << std::endl;
  std::random_device rd_ksi;
  std::mt19937 gen_ksi(rd_ksi());
  std::poisson_distribution<> ksi(lambda_0 * static_cast<double
   >(T));

  int k = ksi(gen_ksi);// count of connections

  auto connections = generate_legal_connections_points(k, T);
  generate_connections_counts(connections, p_0, M,
   ConnectionStatus::LEGAL);

  print_graphics(connections, M);

  add_not_legal_connections(x_vec, lambda_1, connections);
  generate_connections_counts(connections, p_1, M,
```

```
 ConnectionStatus::UNLEGAL);
//std::cout << "Connections" << std::endl;

if (x_vec.front() == 0)
{
  auto k_leg = CreateKFromTimeLineOnInterval(connections, 0,
 M);

  print_conn_histo(k_leg);
  size_t t = 0;
  for (auto x_i : x_vec)
  {
    if (x_i == 1)
    {
      auto k_unleg = CreateKFromTimeLineOnInterval(
 connections, t, M);
      print_conn_histo(k_unleg);
      break;
    }
    t++;
  }
}
else
{
  auto k_unleg = CreateKFromTimeLineOnInterval(connections,
 0, M);
  print_conn_histo(k_unleg);
  size_t t = 0;
  for (auto x_i : x_vec)
  {
    if (x_i == 0)
```

```
      {
        auto k_leg = CreateKFromTimeLineOnInterval(connections,
    t, M);
        print_conn_histo(k_leg);
        break;
      }
      t++;
    }
  }

  return connections;
}
```

A.3 Printing.h(Вивід графіків)

```
#pragma once
#include "HelpersTypes.h"
#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include "..\cvplot\include\cvplot\cvplot.h"

void print_graphics(const std::map<double, std::pair<
   ConnectionStatus, int>>& time_line, unsigned long long M =
   0)
{
  auto window = cvplot::Window("Histogram");
  auto& view =
    window.view("Histogram", { 1400, 900 }).offset({ 0, 0 });
  auto figure = cvplot::Figure(view);
```

```
for (auto time_point : time_line)
{
  if (time_point.second.first == ConnectionStatus::UNLEGAL)
  {
    figure.series("unlegal")
      .type(cvplot::Histogram)
      .add({ std::pair<double, double>(time_point.first,
 time_point.second.second) })
      .color(cvplot::Red);
  }
  else if (time_point.second.first == ConnectionStatus::LEGAL
 )
  {
    figure.series("legal")
      .type(cvplot::Histogram)
      .add({ std::pair<double, double>(time_point.first,
 time_point.second.second) })
      .color(cvplot::Green);
  }
  else
  {
    figure.series("unkonown")
      .type(cvplot::Histogram)
      .add({ std::pair<double, double>(time_point.first,
 time_point.second.second) })
      .color(cvplot::Gray);
  }
}
figure.series("M")
  .type(cvplot::Horizontal)
  .setValue(M)
```

```
    .color(cvplot::Dark);

  figure.show();
  cv::waitKey(0);
}

void print_graphic_convergence(prob_mat A_real, std::vector<
   prob_mat> A_stars)
{
  auto window = cvplot::Window("Histogram");
  auto& view =
    window.view("Graphic convergence", { 1400, 900 }).offset({
   0, 0 });
  auto figure = cvplot::Figure(view);
  for (size_t i = 0; i < A_stars.size(); i++)
  {
    figure.series("A*00")
      .type(cvplot::DotLine)
      .add(i, A_stars[i][0][0])
      .color(cvplot::Red);

    figure.series("A*01")
      .type(cvplot::DotLine)
      .add(i, A_stars[i][0][1])
      .color(cvplot::Aqua);

    figure.series("A*10")
      .type(cvplot::DotLine)
      .add(i, A_stars[i][1][0])
      .color(cvplot::Green);
```

