МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

На правах рукопису

БОБРОВНІКОВА КІРА ЮЛІЇВНА

УДК 004.491. 2

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ВИЯВЛЕННЯ БОТ-МЕРЕЖ У КОРПОРАТИВНИХ МЕРЕЖАХ НА ОСНОВІ АНАЛІЗУ DNS-ТРАФІКА

дисертація на здобуття наукового ступеня
кандидата технічних наук

Спеціальність 05.13.06 – інформаційні технології

Науковий керівник
кандидат технічних наук, доцент
Савенко Олег Станіславович

Хмельницький 2016

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

3LD	-	Third-Level Subdomain
ASN	-	Autonomous System Number
C&C	-	Command-and-Control
DDNS	-	Dynamic Domain Name System
DDOS	-	Distributed Denial-of-Service
DGA	-	Domain Generation Algorithm
DNS	-	Domain Name System
IP	-	Internet Protocol
IRC	-	Internet Relay Chat
MAC	-	Media Access Control
NS	-	Name Server
NXDOMAIN	-	Non-Existent Domain
RCODE	-	Response Code
RR	-	Resource Records
SLD	-	Second-Level Domain
SOA	-	Start of Authority record
TLD	-	Top-Level Domain
TTL	-	Time To Live
АЗ	-	Антивірусний засіб
АПЗ	-	Антивірусне програмне забезпечення
ЖЦ	-	Життєвий цикл
ІТ	-	Інформаційна технологія
ІС	-	Інформаційна система
КС	-	Комп‘ютерна система
ОС	-	Операційна система
ПЗ	-	Програмне забезпечення
ШПЗ	-	Шкідливе програмне забезпечення

ВСТУП

Актуальність роботи. На сьогоднішній день проблема захисту інформації, оброблюваної в інформаційних системах (ІС), не має остаточного вирішення, оскільки в умовах стрімкого розвитку інформаційних технологій постійно виникають нові види загроз, які використовують вразливості програмного та апаратного забезпечення не лише комп'ютерних систем (КС), а й засобів захисту інформації.

Проблема інформаційної безпеки корпоративної мережі є однією з першочергових для бізнесу будь-якого рівня. Важливу роль у вирішенні цієї задачі відіграють програмні засоби забезпечення захисту, проте розробники антивірусного програмного забезпечення (АПЗ) не в змозі передбачити повний спектр можливостей по реалізації загроз з боку висококваліфікованих розробників шкідливого програмного забезпечення (ШПЗ).

Одним з найбільш небезпечних видів ШПЗ на сьогоднішній день є бот-мережі. Створення та використання бот-мереж стає все більш доступною задачею та не вимагає спеціальних знань, оскільки ШПЗ, послуги анонімного хостингу та вже створені бот-мережі пропонуються на ринку бот-мереж за помірними цінами. Переважна більшість бот-мереж для керування інфікованими КС використовує DNS [1]. Динамічна географічно розподілена структура бот-мереж, застосування різноманітних технологій ухилення від виявлення, в тому числі на основі DNS [2-4], та можливість анонімного керування інфікованими КС ускладнює виявлення бот-мереж та вживання контрзаходів з метою припинення їх діяльності.

Перспективним напрямком є підходи виявлення бот-мереж на основі DNS. Дослідження в цій сфері проводять Хольц Т. [5, 6], Девід Дж. Малан [7], Ентонакакіс М. [8-10], Дейгон Д. [11-13], Хюнсанг Чоі [14, 15], Дітріх Д. [16], Ішайз Х. [17, 18], Погребенник В.Д. [19-21], Котенко І.В. [22-24], Рувінська В.М. [25], Сиротинський О.І. [26], Касперський Є.В. [27], Касперські К. [28], Собейкіс В.Г. [29]. Проте, оскільки відомі підходи виявлення бот-мереж на основі аналізу DNS-трафіка не враховують певні важливі фактори, які вказують на функціонування бот-мереж, це призводить до великої кількості хибних спрацювань.

Недоліком традиційних методів виявлення бот-мереж, які ґрунтуються на аналізі трафіка на основі використовуваних портів, фільтрації пакетів, а також відомих сигнатур є можливість ухилення від виявлення шляхом динамічної зміни шкідливого коду, системи керування та портів або використання стандартних портів HTTP/S. Застосування повного аналізу вмісту пакетів є ресурсоємним в мережах з високим навантаженням, а застосування технологій фільтрування підвищує ризик пропуску шкідливих пакетів. Недоліком систем виявлення на основі сигнатур є схильність до високої ймовірності неспрацювань, що є основною причиною порушення безпеки, оскільки такі системи не здатні вирішувати проблеми «нульового дня». Метод систем-«приманок» – замкнених захищених контрольованих областей, що імітують вразливі мережі, ресурси або служби, застосовується передусім з метою спостереження та виявлення спроб вторгнення, дослідження мотивації та технологій, що використовуються злочинцем, збору сигнатур та прогнозування розвитку загроз. Проте, такий метод виявлення неефективний на ранньому етапі поширення інфекції і не піддається масштабуванню.

Як і методи на основі аналізу пакетів, методи виявлення бот-мереж на основі DNS надають можливість виявляти бот-мережі виключно на основі спостереження за роботою мереж, уникаючи прямої взаємодії з середовищем передачі даних та залишаючись непоміченим для зловмисника. Перевагами методів виявлення бот-мереж на основі DNS є порівняно невеликий обсяг трафіка, його висока інформативність, доступність даних для аналізу та відсутність в них конфіденційної інформації. Використання для аналізу лише DNS-трафіка дозволяє зменшити потребу в обчислювальних ресурсах. Важливими перевагами методів на основі DNS є те, що, на відміну від сигнатурних методів, такий підхід здатен виявляти ще невідомі боти, а також можливість здійснення раннього виявлення бот-мереж – на стадії створення їх інфраструктури.

Одним з труднощів виявлення та блокування командно-контролюючих серверів (C&C-серверів) бот-мереж є використання ними технологій ухилення на основі DNS, таких як DNS-тунелювання, технологія «швидкозмінних» мереж (fast-flux service network), технологія «потік доменів» (domain flux) та періодична зміна IP-відображення для шкідливого домена [2-4]. Проте, використання цих технологій ухилення може бути виявлене шляхом аналізу

ознак, вилучених з корисного навантаження DNS-повідомлень. Також, ботам притаманна специфічна поведінка, що відрізняє DNS-запити інфікованих КС від легітимного DNS-трафіка. Це надає можливість виявляти шкідливі доменні імена та інфіковані КС на базі аналізу DNS-трафіка.

При організації процесу виявлення бот-мереж на основі аналізу DNS-трафіка виникає необхідність у забезпеченні однозначної ідентифікації КС в мережі. Оскільки IP-адреса не може бути надійним ідентифікатором для КС в мережі, то в якості ідентифікаторів КС доцільно використовувати їх MAC-адреси за умови забезпечення запобігання підміни MAC-адрес. Для цього виникає необхідність доступу до мережного обладнання та контролю над ним, а саме до керованих комутаторів мережі. Тому в роботі розглядається саме корпоративна комп'ютерна мережа.

В процесі забезпечення функціонування корпоративної мережі виникає протиріччя між двома важливими показниками: інформаційною безпекою корпоративної мережі та оперативністю роботи ІС корпоративної мережі, які можуть бути знижені у зв'язку із впливом на них ще невідомих ботів. У випадку виявлення факту інфікування ІС ботом, вона повинна бути заблокована з метою видалення ШПЗ та усунення наслідків його впливу на систему, що призводить до зниження оперативності роботи ІС. Оскільки хибне спрацювання також призводить до блокування ІС, то для підвищення оперативності роботи ІС корпоративної мережі потрібно знижувати рівень хибних спрацювань, відкидаючи певну частину підозрілих об'єктів аналізу. Проте з метою підвищення рівня інформаційної безпеки корпоративної мережі потрібно забезпечити відповідну реакцію на такі підозрілі об'єкти, зменшуючи рівень пропусків ШПЗ. Отже, з метою підвищення оперативності роботи ІС корпоративної мережі, а також рівня достовірності виявлення бот-мереж виникає потреба у зниженні рівня хибних спрацювань.

Тому, для усунення недоліків відомих підходів запропоновано нові методи виявлення бот-мереж в корпоративних мережах, засновані на аналізі DNS-трафіка, які враховують нову базову ознаку синхронності DNS-запитів та залучають методи інтелектуального аналізу даних, що дозволяє підвищити рівень достовірності виявлення невідомих ботів та оперативність роботи ІС корпоративної мережі.

Зв'язок роботи з науковими програмами, планами, темами. Дослідження, представлені в дисертації, проводились в рамках держбюджетної НДР Хмельницького національного університету №4Б-2015 «Розвиток наукових та інженерних основ надійності електронної техніки шляхом удосконалення технології її тестування на вібрації та удари», виконавцем якої була автор дисертації.

Мета і завдання дослідження. Метою дисертаційної роботи є розроблення інформаційної технології виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка для підвищення достовірності виявлення бот-мереж.

Задачі дослідження формулюються в роботі наступним чином:

1) дослідити особливості функціонування бот-мереж з врахуванням системи доменних імен та розробити відповідні модель бот-мереж з врахуванням системи доменних імен, модель DNS-трафіка та модель процесу виявлення бот-мереж на основі аналізу DNS-трафіка;

2) проаналізувати сучасні інформаційні технології виявлення бот-мереж на основі аналізу DNS-трафіка з метою визначення шляхів підвищення достовірності виявлення бот-мереж;

3) розробити метод ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку, який дозволить ідентифікувати інфіковані комп'ютерні системи, що здійснюють таку активність;

4) розробити метод виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, який дозволить виявляти інфіковані комп'ютерні системи в мережі, що належать до бот-мереж, які використовують технології ухилення на основі DNS;

5) розробити інформаційну технологію виявлення бот-мереж на основі аналізу DNS-трафіка та дослідити достовірність виявлення бот-мереж;

6) розробити алгоритми та реалізувати інформаційну технологію виявлення бот-мереж на основі аналізу DNS-трафіка у вигляді програмного забезпечення та впровадити її у виробництво з метою підвищення достовірності виявлення бот-мереж у корпоративних мережах.

Об'єктом дослідження є процес виявлення бот-мереж у корпоративних мережах на основі аналізу DNS-трафіка.

Предметом дослідження є моделі, методи та програмні засоби інформаційної технології виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка.

Методи дослідження. Для розв'язання поставлених задач використовуються основні положення системного аналізу, методів аналізу даних, теорії мір близькості та теорії комп'ютерних мереж.

Наукова новизна одержаних результатів. В результаті дисертаційного дослідження розв'язано актуальну наукову задачу розроблення інформаційної технології виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка та одержано наступні наукові результати:

1) набула подальшого розвитку модель процесу виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка. Розроблена модель відрізняється від відомих моделей тим, що задіює розроблені модель DNS-трафіка та модель бот-мереж з врахуванням системи доменних імен, що дозволило використання цієї моделі для виявлення вже відомих та нових бот-мереж;

2) вперше розроблено метод ідентифікації бот-мереж у корпоративних мережах на основі їх групової активності в DNS-трафіку, який, на відміну від відомих, уможливлює уточнений поділ періоду моніторингу на інтервали, в межах яких здійснюється пошук груп інфікованих комп'ютерних систем, що ґрунтується на основі аналізу значень TTL, які містяться в DNS-повідомленнях, використовує нову ознаку синхронності DNS-запитів, а також враховує особливості поведінки груп інфікованих комп'ютерних систем, характерні для багатьох видів бот-мереж, що дозволило підвищити достовірність виявлення бот-мереж в порівнянні з відомими антивірусними програмними засобами;

3) вперше розроблено метод виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, у корпоративних мережах, який ґрунтується на залученні кластерного аналізу множини ознак, одержаних з корисного навантаження DNS-повідомлень, які вказують на використання таких технологій ухилення, що дозволило підвищити достовірність виявлення бот-мереж в порівнянні з відомими антивірусними програмними засобами;

4) набула подальшого розвитку інформаційна технологія виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка, яка дозволяє

ідентифікувати боти, що здійснюють групову активність в DNS-трафіку, а також виявляти боти бот-мереж, які застосовують технології ухилення від виявлення на основі DNS. Застосування розробленої інформаційної технології дозволяє підвищити достовірність процесу виявлення бот-мереж в порівнянні з відомими інформаційними технологіями та виявляти нові бот-мережі.

Обґрунтованість і достовірність наукових положень, висновків і рекомендацій. Наукові положення, висновки та рекомендації дисертаційної роботи забезпечуються:

- обґрунтованим викладенням матеріалу теоретичних розробок та коректністю використаних математичних методів;

- апробацією результатів дисертаційних досліджень на наукових конференціях та публікаціями у фахових журналах теоретичного матеріалу;

- результатами проведених експериментів та ефективним практичним впровадженням.

Практичне значення одержаних результатів. В результаті виконаного дисертаційного дослідження розроблено програмне забезпечення інформаційної технології виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка. Дослідження проводились з метою їх подальшої практичної реалізації. Результати експериментальних досліджень з використанням розробленого програмного забезпечення підтверджують вірність наукових положень запропонованої інформаційної технології, оскільки впровадження інформаційної технології підвищує достовірність діагностування на 8-22% у порівнянні з відомими програмними засобами виявлення бот-мереж.

Теоретичні та практичні результати роботи впроваджено:

– державне підприємство «Новатор», відділ автоматизованих систем управління (м. Хмельницький, акт про впровадження від 1.06.2016 р.);

– товариство з обмеженою відповідальністю «ITT - telecommunication company» (м. Хмельницький, акт про впровадження від 24.05.2016 р.);

– у навчальному процесі Хмельницького національного університету при викладанні дисциплін «Програмування комп’ютерних мереж», «Технічна діагностика і надійність комп'ютерних пристроїв та систем» та «Інженерія програмного забезпечення» (акт про впровадження від 5.05.2016 р. ХНУ).

Особистий внесок здобувача. Всі основні результати дослідження, які виносяться на захист, отримані автором особисто. В роботах, опублікованих одноосібно автором, отримано наступні результати: розроблено модель інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка [30], методи та програмне забезпечення інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка [31]. В роботах, опублікованих у співавторстві, автору належать основні ідеї, теоретична та практична розробка положень, відображених у характеристиці наукової новизни отриманих результатів, а саме:

- проведено аналіз відомих методів виявлення бот-мереж на основі DNS [32-34];

- розроблено моделі бот-мережі, DNS-трафіка та модель процесу виявлення бот-мереж в мережах на основі аналізу ознак, які можуть бути вилучені з DNS-трафіка [35];

- розроблено новий метод ідентифікації бот-мереж на основі аналізу DNS-трафіка, який ґрунтується на властивості групової активності ботів в DNS-трафіку та враховує анормальну поведінку груп комп'ютерних систем, властиву для бот-мереж [36, 37];

- розроблено новий метод виявлення бот-мереж в корпоративних мережах на основі пасивного моніторингу DNS-трафіка та активного DNS-зондування з використанням кластерного аналізу векторів ознак, вилучених з корисного навантаження DNS-повідомлень, що дозволило здійснювати виявлення бот-мереж, які використовують технології ухилення від виявлення на основі DNS [38-43];

- розроблено інформаційну технологію виявлення бот-мереж на основі аналізу DNS-трафіка, яка побудована на базі двох нових методів: методу ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку та методу виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS [44];

- розроблено концептуальну схему локалізації бот-мереж в частині застосування інформації з різних комп`ютерних систем мережі для аналізу трафіку з метою виявлення аномальної поведінки комп'ютерних систем, властивої для бот-мереж [45].

Апробація результатів дисертації. Основні положення та результати проведених у дисертаційній роботі досліджень доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях, а саме:

- XII Міжнародна конференція «Контроль і управління в складних системах» (м. Вінниця, 2014 р.);

- Міжнародна науково-практична конференція молодих вчених та студентів «Інформаційне, програмне та технічне забезпечення систем управління організаційно-технологічними комплексами» (м. Луцьк, 2015 р.);

- Computer Networks 22th International Conference (Брунов, Польща, 2015 р.);

- 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (Варшава, Польща, 2015 р.);

- III Міжнародна конференція з автоматичного управління та інформаційних технологій (м. Київ, 2015 р.);

- Міжнародний науково-практичний семінар молодих вчених та студентів «Програмовані логічні інтегральні схеми та мікропроцесорна техніка в освіті і виробництві» (м. Луцьк, 2016 р.);

- XVI Міжнародна наукова конференція ім. Т. А. Таран «Інтелектуальний аналіз інформації» (м. Київ, 2016 р.);

- Cyber Forum DESSERT B2S-S2B (м. Чернівці, 2016 р.);

- Computer Networks 23th International Conference (Брунов, Польща, 2016 р.);

- науково-практичні конференції професорсько-викладацького складу Хмельницького національного університету 2014-2016 рр.

Публікації. Основні матеріали дисертації викладено в 16 наукових публікаціях, з них 6 статей, опублікованих у наукових фахових виданнях України (2 з яких є одноосібними), 3 – у виданнях, зареєстрованих в наукометричній базі Index Copernicus, і 2 – у періодичних закордонних виданнях, зареєстрованих у наукометричній базі Scopus, 1 з яких – у виданні, зареєстрованому в наукометричній базі Web of Science, 8 – у матеріалах конференцій, з них 1 – індексоване у наукометричних базах Scopus та Web of Science, та 1 патент на корисну модель.

Структура дисертації. Дисертація складається зі вступу, чотирьох розділів та висновків, викладених на 150 сторінках основного тексту, списку використаних джерел (138 найменувань). Робота містить 47 рисунків, 7 таблиць та 3 додатки.

РОЗДІЛ 1
АНАЛІЗ ВІДОМИХ ТЕХНОЛОГІЙ ВИЯВЛЕННЯ БОТ-МЕРЕЖ НА ОСНОВІ DNS

1.1 Поняття та типи бот-мереж

Розвиток інформаційних технологій протягом останніх десятиліть, анонімність та швидкість передачі інформації в мережі Інтернет призвели до появи нового виду злочинності – злочинності в кіберпросторі [46-58]. За даними «Лабораторії Касперського» [59], за минулий 2015 рік було зафіксовано 125 262 075 унікальних зразків шкідливих та потенційно небажаних об'єктів, на 34,2% комп'ютерів користувачів Інтернету щонайменше одноразово було здійснено веб-атаки, для чого зловмисниками було використано 6 563 145 унікальних КС. За даними щорічного звіту компанії Symantec щодо загроз в мережі Інтернет [60], за минулий рік її дослідники виявили більш ніж 430 мільйонів унікальних зразків ШПЗ. Кількість вразливостей, що використовувались хакерами, подвоїлась; кількість вразливостей нульового дня збільшилась на 125% порівняно з 2014 роком.

Одним з основних і стабільних джерел доходів для кіберзлочинців на сьогоднішній день є бот-мережі. Щороку по всьому світу бот-мережами інфікується близько 500 млн. КС, кожну секунду – близько 18 [61]. Незначні фінансові витрати та відсутність необхідності у високому рівні спеціальних знань для створення та керування бот-мережею призводять до зростання кількості кіберзлочинів, здійснених із застосуванням бот-мереж [62-65].

Створення в 1999 р. перших програм Backdoor [66] NetBus та BackOrifice2000, що містили повний набір функцій віддаленого керування інфікованою КС, уможливило об'єднання ізольованих окремих одиниць ШПЗ в єдину мережу під управлінням зловмисника – бот-мережу. Отже, бот-мережа – це система контролю над комп'ютерними системами, інфікованими ШПЗ Backdoor (ботом), що дозволяє зловмиснику анонімно та віддалено керувати інфікованими КС, встановлюючи на них необхідну фонову задачу, та використовувати ресурси КС без відому користувача. Бот-мережі мають

динамічну географічно розподілену структуру та можуть об’єднувати ресурси від десятків тисяч до мільйонів інфікованих комп’ютерів, тому володіють потужними обчислювальними ресурсами та небезпечним потенціалом.

Бот-мережі використовуються для здійснення DDoS-атак (розподілені атаки типу «відмова в обслуговуванні»), підбір паролів, поширення ШПЗ, викрадення конфіденційних даних, організації анонімних проксі-серверів, застосування засобів нав’язування реклами, фішингу, поширення спаму, надання сервісу віддалених машин, накрутки клік-лічильників (клікфрод), використання інфікованих КС для зберігання нелегального матеріалу тощо. Бот-мережі використовуються не лише як інструмент для злочинного або нелегального заробітку в мережі Інтернет, а також і як засіб кібершантажу та методів політичного тиску шляхом блокування серверів державних установ.

Більшість ботів бот-мереж є багатофункційними додатками, орієнтованими на певну ОС, що дозволяють реалізовувати широкий асортимент шкідливих дій та з метою поширення і протидії виявленню можуть поєднувати в собі функціонал файлових вірусів, троянців, мережних черв`яків, руткітів.

Найбільш поширеними способами розповсюдження шкідливих ботів є [11]: (1) інфікування електронною поштою із застосуванням методів соціальної інженерії; (2) інфікування через миттєві повідомлення із застосуванням методів соціальної інженерії та вразливостей системи; (3) за допомогою вірусів-черв’яків із використанням віддалених вразливостей ПЗ; (4) завантаження шкідливого коду при відвідуванні веб-сторінок; (5) маскування під корисне ПЗ.

Виявлення та блокування шкідливих процесів ботів на інфікованих КС ускладнюється використанням множини механізмів захисту [67]: (1) протидія відлагодженню; (2) застосування технік обфускації та криптування; (3) використання руткіт-технологій; (4) зараження головного завантажувального запису MBR та зберігання шкідливих модулів поза межами таблиці розділів або організація шифрованого віртуального диску; (5) маскування під системний процес або виконання в контексті інших запущених процесів; (6) використання двох процесів, які самоперезапускаються та перезапускають один одного з метою унеможливлення примусового завершення шкідливого процесу; (7) підміна системних файлів; (8) перезавантаження КС при спробах доступу до

шкідливих виконуваних файлів або ключів автозавантаження, в яких ці файли прописані; (9) використання нестандартних методів запуску; (10) блокування фаєрволів для забезпечення встановлення безперешкодного зв'язку між ботом та C&C-сервером; (11) завершення процесів антивірусного ПЗ тощо.

За типом мережних протоколів, що використовуються для зв'язку з метою забезпечення керування бот-мережею, бот-мережі поділяють на наступні види [11]: (1) IRC-орієнтовані – один з перших видів, де керування здійснюється на основі Internet Relay Chat; (2) IM-орієнтовані – малопоширений вид, для керування використовуються канали служб Instant Messaging (наприклад, AOL, MSN, ICQ); (3) веб-орієнтовані – керування здійснюється через www; (4) інші – керування здійснюється на основі власного протоколу, базуючись на стеку протоколів TCP/IP.

Основною точкою відмови для бот-мереж є використовувана ними архітектура. За архітектурою виокремлюють наступні типи бот-мереж: (1) бот-мережі з єдиним (або декількома) центром керування (C&C, Command&Control center) ; (2) децентралізовані бот-мережі, або P2P-бот-мережі (від англ. «peer-to-peer» – «рівний до рівного»); (3) гібридні бот-мережі.

Централізовані бот-мережі характеризуються простотою створення та керування, високою швидкістю реакції на команди. Для нейтралізації централізованої бот-мережі потрібно ліквідувати її C&C-сервери. В децентралізованих бот-мережах кожен бот з'єднується не з центром керування, а з декількома інфікованими КС мережі, і при отриманні команди від однієї з них передає її решті КС. Для керування децентралізованою бот-мережею зловмиснику необхідний доступ до будь-якої КС, що входить до бот-мережі. Побудова такої бот-мережі складніша, швидкість реакції на команди для неї нижча. Основною перевагою децентралізованих бот-мереж є відсутність центру керування, що ускладнює боротьбу з такими бот-мережами. В гібридній бот-мережі частина ботів, які мають статичні неприватні IP-адреси, та є доступними з глобальної мережі, виступають в ролі клієнтів та серверів одночасно; інша група ботів виступає лише в ролі клієнтів, оскільки вони мають динамічні або приватні IP-адреси, або знаходяться за брандмауерами, тому не доступні для вхідних з'єднань з глобальної мережі. Перевагою гібридних бот-мереж є

поєднання надійності розподілених бот-мереж та наявності багатьох контролюючих центрів. Від'єднання від бот-мережі частини керуючих вузлів може зменшити функціональність, проте не керованість такою бот-мережею.

Використання для організації ботнет-комунікацій DNS дозволяє злочинцям підвищити надійність бот-мережі та анонімність керування нею. Використання сервісів динамічної DNS або власних розподілених DNS-сервісів надає можливість трансформувати C&C-архітектуру з централізованої на розподілену або гібридну топологію (рис. 1.1) [2].

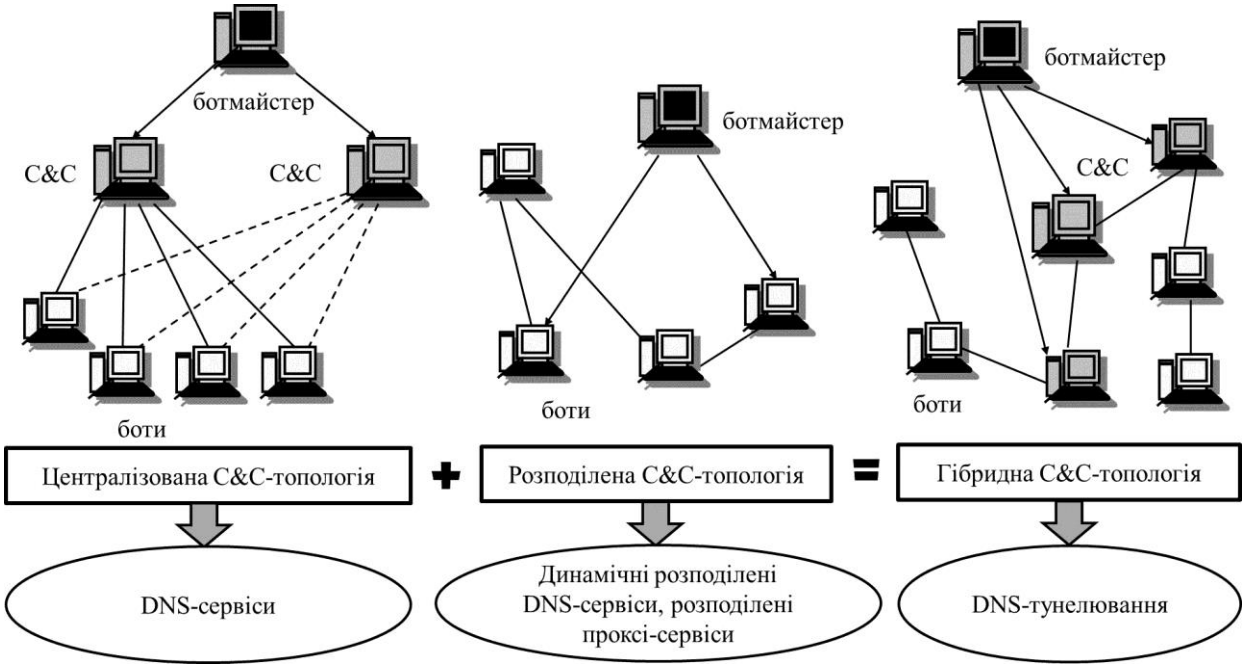


Рисунок 1.1 – Трансформування C&C-архітектури

DNS є всесвітньою ієрархічною розподіленою комп'ютерною системою, що слугує для пошуку та ідентифікації доменів в мережі Інтернет. Сервіс системи доменних імен побудований за схемою «клієнт-сервер», де в якості клієнтської частини виступає прикладний процес, що запитує інформацію щодо відповідності доменного імені IP-адресі або навпаки.

Одна IP-адреса може бути співставлена з великою кількістю доменних імен, що надає можливість забезпечення віртуального хостингу, тобто підтримки на одному вузлі множини веб-сайтів. Також одне доменне ім'я може бути співставлене з множиною IP-адрес з метою забезпечення балансування навантаження (*load balancing*) між мережними пристроями.

Детальна інформація щодо концепції та впровадження DNS подана в RFC-1034 [68] та RFC-1035 [69]. DNS складається з трьох основних компонентів: (1) простір доменних імен та ресурсні записи, або записи DNS (одиниці зберігання та передачі інформації в DNS, *Resource Records – RR*), що визначають деревоподібну структуру простору імен та дані, пов'язані з цими іменами; (2) сервери імен (DNS-сервери, *name servers*) – програмні комплекси, розташовані на спеціалізованих серверах, що містять інформацію щодо структури доменних дерев та блоки даних; (3) DNS-клієнти (*resolvers*) – програмні модулі, що виступають в якості інтерфейсів між програмними комплексами користувачів і DNS-серверами, та отримують інформацію з серверів імен у відповідь на запити клієнтів.