```
    figure.series("A*11")
      .type(cvplot::DotLine)
      .add(i, A_stars[i][1][1])
      .color(cvplot::Blue);
  }

  figure.series("A_real_00")
    .type(cvplot::Horizontal)
    .add(A_stars.size() - 1, A_real[0][0])
    .color(cvplot::Pink);

  figure.series("A_real_01")
    .type(cvplot::Horizontal)
    .add(A_stars.size() - 1, A_real[0][1])
    .color(cvplot::Cyan);

  figure.series("A_real_10")
    .type(cvplot::Horizontal)
    .add(A_stars.size() - 1, A_real[1][0])
    .color(cvplot::Gray);

  figure.series("A_real_11")
    .type(cvplot::Horizontal)
    .add(A_stars.size() - 1, A_real[1][1])
    .color(cvplot::Dark);

  figure.show();
  cv::waitKey(0);
}

void print_conn_histo(const std::map<size_t, int>& data)
```

```
{
  auto window = cvplot::Window("Histogram");
  auto& view =
    window.view("Histogram", { 1400, 900 }).offset({ 0, 0 });
  auto figure = cvplot::Figure(view);
  for (auto [tries, count] : data)
  {
    figure.series("connection")
      .type(cvplot::Histogram)
      .add({ std::pair<double, double>(tries, count) })
      .color(cvplot::Blue);
  }
  figure.show();
  cv::waitKey(0);
}

void print_Markov_chain(const std::vector<int>& X)
{
  auto window = cvplot::Window("Markov Chain");
  auto& view =
    window.view("Markov Chain", { 1400, 900 }).offset({ 0, 0 })
   ;
  view.resize({ 0, 0, 1400, 900 });
  auto figure = cvplot::Figure(view);
  for (size_t t = 0; t < X.size(); t++)
  {
    if (X[t] == 0)
    {
      figure.series("Good request")
        .type(cvplot::Dots)
        .add(t, 0)
```

```
        .color(cvplot::Green);
    }
    if (X[t] == 1)
    {
      figure.series("Bad request")
        .type(cvplot::Dots)
        .add(t, 0)
        .color(cvplot::Red);
    }

  }
  figure
    .show();
  cv::waitKey(0);
}
```

A.4 algs.cpp(Алгоритми)

```
#include "new_algs.h"
#include <cmath>

float factorial(size_t num)
{
  long double fact = 1;
  for (size_t i = 1; i <= num; i++)
  {
    fact *= i;
  }

  return fact;
```

```
}

inline auto q_i(size_t k, float p_i, size_t M)
{
  return std::powf(1 - p_i, k - 1) * ((k < M) ? p_i : 1);
}

float b(int i, size_t M, size_t k, const std::vector<int>& Y,
   float lambda_0, float p_0, float lambda_1, float p_1)
{
  if (k != Y.size())
  {
    throw std::runtime_error("k != Y.size()");
  }

  switch (i)
  {
  case 1:
  {
    if (k == 0)
      return std::exp(-(lambda_0 + lambda_1));
    if (k > 0)
    {
      auto res_value = std::exp(-(lambda_0 + lambda_1));
      res_value /= factorial(k);
      for (auto y_r : Y)
      {
        res_value *= (lambda_0 * q_i(y_r, p_0, M) + lambda_1 *
   q_i(y_r, p_1, M));
      }
      return res_value;
```

```
    }
    return 0.0;
  }
  case 0:
  {
    if (k == 0)
      return std::exp(-lambda_0);
    auto res_value = std::powf(lambda_0, k) * std::exp(-
   lambda_0);
    res_value /= factorial(k);
    for (auto y_r : Y)
    {
      res_value *= q_i(y_r, p_0, M);
    }
    return res_value;
  }
  default:
    throw std::runtime_error("This block mustn't use");
  }
}

std::vector<int> CreateYFromTimeLineOnInterval(const time_space
   & time_line, time_int t)
{
  std::vector<int> Y;
  long double t_double = t;
  for (const auto& [time, statusAndCount] : time_line)
  {
    if (time > t_double + 1)
      break;
    if (time >= t_double)
```