Сервер імен може зберігати в кеш-пам'яті структуру дерева або набір даних щодо будь-якої частини доменного дерева, володіє повною інформацією про підмножину доменного простору (є достовірним джерелом даних (*authority*) для цієї частини доменного дерева) та містить покажчики на інші сервери імен, що можуть бути використані для отримання інформації з будь-якої іншої частини доменного дерева. З метою передачі відповідальності за домен (делегування) певній особі або організації частини доменного дерева виокремлюються в зони.

Оскільки DNS не належить жодній спеціальній організації та не контролюється, DNS-трафік передається між клієнтами та DNS-сервером без захисту та обмежень. Ця властивість активно використовується кіберзлочинцями. У випадку використання зловмисником для керування ботами бот-мережі DNS, для доступу до C&C-сервера інфіковані КС виконують DNS-запити з метою встановлення відповідності між доменним ім'ям C&C-сервера та його IP-адресою. В більшості випадків зовнішньо ініційовані з'єднання з ботами неможливі (наприклад, інфіковані КС знаходяться за брандмауерами, або їх IP-адреси невідомі чи динамічні, або використовується перетворення (трансляція) мережних адрес (Network Address Translation, NAT).

В якості доменного імені C&C-сервера ботмайстер може зареєструвати повне доменне ім'я за допомогою шахрайських способів або скористатися безкоштовним 3LD (піддоменом третього рівня, *third-level subdomain*) від

DDNS-провайдера (*Dynamic Domain Name System*). З метою зменшення витрат та ризиків, ботмайстри, як правило, уникають створення нового домена та використовують піддомени, створюючи доменні імена під спільним SLD (доменом другого рівня, *second-level domain*, наприклад, example.com) в 3LD (наприклад, botnetserver.example.com). Це забезпечує підвищення надійності функціонування бот-мереж за рахунок створення надмірних DDNS-сервісів, оскільки в разі, якщо обслуговування в одному з 3LD призупиняється, обслуговування інших 3LD в межах SLD зазвичай не порушується [70].

Приклад ієрархічної організації простору імен DNS та структури доменного імені відображено на рис. 1.2. Згідно стандарту RFC 2606 [71], який визначає імена доменів верхнього та другого рівня, зарезервованих для використання в якості прикладів в документації та тестування, для прикладу доменного імені C&C-сервера бот-мережі використано ім’я домену третього рівня botnetserver.example.com.

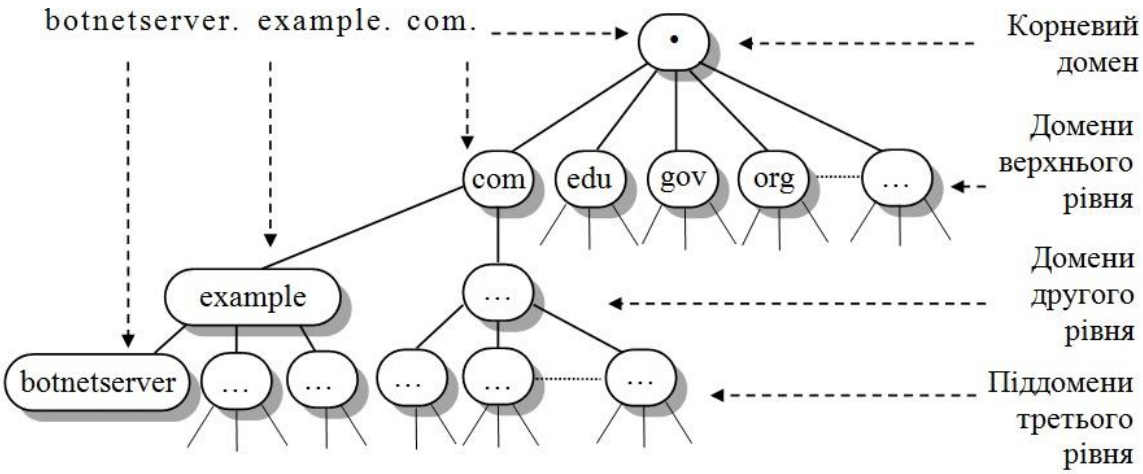


Рисунок 1.2 – Приклад ієрархічної організації простору імен DNS та структури доменного імені

Якщо рекурсивний DNS-сервер зберігає в кеші дані щодо запитуваного доменного імені, то він повертає їх DNS-клієнту. Інакше він послідовно виконує нерекурсивні запити до інших DNS-серверів: (1) до корневого DNS-сервера, який повертає IP-адресу сервера, що є відповідальним за домен верхнього (першого) рівня (*top-level domain, TLD*) – зону «com.»; (2) до DNS-сервера, що є відповідальним за зону «com.», який повертає IP-адресу сервера,

що є відповідальним за домен другого рівня (SLD) «example.com.»; (3) до DNS-сервера, що відповідає за зону «example.com.», де отримує IP-адресу доменного імені третього рівня (3LD) botnetserver.example.com., яку і повертає DNS-клієнту. Відповідь записується в кеш рекурсивного DNS-сервера та DNS-клієнта, додатково кешуються проміжні результати пошуку на період часу, що визначається значенням TTL (*Time To Live*, час життя), яке містилось у відповіді від авторитетного DNS-сервера на DNS-запит. Значення TTL задається в конфігурації опису зони та є 32-бітним беззнаковим полем (від 0 – не кешувати до 2^{32}), та дозволяє визначити, як довго (в секундах) мають утримуватись в кеші записи DNS (RR) [69]. На такий тип запиту DNS-клієнту буде повернута IP-адреса або порожня відповідь та код помилки NXDOMAIN (*Non-Existent Domain*). Узагальнену схему процесу встановлення відповідності між доменним ім’ям C&C-сервера та його IP-адресою подано на рис. 1.3 (на схемі послідовність запитів представлено позначеннями 1…9).

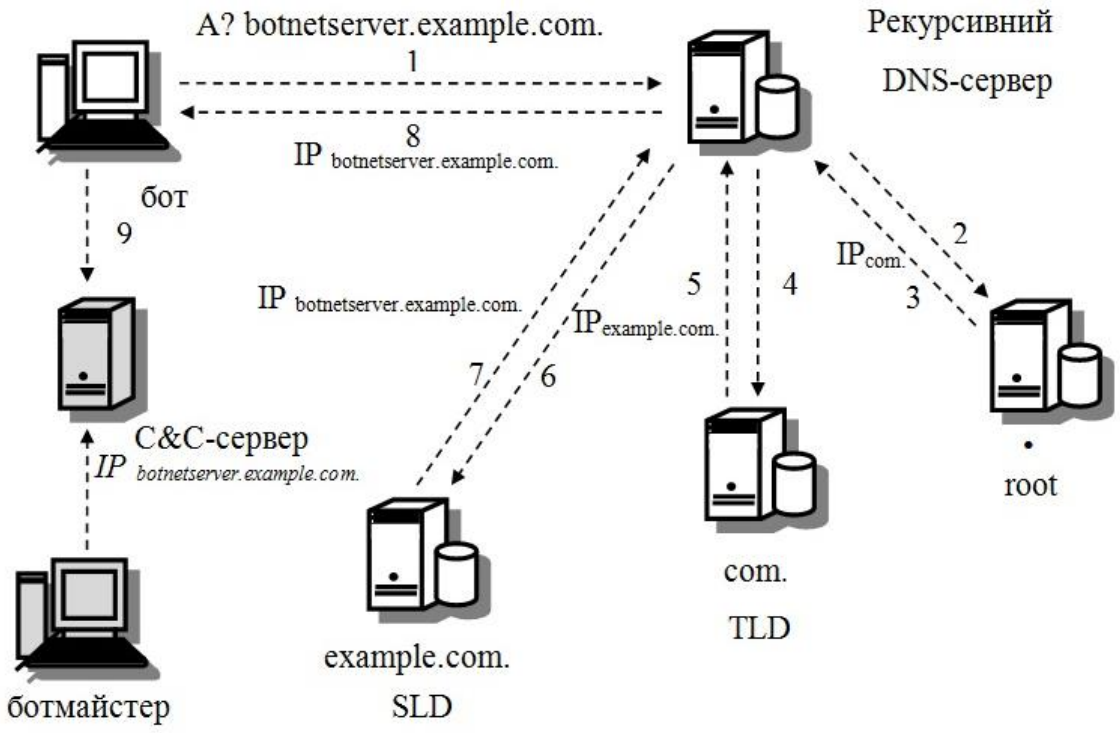


Рисунок 1.3 – Узагальнена схема процесу встановлення відповідності між доменним ім’ям C&C-сервера та його IP-адресою

В разі блокування C&C-сервера мережним адміністратором ботмайстер може налаштувати інший C&C-сервер, використавши DDNS для переміщення доменного імені від заблокованого C&C-сервера на новий з іншою IP-адресою. З метою підвищення завадостійкості бот-мережі окрім прямих спроб з'єднання через процедуру перетворення доменного імені боти також надсилають зворотні DNS-запити для отримання додаткових доменних імен C&C-сервера.

Злочинцями застосовується ряд технологій ухилення від виявлення бот-мереж, які базуються на використанні системи доменних імен: DNS-тунелювання (*DNS-tunneling*), «швидкозмінні» мережі (*fast-flux service network*), технологія (*domain flux*) та періодична зміна IP-відображення для шкідливого домена (*cycling of IP mapping*) [2, 70, 72].

Найбільш простим методом ухилення від занесення до «чорних списків» та блокування є періодична зміна IP-відображення для доменного імені C&C-сервера та встановлення для нього коротких TTL-періодів [70, 72]. Для мінімізації ймовірності порушення функціонування бот-мережі ботмайстер використовує надмірну архітектуру з великою кількістю C&C-серверів, а також регулярно змінює локацію C&C-сервера. Крім того, адреса шкідливого домена може бути встановлена на певний час на адресу, що не маршрутизується, наприклад, на приватну адресу RFC 1918 [73].

На відміну від серверів, що використовуються для шахрайських дій, легітимні сервери зазвичай не змінюють IP-адреси, і тому використовують довгий час кешування (стандартний TTL-період визначено в RFC 1912 [74] як 86400 – 432000 с., або 1-5 діб), що знімає навантаження на авторитетні сервери та забезпечує швидкість DNS-відповідей. Для легітимного сервера TTL-період може бути зменшено перед міграцією сервера або зміною IP-адреси. Зменшення значення TTL-періоду для шкідливого домена забезпечує максимальну кількість інфікованих КС, що мігрують на C&C-сервер. В разі використання довгих TTL-періодів C&C-сервер повинен утримуватись від зміни локації протягом інтервалу часу TTL, інакше кількість активних інфікованих КС в бот-мережі зменшуватиметься з кожною міграцією сервера [70].

Технологія «потік доменів» поєднує короткі TTL-періоди та часті зміни доменного імені C&C-сервера [2]. При застосуванні технології «потік доменів»

автоматична генерація доменного імені може здійснюватися алгоритмічно – комбінація буквено-цифрових символів обирається випадковим чином згідно з алгоритмом генерації доменних імен, вбудованим в тіло бота (*domain generation algorithm, DGA*), або за допомогою комбінацій слів з словника. Можливість занесення до «чорних списків» доменних імен, отриманих за допомогою зворотного інжинірингу бота, долається зловмисником шляхом генерації великої кількості доменних імен. Оскільки в якості дійсних доменних імен C&C-сервера бот-мережі використовується лише незначна частина, і час життя цих доменних імен дуже короткий, це забезпечує прикриття для доменного імені, яке фактично використовується для C&C-сервера. Більшість з алгоритмічно створених доменних імен є недоступними в якості доменних імен C&C-сервера бот-мережі, тому боти генерують велику кількість невдалих DNS-запитів, що призводить до появи в DNS-трафіку NXDOMAIN-відповідей (порожніх DNS-відповідей з кодом помилки RCODE=3, доменне ім’я не існує). Прикладами бот-мереж, які використовують DGA, є Conficker [75-77], Kraken [78, 79], Torpig [80, 81], Bobax [79], Srizbi та BackDoor.Flashback [82].

Високі відмовостійкість, доступність та можливість ухилятись від методу занесення до «чорних списків» DNS (*DNSBL, DNS blacklist*) забезпечує бот-мережам поєднання використання дуже коротких TTL-періодів та циклічного методу round-robin DNS. Round-robin DNS є методом розподілення навантаження та забезпечення відмовостійкості за рахунок надлишковості кількості серверів, коли для відповіді на запити використовується список IP-адрес серверів, що надають ідентичний сервіс, причому послідовність IP-адрес з кожною DNS-відповіддю змінюється. Прикладом таких систем є «швидкозмінні» мережі (*fast-flux service network*) [3], зокрема такі відомі спамерські бот-мережі з функціоналом викрадення конфіденційних даних та здійснення DDoS-атак як Kelihos/Hlux [83] та Waledac/Iksmas [77].

Виокремлюють два типи таких мереж – однопоточні (*single-flux*) та двопоточні (*double-flux*). Однопоточні мережі дозволяють використовувати для повного доменного імені велику кількість пов’язаних з ним IP-адрес, які швидко змінюються, що надає змогу пов’язувати доменне ім’я з новою множиною IP-адрес у визначеному інтервалі часу. Таким чином, при

підключенні до C&C-сервера, наприклад, кожні три хвилини, боти насправді з'єднуватимуться з різними інфікованими вузлами. З метою підвищення безпеки такі скомпрометовані КС, що виконують роль flux-агентів (або flux-ботів), використовуються для TCP- та UDP-переадресації «наосліп» (*blind proxy redirection*) запитів та даних від та до C&C-серверів в якості проксі-серверів, коли множина IP-адрес, що циклічно змінюються, не є кінцевим призначенням для запиту. Переадресація запобігає спробам відстеження вузлів такої мережі. З метою забезпечення стабільної роботи бот-мережі, також здійснюється регулярна перевірка стану вузлів та виключення неактивних вузлів з потоку. Двопоточні мережі забезпечують додатковий рівень захисту за рахунок надмірності: для шкідливого домена змінюються циклічно за алгоритмом round-robin множина DNS A-записів (*A-records*) та NS-записи (*NS-records*), що дозволяє зловмисникам створювати власні розподілені проксі-сервіси.

Застосування DNS-тунелювання дозволяє зловмиснику передавати довільний трафік в межах специфікації протоколу DNS, всередині полів DNS-повідомлення. Це надає можливість використовувати протокол DNS в якості носія для трафіку командування та контролю над бот-мережею [2]. Навіть, якщо КС не має доступу до мережі Інтернет, то сервіс DNS зазвичай є доступним. Це надає можливість боту передавати DNS-запити через локальний або інший DNS-сервер на зовнішній сервер, підконтрольний зловмиснику. В спрощеному вигляді такий механізм здійснення обміну даними може бути описаний наступним чином. Бот надсилає DNS-запит, наприклад, для перетворення доменного імені на IP-адресу. Запитане доменне ім'я складається з двох частин, наприклад, *xxx.z.example.com*, де *xxx* є закодованою інформацією, яку необхідно передати, а *z.example.com* є доменним ім'ям сервера зловмисника. Сервер зловмисника є авторитетним сервером імен для зони, щодо доменних імен якої здійснюються запити ботами. Таким чином, будь-який DNS-запит щодо доменних імен цієї зони буде пересилатись на сервер зловмисника. Після надходження DNS-запита на підконтрольний зловмиснику DNS-сервер закодована інформація, що міститься в запитаному доменному імені, відокремлюється та розшифровується. Зловмисник надсилає боту DNS-відповідь, яка також містить закодовані дані, наприклад, команду для наступних

дій у вигляді CNAME-відповіді: *yyy.z.example.com*, де *yyy* є закодованою командою. Оскільки клієнти зазвичай знаходяться за брандмауерами та не мають служб для прослуховування DNS-запитів, то для підтримки зв'язку з C&C-сервером боти повинні регулярно опитувати сервер зловмисника.

Існує багато технологій DNS-тунелювання, які використовують різні методи кодування та різні типи DNS-записів (як загальновживані, так і рідко використовувані) для передачі інформації. Прикладами бот-мереж, які застосовували цю технологію ухилення, є Morto [84], Feederbot [16], Katusha/Timestamper [85].

Розглянемо типовий механізм організації поширення ШПЗ та встановлення C&C-каналу на прикладі бот-мережі BackDoor.Flashback [67, 82]. Інфікування троянською програмою Backdoor.Flashback.39 здійснюється з використанням вразливостей Java. При відвідуванні користувачем інфікованого сайту за допомогою системи розподілу трафіку (TDS, Traffic Direction System) користувач перенаправляється на шкідливий сайт. Шкідлива сторінка містить Java-скрипт, який завантажує в браузер користувача Java-аплет, який, у свою чергу, містить експлойт. Експлойт зберігає на жорсткому диску інфікованої КС виконуваний файл, який встановлює зв'язок з керуючим сервером, перевіряє одержане від сервера повідомлення на відповідність підпису RSA, і у разі успішного результату перевірки скачує з керуючого сервера та запускає ШПЗ.

Кожен бот реєструється в бот-мережі, передаючи керуючому серверу унікальний ідентифікатор інфікованої КС. Для зв'язку з керуючим сервером використовується DGA, вбудований в тіло бота. Цим видом ШПЗ було інфіковано більш ніж 800000 вузлів під керуванням Mac OS [67, 82].

Отже, бот-мережі є об'єднанням комп'ютерних систем, інфікованих ШПЗ, в єдину мережу під управлінням зловмисника. Ресурси інфікованих КС можуть бути використані з метою здійснення нелегальних, шкідливих та злочинних дій в кіберпросторі, а також виступати в якості об'єкту продажу та оренди, що перетворює бот-мережі на одне з найбільш прибуткових джерел злочинних доходів в мережі Інтернет. Бот-мережі здатні нанести значних економічних втрат як власникам інфікованих ботами КС, так і користувачам комп'ютерних систем та ресурсів, на які було спрямовано атаку бот-мережі.

1.2 Огляд програмних засобів виявлення бот-мереж

Найбільш поширеними антивірусними програмними засобами, призначеними, зокрема, і для виявлення бот-мереж, на сьогоднішній день є [86]: Avira, Avast!, AVG, Bitdefender, Dr. Web, ESET, Kaspersky, McAfee, Microsoft, Symantec, Panda. Розглянемо функційні можливості цих антивірусних засобів, орієнтованих на виявлення ШПЗ в корпоративних мережах.

Avira Small Business Security Suite (Avira Operations GmbH & Co. KG, Німеччина) [87] призначений для захисту кінцевих систем корпоративної мережі (від 5 до 100 КС та серверів вузлів під керуванням Windows). Цей АЗ здатний виявляти відомі типи ШПЗ та приховані процеси, які керують системою, а також здійснює моніторинг мережного трафіка. Особливостями цього засобу є: застосування евристичної технології проактивного захисту AHeAD, що дозволяє перевіряти невідомі коди на наявність підозрілих ознак; застосування алгоритмів навчання для захисту від атак нульового дня; застосування хмарних технологій; виявлення невідомих загроз на основі відслідковування поведінки підозрілих об’єктів.

Avast Endpoint Protection Suite (Plus) (AVAST Software, Чехія) [88] призначений для захисту КС корпоративної мережі, включаючи файлові та поштові сервери (від 5 до 1000 вузлів). Особливостями цього АЗ є застосування інтелектуального сканування SmartScan, безпечного ізольованого середовища для відкриття підозрілих сайтів та можливість запуску підозрілих додатків в ізольованій «пісочниці» (Sandbox).

AVG Internet Security Business Edition (AVG Technologies, Нідерланди) [89] – це АЗ для захисту кінцевих систем (а також поштових і файлових серверів) на платформі Windows в корпоративній мережі. Це АПЗ використовує «хмарний» захист в реальному часі для виявлення нового ШПЗ з використанням технології інтелектуального «хмарного» сканування (Real-time Outbreak Detection), а також інтелектуальну технологію AI Detection для ідентифікації нових загроз в режимі реального часу.

Антивірусне рішення BitDefender Corporate Security (Bitdefender, Румунія) [90] поєднує продукти BitDefender в єдиний засіб для захисту серверів та робочих станцій великої корпоративної мережі від відомих типів ШПЗ. З метою захисту від невідомого ШПЗ використовується евристична технологія у віртуальному оточенні (HiVE), яка емулює віртуальну машину всередині КС та здійснює запуск підозрілого ПЗ для перевірки на наявність поведінки, властивої для ШПЗ.

Dr.Web CureNet! (Doctor Web, Росія) [91] – АПЗ, що здійснює віддалене та централізоване діагностування КС під управлінням Windows в локальній мережі на наявність ШПЗ з використанням, зокрема, превентивного захисту, заснованого на технології несигнатурного (поведінкового) блокування.

Kaspersky Endpoint Security (Kaspersky Lab, Росія) [92] – це АЗ, орієнтований на захист кінцевих систем під управлінням Windows в корпоративній мережі в реальному часі від всіх типів ШПЗ. Kaspersky Endpoint Security містить наступні основні компоненти захисту та контролю: (1) файловий антивірус; (2) моніторинг системи; (3) поштовий антивірус; (4) фаєрвол; (5) контроль запуску та активності програм за визначеним набором правил; (6) моніторинг вразливостей системи в режимі реального часу; (7) контроль зовнішніх пристроїв; (8) веб-контроль доступу до ресурсів мережі Інтернет, який дозволяє створювати групи користувачів з визначеними наборами правил та обмежень доступу до Інтернет-ресурсів.

ESET Endpoint Security (ESET, Словаччина) [93] є АЗ для корпоративних мереж малого та середнього бізнесу. Цей антивірусний засіб базується на хмарних технологіях, застосовує проактивний захист HIPS (Host-based Intrusion Prevention System, система запобігання вторгненням) на основі аналізу поведінки об’єктів, а також використовує технологію сканування на основі бази даних репутації файлів. Захист від бот-мереж здійснюється на основі аналізу схем обміну даними та протоколів.

Symantec Endpoint Protection (Symantec, США) [94] призначений для захисту кінцевих систем корпоративної мережі під управлінням ОС Windows, Mac та Linux (від 100 вузлів). Це АПЗ засноване на технології Insight, що визначає безпечність файлів і додатків, використовуючи зворотний зв'язок

користувачів антивірусних рішень Symantec, а також застосовує поведінковий аналіз та антивірусний сканер на основі агресивного евристичного виявлення.

Panda Endpoint Protection (Panda Security, Іспанія) [95] – АЗ для забезпечення захисту робочих станцій та серверів під керуванням ОС Windows, Linux, Mac, а також Android. Цей АЗ заснований на поведінковому та евристичному аналізі та використовує хмарну базу знань, що містить дані про шкідливі (*malware*) та нешкідливі (*goodware*) файли та процеси, а також інформацію щодо дій, які потрібно виконати стосовно підозрілого об’єкта.

MAC [56] – ПЗ для діагностування КС корпоративних мереж на наявність бот-мереж, що ґрунтується на використанні мультиагентної системи, агенти якої здійснюють висновок щодо інфікування мережі на основі діагностичної інформації від інших агентів шляхом використання системи нечіткого логічного висновку. Висновок стосовно інфікування КС мережі ботом здійснюється на основі аналізу поведінки програмного об’єкту в КС та обміні діагностичною інформацією між активними агентами, встановленими на інших КС корпоративної мережі.

Розглянуті антивірусні засоби побудовані з використанням ІТ, які ґрунтуються на застосуванні синтаксичних сигнатур, контрольних сум, поведінкового аналізу та евристичних методів. Застосування методів поліморфізму та метаморфізму призводить до неможливості виявлення ШПЗ на основі побудови синтаксичних сигнатур. З метою уникнення виявлення евристичними аналізаторами розробники ШПЗ тестують новостворені віруси на можливість виявлення найбільш поширеними засобами антивірусного діагностування. З іншого боку, евристичні методи виявлення ШПЗ засновані на емпіричних припущеннях, отриманих на основі знань щодо сигнатур відомого ШПЗ та можливих механізмів поліморфізму цих сигнатур. Тому, недоліком евристичних методів є висока ймовірність хибних спрацювань у випадках наявності в ПЗ частин коду, що виконують властиві ШПЗ дії або їх послідовності.

Оскільки суть методів виявлення ШПЗ, на яких побудована робота розглянутих антивірусних засобів, є комерційною таємницею, то оцінка достовірності роботи АПЗ може бути сформована виключно на основі аналізу

результатів тестування. Для проведення експерименту було згенеровано множину потенційно невідомого для антивірусних засобів ПЗ з функційним навантаженням ботів бот-мереж різних типів (в загальній кількості 400 зразків) та проведено тестування розглянутих АЗ на достовірність виявлення ними створеного ШПЗ. З метою здійснення порівняльного аналізу достовірності виявлення бот-мереж антивірусними засобами було використано дані рейтингів тестування АПЗ, надані британським журналом Virus Bulletin [96] та німецьким незалежним інститутом AV-TEST [97].

Результати рейтингів тестування від AV-TEST та Virus Bulletin (VB 100), а також результати тестування, отримані в ході проведеного експерименту, наведено в таблиці 1.1 та на рис. 1.4.

Таблиця 1.1

Порівняльний аналіз тестування АПЗ

№ зп	АПЗ	Virus Bulletin, VB 100 [96]	AV-TEST [97]	Виявлено ботів бот-мереж, %
		Виявлено бот-мереж, %		
1.	Avast Endpoint Protection Suite	90	86	82
2.	AVG Internet Security Business Edition	82	81	74
3.	Avira Small Business Security Suite	89	85	81
4.	BitDefender Corporate Security	82	85	77
5.	Dr.Web CureNet!	-	80	79
6.	ESET Endpoint Security	87	83	82
7.	Kaspersky Endpoint Security	89	92	84
8.	McAfee Endpoint Protection Suite	86	84	80
9.	Microsoft System Center Endpoint Protection	80	76	73
10.	Panda Endpoint Protection	83	79	75
11.	Symantec Endpoint Protection	84	87	76
12.	MAC	-	-	88

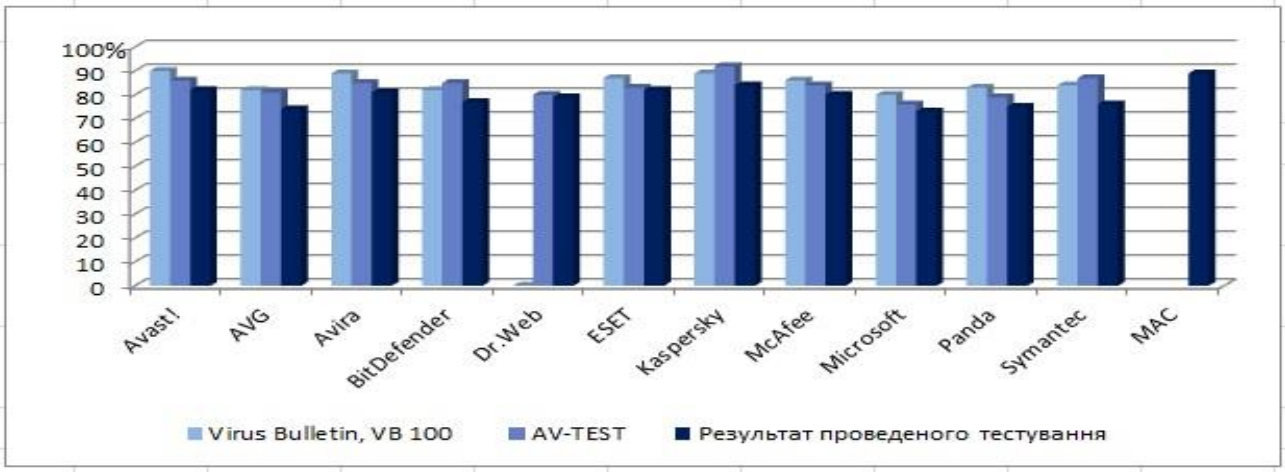


Рисунок 1.4 – Порівняльний аналіз тестування АПЗ

Отже, дані результатів тестування АПЗ, надані сторонніми ресурсами, а також дані, отримані в результаті проведеного експерименту, свідчать про завищення результатів оцінки достовірності виявлення ШПЗ як виробниками антивірусних засобів, так і їх тестувальниками, а також показують недостатньо високу достовірність виявлення нових бот-мереж найбільш поширеними антивірусними засобами.

1.3 Аналіз методів виявлення бот-мереж на основі DNS

На сьогодні відомо ряд методів виявлення бот-мереж на основі DNS.

В [12] побудовано моделі, які описують динаміку активності бот-мереж в залежності від часових зон та допомагають передбачати зростання чисельності популяції бот-мережі та спалахи бот-епідемій. Інфіковані КС, що входять до складу бот-мережі, можуть бути розосереджені по всьому світу і, як правило, знаходяться в різних часових зонах. Оскільки користувачі зазвичай вимикають комп'ютери вночі, в цей час боти проявляють меншу активність. Після перезавантаження боти знову виконують DNS-запити. Таким чином, щогодини до мережі підключається частина ботів, тому в графіку активності бот-мереж за країнами походження інфікованих комп'ютерів чітко проявляється добовий шаблон (pattern), утворений значним підвищенням обсягів рекурсивного DNS-трафіку.

В [8] запропоновано динамічну систему оцінки репутації доменних імен, яка використовує три групи ознак для побудови моделей легітимних та шкідливих

доменів – мережні, зональні та доказові ознаки, на основі яких здійснюється оцінка репутації домена. Мережні ознаки доменів (загальна кількість IP-адрес, пов'язаних з доменним ім'ям; їх географічна локація; кількість різних номерів автономних систем (ASN) для них тощо) та зональні ознаки (середня довжина доменних імен; кількість різних доменів верхнього рівня для них; частота, з якою з'являються в доменному імені різні символи тощо) одержуються на базі аналізу DNS-запитів. Доказові ознаки базовані на даних «чорних списків» та систем-«приманок» і дозволяють визначити, в якій мірі домен пов'язаний з відомими шкідливими доменними іменами або IP-адресами.

В [98] запропоновано евристику для виявлення DNSBL-розвідувальної діяльності ботмайстра та складання списків ймовірних ботів. З метою визначення, чи знаходяться спам-боти в «чорному списку», ботмайстер виконує DNSBL-запити. DNSBL (*DNS blacklist*) є списками хостів, що зберігаються з використанням системи DNS та застосовуються для боротьби зі спамом. Згідно методу будується граф DNSBL-запитів, де ребро від вузла A до B вказує на те, що вузол A надсилав запит з метою визначення, чи є вузол B у списку DNSBL. На базі графа визначено очікувані просторову та часову характеристики легітимних пошуків в порівнянні з розвідувальними пошуками, що допомагають відрізнити DNSBL-запити ботмайстра від запитів легітимних поштових серверів. Кількісне визначення просторових відносин визначається за формулою:

$$\lambda_n = \frac{d_{n,out}}{d_{n,in}}, \tag{1.1}$$

де λ_n – відношення пошуку для вузла n;

$d_{n,out}$ – кількість унікальних IP-адрес, що були запитані вузлом n;

$d_{n,in}$ – кількість унікальних IP-адрес, що запитували вузол n.

Високий показник λ_n свідчить про виконання хостом розвідки. Якщо $d_{n,in} = 0$ (що зазвичай має місце), λ_n розглядається як велике число. Часові відносини полягають в тому, що легітимний поштовий трафік має тенденцію бути добовим. Тому, легітимні запити, спричинені приходом реальної

електронної пошти, можна відрізнити від розвідувальних запитів, оскільки останні не відображатимуть реалістичних шаблонів (patterns) надходження електронної пошти. Недоліком методу є те, що він передбачає використання одного хоста для вхідного та вихідного поштового серверів. Проте, в великих мережах вони можуть бути розділені, і тоді запити від вхідного поштового сервера можуть бути розцінені, як спроба розвідки.

В [99] запропоновано хост-орієнтований підхід, заснований на аналізі поведінки ботів і не ботів, пов’язаної з реакцією на DNS-відповіді. З метою створення надмірності та підвищення завадостійкості бот-мережі, окрім прямих спроб з'єднання через процедуру перетворення доменного імені, боти виконують зворотні DNS-запити для отримання додаткових доменних імен C&C-сервера. При цьому IP-адреси неуспішних зворотних DNS-запитів ігноруються доброякісним ПЗ, проте часто використовуються ботами та іншим ШПЗ. Визначено чотири підозрілі процеси, що можуть мати місце в RD-поведінці (*reaction-to-DNS response behavior, RD-behavior*): (1) успішне пряме перетворення імені, підключення не відбулось – аномальний процес; (2) успішне зворотне перетворення імені, підключення не відбулось – притаманний для ШПЗ, в тому числі ботів; (3) неуспішне зворотне перетворення імені, підключення відбулось – притаманний для ШПЗ, в тому числі ботів; (4) неуспішне зворотне перетворення імені, підключення не відбулось – домінуючий для ботів.

В [100] визначено наступні особливості бот-мереж, що характеризують їх поведінку: (1) зв’язки (відношення) ботів – зв’язок між власником бот-мережі (ботмайстром) та ботами – один до багатьох. Ботмайстер контролює множину ботів та видає одну команду для всіх, тому боти виявляють групову поведінку, що дозволяє виявити активність ботів шляхом моніторингу мережного трафіка; (2) синхронізація ботів – боти отримують одну команду від ботмайстра, підтримують зв’язки між собою та атакують одночасно, що дозволяє виявити групу ботів за допомогою високого показника трафіку та в деяких випадках – його дискретності в часі; (3) відповіді ботів – бот відповідає на команди ботмайстра та одразу виконує їх. Необхідний для цього час (час відгуку) постійний, що також може бути використано для виявлення.

Ряд методів [14, 15, 101] ґрунтуються на властивості групової активності бот-мереж в DNS-трафіку.

Підхід [101] передбачає моніторинг, захоплення DNS-трафіка в різних часових інтервалах та вимірювання відношення подібності між будь-якими двома групами КС, що запитують одне й те саме доменне ім’я. Для обчислення значення подібності між групами КС використовується коефіцієнт Жаккара. Недоліком методу є те, що він спирається на групові запити лише однакових доменних імен, не враховуючи міграцій C&C-серверів та інших DNS-запитів, пов’язаних з діяльністю бот-мережі.

Підхід [14], орієнтований на виявлення ботів в мережах класу C, для обчислення подібності між двома групами КС використовує коефіцієнт Кульчинського. З метою виявлення міграцій C&C-серверів бот-мереж порівнюються списки IP-адрес КС, що запитували різні доменні імена, але які подібні за розмірами в межах 10%. Недоліками цього підходу є значне зростання часу обробки та потреба у великих обсягах обчислювальних ресурсів при застосуванні до великих мереж, недостатня гнучкість механізму виявлення міграцій C&C-серверів та пов’язаних з функціонуванням бот-мережі DNS-запитів.

З метою визначення множини доменних імен бот-мережі, і таким чином виявлення міграцій C&C-серверів, підхід [15] використовує кластеризацію методом x-середніх (*x-means*) ознак, вилучених з DNS-трафіка. Виокремлено 3 групи таких ознак: (1) засновані на DNS-лексикології; (2) засновані на інформації, вилученій з DNS-запитів; (3) засновані на інформації, вилученій з DNS-відповідей. Перша група ознак об’єднує наступні ознаки, вилучені з доменного імені: (1) кількість міток в доменному імені; (2) середня довжина мітки домена; (3) найбільша довжина мітки домена; (4) наявність домена другого рівня, занесеного до «чорних списків». Друга група містить ознаки: (1) кількість надісланих запитів щодо доменного імені; (2) кількість різних IP-адрес КС, що надсилали запити; (3) кількість різних номерів автономних систем (ASN), до яких належать IP-адреси КС, що надсилали запити; (4) тип запиту (A, NS, CNAME, MX, PTR); (5) оцінка подібності груп КС для домена. До третьої групи віднесено ознаки: (1) кількість різних повернутих IP-адрес доменного імені; (2) кількість різних номерів автономних систем, до яких належать IP-

адреси доменного імені; (3) кількість різних країн локації IP-адрес доменного імені; (4) значення поля TTL в DNS-відповіді.

З метою обчислення значення подібності між двома групами КС використовується косинусний коефіцієнт. Недоліками методу, запропонованого в [15], є те, що він орієнтований на виявлення ботів у великих розподілених мережах, тому не придатний для невеликих локальних мереж. Коротка тривалість періоду моніторингу (одна година), зумовлена зменшенням ймовірності зміни динамічних IP-адрес як ідентифікаторів КС в мережі, призводить до неспроможності виявлення групових запитів, якщо повторний запит групи КС відбувся поза межами періоду моніторингу.

Спільним недоліком підходів [14, 15, 101] є довільний поділ періоду моніторингу на інтервали, в межах яких здійснюється пошук груп інфікованих КС, що призводить до зменшення рівня виявлення. Іншим недоліком описаних методів є використання для порівняння двох груп КС симетричних мір подібності (Жаккара, Кульчинського, косинусного коефіцієнта), що є доцільним для оцінки подібності рівновеликих груп, тому може призводити до хибних спрацювань.

В [102] представлено підхід, який використовує виявлення групових неуспішних DNS-запитів ботів в мережі, що здійснюють неуспішні спроби мережних з'єднань та мають подібний розмір корисного навантаження пакетів при спробі мережного з'єднання. З метою визначення шкідливих доменних імен та IP-адрес використовується адаптивна мережа нечіткого висновку. Недоліком методу є необхідність у додатковому зборі та аналізі TCP-трафіка.

В [11] запропоновано метод виявлення C&C-серверів бот-мереж за аномально високими або сконцентрованими в часі рівнями DDNS-запитів. Метод пов'язує рівні надходження DNS-запитів щодо доменів другого рівня з рівнями DNS-запитів щодо піддоменів третього рівня:

$$C_{SLD_i} = R_{SLD_i} + \sum_{j=1}^{|SLD_i|} R_{3LD_j}\ , \qquad (1.2)$$

де R_{SLD_i} та R_{3LD_j} – рівні запитів щодо домена другого рівня та піддомена третього рівня відповідно;

C_{SLD_i} – канонічний рівень DNS-запитів (*Canonical DNS Request Rate, CDRR*) щодо домена другого рівня.

Таким чином, можна оцінити рейтинг доменів на основі обсягу трафіка, надісланого щодо піддерева батьківської зони, що дозволяє відрізняти легітимний трафік від трафіка бот-мереж. Якщо рівень CDRR для доменного імені є аномальним згідно нерівності Чебишева, доменне ім'я, ймовірно, є C&C-сервером. Також, запропоновано виявлення C&C-серверів за аномальним розподілом CDRR доменних імен в часі. Для цього часові CDRR для кожного домена другого рівня протягом дня сортуються в порядку спадання, утворюючи вектор ознак. Доменні імена, вектори ознак для яких відрізняються від нормального більше, ніж на порогове значення, ймовірно, є C&C-серверами. Для вимірювання відстані між векторами ознак використано відстань Махаланобіса.

Також, запропоновано оцінку загальної чисельності популяції бот-мережі за методом аналізу замкнених систем Лінкольна-Петерсена за двома вибірками ботів, виявлених через певний проміжок часу:

$$\hat{N} = \frac{(M+1)(C+1)}{R+1} + 1, \tag{1.3}$$

де $\hat{N}$ – оцінка загальної чисельності популяції бот-мережі;

M та C – загальна кількість ботів, виявлених та промаркованих в першій та другій вибірках відповідно;

R – кількість ботів другої вибірки, що були промарковані в перший раз.

Якщо $M + C \geq N$, де N – загальна чисельність популяції бот-мережі, оцінка популяції $\hat{N}$ є неупередженою навіть при невеликих розмірах вибірки. Для нормального розподілу для $\hat{N}$ може бути підраховано 95% довірчий інтервал для популяції $\hat{N} \pm 1{,}96\sqrt{V}$, де

$$V = \frac{(M+1)(C+1)(M-R)(C-R)}{(R+1)^2(R+2)}. \quad (1.4)$$

Проте, метод Лінкольна-Петерсена не може надати об'єктивної оцінки чисельності бот-мережі, оскільки він відповідає моделі замкненої популяції, яка передбачає, що за період дослідження ймовірності зустрічі екземплярів не змінюються, а загальна та промаркована чисельності екземплярів залишаються постійними, не враховуючи припинення існування та появи нових екземплярів.

В [103] надано експериментальну оцінку двох відомих підходів до виявлення C&C-серверів бот-мереж на основі аномального DDNS-трафіка – заснованих на виявленні аномально високих або сконцентрованих в часі обсягах запитів доменних імен та аномально високому рівні NXDOMAIN-відповідей. Відзначено, що багато з популярних доменів використовують короткі TTL-періоди (наприклад, *gmail.com*, *yahoo.com*, *mozilla.com*), тому підходи першої групи оцінюють їх як C&C-сервери бот-мереж. Аномальні домени другого рівня насправді мають більш рівномірний розподіл CDRR в часі, ніж нормальні, що суперечить [11]. Підходи на основі виявлення аномального DDNS-трафіка мають розглядати лише DDNS-запити. Проте, відокремлення DDNS-запитів від інших DNS-запитів складне. Низьке значення TTL – необхідна, проте недостатня умова для DDNS-відповіді, що може бути властивим також для легітимних доменних імен. Деякі легітимні та популярні доменні імена також використовують послуги DDNS-провайдерів. Нездатність виключити доменні імена, що не використовують DDNS, сприяє хибним спрацюванням. Підходи, засновані на аномально високому рівні NXDOMAIN-відповідей, що є більш ймовірними для DDNS-імен, ніж для інших, є менш схильними до хибних спрацювань.

Існує також ряд підходів, спрямованих на виявлення використання бот-мережами технологій ухилення на основі DNS [4, 6, 16, 18, 104-125].

В [114] запропоновано використання сервісу пасивної реплікації DNS, що надає можливість виявлення застосування періодичної зміни IP-відображення для доменних імен C&C-серверів бот-мереж. Метод полягає в спостереженні

DNS-трафіка, зберіганні всіх завершених DNS-резолюцій та побудові частин файлу зони. Реплікація зберігається в базі даних, що надає можливість отримати відповідь на запит не лише про поточний стан домена, а й кожну встановлену для домена адресу. Недоліком методу є те, що такі сенсори корисні лише на ділянках мережі з передбачуваним високим трафіком.

В [115] запропоновано підхід, який дозволяє виявляти доменні імена бот-мереж, що використовують технологію «швидкозмінних» мереж. Висновок щодо шкідливості доменного імені здійснюється за рядом ознак, отриманих на основі аналізу A-, NS-, SOA- та BGP-запитів щодо доменного імені: (1) значення часу життя A-записів (TTL-періоду); (2) кількість IP-адрес, що відповідають доменному імені; (3) середня дистанція між IP-адресами, що відповідають доменному імені; (4) кількість різних номерів автономних систем (ASN), до яких належать IP-адреси, що відповідають доменному імені; (5) дистанція між IP-адресами серверів імен (*name server*, NS); (6) кількість NS-записів; (7) кількість різних номерів автономних систем для IP-адрес серверів імен; (8) значення таймера “retry” (визначає, як довго вторинний сервер імен повинен чекати перед тим, як зробити повторну спробу запиту первинного сервера щодо зміни серійного номера зони, якщо попередня спроба була невдалою). Недоліком методу є необхідність активного DNS-зондування, тому неможливість реалізації виключно на основі пасивного аналізу DNS-трафіка.

В [116] запропоновано систему виявлення шкідливих доменів на основі пасивного DNS-аналізу, що здійснює класифікацію доменних імен за чотирма групами ознак, які можуть бути вилучені з DNS-трафіка: (1) часові ознаки; (2) ознаки, що базуються на DNS-відповідях; (3) ознаки, що базуються на значеннях TTL; (4) ознаки, що базуються на доменному імені. Перелік ознак шкідливих доменних імен подано в табл. 1.2. Недоліком підходу є необхідність залучення інформації, отриманої від інших сервісів (WHOIS тощо).

Таблиця 1.2

Ознаки шкідливих доменних імен, які можуть бути вилучені на основі пасивного DNS-моніторингу

№ з/п	Група ознак	Назва ознаки
1.	Часові ознаки	коротка тривалість життя домена
2.		добова подібність в зміні кількості запитів з плином часу
3.		шаблони запитів, що регулярно повторюються
4.		відношення доступу – домен перебуває здебільшого в стані очікування, чи в стані доступу постійно (в разі популярного домена)
5.	Ознаки, що базуються на DNS-відповідях	кількість різних IP-адрес для домена протягом часу спостереження
6.		кількість різних країн локації для доменного імені
7.		кількість доменів, що спільно використовують IP-адресу
8.		результати зворотних DNS-запитів - кількість зворотних DNS-запитів, що повернули IP-адресу
9.	Ознаки, що базуються на значеннях TTL	середнє значення TTL-періоду
10.		стандартне відхилення TTL-періоду
11.		кількість різних значень TTL-періоду
12.		кількість змін TTL-періоду
13.		відсоток використання специфічних значень TTL-періоду
14.	Ознаки, що базуються на доменному імені	відсоток цифрових символів в імені домена
15.		відсоток довжини найдовшого значимого підрядка в доменному імені

В [17, 117] запропоновано метод виявлення бот-мереж на основі аналізу історії NS-записів. Ключова ідея підходу полягає в тому, що для більшості резолюцій доменних імен спочатку одержується відповідний запис NS від авторитетного сервера імен, в той час як підозрілі комунікації можуть не включати такої процедури для приховування своєї активності. Тому, з метою виявлення комунікацій бот-мереж здійснюється перевірка, чи включається ім'я сервера імен (IP-адреса), що відповідає DNS-запиту, до історії NS-записів.

В [118] запропоновано метод, який дозволяє здійснювати ідентифікацію легітимних доменів CDN (*Content Delivery Network*) та доменних імен C&C-серверів бот-мереж, які використовують технологію ухилення «швидкозмінних» мереж (*Fast Flux Service Networks, FFSN*). Метод ґрунтується

на виявленні ознак, отриманих на основі активного DNS-зондування, а саме співставлення домена з IP-адресами, які знаходяться в різних сегментах мережі, а також врахування коливань часу обробки запитів, що пов’язані з відмінностями у рівні продуктивності різних проксі-серверів в «швидкозмінних» мережах, для множини послідовних в часі DNS-запитів щодо певного доменного імені.

В [119] описано метод виявлення бот-мереж, що використовують технологію «потік доменів». Оскільки в таких бот-мережах генерація доменного імені зазвичай здійснюється з використанням DGA, то їх доменні імена характеризуються високою інформаційною ентропією. Метод заснований на аналізі успішних та невдалих DNS-запитів, представлених в інтервалі часу, що містить успішні запити. Невдалі DNS-запити можуть вказувати на наявність ботів на клієнтах, тоді як успішні запити, які відбуваються в часі поруч з невдалими, ймовірно пов’язані з неінфікованими КС. Для визначення IP-адрес, що належать бот-мережам, аналізуються зв'язані параметри ентропії доменних імен та кореляція в часі між успішними та неуспішними DNS-запитами. Ентропія доменів, що відповідають успішним DNS-запитам, визначається як середнє значення нормалізованої відстані Левенштейна (або дистанції редагування) для всіх пар таких доменних імен.

Перевагою методу є те, що він не покладається на високі рівні DNS-запитів. Недоліком методу є спирання на невдалі доменні імена, а також чутливість до вибору часового інтервалу, збільшення якого підвищує ймовірність хибних спрацювань, а зменшення знижує відсоток виявлення.

Методи виявлення використання DNS-тунелювання [4] можна розподілити на дві групи: методи, засновані на аналізі корисного навантаження DNS-повідомлень, та методи, засновані на аналізі DNS-трафіка. Ознаки, що вказують на застосування DNS-тунелювання, які використовуються відомими методами [4], подано в табл. 1.3. Оскільки тунельовані канали передають зашифровану інформацію, то такі повідомлення характеризуються високою ентропією. Обсяг переданих даних між C&C-сервером і ботом є значно більшим, порівняно зі звичайним використанням DNS. Це призводить до збільшення розміру повідомлень та/або до збільшення смуги пропускання між ботом та C&C-

сервером. Також, потік інформації між ботом та C&C-сервером є більш тривалим в часі. Спільним недоліком підходів [4] є зосередження на виявленні вузького кола ШПЗ, що використовує окремі технології DNS-тунелювання.

Таблиця 1.3

Групи методів виявлення застосування DNS-тунелювання
та використовувані ними ознаки

№ з/п	Група методів	Ознаки, які вказують на застосування DNS-тунелювання
1.	Аналіз корисного навантаження DNS-повідомлень	підвищена довжина запитаного імені хоста
2.		кількість унікальних символів в доменному імені
3.		висока ентропія запитаного доменного імені
4.		ознаки, засновані на частотному аналізі доменного імені
5.		використання рідковживаних типів записів DNS (KEY, NULL тощо), або таких, які зазвичай не використовуються клієнтами (наприклад, TXT, які найбільш часто використовуються для тунелювання)
6.		висока ентропія записів DNS, які містяться в DNS-повідомленнях (CNAME, TXT, NS, MX, KEY, NULL тощо)
7.		підвищена довжина DNS-повідомлення
8.	Аналіз DNS-трафіка	підвищена кількість та частота DNS-запитів
9.		підвищений обсяг DNS-трафіка від клієнта
10.		підвищений обсяг DNS-трафіка щодо домена
11.		велика кількість імен хостів для домена
12.		надмірна кількість NXDOMAIN-відповідей (є характерним для утиліти Heyoka)
13.		виявлення DNS-запитів, які не мають у відповідність наступного запиту за допомогою іншого додатку (наприклад, HTTP)

В [122] проведено аналіз можливостей використання DNS-запитів для встановлення прихованих комунікацій. Надано оцінку статистичних методів виявлення аномалій у вмісті DNS-пакетів шляхом порівняння ймовірнісних розподілів нормального та тунельованого DNS-трафіка. З метою висвітлення потенційної загрози розглянуто можливість здійснення контрзаходів з боку зловмисника.

В [123, 124] проведено аналіз великої кількості реальних DNS-запитів та обчислено порогові значення довжини запитаного імені хоста та кількості унікальних символів в ньому, що дозволяють відрізнити легітимний DNS-трафік від тунельованого.

В [16] описано бот-мережу Feederbot, в якій використовується технологія DNS-тунелювання. На відміну від відомих видів бот-мереж, C&C-сервери описаної бот-мережі не змінювали локацію протягом часу спостереження, і DNS-запити ботів не були синхронними в часі. Запропоновано метод виявлення цього класу ШПЗ, заснований на використанні двох ознак: (1) підвищена ентропія поля RDATA (застосовується міра ентропії Шеннона); (2) комунікативна поведінка, що відрізняється збільшенням розміру DNS-повідомлень, смуги пропускання та підвищеною тривалістю в часі. Підхід використовує кластеризацію цих ознак за методом k-середніх із застосуванням Евклідової відстані для виявлення такого типу ботів.

Отже, з огляду на надзвичайно великий шкідливий функціонал бот-мереж, а також застосування ними технологій ухилення різних типів, актуальною є задача розробки нових методів і інформаційної технології виявлення бот-мереж в корпоративних комп’ютерних мережах, які б враховували особливості поведінки ботів бот-мережі в DNS-трафіку, а також дозволили б консолідувати переваги методів, заснованих на пасивному моніторингу DNS-трафіка та активному DNS-зондуванні, в єдину систему з метою підвищення ефективності виявлення бот-мереж.

1.4 Постановка задачі

Представлені в результаті проведеного дослідження недоліки методів і засобів, на яких засновані відомі ІТ виявлення бот-мереж в корпоративних комп’ютерних мережах, а також антивірусних засобів свідчать про невідповідність можливостей відомих методів та засобів вимогам, які до них висуваються, оскільки вони не здатні здійснювати виявлення нових ботів бот-мереж з високою достовірністю.

Разом з тим, розробники бот-мереж володіють сучасними технічними

засобами для розробки бот-мереж та супроводження ШПЗ, які ускладнюють виявлення бот-мереж відомими антивірусними засобами та знижують його достовірність.

Враховуючи переваги підходів виявлення бот-мереж на основі DNS, важливою задачею є розробка нових методів виявлення бот-мереж в мережах на основі аналізу DNS-трафіка, які уможливлять виявлення як вже відомих, так і нових ботів бот-мереж. Для підвищення достовірності процесу виявлення бот-мереж необхідно вирішити наступні задачі:

1) проаналізувати сучасні інформаційні технології виявлення бот-мереж на основі аналізу DNS-трафіка з метою визначення шляхів підвищення достовірності виявлення;

2) дослідити особливості функціонування бот-мереж з врахуванням системи доменних імен та розробити відповідні модель бот-мереж з врахуванням системи доменних імен, модель DNS-трафіка та модель процесу виявлення бот-мереж на основі аналізу DNS-трафіка;

3) розробити метод ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку, який дозволить ідентифікувати інфіковані комп’ютерні системи, що здійснюють таку активність;

4) розробити метод виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, який дозволить виявляти інфіковані комп’ютерні системи в мережі, що належать до бот-мереж, які використовують технології ухилення на основі DNS;

5) розробити інформаційну технологію виявлення бот-мереж на основі аналізу DNS-трафіка та дослідити достовірність виявлення бот-мереж;

6) розробити алгоритми та реалізувати інформаційну технологію виявлення бот-мереж на основі аналізу DNS-трафіка у вигляді програмного забезпечення та впровадити її у виробництво з метою підвищення достовірності виявлення бот-мереж у корпоративних мережах.

РОЗДІЛ 2
МОДЕЛІ БОТ-МЕРЕЖ, DNS-ТРАФІКА ТА ПРОЦЕСУ ВИЯВЛЕННЯ БОТ-МЕРЕЖ НА ОСНОВІ АНАЛІЗУ DNS-ТРАФІКА

2.1 Визначення області дослідження

При організації процесу виявлення бот-мереж на основі аналізу DNS-трафіка важливим питанням є можливість однозначної ідентифікації комп'ютерних систем в мережі. IP-адреса КС може бути як статичною, так і динамічною, тобто призначатись автоматично при підключенні КС до мережі і використовуватись протягом обмеженого проміжку часу, який визначається сервісом, що призначав цю IP-адресу. Тому, IP-адреса не може бути надійним ідентифікатором для комп'ютерної системи в мережі.

Враховуючи вищесказане, в якості ідентифікаторів КС мережі використовуватимемо їх MAC-адреси за умови забезпечення запобігання підміни MAC-адрес (*MAC-spoofing*). Для цього виникає необхідність доступу до мережного обладнання та контролю над ним, а саме до керованих комутаторів мережі, які здатні виконувати функцію Port Security, що унеможливить підміну MAC-адрес шляхом контролю доступу КС до мережі на основі їх MAC-адрес та портів підключення, а також функцію дзеркалювання (*Port mirroring*) DNS-трафіка на підключені до дзеркалюючих портів комутаторів мережні давачі з метою збору DNS-трафіка мережі. Тому, в роботі розглядається саме корпоративна комп'ютерна мережа.

Отже, розглянемо корпоративну мережу, яка є мультисервісною мережею передачі даних, що керована єдиною організацією та призначена для задоволення її виробничих вимог, як об'єкт діагностування на наявність бот-мереж (рис. 2.1).

Інформаційні ресурси організації мають певну цінність, що може бути визначена матеріально, і тому вимагають захисту від загроз, які можуть призвести до порушення політики безпеки інформації, втрати інформації, а також нанесення збитків автоматизованим системам (АС) корпоративної

мережі. Загроза (threat) в [126] визначена як будь-які обставини або події, які можуть бути причиною порушення політики безпеки інформації та/або нанесення збитків АС.

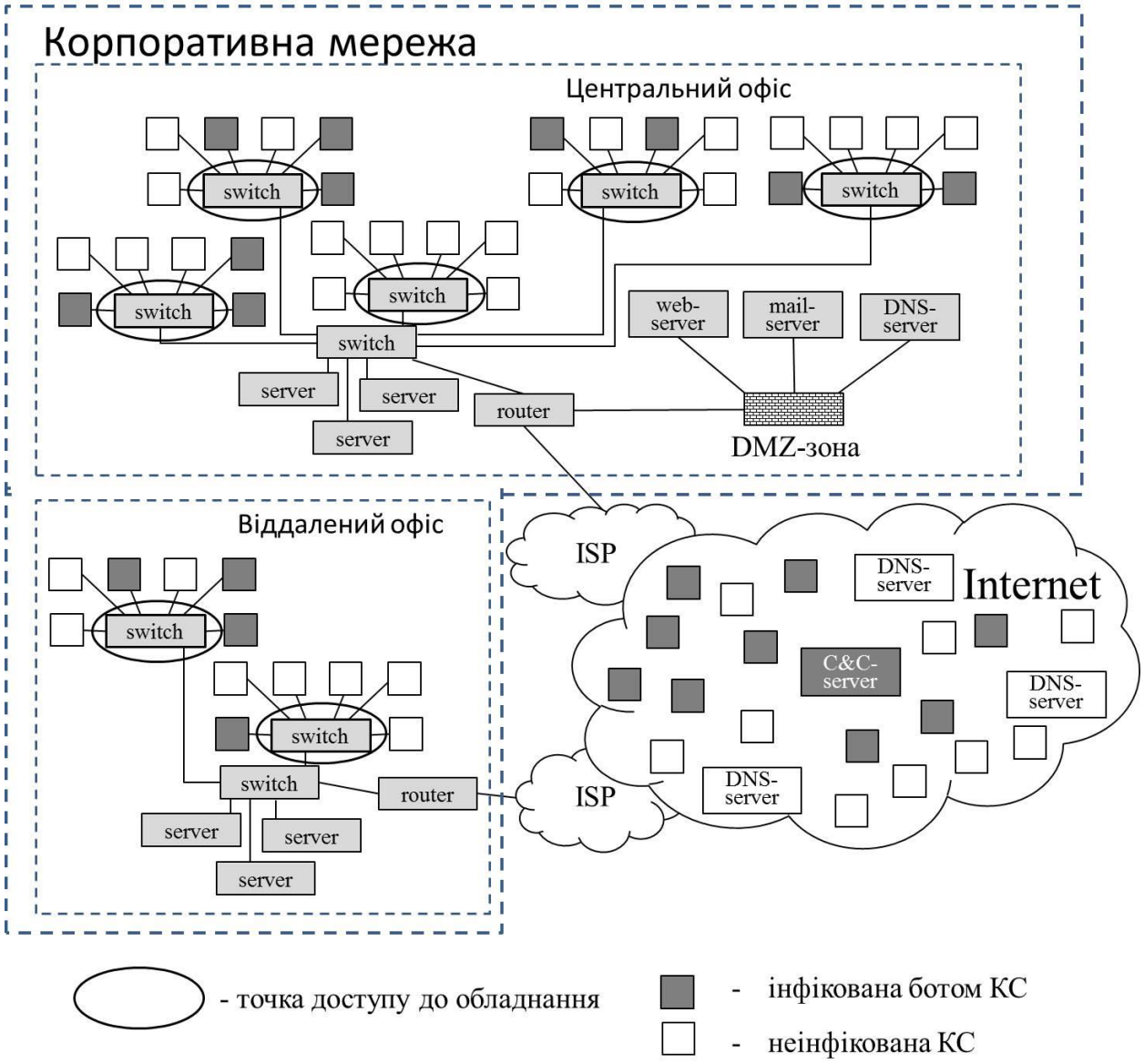


Рисунок 2.1 – Узагальнена схема корпоративної мережі як об’єкту діагностування на наявність бот-мереж

Джерелами загроз ДСТУ 3396.0-96 [127] визначено діяльність іноземних розвідок, а також навмисні або ненавмисні дії юридичних і фізичних осіб. Згідно [128], спроба реалізації загрози називається атакою. Уможливлення несанкціонованого доступу до корпоративної інформації в разі успішного здійснення атаки може призвести до значних фінансових втрат. Викрадені

конфіденційні дані, інформація щодо фінансової діяльності або така, що є інтелектуальною власністю організації, можуть бути використані з метою продажу або безпосереднього отримання прибутку.