```
    {
      int conn_count = statusAndCount.second;
      Y.push_back(conn_count);
    }
  }
  return Y;
}

std::map<size_t, int> CreateKFromTimeLineOnInterval(const std::
   map<double, std::pair<ConnectionStatus, int>>& time_line,
   time_int t, size_t M)
{
  std::map<size_t, int> K;
  for (size_t i = 1; i <= M; i++)
  {
    K[i] = 0;
  }

  long double t_double = t;
  for (const auto& [time, statusAndCount] : time_line)
  {
    if (time > t_double + 1)
      break;
    if (time >= t_double)
    {
      K[statusAndCount.second]++;
    }
  }
  return K;
}
```

```
std::pair<std::vector<std::vector<float>>, std::vector<float>>
   ForwardAlg(
  time_int T,
  const time_space& time_line,
  const std::vector<float>& mu,
  const prob_mat& A,
  size_t M,
  float lambda_0,
  float lambda_1,
  float p_0,
  float p_1)
{
  auto N = mu.size();
  float one = 1.f;
  std::vector<std::vector<float>> alpha(T, std::vector<float>(N
   , 0.0f));
  std::vector<float> c(T, 0.0f);
  auto Y_0 = CreateYFromTimeLineOnInterval(time_line, 0);
  c[0] = 0.0f;
  for (size_t i = 0; i < N; i++)
  {
    alpha[0][i] = mu[i] * b(i, M, Y_0.size(), Y_0, lambda_0,
   p_0, lambda_1, p_1);
    c[0] += alpha[0][i];
  }
  c[0] = one / c[0];
  for (size_t i = 0; i < N; i++)
  {
    alpha[0][i] *= c[0];
  }
```

```
  for (size_t t = 1; t < T; t++)
  {
    c[t] = 0;
    auto Y_t = CreateYFromTimeLineOnInterval(time_line, t);
    for (size_t i = 0; i < N; i++)
    {
      for (size_t j = 0; j < N; j++)
      {
        alpha[t][i] += (alpha[t - 1][j] * A[j][i]);
      }
      alpha[t][i] *= b(i, M, Y_t.size(), Y_t, lambda_0, p_0,
   lambda_1, p_1);
      c[t] += alpha[t][i];
    }
    c[t] = one / c[t];
    for (size_t i = 0; i < N; i++)
    {
      alpha[t][i] *= c[t];
    }
  }

  return { alpha, c };
}

std::vector<std::vector<float>> BackwardAlg(
  time_int T,
  const time_space& time_line,
  const std::vector<float>& mu,
  const prob_mat& A,
  std::vector<float> c,
  size_t M,
```

```
  float lambda_0,
  float lambda_1,
  float p_0,
  float p_1)
{
  auto N = mu.size();
  std::vector<std::vector<float>> beta(T, std::vector<float>(N,
    0.0f));
  for (size_t i = 0; i < N; i++)
  {
    beta[T - 1][i] = c[T - 1];
  }
  for (size_t k = T - 1; k > 0 ; k--)
  {
    size_t t = k - 1;//starts from T - 2
    auto Y_t_p_1 = CreateYFromTimeLineOnInterval(time_line, t +
    1);
    for (size_t i = 0; i < N; i++)
    {
      beta[t][i] = 0;
      for (size_t j = 0; j < N; j++)
      {
        beta[t][i] += (A[i][j] * b(j, M, Y_t_p_1.size(),
   Y_t_p_1, lambda_0, p_0, lambda_1, p_1) * beta[t + 1][j]);
      }
      beta[t][i] *= c[t];
    }
  }
  return beta;
}
```

```
float ComputeLogProb(const std::vector<float>& c)
{
  float logProb = 0.0f;
  for (auto c_i : c)
  {
    logProb += std::logf(c_i);
  }
  logProb = -logProb;
  return logProb;
}

std::pair<std::vector<std::vector<std::vector<float>>>, std::
   vector<std::vector<float>>> computeGammasAndDiGammas(
  const std::vector<std::vector<float>> alpha,
  const std::vector<std::vector<float>> beta,
  time_int T,
  const time_space& time_line,
  const std::vector<float>& mu,
  const prob_mat& A,
  size_t M,
  float lambda_0,
  float lambda_1,
  float p_0,
  float p_1
)
{
  auto N = mu.size();
  std::vector<std::vector<std::vector<float>>> gamma_ij(T - 1,
   std::vector<std::vector<float>>(N, std::vector<float>(N, 0.0
   f)));
  std::vector<std::vector<float>> gamma_i(T, std::vector<float
```

```
 >(N, 0.0f));
for (size_t t = 0; t < T - 1; t++)
{
  float denom = 0;
  auto Y_t_p_1 = CreateYFromTimeLineOnInterval(time_line, t +
  1);
  for (size_t i = 0; i < N; i++)
  {
    for (size_t j = 0; j < N; j++)
    {
      denom += (alpha[t][i] * A[i][j] * b(j, M, Y_t_p_1.size
 (), Y_t_p_1, lambda_0, p_0, lambda_1, p_1) * beta[t+1][j]);
    }
  }
  for (size_t i = 0; i < N; i++)
  {
    for (size_t j = 0; j < N; j++)
    {
      gamma_ij[t][i][j] = ((alpha[t][i] * A[i][j] * b(j, M,
 Y_t_p_1.size(), Y_t_p_1, lambda_0, p_0, lambda_1, p_1) *
 beta[t + 1][j]) / denom);
      gamma_i[t][i] += gamma_ij[t][i][j];
    }
  }
}

//calculate gamma[T-1](i)
float denom = 0;
for (size_t i = 0; i < N; i++)
{
  denom += alpha[T - 1][i] * beta[T - 1][i];
```

```
  }
  for (size_t i = 0; i < N; i++)
  {
    gamma_i[T - 1][i] += ((alpha[T - 1][i] * beta[T - 1][i] )/
   denom);
  }
  return { gamma_ij, gamma_i };
}

std::pair<std::vector<prob_mat>, std::vector<float>>
   BaumaVelshaAlgSc(
  time_int T,
  const time_space& time_line,
  const std::vector<float>& mu,
  const prob_mat& A,
  size_t M,
  float lambda_0,
  float lambda_1,
  float p_0,
  float p_1,
  size_t iter_count)
{
  std::vector<prob_mat> A_stars_by_iters;
  A_stars_by_iters.reserve(iter_count + 1);

  auto A_star = A;
  auto mu_star = mu;
  auto N = mu.size();
  A_stars_by_iters.push_back(A_star);
  for (size_t it = 0; it < iter_count; it++)
  {
```

```
 auto [alpha, c] = ForwardAlg(T, time_line, mu_star, A_star,
M, lambda_0, lambda_1, p_0, p_1);
 auto beta = BackwardAlg(T, time_line, mu_star, A_star, c, M
, lambda_0, lambda_1, p_0, p_1);
 auto [gamma_ij, gamma_i] = computeGammasAndDiGammas(alpha,
beta, T, time_line, mu_star, A_star, M, lambda_0, lambda_1,
p_0, p_1);

 //mu*
 for (size_t i = 0; i < N; i++)
 {
   mu_star[i] = gamma_i[0][i];
 }

 //A*
 for (size_t i = 0; i < N; i++)
 {
   for (size_t j = 0; j < N; j++)
   {
     float numer = 0.0f;
     float denom = 0.0f;
     for (size_t t = 0; t < T - 1; t++)
     {
       numer += gamma_ij[t][i][j];
       denom += gamma_i[t][i];
     }
     A_star[i][j] = (numer / denom);
   }
 }
 A_stars_by_iters.push_back(A_star);
```