За даними лабораторії Касперського [129], в 2015 році на 58% корпоративних комп'ютерів було спрямовано щонайменше по одній атаці ШПЗ, на 29% (тобто, майже на кожен третій корпоративний вузол) була спрямована щонайменше одна атака з глобальної мережі.

Враховуючи зростання кількості кіберзлочинів, здійснених із застосуванням бот-мереж [61], актуальною задачею є розробка моделей бот-мереж з метою виявлення та блокування відомих та нових ботів в корпоративних мережах.

2.2 Модель бот-мереж з врахуванням системи доменних імен

З метою організації процесу виявлення бот-мереж в мережах побудуємо модель бот-мереж з врахуванням системи доменних імен. Для побудови моделі бот-мережі розглянемо її життєвий цикл з врахуванням системи доменних імен.

Життєвий цикл бот-мережі розділимо на п'ять фаз: (1) інфікування; (2) первинна реєстрація або з'єднання; (3) здійснення шкідливої активності; (4) супроводження; (5) припинення функціонування бот-мережі. Схему життєвого циклу бот-мережі подано на рис. 2.2.

Кожен бот, який входить до складу бот-мережі, може здійснювати пошук вразливих КС з метою їх інфікування та розширення бот-мережі. На фазі інфікування відбувається подолання існуючих в КС механізмів захисту, виконання шкідливого скрипта (shell-коду) та встановлення програми-бота на інфікованій комп'ютерній системі.

Інфікування КС ботом може бути як наслідком проведення успішної атаки на етапі здійснення шкідливої активності, так і окремою фазою ЖЦ. Наприклад, одним з найпоширеніших способів інфікування є drive-by завантаження [66], коли інфікування КС здійснюється під час відвідування скомпрометованого веб-сайту, на якому розміщено шкідливий код.

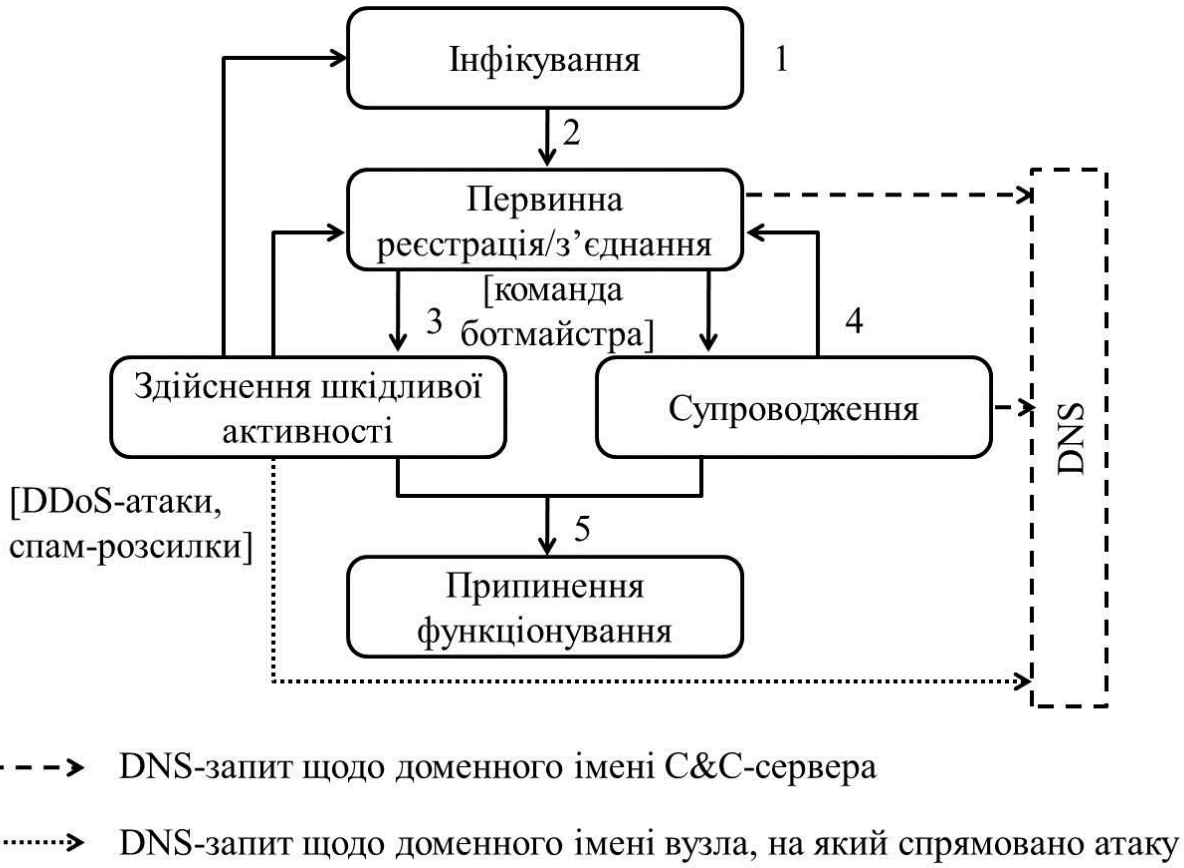


Рисунок 2.2 – Схема життєвого циклу бот-мережі з врахуванням системи доменних імен

Шкідливий код може бути завантажений на жорсткий диск КС або впроваджений у вигляді динамічних бібліотек в пам'ять легітимного процесу без збереження на диску. В цьому випадку шкідливий процес існує лише в ОП інфікованої КС до перезавантаження системи (так званий «безтілесний» бот). Здійснення інфікування з використанням популярного ресурсу, який відвідується користувачем регулярно, надає можливість багаторазового інфікування КС та утримування її в бот-мережі.

В разі успішного інфікування КС бот ініціює DNS-запит з метою встановлення відповідності між доменним ім'ям C&C-сервера та його IP-адресою. Використовуючи DNS-клієнт (*stub resolver*), інфікована КС надсилає рекурсивному DNS-серверу рекурсивний DNS-запит щодо доменного імені C&C-сервера. В разі успішного DNS-запиту бот здійснює первинну реєстрацію в бот-мережі та встановлює канал зв'язку з C&C-сервером – C&C-канал. Після спливання TTL-періоду DNS або перезавантаження КС бот знову ініціює DNS-

запит і оновлює записи локального кешу DNS та кешу рекурсивного DNS-сервера.

На відміну від неінфікованих КС в локальній мережі, які зазвичай використовують локальні DNS-сервери для DNS-запитів, інфіковані ботами КС можуть використовувати також безкоштовні сервіси DNS (OpenDNS, FreeDNS) або власні DNS-сервери. На рис. 2.3 подано узагальнену схему встановлення зв’язку між ботами бот-мережі та C&C-сервером з використанням DNS (на схемі послідовність запитів та відповідей представлено позначеннями 1…10).

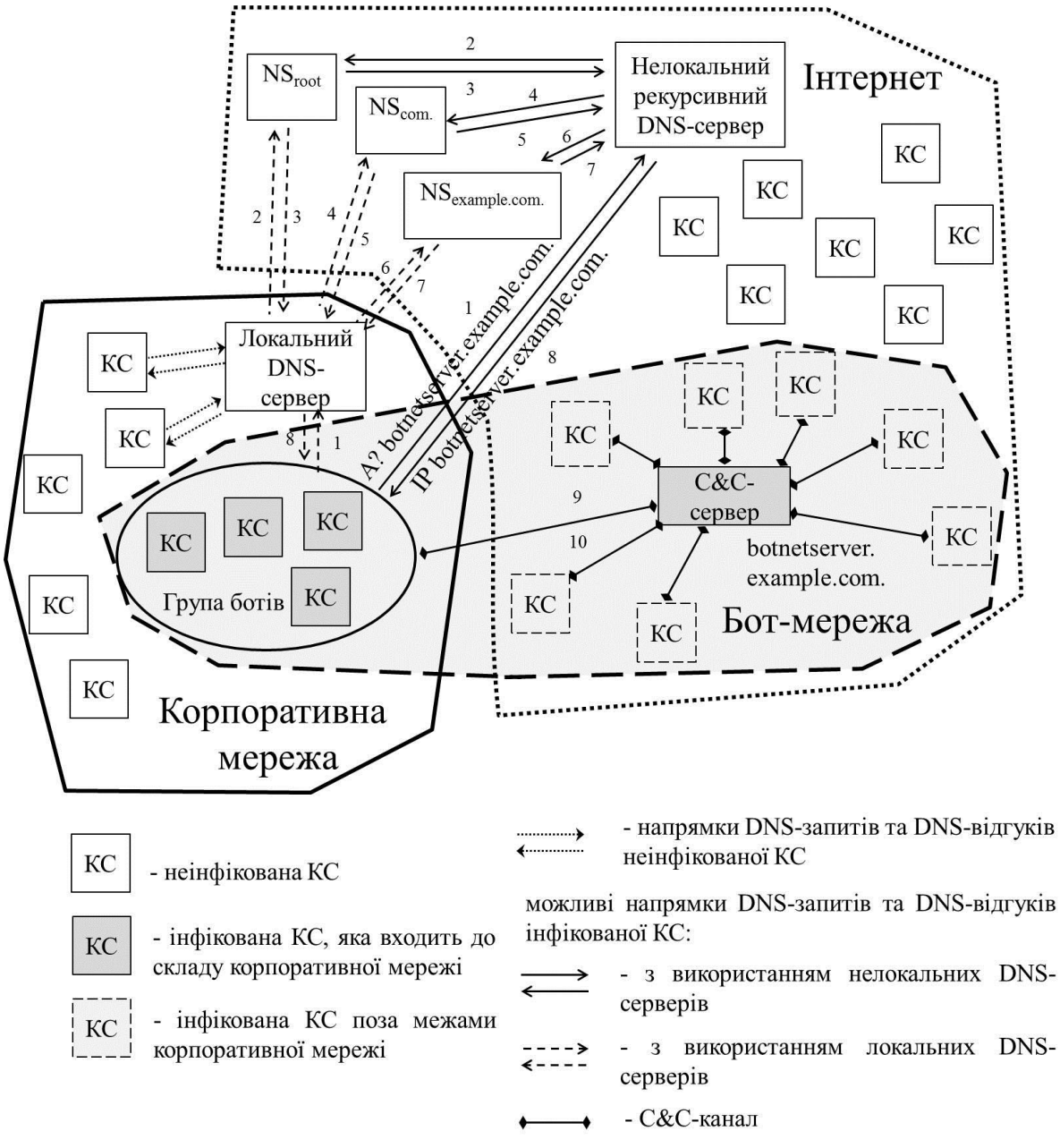


Рисунок 2.3 – Узагальнена схема встановлення зв’язку між ботами бот-мережі та C&C-сервером з використанням DNS

Дії на фазі здійснення шкідливої активності визначаються функційним призначенням ботів бот-мережі та є наслідком виконання ботами команд власника бот-мережі (ботмайстра).

Фаза супроводження полягає у внесенні змін до коду ботів з метою додавання ботам бот-мережі нової функціональності, забезпечення ухилення від виявлення, виправлення дефектів в ШПЗ, надання вказівки щодо зміни локації командно-контролюючого сервера бот-мережі тощо.

Настання фази припинення функціонування може бути зумовлене як виконанням відповідної команди власника бот-мережі, так і ліквідацією бот-мережі експертами в галузі ІТ-безпеки, правоохоронними органами тощо.

Фази первинної реєстрації / з'єднання та супроводження вимагають надсилання ботами DNS-запитів з метою отримання інформації щодо локації C&C-сервера або сервісів оновлень ШПЗ бот-мережі. Частина атак на фазі здійснення шкідливих дій також вимагають використання сервісу DNS.

DNS-запити ініціюються ботом в наступних ситуаціях: (1) для первинної реєстрації бота та згуртування з бот-мережею після успішного інфікування КС; (2) після збоїв з'єднання з C&C-сервером (після збою 3-етапного «рукостискання» боти розпочинають надсилати запити до DNS-сервера); (3) після міграції C&C-сервера; (4) після зміни IP-адреси C&C-сервера; (5) при здійсненні шкідливих дій (DDoS-атак, спам-розсилок); (6) після перезавантаження інфікованої КС; (7) з метою підвищення завадостійкості (для отримання додаткових доменних імен, пов'язаних з функціонуванням бот-мережі, бот здійснює зворотні DNS-запити).

DNS-запити інфікованих КС вирізняються характерною ознакою – скоординованістю (груповою активністю), яка полягає в тому, що боти бот-мережі здійснюють одночасні або зосереджені в невеликому проміжку часу DNS-запити під час спроб доступу до C&C-серверів, їх міграціях, виконанні команд або скачуванні оновлень ШПЗ.

В разі, якщо ресурсні записи DNS для доменного імені були кешовані DNS-клієнтом КС, повторний DNS-запит не виходить за межі локального кеша DNS комп'ютерної системи до спливання TTL. Для багатьох видів бот-мереж

характерним є ігнорування ботами TTL-періоду, тривалість якого містилась у відповіді від авторитетного DNS-сервера на DNS-запит. Це означає, що бот виконує очищення локального кеша DNS та здійснює повторний DNS-запит щодо доменного імені до завершення TTL-періоду, що надає можливість підвищити гнучкість та надійність керування бот-мережею (рис. 2.4 а, б, на схемі послідовність дій представлено позначеннями 1…6).

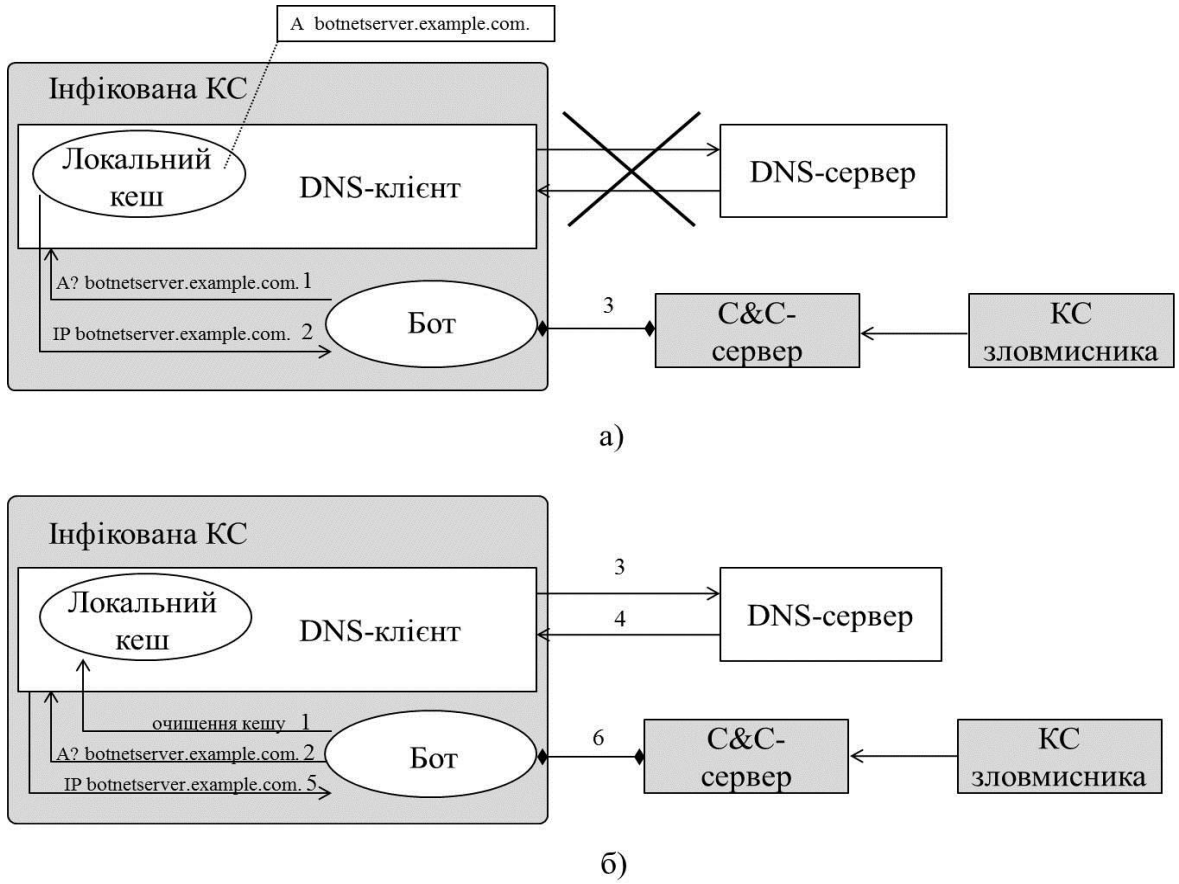


Рисунок 2.4 – Здійснення повторного DNS-запиту щодо доменного імені до завершення TTL-періоду: а) за наявності в локальному кеші DNS кешованої DNS-відповіді щодо доменного імені; б) після очищення локального кешу DNS

Розглянемо модель бот-мережі як систему керування інфікованими ботами КС з врахуванням використання DNS на різних фазах життєвого циклу бот-мережі.

Представимо модель бот-мережі у вигляді кортежу:

$$M_{BN} = \langle C, A, B, \Psi, Z, L, F \rangle, \tag{2.1}$$

де $C = \{c_j\}_{j=1}^{N_C}$ – множина контролюючих елементів бот-мережі, N_C – кількість контролюючих елементів бот-мережі;

$A = \{a_j\}_{j=1}^{3}$ – тип архітектури бот-мережі;

$B = \{b_j^p\}_{j=1}^{N_B}$ – множина мережних протоколів, що використовуються для керування бот-мережею, N_B – кількість мережних протоколів, $p \in P$, $P = \{1..65535\}$– множина портів, що використовуються для керування бот-мережею;

$\Psi = \{\psi_j\}_{j=1}^{4}$ – множина технологій ухилення від виявлення бот-мереж на основі DNS;

$Z = \{z_j\}_{j=1}^{N_Z}$ – множина ботів, що входять до складу бот-мережі, N_Z – кількість ботів бот-мережі;

$L = \{l_j\}_{j=1}^{5}$ – множина стадій життєвого циклу бот-мережі;

$F = \{f_j\}_{j=1}^{N_F}$ – множина функцій ботів, що визначається відповідною фазою життєвого циклу бот-мережі, N_F – кількість функцій ботів бот-мережі; функція інфікування вузла $l_1 \Rightarrow \mathrm{Y} \xrightarrow{f_1} \{h_{\inf} | h_{\inf} \in H\}$, де Y – множина шкідливих дій, закладених в функціонал бот-мережі, H – множина КС в глобальній мережі, $h_{\inf}$ – інфікована ботом КС; функція приєднання інфікованої КС до бот-мережі $l_2 \Rightarrow Z \cup \{h_{\inf} | h_{\inf} \in H\} \xrightarrow{f_2} Z'$; функція оновлення версії ШПЗ бота бот-мережі $l_3 \Rightarrow z \times z' \xrightarrow{f_3} z'$; функція виконання команди на здійснення шкідливої активності $l_4 \Rightarrow Z \times \{p | p \in P\} \xrightarrow{f_4} \mathrm{Y}$, де P – множина команд, які можуть бути виконані ботами бот-мережі; функція припинення функціонування бота бот-мережі $l_5 \Rightarrow Z \setminus \{z | z \in Z\} \xrightarrow{f_5} Z'$.

Розглянемо більш детально принципи функціонування складових моделі бот-мережі.

Для керування бот-мережею найбільш часто використовуються наступні стандартні протоколи та порти:

- $b_1^{20,21}$ – FTP (File Transfer Protocol), мережний протокол, призначений для передачі файлів у комп'ютерних мережах;

- b_2^{22} – SSH (Secure Shell), мережний протокол, що дозволяє здійснювати віддалене керування ОС та тунелювання TCP-з'єднань;

- b_3^{23} – TELNET (Terminal Network), мережний протокол для забезпечення взаємодії термінальних пристроїв та процесів;

- b_4^{25} – SMTP (Simple Mail Transfer Protocol), мережний протокол, призначений для передачі електронної пошти у мережах TCP/IP;

- b_5^{43} – WHOIS, мережний протокол для отримання реєстраційних даних щодо власників доменних імен, автономних систем, IP-адрес;

- b_6^{53} – DNS;

- $b_7^{67,68}$ – DHCP (Dynamic Host Configuration Protocol), мережний протокол, що дозволяє КС автоматично одержувати IP-адресу та інші параметри, необхідні для роботи в мережі TCP/IP;

- b_8^{69} – TFTP (Trivial File Transfer Protocol), протокол передачі файлів, який використовується для початкового завантаження бездискових робочих станцій;

- b_9^{80} – HTTP (Hyper Text Transfer Protocol), протокол, який використовується для одержання інформації з веб-сайтів;

- b_{10}^{110} – POP3 (Post Office Protocol Version 3), мережний протокол, що використовується клієнтами електронної пошти для вилучення електронного повідомлення з віддаленого сервера по TCP/IP з'єднанню;

- b_{11}^{119} – NNTP (Network News Transfer Protocol), протокол обміну повідомленнями між сервером груп новин та клієнтом;

- b_{12}^{143} – IMAP (Internet Message Access Protocol), протокол прикладного рівня для доступу до електронної пошти;

- $b_{13}^{194,6667}$– IRC (Internet Relay Chat), протокол для обміну повідомленнями в режимі реального часу;

- b_{14}^{5190} – OSCAR (Open System for CommunicAtion in Realtime), мережний протокол для обміну миттєвими та офлайновими текстовими повідомленнями;

- b_{15}^{5222} – XMPP (Extensible Messaging and Presence Protocol, або Jabber),

відкритий протокол для миттєвого обміну повідомленнями в режимі, близькому до режиму реального часу.

Представимо множину командно-контролюючих елементів бот-мережі наступним чином:

$$C = \{c_j\}_{j=1}^{N_C} = \{\langle\langle D, I\rangle, \langle N, E\rangle\rangle_j\}_{j=1}^{N_C}, \tag{2.2}$$

де $D = \{d_j\}_{j=1}^{N_D}$, $I = \{i_j\}_{j=1}^{N_I}$, $N = \{n_j\}_{j=1}^{N_N}$, $E = \{e_j\}_{j=1}^{N_E}$ – множини доменних імен та IP-адрес контролюючих елементів бот-мережі, множини доменних імен та IP-адрес авторитетних серверів імен для d відповідно, N_D, N_I, N_N, N_E – кількість доменних імен, які відповідають контролюючим елементам бот-мережі, IP-адрес, які співставляються з цими доменними іменами, доменних імен авторитетних серверів імен та їх IP-адрес відповідно.

За командно-контролюючі елементи приймемо командно-контролюючі сервери та сервіси технічного обслуговування бот-мереж.

Представимо тип архітектури бот-мережі наступним чином: $A = \{a_j\}_{j=1}^{3}$, де a_1 – централізована, a_2 – розподілена, a_3 – гібридна.

В бот-мережі централізованої архітектури кожен бот з множини Z встановлює зв’язок з одним (або декількома) командно-контролюючим сервером з множини C (рис. 2.5).

Розподіленість архітектури бот-мережі може бути забезпечена шляхом застосування власних розподілених DNS-сервісів або сервісів динамічної DNS, що дозволяє оновлювати інформацію на DNS-сервері в режимі реального часу та в автоматичному режимі і уможливлює використання великої кількості інфікованих КС з метою перенаправлення трафіка від ботів бот-мережі до групи контролюючих вузлів з множини C та навпаки (рис. 2.6).

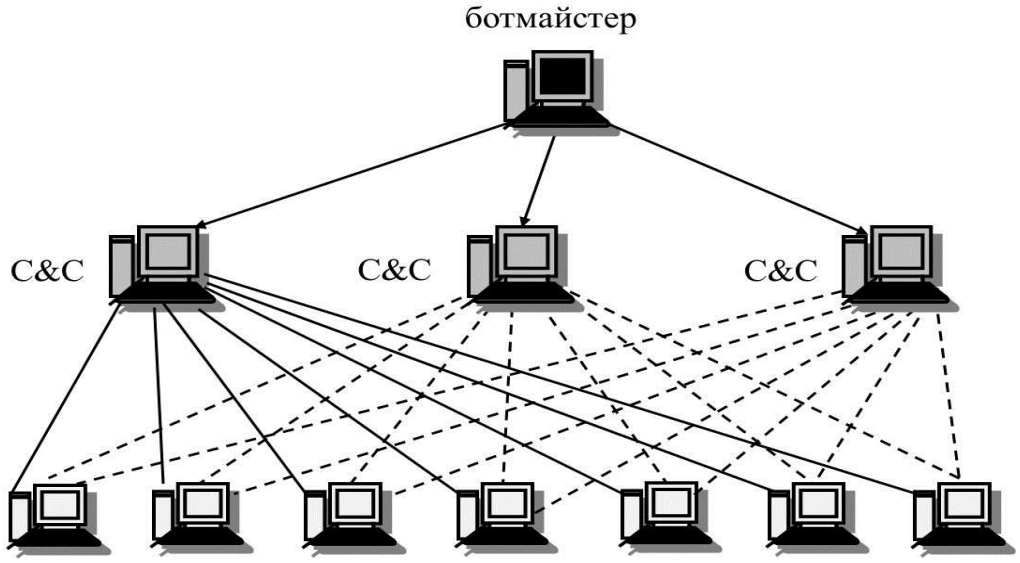


Рисунок 2.5 – Схематичне зображення бот-мережі централізованої архітектури

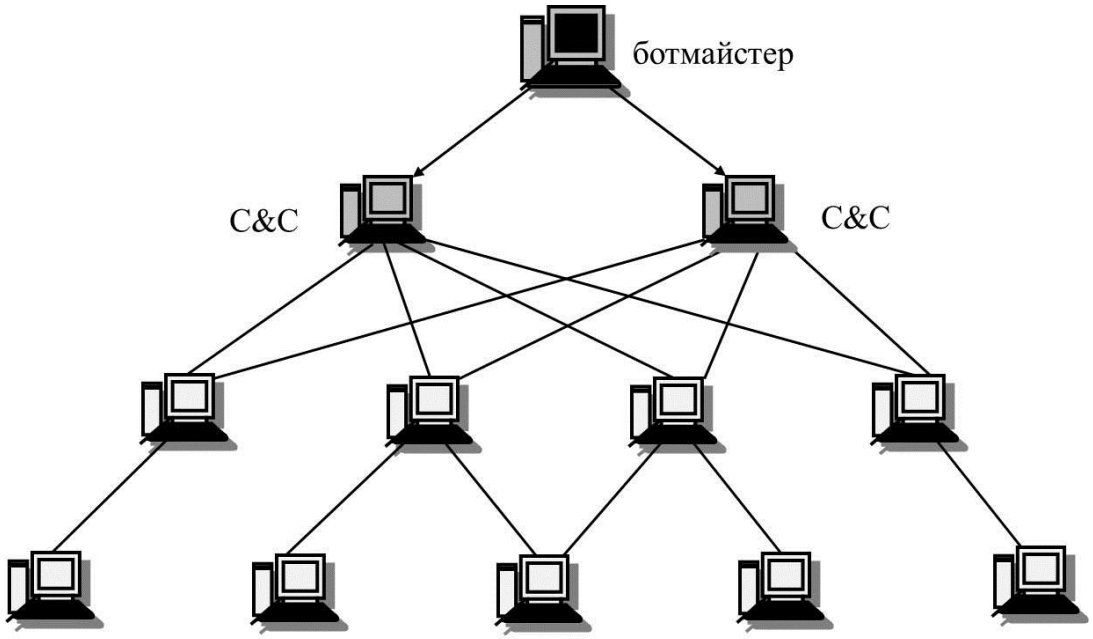


Рисунок 2.6 – Схематичне зображення бот-мережі розподіленої архітектури

В бот-мережі гібридної архітектури частина ботів виконують функції клієнтів та серверів одночасно, тобто належать до множини $C \bigcap Z \coloneqq \{x \mid x \in Z \wedge x \in C\}$, а решта ботів є тільки клієнтами, що дозволяє підвищити надійність та швидкість реакції такої бот-мережі. Бот-мережа може бути розподілена на кластери, які використовують протокол P2P для внутрішнього спілкування. В кожному кластері виокремлюється головний вузол, який може з'єднуватись з головними вузлами інших кластерів вищого рівня ієрархії, підтримуючи зв'язок між кластерами (рис. 2.7).

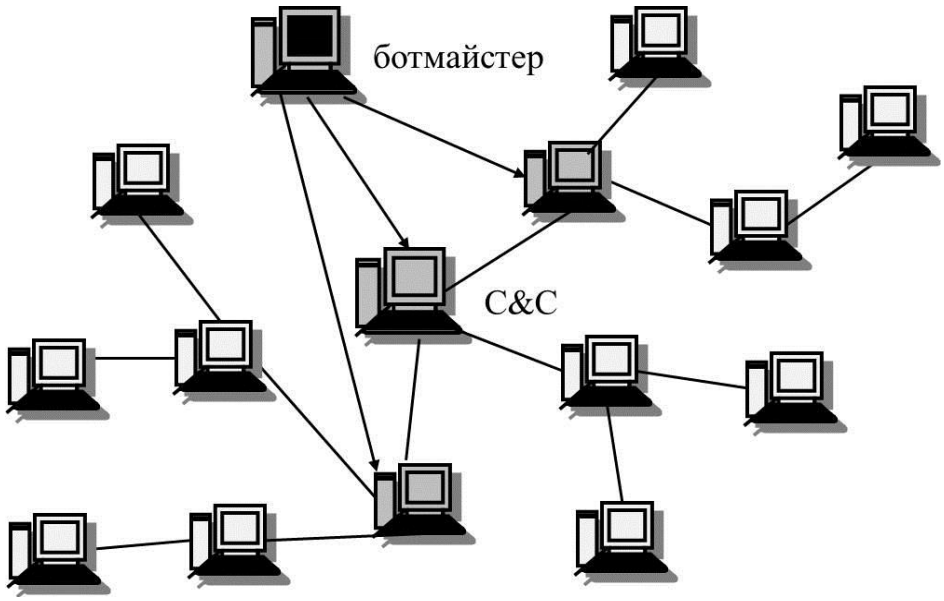


Рисунок 2.7 – Схематичне зображення бот-мережі гібридної архітектури

Представимо множину технологій ухилення від виявлення бот-мереж на основі використання DNS наступним чином: $\Psi = \{\psi_j\}_{j=1}^{4}$, де ψ_1 – періодична зміна IP-відображення (cycling of IP mapping), ψ_2 – «потік доменів» (domain flux), ψ_3 – «швидкозмінні мережі» (fast flux), ψ_4 – DNS-тунелювання (DNS-tunneling).

При періодичній зміні IP-відображення C&C-сервер бот-мережі з множини $c \in C$ періодично змінює локацію, при цьому доменне ім'я d, пов'язане з C&C-сервером, співставляється з однією IP-адресою з множини $i \in I$, $d \to \{i_1,...,i_n\}$ (рис. 2.8, на схемі періодичну зміну локації C&C-сервера бот-мережі представлено позначеннями 1…n). Зазвичай функції C&C-серверів виконують географічно розподілені в глобальній мережі скомпрометовані вузли, підконтрольні зловмиснику. Тип архітектури бот-мережі – централізована, $\psi_1 \Rightarrow a_1$.

При використанні технології ухилення «потік доменів» C&C-сервер бот-мережі $c \in C$ періодично мігрує на нові доменні імена зі списку, який формується за допомогою використання алгоритму генерації доменних імен, DGA (рис. 2.9). Ботом та командно-контролюючим сервером за визначеним алгоритмом генерується дуже велика кількість доменних імен, певна незначна

частина з яких використовуються в якості доменних імен C&C-сервера бот-мережі.

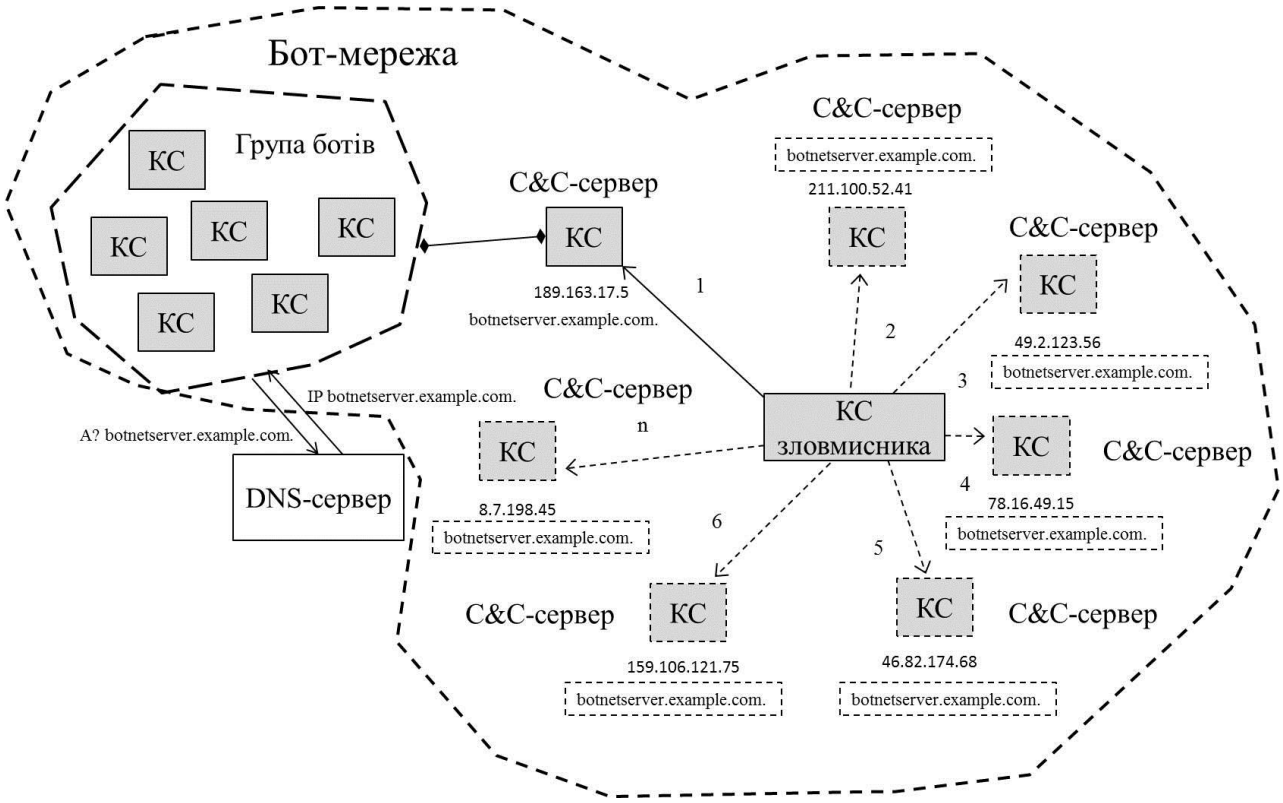


Рисунок 2.8 – Періодична зміна IP-відображення для доменного імені C&C-сервера

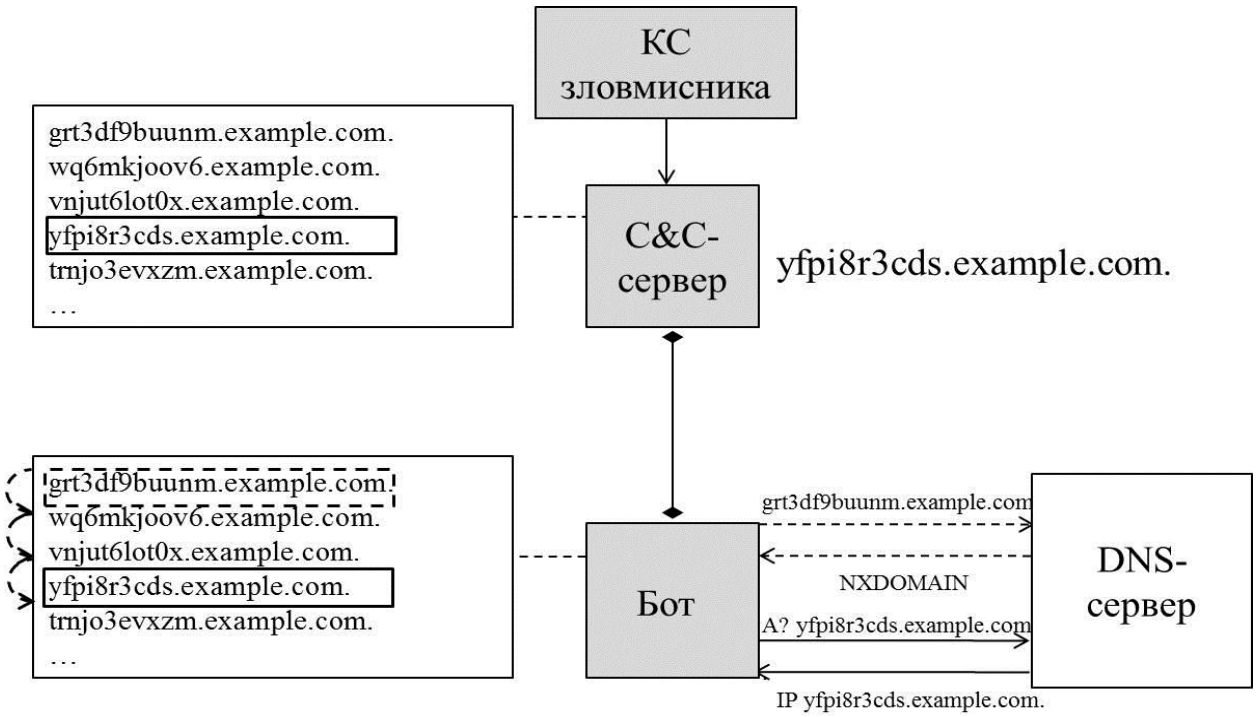


Рисунок 2.9 – Схема застосування алгоритму генерації доменних імен (DGA)

З метою встановлення зв’язку з C&C-сервером бот послідовно перебирає список згенерованих доменних імен та надсилає стосовно них DNS-запити, допоки не отримає успішну DNS-відповідь з IP-адресою C&C-сервера. Таким чином, IP-адресі C&C-сервера $i \in I$ в межах кожного інтервалу часу, визначеного TTL-періодом для записів DNS щодо доменного імені C&C-сервера, може відповідати нове доменне ім’я $d \in D$, тобто $\{i\} \to \{d_1,...,d_n\}$, або в разі, якщо C&C-сервер також змінює і локацію, то $\{i_1,...,i_n\} \to \{d_1,...,d_m\}$. Тип архітектури бот-мережі – централізована, $\psi_2 \Rightarrow a_1$.

В зв’язку з частою зміною доменного імені C&C-сервера або проблемами з його доступністю для бот-мереж є характерною підвищена кількість DNS-відповідей з кодом помилки RCODE=3 (NXDOMAIN).

Контролюючі елементи «швидкозмінної» мережі мають більш розширений функціонал в порівнянні з типовими C&C-серверами бот-мереж та приховані за мережею з множини зовнішніх проксі-серверів – «flux-агентів», які слугують для перенаправлення запитів ботів бот-мережі та даних до та від внутрішніх контролюючих серверів (рис. 2.10, 2.11, на схемах послідовність запитів та відповідей представлено позначеннями 1…11).

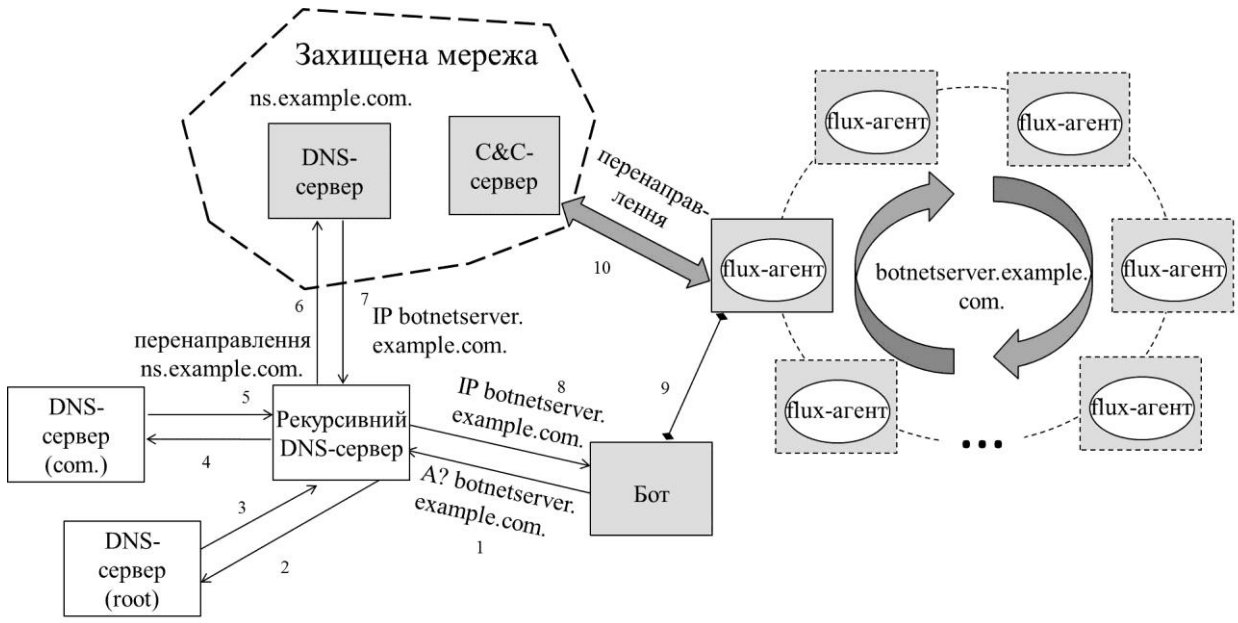


Рисунок 2.10 – Схема функціонування однопоточної «швидкозмінної» мережі

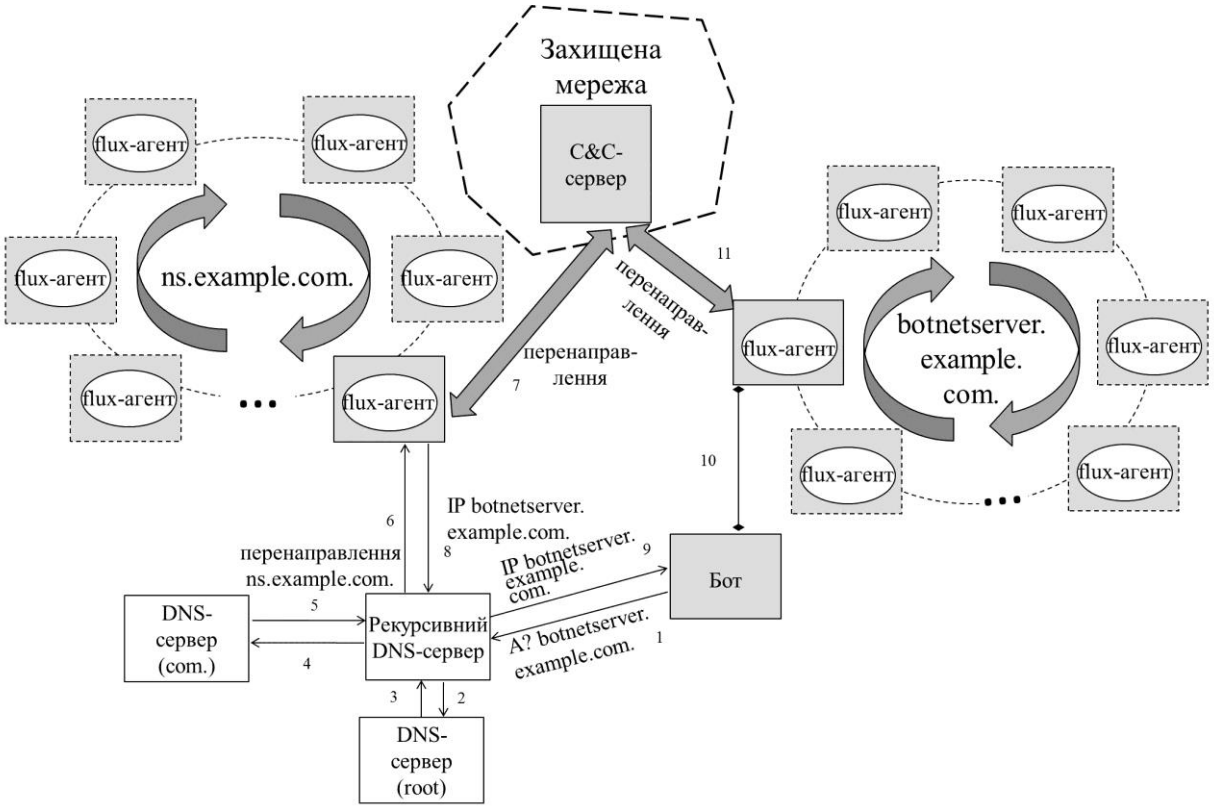


Рисунок 2.11 – Схема функціонування двопоточної «швидкозмінної» мережі

Таким чином, в межах інтервалу часу, визначеного TTL-періодом DNS, для «однопоточної» мережі доменне ім’я d, яке використовується для зв’язку ботів з контролюючими елементами $\{c_1,...,c_n\}$, співставляється з новою підмножиною IP-адрес, що циклічно змінюється $d \to \{i_1,...,i_n\}$, і які насправді є IP-адресами географічно розподілених інфікованих вузлів бот-мережі, що перенаправляють трафік до контролюючих елементів, тобто фактично $\{c_1,...,c_n\} \coloneqq \{\mathrm{x} \mid \mathrm{x} \in Z \wedge x \in C\}$.

Для «двопоточної» мережі додатково доменне ім’я кожного авторитетного сервера імен n співставляється з підмножиною IP-адрес, що циклічно змінюється, тобто $d \to \{i_1,...,i_n\}$, $n \to \{e_1,...,e_m\}$, і які насправді є IP-адресами географічно розподілених інфікованих вузлів бот-мережі, тобто $\{n_1,...,n_m\} \coloneqq \{\mathrm{x} \mid \mathrm{x} \in Z \wedge x \in N\}$. Враховуючи, що кількість серверів імен для таких бот-мереж зазвичай більше одного, то $\{n_1,...,n_m\} \to \{e_1,...,e_n\}$. Використання в якості проксі-серверів великої кількості інфікованих вузлів, які знаходяться в різних частинах світу, та численні перенаправлення ускладнюють відстеження

та відключення командно-контролюючих центрів такої бот-мережі. Тип архітектури бот-мережі – розподілена, $\psi_3 \Rightarrow a_2$.

При використанні технології DNS-тунелювання для передачі трафіку командування та контролю зловмисником використовується фальшивий DNS-сервер, який є авторитетним DNS-сервером імен для зони DNS, щодо доменних імен якої здійснюються DNS-запити ботами бот-мережі. Це уможливлює надсилання ботами на сервер зловмисника закодованих повідомлень під виглядом DNS-запитів стосовно доменних імен цієї зони, та отримання ботами команд від зловмисника, також закодованих в DNS-відповідях (рис. 2.12, на схемі послідовність запитів та відповідей представлено позначеннями 1…4).

В цьому випадку множина доменних імен D зони DNS фактично виступає в ролі аналога доменних імен C&C-сервера бот-мережі, IP-адреса e фальшивого DNS-сервера зазвичай залишається сталою, тобто $\{d_1,...,d_n\} \to e$. Тип архітектури бот-мережі – централізована або гібридна, $\psi_4 \Rightarrow a_1 \vee a_3$.

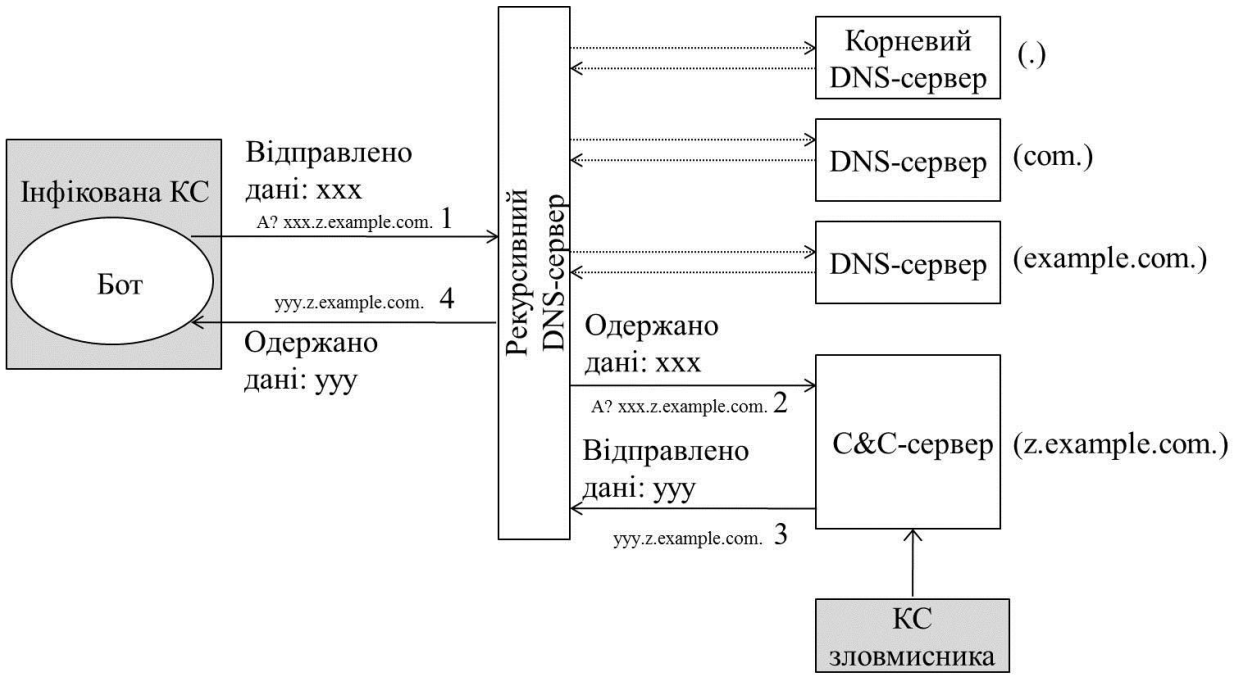


Рисунок 2.12 – Схема передачі довільного трафіку всередині полів DNS-повідомлення

Отже, побудовано модель бот-мереж з врахуванням системи доменних імен. Побудована модель бот-мереж враховує застосування бот-мережами

технологій ухилення від виявлення на основі DNS та трансформування C&C-архітектури бот-мережі з централізованої на розподілену або гібридну топологію. Модель бот-мереж з врахуванням системи доменних імен є необхідною для побудови моделі процесу виявлення бот-мереж на основі аналізу DNS-трафіка.

2.3 Модель DNS-трафіка та модель процесу виявлення бот-мереж на основі аналізу DNS-трафіка

Оскільки об'єктом дослідження є процес виявлення бот-мереж у корпоративних мережах на основі аналізу DNS-трафіка, то важливою задачею є розроблення його моделі, яка ґрунтується на моделі бот-мереж з врахуванням системи доменних імен та моделі DNS-трафіка.

Представимо модель DNS-трафіка корпоративної мережі у вигляді кортежу:

$$M_T = \langle \chi, H, S, D \rangle, \tag{2.3}$$

де χ – множина DNS-повідомлень, надісланих від та до множини H комп'ютерних систем корпоративної мережі, $\chi = \chi^O \cup \chi^I$, де χ^O – множина вихідних DNS-повідомлень мережі, χ^I – множина вхідних DNS-повідомлень мережі;

S – множина DNS-серверів, до яких було надіслано DNS-запити та від яких було одержано DNS-відповіді комп'ютерними системами мережі, $S = S^L \cup S^N$, де S^L – множина локальних DNS-серверів корпоративної мережі, S^N – множина нелокальних DNS-серверів;

D – множина запитаних комп'ютерними системами мережі доменних імен, $D = \{d_i\}_{i=1}^{N_D}$, де N_D – кількість різних доменних імен.

Представимо множину КС мережі, які здійснювали DNS-запити протягом часу спостереження, наступним чином: $H = \bigcup_{j=d_1}^{d_{N_D}} \bigcup_{k=1}^{N_{TTL}} H_{j,k}$, де H_j – підмножини MAC-адрес КС, які надсилали DNS-запити щодо певного доменного імені протягом часу спостереження; $H_{j,k}$ – підмножини MAC-адрес КС, які надсилали DNS-запити щодо певного доменного імені в межах певного TTL-періоду; N_{TTL} – загальна кількість таких підмножин; $H_{j,k} = \{h_{j,k,i}\}_{i=1}^{N_{H,j,k}}$, де $h_{j,k,i}$ – MAC-адреса певної КС мережі; $N_{H,j,k}$ – кількість КС мережі, які надсилали DNS-запити в межах певного TTL-періоду.

Аналогічно множину захоплених вхідних DNS-повідомлень представимо як $\chi^T = \bigcup_{j=d_1}^{d_{N_D}} \bigcup_{k=1}^{N_{TTL}} \chi_{j,k}$, де χ_j – підмножини вхідних DNS-повідомлень щодо певного доменного імені, захоплені протягом часу спостереження; $\chi_{j,k}$ – підмножини вхідних DNS-повідомлень щодо певного доменного імені, захоплені в межах певного TTL-періоду; $\chi_{j,k} = \{\chi_{j,k,i}\}_{i=1}^{N_{\chi,j,k}}$, де $\chi_{j,k,i}$ – DNS-повідомлення, захоплене в межах певного TTL-періоду, $N_{\chi,j,k}$ – кількість DNS-повідомлень, захоплених в межах певного TTL-періоду.

З врахуванням полів вхідного DNS-повідомлення, дані з яких можуть бути використані для виявлення DNS-запитів бот-мереж, згідно стандарту RFC 1035 [69] опишемо захоплений DNS-відгук щодо певного доменного імені кортежем:

$$\chi_{j,k,i} = \left\langle \chi_{j,k,i,\text{H}}, \chi_{j,k,i,TS}, \chi_{j,k,i,IP}, \left\langle \chi_{j,k,i,HD}, \chi_{j,k,i,ANS}, \chi_{j,k,i,ATH}, \chi_{j,k,i,ADD} \right\rangle \right\rangle, \quad (2.4)$$

$$j = d_1, ..., d_{N_D}, k = \overline{1, N_{TTL}}, i = \overline{1, N_{\chi,j,k}},$$

де $\chi_{j,k,i,\text{H}}$ – MAC-адреса КС, що здійснювала DNS-запит;

$\chi_{j,k,i,TS}$ – часовий штамп (час надходження DNS-пакета);

$\chi_{j,k,i,IP}$ – IP-адреса джерела DNS-пакета;

$\chi_{j,k,i,HD}, \chi_{j,k,i,ANS}, \chi_{j,k,i,ATH}, \chi_{j,k,i,ADD}$ – секції DNS-повідомлення: заголовок (Header), секція відгуків (Answer), секція серверів імен (Authority) та секція додаткової інформації (Additional) відповідно.