```
    //----------OUTPUTING------------
    std::cout << "----------ITERATION " << it + 1 << "
   --------------" << std::endl;
    std::cout << "A*: " << std::endl;
    for (size_t i = 0; i < N; i++)
    {
      for (size_t j = 0; j < N; j++)
      {
        std::cout << A_star[i][j] << " ";
      }
      std::cout << std::endl;
    }

    std::cout << "mu*: " << std::endl;
    for (size_t i = 0; i < N; i++)
    {
      std::cout << mu_star[i] << " ";

    }
    std::cout << std::endl;

    std::cout << "log(P(O | lambda)): " << ComputeLogProb(c) <<
    std::endl;

    std::cout << "---------------------------------------" <<
   std::endl;

  }
  return { A_stars_by_iters, mu_star };
}
```

```
std::vector<int> Viterbi(
  time_int T,
  const time_space& time_line,
  const std::vector<float>& mu,
  const prob_mat& A,
  size_t M,
  float lambda_0,
  float lambda_1,
  float p_0,
  float p_1)
{
  auto N = mu.size();
  std::vector<std::vector<float>> sigma(T, std::vector<float>(N
   , 0.0f));
  std::vector<int> ksi(T, 0);
  auto Y_0 = CreateYFromTimeLineOnInterval(time_line, 0);
  for (size_t i = 0; i < N; i++)
  {
    sigma[0][i] = std::log(mu[i] * b(i, M, Y_0.size(), Y_0,
   lambda_0, p_0, lambda_1, p_1));
  }

  for (size_t t = 1; t < T; t++)
  {
    auto Y_t = CreateYFromTimeLineOnInterval(time_line, t);
    for (size_t i = 0; i < N; i++)
    {
      for (size_t j = 0; j < N; j++)
      {
        auto sigma_candidate = sigma[t - 1][j] + std::log(A[j][
   i]) + std::log(b(i, M, Y_t.size(), Y_t, lambda_0, p_0,
```

```
    lambda_1, p_1));
          if ((sigma[t][i] < sigma_candidate) || (j == 0))
          {
            sigma[t][i] = sigma_candidate;
          }
        }
      }

      for (size_t i = 1; i < N; i++)
      {
        ksi[t] = (sigma[t][i] > sigma[t][i - 1]) ? i : i - 1;
      }
    }

    return ksi;
}
```

A.5 main.cpp(Експеримент)

```
#include <vector>
#include <array>
#include "Generating.h"

int main()
{
    //Parameters
    std::array<double, 2> prob_mat_0 = { 0.6, 0.4 };
    std::array<double, 2> prob_mat_1 = { 0.3, 0.7 };
    std::array<std::array<double, 2>, 2> A = { prob_mat_0,
     prob_mat_1 };
```

```
std::array<double, 2> mu = { 0.43, 0.57 };
double lambda_0 = 2;
double lambda_1 = 3;
time_int T = 50;
double p_0 = 0.7;
double p_1 = 0.08;
size_t M = 7;

auto connections = generate_data(lambda_0, lambda_1, T, p_0,
 p_1, M, A, mu[0]);
print_graphics(connections, M);

std::vector<float> prob_test_mat_0 = { 0.3, 0.7 };
std::vector<float> prob_test_mat_1 = { 0.6, 0.4 };
std::vector<std::vector<float>> A_start = { prob_test_mat_0,
  prob_test_mat_1 };
std::vector<float> mu_start = { 0.6, 0.4 };


auto [A_stars, mu_star] = BaumaVelshaAlgSc(T, connections,
 mu_start, A_start, M, lambda_0, lambda_1, p_0, p_1, 50);

prob_mat A_vec;
for (const auto& line : A)
{
  A_vec.push_back(std::vector<float>(line.begin(), line.end()
 ));
}

print_graphic_convergence(A_vec, A_stars);
```

```
    auto X_res = Viterbi(T, connections, mu_star, A_stars.back(),
      M, lambda_0, lambda_1, p_0, p_1);

    for (auto x_i : X_res)
    {
      std::cout << x_i << " ";
    }
    std::cout << std::endl;

    print_Markov_chain(X_res);

    return 0;
}
```