Заголовок DNS-повідомлення може бути описаний наступним чином:

$$\chi_{j,k,i,HD} = \left\langle \chi_{j,k,i,HD,ID}, \chi_{j,k,i,HD,OPC}, \chi_{j,k,i,HD,RC}, \chi_{j,k,i,HD,QDC}, \chi_{j,k,i,HD,ANC}, \chi_{j,k,i,HD,NSC}, \chi_{j,k,i,HD,ARC} \right\rangle,$$
$$j = d_1, ..., d_{N_D}, k = \overline{1, N_{TTL}}, i = \overline{1, N_{\chi,j,k}}, \quad (2.5)$$

де $\chi_{j,k,i,HD,ID}$ – ідентифікатор, що дозволяє пов'язати DNS-запит та DNS-відгук (поле ID);

$\chi_{j,k,i,HD,OPC}$ – тип запиту (поле OPCODE), $\chi_{j,k,i,HD,OPC} \in \{0,..,2\}$, 0 – стандартний, 1 – інверсний, 2 – запит стану сервера;

$\chi_{j,k,i,HD,RC}$ – код відгуку (поле RCODE), $\chi_{j,k,i,HD,RC} \in \{0,..,5\}$, 0 – немає помилки, 1 – помилка в форматі запиту, 2 – збій сервера, 3 – доменне ім'я не існує тощо;

$\chi_{j,k,i,HD,QDC}$ – кількість записів в секції запитів (поле QDCOUNT);

$\chi_{j,k,i,HD,ANC}, \chi_{j,k,i,HD,NSC}, \chi_{j,k,i,HD,ARC}$ – кількість ресурсних записів в секціях відгуків, серверів імен та додаткової інформації (поля ANCOUNT, NSCOUNT, ARCOUNT) відповідно.

Секції відгуків, серверів імен та додаткової інформації мають однаковий формат та можуть бути описані як множини ресурсних записів наступним чином:

$$\chi_{j,k,i,S} = \left\{ \left(\chi_{j,k,i,S,NM}, \chi_{j,k,i,S,TP}, \chi_{j,k,i,S,TTL}, \chi_{j,k,i,S,RDL}, \chi_{j,k,i,S,RDT} \right)_n \right\}_{n=1}^{N_{RR,S}},$$
$$j = d_1, ..., d_{N_D}, k = \overline{1, N_{TTL}}, i = \overline{1, N_{\chi,j,k}}, \quad (2.6)$$

де $S \in \{"ANS","ATH","ADD"\}$, $\chi_{j,k,i,S,NM}$ – ім'я домена, до якого відноситься ресурсний запис (поле NAME);

$\chi_{j,k,i,S,TP}$ – тип коду ресурсного запису (поле TYPE), визначає значення та формат даних в полі RDATA [69];

$\chi_{j,k,i,S,TTL}$ – час життя записів DNS (поле TTL); $\chi_{j,k,i,S,RDL}$ – довжина поля RDATA (поле RDLENGTH);

$\chi_{j,k,i,S,RDT}$ – рядок, що описує ресурс (поле RDATA);

$N_{RR,S}$ – кількість ресурсних записів в секції (дорівнює значенню $\chi_{j,k,i,HD,ANC}, \chi_{j,k,i,HD,NSC}, \chi_{j,k,i,HD,ARC}$ для відповідної секції).

Представимо концептуально модель процесу виявлення бот-мереж на основі аналізу DNS-трафіка наступним чином:

$$M_D = \left\langle \chi^T, f_1^T, C_{GA}{}^T, C_{ET}{}^T, f_2^T, T \right\rangle, \tag{2.7}$$

де χ^T – множина захоплених вхідних DNS-повідомлень (DNS-відгуків) до множини H комп'ютерних систем корпоративної мережі;

f_1^T – функція співставлення доменних імен з «білим» та «чорним» списками;

$C_{GA}{}^T$ – множина алгоритмів ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку;

$C_{ET}{}^T$ – множина алгоритмів виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS;

f_2^T – функція локалізації КС, інфікованих ботами, та блокування дій ботів;

$T = \{t_m\}_{m=0}^{N_T}$ – інтервал часу спостереження, де N_T – кількість ітерацій спостереження.

Формалізована схема процесу виявлення бот-мереж на основі аналізу DNS-трафіка подана на рис. 2.13.

Процес виявлення бот-мереж на основі аналізу DNS-трафіка може бути поданий у вигляді часової діаграми (рис. 2.14), де t_0 – час початку спостереження (збору вхідного DNS-трафіка), $\{t_1,...,t_n\}$ – тривалості ітерацій спостереження.

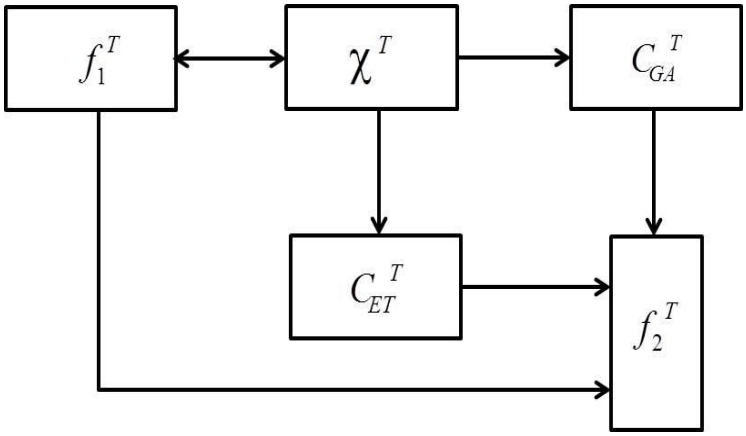


Рисунок 2.13 – Формалізована схема процесу виявлення бот-мереж на основі аналізу DNS-трафіка

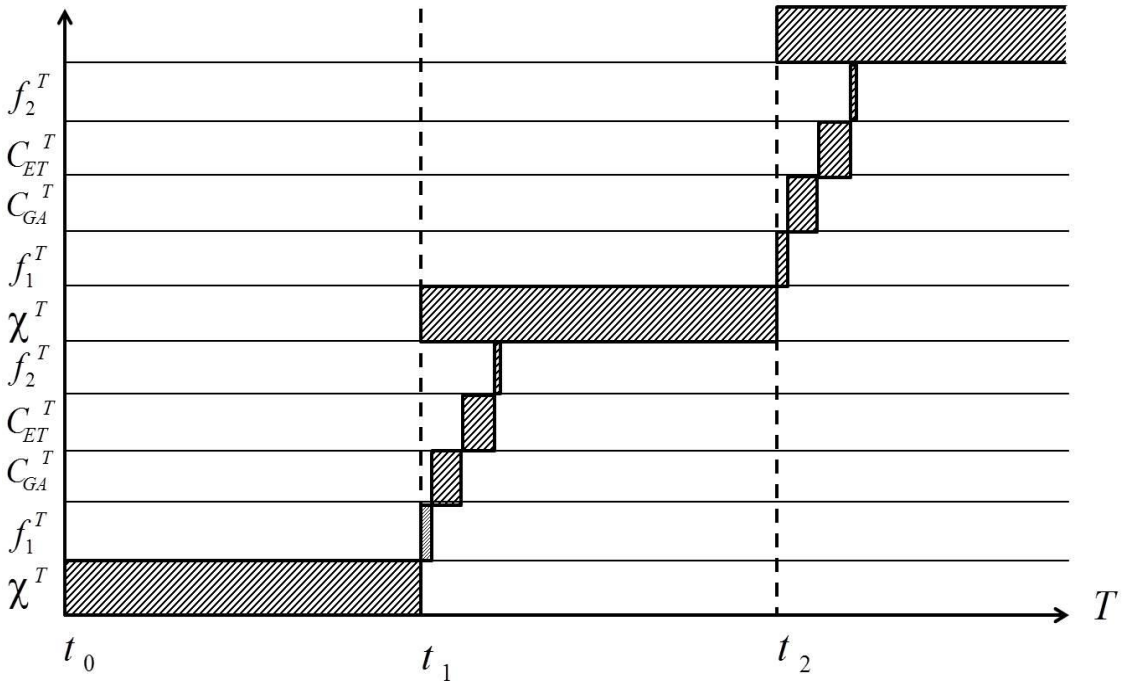


Рисунок 2.14 – Часова діаграма процесу виявлення бот-мереж на основі аналізу DNS-трафіка

Функцію f_1^T співставлення доменних імен з «білим» та «чорним» списками опишемо кортежем:

$$f_1^T = \langle D, W, B \rangle, \tag{2.8}$$

де W – множина доменних імен, занесених до «білого» списку популярних легітимних доменних імен;

B – множина доменних імен, занесених до «чорного» списку відомих шкідливих доменних імен відповідно.

Визначимо функцію f_1^T наступним чином:

$$f_1^T : \{d | d \in W \vee d \in B\} \to \{\text{Legitimate}, \text{Malicious}\} \ . \quad (2.9)$$

Множина алгоритмів ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку складається з наступних елементів:

$$C_{GA}^T = \{c_f, c_s, c_m, c_e, c_b, c_p, f\}, \quad (2.10)$$

де c_f – алгоритм виявлення групового ігнорування TTL-періоду;

c_s – алгоритм побудови вектора щільності розподілу DNS-запитів в часі та перевірки синхронності запитів;

c_m – алгоритм побудови матриці спостереження для аналізу вхідного DNS-трафіка;

c_e – алгоритм виявлення групової активності шляхом аналізу групових запитів щодо одного й того самого доменного імені;

c_b – алгоритм побудови нижньотрикутної матриці мір Браун-Бланке для порівняння груп;

c_p – алгоритм формування векторів ознак для пар групових DNS-запитів;

f – алгоритм аналізу векторів ознак для пар групових DNS-запитів з метою ідентифікації інфікованих КС.

Зазвичай неінфіковані КС в локальній мережі здійснюють DNS-запити до локальних DNS-серверів. У випадку використання локального DNS-сервера для DNS-запитів щодо доменного імені d в інтервалі часу Δt від першої кешованої DNS-сервером DNS-відповіді щодо d до спливання TTL КС одержуватимуть

послідовно зменшувані в часі значення TTL_d, що визначатимуть залишок часу зберігання записів DNS щодо d в кеші DNS-сервера, а також час, протягом якого записи мають зберігатись в локальному кеші DNS на КС. Кешування DNS-записів має на меті зменшення кількості DNS-запитів до локальних та зовнішніх DNS-серверів, а також скорочення часу, необхідного для переведення доменних імен в IP-адреси.

Якщо КС надсилала DNS-запит щодо d та отримала DNS-відповідь, то для наступних запитів щодо d DNS-записи братимуться з локального кеша DNS комп'ютерної системи, допоки не сплине час їх зберігання. Якщо в інтервалі часу Δt до закінчення TTL-періоду TTL_d КС повторно надсилала DNS-запит щодо d, це означає, що було здійснено очищення локального кеша DNS на КС. Якщо очищення локального кеша DNS спостерігається на окремих КС, це може бути наслідком дій користувачів; якщо така поведінка спостерігається за групою КС, що синхронно запитує доменне ім'я, це може свідчити про ігнорування TTL-періоду, що є властивим для багатьох видів бот-мереж.

Боти в локальній мережі можуть використовувати як локальні, так і власні DNS-сервери, а також безкоштовні сервіси DNS (OpenDNS, FreeDNS). При використанні комп'ютерними системами різних DNS-серверів в DNS-відгуках можуть міститись різні значення TTL-періодів в залежності від того, чи були, і як давно, кешовані DNS-записи для доменного імені на цих серверах. Боти бот-мереж централізованої та розподіленої архітектури зберігають синхронність в діях, тому прояв такої неузгодженості DNS для бот-мереж цих типів виключений і може бути викликаний лише діями користувачів, що не впливатиме на точність виявлення.

Алгоритм виявлення групового ігнорування TTL-періоду визначимо парою:

$$c_f = \langle \eta, \gamma \rangle, \tag{2.11}$$

де η – алгоритм побудови матриці спостереження V_{MAC};

γ – функція порівняння множин КС для попереднього та повторного групових запитів в межах TTL-періоду.

Алгоритм побудови матриці спостереження V_{MAC} опишемо кортежем:

$$\eta = \left\langle D, \chi_{j,k}, \chi_{j,k,i,TS}, \chi_{j,k,i,ANS,TTL}, H_{j,k} \right\rangle, \; j = d_1, ..., d_{N_D}, k = \overline{1, N_{TTL}}, i = \overline{1, N_{\chi,j,k}} . \quad (2.12)$$

Представимо матрицю спостереження V_{MAC} у вигляді набору векторів:

$$\overline{W_{V,i}} = \left(h_{j,i}\right), \; j = \overline{1, N_G} , \quad (2.13)$$

де $h_{j,i}$ – MAC-адреси КС, які здійснювали DNS-запити щодо d в межах TTL, i – номер рядка матриці, $i > 1$, якщо було зафіксовано повторні DNS-запити КС щодо доменного імені d в межах TTL;

N_G – кількість різних КС, які запитували доменне ім’я d в межах TTL.

В матрицю спостереження V_{MAC} заносяться MAC-адреси КС, що здійснювали DNS-запити щодо доменного імені d від першого зафіксованого DNS-відгуку щодо d до спливання TTL-періоду, отриманого в DNS-відгуку. Або, якщо $TTL^r_{d^{m'}} < TTL_{d^{m'+1}}$, тоді формування груп триває до спливання $TTL_{d^{m'+1}}$, де $TTL^r_{d^{m'}} = TTL_{d^{m'}} - t_{d^{m'+1}}$ – залишок часу до видалення DNS-записів з локальних DNS-кешів на КС, що запитували d, згідно попереднім відгукам, $TTL_{d^{m'}}$ – TTL-період, в межах якого здійснюється пошук, $t_{d^{m'+1}}$ – час надходження наступного DNS-відгуку відносно початку зворотнього відліку періоду $TTL_{d^{m'}}, TTL_{d^{m'+1}}$ – TTL-період, отриманий в наступному DNS-відгуку.

Визначимо функцію γ порівняння двох множин КС для попереднього та повторного групових запитів в межах TTL-періоду з матриці спостереження V_{MAC} наступним чином:

$$\gamma : \left\{ g_{1,i} \le g_{2,i}, K_B \right\}, \; K_B : \left\{ K_B \in [0;1]; G_1, G_2 \to K_B \right\}, \quad (2.14)$$

де G_1 та G_2 – множини MAC-адрес КС (групи), що здійснювали повторні DNS-запити в межах TTL-періоду, $G_1 \subseteq H_{j,k}, G_2 \subseteq H_{j,k}$, $g_{1,i} \in G_1, g_{2,i} \in G_2$;

K_{B} – коефіцієнт подібності Браун-Бланке для двох порівнюваних груп;

δ – порогове значення подібності.

Якщо $\gamma < \delta$, то рядок матриці V_{MAC} для групового запиту $\min(G_1, G_2)$ видаляється.

Алгоритм побудови вектора щільності розподілу DNS-запитів в часі та перевірки синхронності DNS-запитів визначимо кортежем:

$$c_s = \langle \theta, \varphi, \vartheta \rangle, \tag{2.15}$$

де θ – алгоритм побудови вектора щільності розподілу DNS-запитів в часі;

φ – функція визначення синхронності DNS-запитів;

ϑ – функція об’єднання множин MAC-адрес груп КС з матриці V_{MAC}.

З метою перевірки синхронності DNS-запитів побудуємо вектор щільності розподілу DNS-запитів в часі. Групи DNS-запитів вважатимемо синхронними, якщо спостерігається велика кількість DNS-запитів щодо доменного імені в межах певного інтервалу часу ts, який назвемо часом синхронізації ботів. Алгоритм побудови вектора щільності розподілу DNS-запитів в часі визначимо кортежем:

$$\theta = \left\langle \chi_{j,k}, \chi_{j,k,i,TS}, t_{first}, t_{last}, t_s \right\rangle, \; j = d_1, ..., d_{N_D}, k = \overline{1, N_{TTL}}, i = \overline{1, N_{\chi,j,k}}, \tag{2.16}$$

де t_{first} та t_{last} – час надходження відповідно першого та останнього DNS-відгуків щодо доменного імені в межах TTL-періоду, протягом якого здійснюється пошук групової активності, або DNS-відгуків, що містили код помилки NXDOMAIN, або DNS-відгуків в межах інтервалу часу, коли було зафіксовано групове очищення локальних кешів DNS.

Вектор щільності розподілу DNS-запитів в часі опишемо наступним чином:

$$\overline{W_d} = (\Omega_j)_{j=1}^{z}, \tag{2.17}$$

де Ω_j – кількість DNS-запитів в межах z-го інтервалу; $z = (t_{last} - t_{first}) / (\frac{1}{3} ts)$.

Функцію визначення синхронності DNS-запитів опишемо кортежем:

$$\varphi = \langle \overline{W_d}, \Omega_{\max}, \Omega_{\max \pm 2}, Sum_s, Sum_r, \delta \rangle, \tag{2.18}$$

де $\Omega_{\max}$ – елемент вектора $\overline{W_d}$ з максимальним значенням;

$\Omega_{\max \pm 2}$ – елементи вектора, суміжні з $\Omega_{\max}$;

Sum_s – сума значень максимального та двох суміжних з ним елементів вектора з найбільшими значеннями, які описують розподіл DNS-запитів неперервного інтервалу часу;

Sum_r – сума значень решти елементів вектора.

Визначимо функцію φ наступним чином: $\varphi(\overline{W_d}) = true$, якщо $(1-\delta)(Sum_s + Sum_r) \geq Sum_r$, інакше $\varphi(\overline{W_d}) = false$. Якщо $\varphi(\overline{W_d}) = true$, то груповий запит підлягає подальшому аналізу, інакше – відкидається.

Якщо було виявлено повторні групові запити в межах TTL-періоду та $\varphi(\overline{W_d}) = true$ для цих DNS-запитів, множини MAC-адрес груп КС, що здійснювали повторні запити, об'єднуються. Визначимо функцію об'єднання множин MAC-адрес груп КС $\vartheta : \bigcup_{j=1}^{q} G_j \to G$, де q – кількість повторних запитів.

Алгоритм побудови матриці спостереження M_m визначимо кортежем:

$$c_m = \langle G_i, \chi_j, \overline{W_{M,i}} \rangle, i = \overline{1, N_{GQ}}, j = d_1, ..., d_{N_D}. \tag{2.19}$$

Матриця спостереження M_m будується для кожного визначеного інтервалу часу спостереження t_m для множини часових інтервалів $T = \{t_m\}_{m=1}^{N_T}$, де

N_T – кількість інтервалів часу спостереження, m – номер ітерації спостереження. Таким чином, одержуємо множину матриць спостереження $M = \{M_m\}_{m=1}^{N_T}, M_m = (m_{ij})_{i=1,j=1}^{N_q, N_h+\varepsilon}$, де N_q – кількість групових DNS-запитів, N_h – кількість різних MAC-адрес, представлених в групах, ε – кількість ознак групових DNS-запитів.

Якщо для групового DNS-запиту $\varphi(\overline{W_d}) = true$, то множини MAC-адрес груп КС з матриці v_{MAC} переносяться до матриці спостереження M_m.

Представимо матрицю M_m у вигляді набору векторів:

$$\overline{W_{M,i}} = (d_i, G_i, N_{G,i}, S_i, F_i, R_i, M_i, N_i), i = \overline{1, N_{GQ}}, \quad (2.20)$$

де $N_{G,i}$ – кількість КС у групі;

S_i, F_i, R_i – ознаки наявності для групи КС нетипових для звичайних користувачів особливостей поведінки, властивих для бот-мереж;

S_i – ознака звертання до локальних/нелокальних DNS-серверів;

F_i – ознака повторного запиту в межах TTL-періоду;

R_i – ознака наявності в DNS-відповідях коду помилки NXDOMAIN;

M_i – ознака “інфікований” чи “підозрілий” щодо групи КС, отримана на проміжних етапах аналізу;

N_i – номер ітерації спостереження, на якій зафіксовано ознаку “підозрілий”;

N_{GQ} – кількість групових DNS-запитів.

Алгоритм виявлення групової активності шляхом аналізу групових запитів щодо одного й того самого доменного імені опишемо кортежем:

$$c_e = \langle \overline{W_{M,i}}, \tau, \varpi \rangle, i = \overline{1, N_{GQ}}, \quad (2.21)$$

де τ – функція порівняння груп КС з матриці спостереження M_m, що запитували однакові доменні імена;

ϖ – функція об'єднання рядків матриці M_m.

Визначимо функцію τ наступним чином: $\tau: G_1 \times ... \times G_n \to \{\text{Infected}, \text{Suspicious}, \text{Not\_Infected}\}$, або:

$$\tau(G_1,...,G_n) = \begin{cases} \textit{Infected, if } (K_S(G_1,...,G_n) \geq \delta) \vee (\delta` \leq K_S(G_1,...,G_n) < \delta \wedge \\ \wedge (\exists F = 1 \vee \forall S = 1), \\ \textit{Suspicious, if } (\delta` \leq K_S(G_1,...,G_n) < \delta) \wedge (\forall F \neq 1 \vee \exists S \neq 1), \\ \textit{Not\_Infected otherwise}, \end{cases} \quad (2.22)$$

де $G_1,...,G_n$ – групи КС, що запитували однакові доменні імена;

K_S – коефіцієнт подібності для порівняння груп КС;

$\delta`$ – порогове значення подібності, за якого групи КС вважатимуться підозрілими;

$\{S | S \in \{M_m(d_j, S)\}_{j=G_1}^{G_n}\}, \{F | F \in \{M_m(d_j, F)\}_{j=G_1}^{G_n}\}$

Якщо $\tau(G_1,...,G_n) = \text{Not\_Infected}$, то групові запити для доменного імені видаляються з матриці спостереження M_m.

Визначимо функцію ϖ наступним чином:

$$\varpi: \bigcup_{j=1}^{n} \langle G_j, S_j, F_j, R_j \rangle \to \langle G, S, F, R \rangle. \quad (2.23)$$

Представимо нижньотрикутну матрицю мір Браун-Бланке B_m у вигляді набору векторів:

$$\overline{W_{B,i}} = (d_i, K_B(G_i, G_j), N_{G,i}, S_i, F_i, R_i, M_i, N_i), i = \overline{1, N_{GQ}}, j = \overline{1, N_{GQ}}, \quad (2.24)$$

де $K_B(G_i, G_j)$ – коефіцієнт Браун-Бланке, обчислений для пари груп КС G_i та G_j, що запитували різні доменні імена.

Алгоритм побудови матриці мір Браун-Бланке B_m визначимо парою:

$$c_b = \langle \overline{W_{M,i}}, \overline{W_{B,j}} \rangle, i = \overline{1, N_{GQ}}, j = d_1,..., d_{N_D}. \quad (2.25)$$

Алгоритм формування векторів ознак для пар групових DNS-запитів на основі матриці мір Браун-Бланке визначимо парою:

$$c_p = \left\langle \overline{W_{B,i}}, \overline{W_{G_i,G_j}} \right\rangle, i = \overline{1, N_{GQ}}, j = \overline{1, N_{GQ}}, \quad (2.26)$$

де $\overline{W_{G_i,G_j}}$ – вектор ознак для пари групових DNS-запитів, для яких $K_B \geq \delta`$; $\overline{W_{G_i,G_j}} = \left(K_B\left(G_i, G_j\right), S_{G_i,G_j}, F_{G_i,G_j}, R_{G_i,G_j}, M_{G_i,G_j}\right)$, де $S_{G_i,G_j}, F_{G_i,G_j}, R_{G_i,G_j}, M_{G_i,G_j}$ – зведені поведінкові ознаки для двох груп КС.

Якщо для обох порівнюваних груп КС було зафіксовано наявність ознаки, яка свідчить про поведінку, властиву для бот-мереж (групове звертання до нелокальних DNS-серверів, групове ігнорування TTL-періоду, груповий DNS-запит з кодом помилки NXDOMAIN), відповідна ознака вектора приймає значення «Dangerous», якщо наявність такої ознаки зафіксовано для однієї групи КС – «Suspicious», якщо значення такої ознаки для обох груп вказує на невластиву для бот-мереж поведінку КС – «Unusual», інакше – «Neutral».

Висновок щодо наявності в мережі групової активності ботів здійснюється на основі аналізу векторів ознак $\overline{W_{G_i,G_j}}$ для пар групових DNS-запитів за алгоритмом f. Результат аналізу може приймати чотири значення: «Not_Infected» (неінфіковані), «Not_Suspicious» (не підозрілі), «Suspicious» (підозрілі), «Infected» (інфіковані).

Множина алгоритмів виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, складається з наступних елементів:

$$C_{ET}{}^{T} = \{c_t, c_w, c_d, \wp, c_u, \beta\}, \quad (2.27)$$

де c_t – алгоритм аналізу полів TTL вхідних DNS-повідомлень;

c_w – алгоритм вилучення ознак, які вказують на застосування технологій ухилення від виявлення на основі DNS, з вхідних DNS-повідомлень щодо певного доменного імені та побудови вектора ознак;

c_d – алгоритм побудови матриці даних на основі векторів ознак для різних доменних імен;

$\wp$ – алгоритм здійснення нечіткої кластеризації c-means з частковим навчанням;

c_u – алгоритм здійснення активного DNS-зондування;

β – алгоритм здійснення висновку щодо наявності застосування технологій ухилення на основі DNS.

Алгоритм аналізу полів TTL вхідних DNS-повідомлень з метою відбору DNS-відгуків для подальшого аналізу визначимо кортежем:

$$c_t = \left\langle D, \chi_j, \chi_{j,k}, H_j, \chi_{j,k,i,TS}, \chi_{j,k,i,ANS,TTL} \right\rangle, j = d_1, ..., d_{N_D}, k = \overline{1, N_{TTL}}, i = \overline{1, N_{\chi,j,k}}. \quad (2.28)$$

Алгоритм вилучення ознак з вхідних DNS-повідомлень щодо певного доменного імені та побудови вектора ознак визначимо кортежем:

$$c_w = \left\langle D, \chi^T, \chi_j, \chi_{j,k}, T_{\psi_1}, T_{\psi_2}, T_{\psi_3}, T_{\psi_4}, \upsilon \right\rangle, j = d_1, ..., d_{N_D}, k = \overline{1, N_{TTL}}, \quad (2.29)$$

де T_{ψ_1}, T_{ψ_2}, T_{ψ_3}, T_{ψ_4} – множини ознак, які вказують на застосування технологій періодичної зміни IP-відображення для шкідливого домена, «потік доменів», «швидкозмінних» мереж, DNS-тунелювання відповідно; υ – функція побудови вектора ознак.

Визначимо функцію υ наступним чином:

$$\upsilon : \left\{ D, \chi^T, \chi_j, \chi_{j,k} \right\} \to W_{e,j}, \quad (2.30)$$

де $\overline{W_{e,j}}$ – вектор ознак вхідних DNS-повідомлень щодо доменного імені.

Множина ознак, які вказують на застосування періодичної зміни IP-відображення для шкідливого домена, складається з наступних елементів:

$$T_{\psi_1} = \left\{ t_{\text{mod}}, t_{med}, t_{aver}, n_{IP}, s_{IP} \right\}, \quad (2.31)$$

де t_{mod} – TTL-період, мода;

t_{med} – TTL-період, медіана;

t_{aver} – TTL-період, середнє арифметичне значення;

n_{IP} – кількість IP-адрес, пов'язаних з доменним ім'ям;

s_{IP} – середня дистанція між IP-адресами, пов'язаними з доменним ім'ям.

Множина ознак, які вказують на застосування технології «потік доменів», складається з наступних елементів:

$$T_{\psi_2} = \{t_{\text{mod}}, t_{med}, t_{aver}, f_S, n_D\}, \tag{2.32}$$

де f_S – бінарна ознака успішності DNS-запиту;

n_D – кількість доменних імен, які спільно використовують IP-адресу.

Множина ознак, які вказують на застосування технології «швидкозмінних» мереж, складається з наступних елементів:

$$T_{\psi_3} = \{t_{\text{mod}}, t_{med}, t_{aver}, n_A, s_A, n_{UA}, s_{UA}\}, \tag{2.33}$$

де n_A – кількість A-записів, що відповідають доменному імені, у вхідному DNS-повідомленні;

s_A – середня дистанція між IP-адресами в множині A-записів для доменного імені у вхідному DNS-повідомленні;

n_{UA} – кількість унікальних IP-адрес в множинах A-записів, що відповідають доменному імені, у вхідних DNS-повідомленнях;

s_{UA} – середня дистанція між унікальними IP-адресами в множинах A-записів, що відповідають доменному імені, у вхідних DNS-повідомленнях.

Множина ознак, які вказують на застосування DNS-тунелювання, складається з наступних елементів:

$$T_{\psi_4} = \{l_N, n_U, e_N, e_R, f_{UR}, l_P, f_{E_B}\}, \tag{2.34}$$

де l_N – довжина доменного імені;

n_U – кількість унікальних символів в доменному імені;

e_N – ентропія доменного імені;

e_R – максимальне значення ентропії ресурсних записів DNS, які містяться в DNS-повідомленнях;

f_{UR} – бінарна ознака використання рідковживаних типів записів DNS, або таких, які зазвичай не використовуються клієнтами;

l_P – середній розмір DNS-повідомлень щодо доменного імені;

f_{E_B} – функція залежності ентропії поля DNS-повідомлення від його довжини [16].

Тоді, вектор ознак $\overline{W_{e,j}}$ вхідних DNS-повідомлень щодо доменного імені подамо наступним чином:

$$\overline{W_{e,j}} = (l_{N,j}, n_{U,j}, e_{N,j}, t_{\mathrm{mod},j}, t_{med,j}, t_{aver,j}, n_{A,j}, n_{IP,j}, s_{IP,j}, s_{A,j}, n_{UA,j}, s_{UA,j}, n_{N,j}, f_{UR,j}, e_{R,j}, l_{P,j}, f_{S,j}), \quad (2.35)$$

$$j = d_1, ..., d_{N_D}.$$

Алгоритм побудови матриці даних на основі векторів ознак визначимо парою:

$$c_d = \left\langle \overline{W_{e,j}}, V \right\rangle, \quad (2.36)$$

де $V = \left(v_{ji}\right)_{j=1,i=1}^{N_D,N_S}$ – матриця даних, $V(j,) = \overline{W_{e,j}}$, $N_S = \left|T_{\psi_1}\right| + \left|T_{\psi_2}\right| + \left|T_{\psi_3}\right| + \left|T_{\psi_4}\right|$ – загальна кількість ознак, які вказують на використання технологій ухилення від виявлення на основі DNS.

Алгоритм $\wp$ здійснення нечіткої кластеризації c-means з частковим навчанням опишемо кортежем:

$$\wp = \langle V, X, U \rangle, \quad (2.37)$$

де $X = \{x_i\}_{i=1}^{N_x}$ – промаркована вибірка даних, яка формується на основі множини знань щодо ознак вхідних DNS-повідомлень до ботів, які застосовують технології ухилення від виявлення на основі DNS для здійснення часткового навчання кластеризатора, N_x – кількість об'єктів в промаркованій вибірці;

U – матриця нечіткого розбиття, елементи якої визначають ступінь приналежності кожного з множини об'єктів кластеризації до кожного з п'яти наперед визначених кластерів, чотири з яких відповідають технологіям ухилення на основі DNS з множини Ψ, а п'ятий – нормальним запитам.

Тоді, алгоритм нечіткої кластеризації c-means з частковим навчанням може бути визначений наступним чином: $\wp : V \to U$.

Представимо матрицю нечіткого розбиття U наступним чином:

$$U = [u_{ij}], u_{ij} \in [0,1], i = \overline{1, N_D}, j = \overline{1,5}, \sum_{j=\overline{1,5}} u_{ij} = 1. \quad (2.38)$$

Алгоритм здійснення активного DNS-зондування визначимо кортежем:

$$c_u = \langle D, U, \hbar, \beta \rangle, \quad (2.39)$$

де $\hbar$ – функція здійснення активного DNS-зондування; β – функція здійснення висновку щодо наявності застосування технологій ухилення на основі DNS.

Визначимо функцію $\hbar$ наступним чином:

$$\hbar : \{d | \lambda' \le u_{d,j} < \lambda, \; j = \overline{1,4}\} \to T_A, \quad (2.40)$$

де λ' та λ – порогові значення приналежності об'єкта кластеризації до кластера, за яких доменне ім'я вважається підозрілим та шкідливим відповідно; якщо $\lambda' \le u_{d,j} < \lambda, \; j = \overline{1,4}$, то має місце невизначеність результату кластеризації;

T_A – множина додаткових ознак, які вказують на застосування технологій ухилення бот-мереж на основі DNS та можуть бути отримані засобами активного DNS-зондування.

Множина ознак T_A складається з наступних елементів:

$$T_A = \{n_{NS}, s_{NS}, v_{retry}, n_{ASN}, n_{ASA}\}, \tag{2.41}$$

де n_{NS} – кількість NS-записів у DNS-відповіді;

s_{NS} – середня дистанція між IP-адресами для множини NS-записів щодо доменного імені;

v_{retry} – значення поля retry, отримане у DNS-відповіді на SOA-запит;

n_{ASN} – кількість різних номерів автономних систем (ASN), до яких належать IP-адреси, пов'язані з серверами імен;

n_{ASA} – кількість різних номерів автономних систем, до яких належать IP-адреси, пов'язані з доменним іменем.

Алгоритм здійснення висновку щодо наявності застосування технологій ухилення на основі DNS подамо у вигляді пари:

$$\beta = \langle U, T_A \rangle. \tag{2.42}$$

Функцію f_2^T локалізації КС, інфікованих ботами, та блокування дій ботів опишемо кортежем:

$$f_2^T = \left\langle \text{H}, f\left(\overline{W_{G_i,G_j}}\right), \wp, \beta, \zeta \right\rangle, i = \overline{1, N_{GQ}}, j = \overline{1, N_{GQ}}, \tag{2.43}$$

де ζ – множина заходів, які застосовуються до КС, визначених як інфіковані, з метою ліквідації інфекції (блокування, усунення вразливостей систем, встановлення (оновлення) антивірусного ПЗ тощо).

Отже, модель процесу виявлення бот-мереж на основі аналізу DNS-трафіка, яка ґрунтується на моделі бот-мереж з врахуванням системи доменних імен та моделі DNS-трафіка, є основою для побудови інформаційної технології виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка, що дозволить здійснювати виявлення ботів бот-мереж централізованої, розподіленої та гібридної архітектури з врахуванням використання ними системи доменних імен на різних стадіях життєвого циклу бот-мережі. Застосування розробленої моделі уможливить підвищення достовірності виявлення бот-мереж в корпоративних мережах.

Висновки до розділу 2

1. Розроблено модель бот-мереж з врахуванням системи доменних імен, а також використання бот-мережами технологій ухилення від виявлення на основі DNS, що надає можливість підвищити достовірність виявлення бот-мереж в корпоративній мережі.

2. Розроблено модель DNS-трафіка та модель процесу виявлення бот-мереж в корпоративних мережах на основі аналізу вхідного DNS-трафіка, яка ґрунтується на розроблених моделі бот-мереж та моделі DNS-трафіка. Розроблена модель заснована на властивості групової активності ботів в DNS-трафіку та застосовує кластерний аналіз векторів ознак, вилучених з корисного навантаження DNS-повідомлень, які вказують на використання бот-мережами технологій ухилення від виявлення на основі DNS. Відмінними рисами запропонованої моделі є уточнений поділ періоду спостереження на інтервали, в межах яких здійснюється пошук груп інфікованих комп’ютерних систем, що ґрунтується на основі аналізу значень полів TTL, які містяться в DNS-повідомленнях; використання нової ознаки синхронності DNS-запитів; врахування особливостей поведінки груп інфікованих комп’ютерних систем, характерних для бот-мереж; залучення кластерного аналізу множини ознак, одержаних з корисного навантаження DNS-повідомлень, які вказують на використання технологій ухилення від виявлення на основі DNS, та здійснення активного DNS-зондування з метою усунення невизначеності частини

результатів кластеризації, що дозволило підвищити достовірність виявлення бот-мереж.

3. Розроблена модель процесу виявлення бот-мереж в корпоративних мережах на основі аналізу вхідного DNS-трафіка є основою методів виявлення бот-мереж на основі аналізу DNS-трафіка, які дозволять виявляти ще невідомі боти, а також здійснювати виявлення вже на початковій стадії поширення ботів в мережі.

РОЗДІЛ 3
МЕТОДИ ВИЯВЛЕННЯ БОТ-МЕРЕЖ НА ОСНОВІ АНАЛІЗУ DNS-ТРАФІКА

3.1 Основи інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка

Аналіз відомих інформаційних технологій виявлення бот-мереж показав низький рівень достовірності виявлення ними невідомих ботів бот-мереж. Враховуючи той факт, що більшість бот-мереж в процесі функціонування використовують систему доменних імен, а також розглянуті вище переваги підходів виявлення бот-мереж на основі DNS, актуальною задачею є розробка ІТ виявлення бот-мереж в корпоративних мережах, яка дозволить консолідувати переваги відомих технологій виявлення бот-мереж на основі DNS.

При здійсненні виявлення невідомих ботів виникає протиріччя між двома важливими показниками: інформаційною безпекою корпоративної мережі та оперативністю роботи ІС корпоративної мережі. Для підвищення оперативності роботи ІС корпоративної мережі потрібно знижувати рівень хибних спрацювань, відкидаючи певну частину підозрілих об’єктів аналізу, проте з метою підвищення рівня інформаційної безпеки корпоративної мережі потрібно забезпечити відповідну реакцію на такі підозрілі об’єкти, зменшуючи рівень пропусків шкідливого ПЗ.

Тому, з метою підвищення рівня достовірності виявлення бот-мереж було розроблено інформаційну технологію виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка. Використання для аналізу виключно DNS-трафіка дозволяє підвищити оперативність виявлення бот-мереж.

Основними засадами, на яких ґрунтується запропонована інформаційна технологія, є:

1) виявлення бот-мереж здійснюється в межах корпоративної мережі;

2) вхідними даними для ІТ є вхідний DNS-трафік, який збирається за допомогою множини мережних давачів, підключених до дзеркалюючих портів керованих комутаторів;

3) інформаційна технологія виявлення бот-мереж заснована на використанні розробленої моделі процесу виявлення бот-мереж в мережах на основі аналізу DNS-трафіка M_D (2.7);

4) засобами ІТ є підсистема ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку та підсистема виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS;

5) метою діагностування є підвищення достовірності виявлення відомих та невідомих бот-мереж в корпоративних мережах.

Процес виявлення бот-мереж включає дві основні складові: ідентифікацію бот-мереж на основі їх групової активності в DNS-трафіку та виявлення бот-мереж, які використовують технології ухилення від виявлення на основі DNS. Укрупнена схема процесу виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка подана на рис. 3.1.

Згідно із запропонованою моделлю, процес ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку складається з семи етапів $C_{GA}{}^{T} = \{c_f, c_s, c_m, c_e, c_b, c_p, f\}$, на яких здійснюються поділ КС корпоративної мережі на групи, які імовірно здійснювали групову активність, та оцінка подібності груп КС з врахуванням особливостей поведінки КС в DNS-трафіку, властивих для ботів бот-мереж.

Процес виявлення бот-мереж, які використовують технології ухилення від виявлення на основі DNS, складається з шести складових $C_{ET}{}^{T} = \{c_t, c_w, c_d, \wp, c_u, \beta\}$, призначених для побудови векторів ознак вхідних DNS-повідомлень щодо запитаних комп'ютерними системами корпоративної мережі доменних імен, здійснення кластерного аналізу векторів ознак з метою визначення наявності застосування технологій ухилення бот-мереж на основі DNS та уточнення частини результатів кластеризації в разі їх невизначеності засобами активного DNS-зондування.

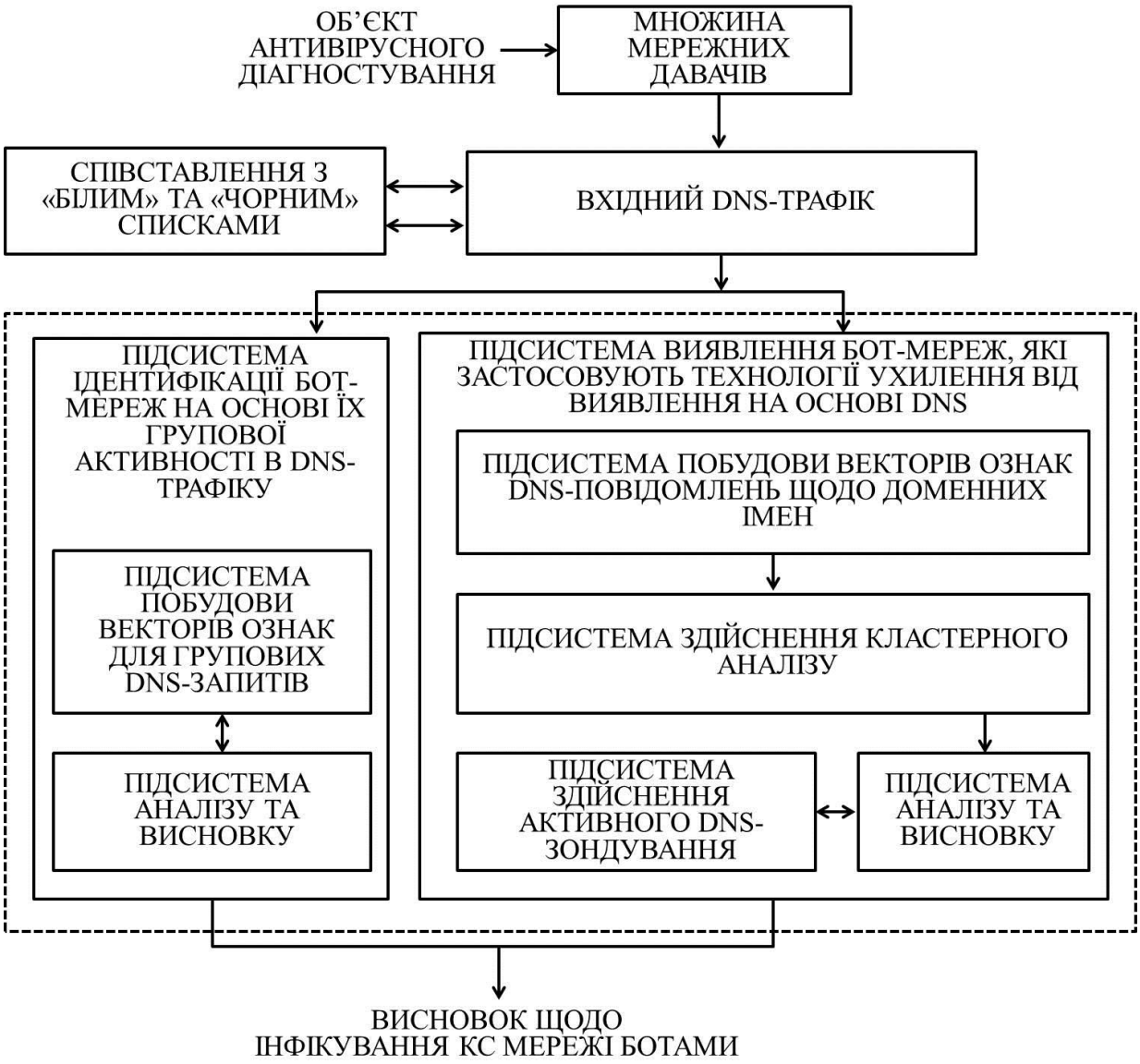


Рисунок 3.1 – Укрупнена схема процесу виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка

Розглянемо більш детально етапи складових частин процесу виявлення бот-мереж в мережах. Підсистема ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку включає підсистему побудови векторів ознак для групових DNS-запитів та підсистему аналізу та висновку.

Підсистема побудови векторів ознак для групових DNS-запитів виконує наступні задачі: аналіз полів TTL з метою виявлення ігнорування TTL-періодів DNS та поділу КС мережі, які здійснювали DNS-запити, на групи; перевірка синхронності DNS-запитів групи КС, які імовірно здійснювали групову активність; побудова векторів ознак для пар групових DNS-запитів щодо різних доменних імен.

Підсистема аналізу та висновку здійснює аналіз групових DNS-запитів щодо однакових доменних імен та аналіз векторів ознак для пар групових DNS-запитів щодо різних доменних імен та формує за визначеними правилами висновок щодо інфікування КС в корпоративній мережі ботами бот-мереж.

Підсистема виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, включає підсистему побудови векторів ознак DNS-повідомлень щодо доменних імен, систему здійснення кластерного аналізу, систему здійснення активного DNS-зондування та систему аналізу та висновку.

Підсистема побудови векторів ознак DNS-повідомлень щодо доменних імен виконує наступні задачі: відбір вхідних DNS-повідомлень на основі значень полів TTL DNS для їх подальшого аналізу; вилучення з DNS-повідомлень ознак, які можуть вказувати на застосування технологій ухилення бот-мереж на основі DNS, та побудова векторів ознак DNS-повідомлень щодо різних доменних імен. Підсистема здійснення кластерного аналізу здійснює нечітку кластеризацію c-means побудованих векторів ознак з частковим навчанням. Результати кластеризації подаються на вхід підсистеми аналізу та висновку, яка за визначеними правилами здійснює висновок щодо інфікування КС в корпоративній мережі ботами бот-мереж.

Підсистема здійснення активного DNS-зондування призначена для здійснення DNS-запитів різних типів з метою вилучення додаткових ознак доменних імен, які можуть вказувати на застосування технологій ухилення бот-мереж на основі DNS, для усунення невизначеності частини результатів кластеризації. Додаткові ознаки, вилучені засобами активного DNS-зондування, аналізуються підсистемою аналізу та висновку, яка за визначеними правилами формує результуючий висновок щодо інфікування КС в корпоративній мережі ботами бот-мереж.

Таким чином, розглянуто основні засади, на яких ґрунтується інформаційна технологія виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка, описано складові та підсистеми процесу виявлення бот-мереж в мережах на основі аналізу DNS-трафіка та виокремлено основні потоки діагностичної інформації.

3.2 Метод ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку

Розроблений метод ідентифікації бот-мереж в корпоративних мережах ґрунтується на властивості групової активності ботів в DNS-трафіку. Така активність проявляється в зосереджених в невеликому проміжку часу групових DNS-запитах КС під час спроб доступу до командно-контролюючих серверів, їх міграціях, виконанні команд або скачуванні оновлень шкідливого програмного забезпечення. Метод враховує особливості поведінки інфікованих груп КС, характерні для багатьох видів бот-мереж: групи КС ігнорують TTL-період DNS, тобто очищують локальні кеші DNS та здійснюють повторні запити щодо доменного імені до завершення TTL-періоду, а також здійснюють DNS-запити, використовуючи нелокальні DNS-сервери. Розроблений метод також дозволяє відслідковувати підвищену кількість порожніх DNS-відповідей з кодом помилки RCODE=3 (NXDOMAIN).

Розроблений метод враховує міграції C&C-серверів та інші DNS-запити, пов’язані з функціонуванням бот-мережі. Метод враховує можливу зміну розмірів груп КС в результаті поширення ботів, і не вимагає великих обсягів обчислювальних ресурсів і значного часу обробки даних.

В [14, 15, 101] для порівняння груп КС використовуються симетричні міри подібності, які доцільно використовувати для оцінки подібності рівновеликих груп. Тому, в роботі використовується несиметрична міра подібності, яка є придатною для оцінки різних за розмірами груп. На відміну від [102], розроблений метод передбачає збір лише DNS-трафіка. На відміну від [14, 15, 101], в розробленому методі поділ періоду моніторингу на інтервали, в межах яких здійснюється пошук інфікованих груп КС, грунтується на основі врахування значеннь TTL, які містяться в DNS-повідомленнях.

Метод складається з наступних кроків:

1) збір вхідного DNS-трафіка;

2) співставлення з «білим» та «чорним» списками доменних імен;

3) виявлення груп КС, які ігнорують TTL-період;

4) побудова вектору щільності розподілу запитів в часі для перевірки синхронності запитів;

5) побудова матриці спостереження для збору та аналізу вхідного DNS-трафіка;

6) виявлення групової активності шляхом аналізу групових запитів щодо одного й того самого доменного імені;

7) побудова нижньотрикутної матриці мір Браун-Бланке для порівняння груп;

8) формування векторів ознак для пар групових запитів;

9) аналіз векторів ознак для ідентифікації інфікованих КС.

Принцип функціонування методу подано на рис.3.2.

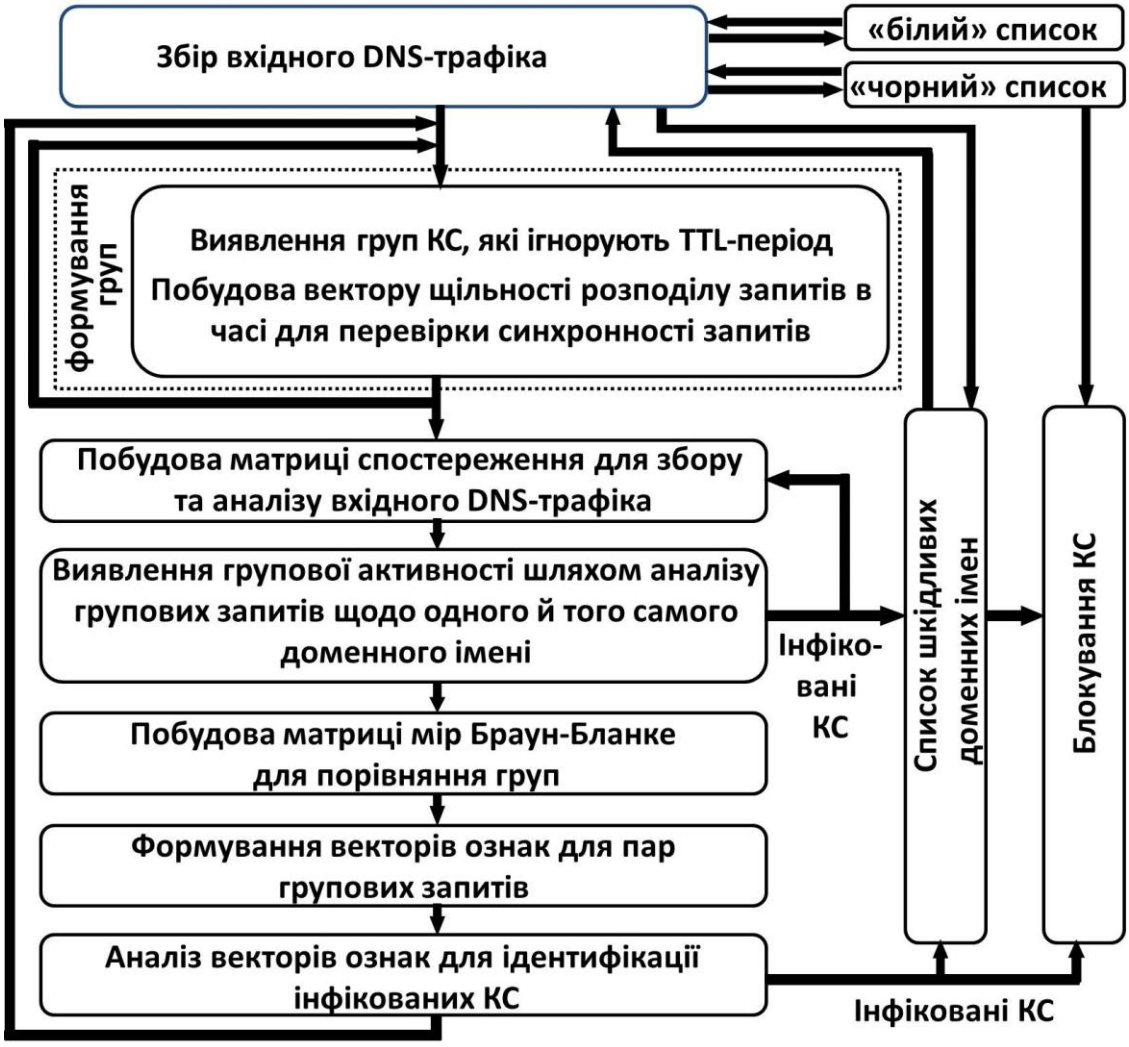


Рисунок 3.2 – Принцип функціонування методу ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку

Розглянемо більш детально етапи методу ідентифікації бот-мереж в корпоративних мережах на основі їх групової активності в DNS-трафіку.

Вхідний DNS-трафік збирається за допомогою множини мережних давачів $E = \{e_i\}_{i=1}^{N_E}$, де N_E – кількість давачів, підключених до дзеркалюючих портів комутаторів.

З метою відкидання легітимних DNS-запитів здійснюється співставлення зібраних даних з «білим» списком відомих легітимних доменних імен. Для виявлення DNS-запитів до відомих шкідливих доменних імен здійснюється співставлення зібраних даних з «чорним» списком відомих шкідливих доменних імен.

На наступному етапі здійснюється виявлення груп КС корпоративної мережі, які ігнорують TTL-період. З цією метою КС очищають локальні кеші DNS для уможливлення здійснення повторних DNS-запитів в межах TTL-періоду DNS. Для виявлення цього факту будується матриця спостереження V_{MAC}, кожен рядок якої містить MAC-адреси КС, які здійснювали запити щодо конкретного доменного імені в межах TTL-періоду.

Таким чином, рядки матриці V_{MAC} містять MAC-адреси КС, які, ймовірно, здійснюють групову активність. Якщо MAC-адреса КС представлена в групі G_d, то у відповідній комірці матриці позначається «1», інакше – «0». Якщо КС повторно надсилала запит щодо доменного імені d, то MAC-адреса КС позначається «1» в рядку матриці V_{MAC}, створеному для повторного запиту:

$$V_{MAC}(h_{j,i}) = \begin{cases} 0, if\ h_j \notin G_d \\ 1, if\ h_j \in G_d \end{cases}, where\ j = \overline{1, N_G}, \quad (3.1)$$

де h_j – MAC-адреси КС, які здійснювали DNS-запити щодо d в межах TTL, i – номер рядка матриці.

Приклад матриці спостереження V_{MAC} подано на рис. 3.3.

h_1	h_2	h_3	h_4	…	h_j
1	1	1	1	…	1
1	0	1	1	...	0

Рисунок 3.3 – Матриця спостереження V_{MAC}

Формування груп триває до спливання найбільшого значення TTL-періоду, отриманого в DNS-відповіді щодо повторного запиту.

Позначимо N_G та $N_{G_{rep}}$ розміри груп для попереднього та повторного групових запитів, δ – порогове значення подібності між двома групами. Якщо $\delta \cdot N_G > N_{G_{rep}}$, то рядок матриці V_{MAC} для повторного запиту відкидається. Для групових запитів, які не були відкинуті на цьому етапі, перевіряється їх синхронність, як описано нижче.

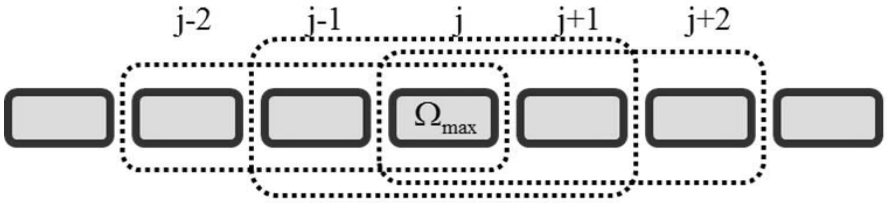
На наступному етапі здійснюється побудова вектору щільності розподілу DNS-запитів в часі для перевірки синхронності DNS-запитів. Групи запитів можна розглядати як синхронні, якщо спостерігається велика кількість запитів для доменного імені в межах часу, коли боти бот-мережі здійснюють запити – часу синхронізації ботів ts. Тому, для перевірки синхронності DNS-запитів виконуються наступні кроки. Якщо інтервал часу між першим та останнім DNS-відгуками Δt для групового запиту щодо доменного імені d більший, ніж тривалість часового вікна ts, то інтервал часу Δt розбивається на z підінтервалів: $z = (t_{last} - t_{first}) / (\frac{1}{3} ts)$, де t_{last} та t_{first} – час надходження останнього та першого DNS-відгуків щодо доменного імені d в межах TTL-періоду, протягом якого здійснюється пошук групової активності або зафіксовано групове очищення локальних кешів DNS. Такий поділ надає можливість мінімізувати кількість DNS-запитів, які не потрапили в інтервал ts (рис. 3.4, а).

Для групового запиту будується z-елементний вектор щільності розподілу запитів в часі $\overline{W_d} = (\Omega_j)_{j=1}^{z}$, де Ω_j – кількість запитів в межах z-го інтервалу. Для елемента вектора $\overline{W_d}$ з максимальним значенням $\Omega_{\max}$ в межах $j = \max \pm 2$ відшукуються два суміжні елементи з найбільшими значеннями

таким чином, щоб всі три елементи описували розподіл запитів неперервного інтервалу часу, та обчислюється їх сума (Sum_s).

Якщо $(1-\delta)(Sum_s + Sum_r) \geq Sum_r$, то множини MAC-адрес груп КС в матриці V_{MAC} об'єднуються, і груповий запит підлягає подальшому аналізу, інакше така група відкидається, де Sum_r – сума значень решти елементів вектора $\overline{W_d}$ (рис. 3.4, б).

а)



б)

Рисунок 3.4 – Перевірка синхронності запитів:

а) поділ інтервалу Δt на z підінтервалів;

б) пошук найбільшої кількості DNS-запитів в межах інтервалу часу ts

На наступному етапі з метою збору та подальшого аналізу вхідного DNS-трафіка для кожного визначеного інтервалу часу моніторингу t_m будується матриця спостереження M_m, m – номер ітерації спостереження. Вона містить доменні імена d_i, запитані групами КС; MAC-адреси груп КС h_j, отримані з матриці V_{MAC}; ознаку звертання до локальних/нелокальних DNS-серверів, S, ознаку повторного запиту в межах TTL-періоду, F; ознаку наявності у DNS-відповідях коду помилки NXDOMAIN, R; ознаку "інфікований" чи "підозрілий" щодо групи КС, отриману на проміжних етапах аналізу, M; номер

ітерації спостереження, на якій зафіксовано ознаку “підозрілий”, N; кількість КС у групі, N_G.

Якщо було виявлено синхронність запитів, то множини MAC-адрес h_j груп КС переносяться з матриці V_{MAC} до матриці спостереження M_m.

Якщо було виявлено групове очищення локальних кешів DNS, то у комірці матриці спостереження $M_m(d_i, F)$ позначається «1», інакше – «0»:

$$M_m(d,F)=\begin{cases}0, if\ \left(\forall n\left(h_j^{\Delta t}\right)=1\right)\vee \gamma<\delta,\\ 1, if\ \gamma\geq\delta,\end{cases} \tag{3.2}$$

де $n\left(h_j^{\Delta t}\right)$ – кількість поява MAC-адреси КС в Δt.

Якщо група КС надсилала запити щодо доменного імені d_i як до локального, так і до інших DNS-серверів, то у комірці матриці спостереження $M_m(d_i, S)$ позначається «0», якщо лише до локального DNS-сервера – «0.5», якщо лише до нелокальних DNS-серверів – «1»:

$$M_m(d,S)=\begin{cases}0, if\ \left(\exists\chi_{d,k,i,IP}\in\Xi\right)\wedge\left(\exists\chi_{j,k,i,IP}\notin\Xi\right),\\ 0.5, if\ \forall\chi_{d,k,i,IP}\in\Xi,\\ 1, if\ \forall\chi_{d,k,i,IP}\notin\Xi,\end{cases} \tag{3.3}$$

де Ξ – множина IP-адрес локальних DNS-серверів мережі.

Якщо DNS-відгуки для групи містили код помилки NXDOMAIN, то у комірці матриці спостереження $M_m(d_i, R)$ позначається «1», інакше «0»:

$$M_m(d,R)=\begin{cases}1, if\ \chi_{j,k,i,HD,RC}=3,\\ 0\ otherwise.\end{cases} \tag{3.4}$$

Комірки матриці спостереження $M_m(d_i, M)$ та $M_m(d_i, N)$ заповнюються нулями. В комірку матриці спостереження $M_m(d_i, N_G)$ заноситься кількість MAC-адрес, представлених у відповідній групі.

З метою зниження рівня хибних спрацювань прийнято пороговий розмір інфікованих груп n_t=4, які можуть бути виявлені запропонованим методом. DNS-запити груп меншого розміру відкидаються. Приклад побудови матриці спостереження M_m представлено на рис. 3.5.

	h_1	h_2	…	hj	N_G	S	F	R	M	N
d_1	1	1	…	1	5	1	1	0	0	0
d_1	1	0	…	1	4	1	0	0	0	0
d_2	1	0	…	1	6	0.5	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…

Рисунок 3.5 – Матриця спостереження M_m

Для порівняння *двох* груп КС G_1 та G_2, які надсилали DNS-запити щодо двох доменних імен d_1 та d_2 в інтервалах часу Δt_1 та Δt_2 відповідно, використано коефіцієнт Браун-Бланке [130-132], який є несиметричною мірою подібності, а тому придатний для оцінки різних за розмірами груп і дозволяє оцінити подібності двох груп КС з високою точністю:

$$K_B(G_1, G_2) = \frac{N_o}{\max\left[N_{G_1}, N_{G_2}\right]}, \tag{3.5}$$

де N_o – кількість спільних елементів в групах G_1 та G_2;

N_{G_1} та N_{G_2} – кількість КС в групах G_1 та G_2 відповідно; $K_B(G_1, G_2) \in [0,1]$.

Якщо кількість порівнюваних груп більша двох, то для оцінки подібності груп КС використовується індекс дисперсності Коха [133]:

$$K_K(G_1,...,G_q) = \frac{C - A}{(q-1)\cdot A}, \tag{3.6}$$

де $G_1,...,G_q$ – порівнювані групи КС;

q – кількість порівнюваних груп;

$C = \sum_{i=1}^{q} N_{G_i}$ – загальна кількість MAC-адрес в усіх групах;

A – кількість різних MAC-адрес, представлених в групах; $K_K(G_1,...,G_q) \in [0,1]$.

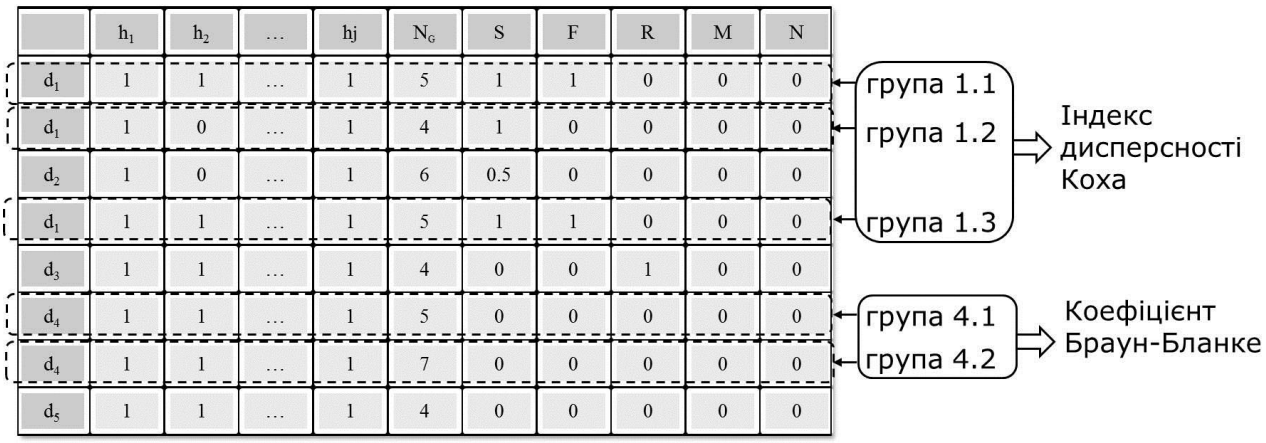
Групи КС вважатимуться інфікованими, якщо коефіцієнт подібності для груп перевищує порогове значення $K_B \geq \delta$ або $K_K \geq \delta$, де δ – порогове значення подібності. Також, введемо додатково порогове значення подібності δ`, яке вказує на підозрілість груп КС, якщо $\delta` \leq K_B < \delta$ або $\delta` \leq K_K < \delta$. В якості ідентифікаторів КС в мережі використаємо MAC-адреси за умови забезпечення запобігання підміни MAC-адрес.

На наступному етапі здійснюється аналіз матриці спостереження M_m з метою виявлення групових запитів щодо однакових доменних імен. Для цього порівнюються групи за MAC-адресами. В залежності від кількості групових запитів щодо певного доменного імені d обирається коефіцієнт Браун-Бланке для порівняння двох груп або індекс дисперсності Коха для 3 і більше груп (рис. 3.6, а). Якщо результат порівняння перевищує поріг $K_B \geq \delta$ або $K_K \geq \delta$, то групи КС вважаються інфікованими.

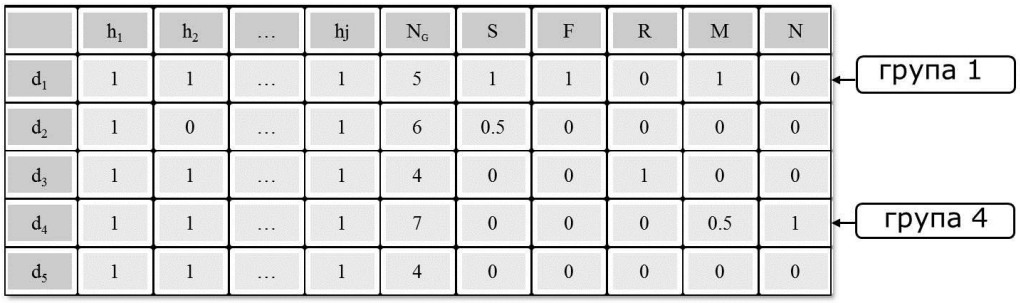
Якщо результат порівняння становить $\delta` \leq K_B < \delta$ або $\delta` \leq K_K < \delta$, то здійснюється додатковий аналіз матриці спостереження M_m щодо наявності факту ігнорування TTL-періоду групами, $M_m(d_i, F) = 1$, та використання групами нелокальних DNS-серверів, $M_m(d_i, S) = 1$. Якщо для будь-якого з групових запитів спостерігалось групове ігнорування TTL-періоду або для всіх запитів спостерігалось звертання до нелокальних DNS-серверів, то групи КС вважаються інфікованими. Інакше, групи КС вважаються підозрілими.

Якщо групи, які запитували одне й те саме доменне ім’я, визначені інфікованими або підозрілими, то множини їх MAC-адрес об’єднуються в один рядок для доменного імені d в матриці спостереження M_m (рис. 3.6, б) з метою подальшого пошуку пов’язаних з групою DNS-запитів. Якщо група КС була визначена як інфікована, то в комірці матриці спостереження $M_m(d, M)$ проставляється «1», якщо група КС була визначена як підозріла – «0.5»:

$$M_m(d,M)=\begin{cases}1, if\ \tau(G_1,...,G_n)=\text{infected},\\ 0.5, if\ \tau(G_1,...,G_n)=\text{suspicious}.\end{cases} \quad (3.7)$$



а)



б)

Рисунок 3.6 – Матриця спостереження M_m:

а) порівняння групових запитів для одного й того самого доменного імені d;

б) матриця спостереження M_m після об'єднання множин MAC-адрес в один рядок для доменного імені d

Якщо група КС була визначена інфікованою, доменне ім'я d заноситься до списку шкідливих доменних імен.

У випадку об'єднання множин MAC-адрес КС комірки матриці спостереження M_m заповнюються за наступними правилами. Якщо для будь-якого з групових запитів спостерігалось групове ігнорування TTL-періоду, в комірці матриці спостереження $M_m(d,F)$ проставляється «1», інакше – «0»:

$$M_m(d,F)=\begin{cases}1, if\ \exists F=1,\\ 0\ otherwise,\end{cases} \tag{3.8}$$

де $\left\{F|F\in\left\{M_m(d_j,F)\right\}_{j=G_1}^{G_n}\right\}$.

Якщо запити груп КС здійснювались як до локального, так і до інших DNS-серверів, то у комірці матриці спостереження $M_m(d,S)$ проставляється «0»; якщо запити груп здійснювались лише до локального DNS-сервера, то у комірці матриці спостереження $M_m(d,\mathrm{S})$ проставляється «0.5», якщо запити груп здійснювались до нелокальних DNS-серверів, то у комірці матриці спостереження $M_m(d,\mathrm{S})$ проставляється «1»:

$$M_m(d,S)=\begin{cases}0.5, if\ \forall S=0.5,\\ 1, if\ \forall S=1,\\ 0\ otherwise,\end{cases} \tag{3.9}$$

де $\left\{S|S\in\left\{M_m(d_j,S)\right\}_{j=G_1}^{G_n}\right\}$

Якщо DNS-відгуки для останньої групи КС містили код помилки NXDOMAIN, то у комірці матриці спостереження $M_m(d,R)$ проставляється «1», інакше «0»:

$$M_m(d,R)=\begin{cases}1, if\ M_m(d_{G_n},R)=1,\\ 0\ otherwise,\end{cases} \tag{3.10}$$

де $M_m(d_{G_n},R)$ – значення комірки для групи КС, що останньою запитувала доменне ім’я.

В комірках матриці спостереження $M_m(d,N)$ для груп КС, які вважаються підозрілими, проставляється номер ітерації спостереження:

$$M_m(d,N)=m. \tag{3.11}$$

Якщо жодна з умов не задовольняється, то групові запити для таких доменних імен видаляються з матриці спостереження M_m.

На наступному етапі на основі матриці спостереження M_m будується нижньотрикутна матриця мір Браун-Бланке B_m. Ознаки N_G, S, F, R, M, N з матриці M_m переносяться до матриці B_m. Рядки матриці B_m формуються за зростанням кількості MAC-адрес в групах N_G, по стовпцях. Також, в матрицю B_m заносяться коефіцієнти Браун-Бланке, обчислені для пар груп КС (рис. 3.6). Обчислення значень комірок для кожного стовпця припиняється, якщо $N_{Gi} / N_{Gi+1} < \delta`$ (тобто відношення розмірів порівнюваних груп є меншим за порогове значення δ' – порожні комірки матриці B_m, показані на рис. 3.7). Це дозволить зменшити час та обчислювальні ресурси, необхідні для аналізу.

	d_3	d_5	d_1	d_2	d_4	…	N_G	S	F	R	M	N
d_3	1					…	4	0	0	1	0	0
d_5	1	1				…	4	0	0	0	0	0
d_1	0.8	0.8	1			…	5	1	1	0	1	0
d_2			0.5	1		…	6	0.5	0	0	0	0
d_4			0.71	0.43	1	…	7	0	0	0	0.5	1

Рисунок 3.7 – Матриця мір Браун-Бланке B_m

Для кожної пари групових запитів, якщо виконується умова $K_B \geq \delta`$, згідно матриці мір Браун-Бланке B_m формується вектор ознак $\overline{W_{G_1,G_2}}$ (рис. 3.8).

Вектор складається з п’яти елементів: коефіцієнт Браун-Бланке та зведені поведінкові ознаки для двох порівнюваних груп, отримані на основі матриці B_m, які можуть приймати наступні значення: «Unusual» (непритаманна ботам), «Neutral» (властива як користувачам, так і ботам), «Suspicious», «Dangerous» (властива ботам):

$$\overline{W_{G_1,G_2}} = \left(K_B(G_1,G_2), S_{G_1,G_2}, F_{G_1,G_2}, R_{G_1,G_2}, M_{G_1,G_2}\right), \quad (3.12)$$

де $S_{G_1,G_2}, F_{G_1,G_2}, R_{G_1,G_2}, M_{G_1,G_2}$ – зведені поведінкові ознаки для двох порівнюваних груп.

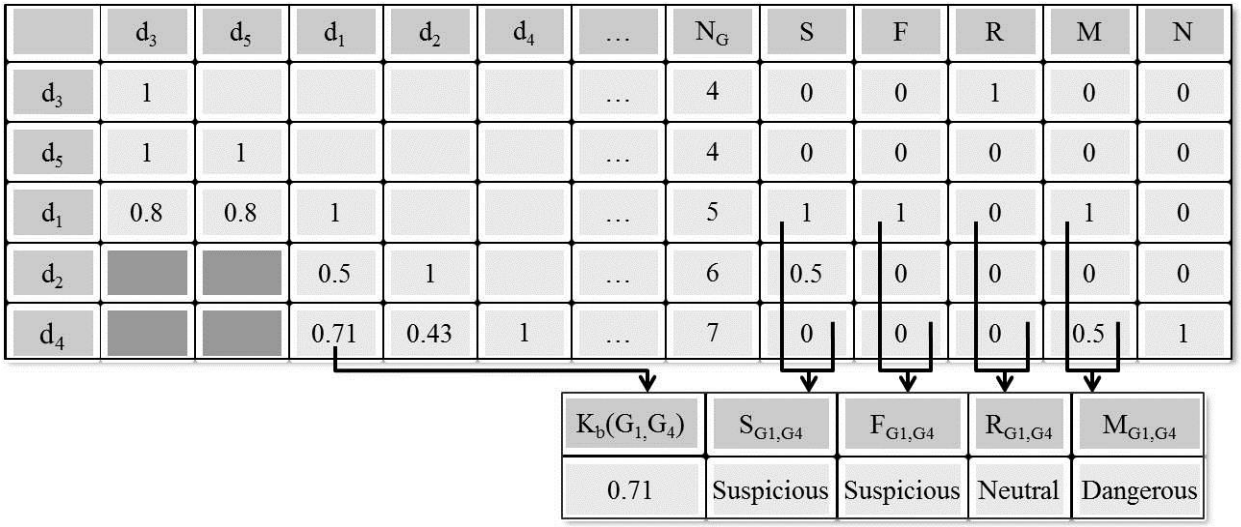


Рисунок 3.8 – Формування векторів ознак для пар групових запитів з матриці мір Браун-Бланке B_m

Зведені поведінкові ознаки S_{G_1,G_2} та M_{G_1,G_2} можуть бути визначені наступним чином:

$$S_{G_1,G_2} = \begin{cases} Unusual, if\ B_m(d_1,S) = B_m(d_2,S) = 0, \\ Neutral, if\ B_m(d_1,S) = B_m(d_2,S) = 0.5, \\ Dangerous, if\ B_m(d_1,S) = B_m(d_2,S) = 1, \\ Suspicious\ otherwise. \end{cases} \tag{3.13}$$

$$M_{G_1,G_2} = \begin{cases} Neutral, if\ B_m(d_1,M) = B_m(d_2,M) = 0, \\ Suspicious, if\ ((B_m(d_1,M) = 0.5 \vee B_m(d_2,M) = 0.5) \wedge \\ \wedge B_m(d_1,M) \neq 1 \wedge B_m(d_2,M) \neq 1) \wedge B_m(d_1,M) \neq B_m(d_2,M), \\ Dangerous, if\ B_m(d_1,M) = 1 \vee B_m(d_2,M) = 1 \vee \\ \vee (B_m(d_1,M) = B_m(d_2,M) = 0.5 \wedge B_m(d_1,N) \neq B_m(d_2,N) \vee \\ \vee B_m(d_1,N) = B_m(d_2,N) = 0). \end{cases} \tag{3.14}$$

Зведені ознаки F_{G_1,G_2} та R_{G_1,G_2} визначаються аналогічно. Нижче наведено приклад для першої з них:

$$F_{G_1,G_2} = \begin{cases} Neutral, if\ B_m(d_1,F) = B_m(d_2,F) = 0, \\ Suspicious, if\ B_m(d_1,F) \neq B_m(d_2,F), \\ Dangerous, if\ B_m(d_1,F) = B_m(d_2,F) = 1. \end{cases} \quad (3.15)$$

На наступному етапі з метою ідентифікації інфікованих КС здійснюється аналіз векторів ознак для пар групових запитів. Аналіз векторів ознак $\overline{W_{G_1,G_2}}$ здійснюється за наступними правилами, де функція виходу $f\left(\overline{W_{G_1,G_2}}\right)$ може приймати чотири значення: «Not_Infected» (неінфіковані), «Not_Suspicious» (не підозрілі), «Suspicious» (підозрілі), «Infected» (інфіковані):

$$f\left(\overline{W_{G_1,G_2}}\right) = \begin{cases} Not\_Infected, if\ K_B(G_1,G_2) < \delta \wedge S_{G_1,G_2} = Unusual \wedge \\ \wedge \forall \overline{W_{G_1,G_2}}(j) \neq Suspicious\ \wedge \forall \overline{W_{G_1,G_2}}(j) \neq Dangerous, \\ Not\_Suspicious, if\ K_B(G_1,G_2) < \delta \wedge S_{G_1,G_2} \neq Unusual \wedge \\ \wedge \forall \overline{W_{G_1,G_2}}(j) \neq Suspicious\ \wedge \forall \overline{W_{G_1,G_2}}(j) \neq Dangerous, \\ Infected, if\ \exists \overline{W_{G_1,G_2}}(j) = Dangerous\ \vee K_B(G_1,G_2) \geq \delta, \\ Suspicious\ otherwise. \end{cases} \quad (3.16)$$

де $j = \overline{2,5}$ – номер елемента в векторі ознак.

Одна й та сама група в межах ітерації може отримати декілька різних оцінок. В такому випадку пріоритет має оцінка з вищим ступенем небезпечності. Групи КС, які було визначено як не інфіковані, відкидаються. Щодо груп КС, визначених як інфіковані, здійснюються заходи з метою ліквідації інфекції (блокування, усунення вразливостей системи, встановлення (оновлення) антивірусного ПЗ тощо).

Групи КС з матриці спостереження M_m, які не потрапили до матриці мір Браун-Бланке B_m, та групи, для яких не було виконано умову $K_B \geq \delta'$, а також групи, визначені як не підозрілі та підозрілі, аналізуються разом з даними, що будуть отримані на наступній ітерації спостереження (матриця спостереження M_{m+1}) з метою виявлення можливих повторних групових запитів. При цьому, якщо група, яка запитувала доменне ім'я d, була визначена підозрілою, в комірці матриці спостереження $M_{m+1}(d,M)$ для цієї групи проставляється «0.5», а в комірці $M_{m+1}(d,N)$ – номер ітерації m.

Отже, розроблено новий метод ідентифікації бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка. Розроблений метод ґрунтується на властивості групової активності ботів в DNS-трафіку та враховує анормальну поведінку груп КС, властиву багатьом видам бот-мереж.

Розроблений метод дозволяє виявляти ще невідомі боти вже на початковій стадії поширення інфекції в корпоративній мережі та може бути застосований як до малих, так і до великих мереж, і не вимагає значних обсягів обчислювальних ресурсів для обробки даних.

3.3. Метод виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS

З метою виявлення бот-мереж, що використовують технології ухилення на основі DNS, розроблено новий метод, який базується на кластерному аналізі ознак, отриманих шляхом дослідження корисного навантаження DNS-повідомлень, та використовує активне DNS-зондування. Метод використовує нечітку кластеризацію с-середніх з частковим навчанням. Застосування нечіткої кластеризації дозволяє отримувати високу точність та змістовність результату кластеризації [134, 135]. Об’єктами кластеризації є вектори ознак використання ботами технологій ухилення від виявлення на основі DNS, які отримано на основі пасивного моніторингу вхідного DNS-трафіка.

Метод складається з наступних кроків (рис.3.9):

1) збір вхідного DNS-трафіка мережі;

2) аналіз полів TTL вхідних DNS-повідомлень щодо певного доменного імені;

3) вилучення ознак з вхідних DNS-повідомлень щодо певного доменного імені та побудова вектору ознак;

4) побудова матриці даних на основі векторів ознак;

5) здійснення нечіткої кластеризації з частковим навчанням з метою виявлення запитів, які можуть свідчити про функціонування ботів, що належать

до бот-мереж, які використовують технології ухилення від виявлення на основі DNS;

6) здійснення активного DNS-зондування;

7) локалізація КС, інфікованих ботами, що належать до бот-мереж, які використовують технології ухилення від виявлення на основі DNS, та блокування дій ботів.

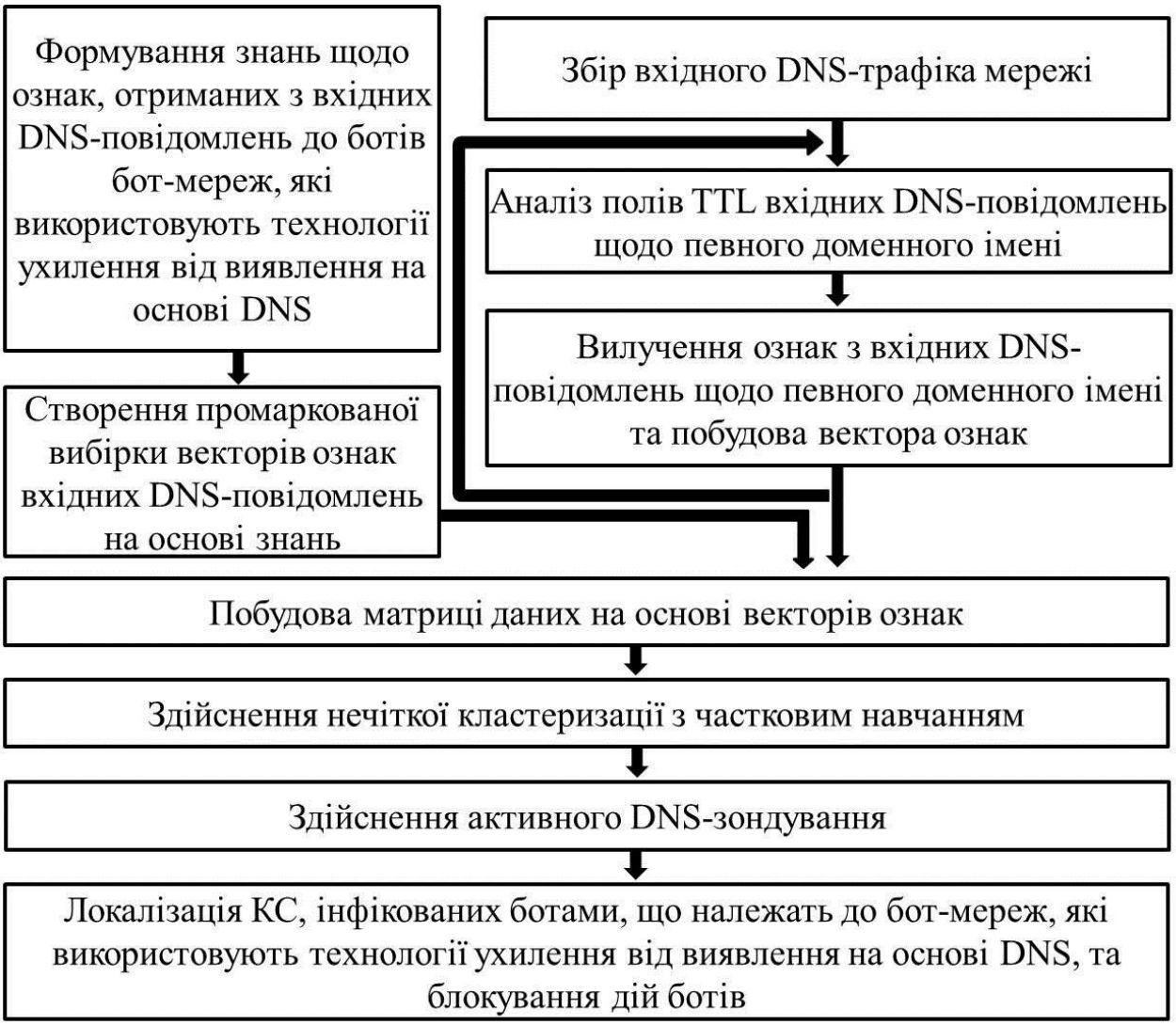


Рисунок 3.9 – Принцип функціонування методу виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS

Подамо вектор ознак, отриманих з вхідних DNS-повідомлень щодо певного доменного імені d, наступним чином:

$$\overline{W_e} = (l_N, n_U, e_N, t_{\text{mod}}, t_{med}, t_{aver}, n_A, n_{IP}, s_{IP}, s_A, n_{UA}, s_{UA}, n_D, f_{UR}, e_R, l_P, f_S), \quad (3.17)$$

де l_N – довжина доменного імені;

n_U – кількість унікальних символів в доменному імені;

e_N – ентропія доменного імені;

t_{mod} – TTL-період, мода (значення у множині спостережень, яке зустрічається найбільш часто; застосуємо мінімальне значення у випадку, якщо множина є мультимодальною);

t_{med} – TTL-період, медіана (значення ознаки, яке розділяє ранжовану сукупність на дві рівні частини, де 50% нижніх одиниць ряду даних мають значення ознаки, не більше, ніж медіана, а 50% верхніх – не менше, ніж медіана);

t_{aver} – TTL-період, середнє арифметичне значення;

n_A – кількість A-записів, що відповідають доменному імені, у вхідному DNS-повідомленні (ознака використовується, якщо $n_A > 1$);

n_{IP} – кількість IP-адрес, пов'язаних з доменним ім'ям (ознака використовується, якщо $n_A = 1$);

s_{IP} – середня дистанція між IP-адресами, пов'язаними з доменним ім'ям (ознака використовується, якщо $n_A = 1$);

s_A – середня дистанція між IP-адресами в множині A-записів для доменного імені у вхідному DNS-повідомленні (ознака використовується, якщо $n_A > 1$);

n_{UA} – кількість унікальних IP-адрес в множинах A-записів, що відповідають доменному імені, у вхідних DNS-повідомленнях (ознака використовується, якщо $n_A > 1$);

s_{UA} – середня дистанція між унікальними IP-адресами в множинах A-записів, що відповідають доменному імені, у вхідних DNS-повідомленнях (ознака використовується, якщо $n_A > 1$);

n_D – кількість доменних імен, які спільно використовують IP-адресу;

f_{UR} – бінарна ознака використання рідковживаних типів записів DNS (KEY, NULL тощо), або таких, які зазвичай не використовуються клієнтами (наприклад, TXT, які найбільш часто використовуються для тунелювання);

e_R – максимальне значення ентропії записів DNS, які містяться в DNS-повідомленнях (CNAME, TXT, NS, MX, KEY, NULL тощо);

l_P – середній розмір DNS-повідомлень щодо доменного імені;

f_S – бінарна ознака успішності DNS-запиту ($f_S = 0$, якщо DNS-запит невдалий, $f_S = 1$, якщо DNS-запит успішний).

Також, метод використовує функцію залежності $f_{E_{Bn}}$ ентропії поля DNS-повідомлення від його довжини [16], де n – основа кодування.

На основі ознак, властивих вхідним DNS-повідомленням до ботів, з метою визначення технології ухилення від виявлення на основі DNS [4, 16, 72, 115, 116, 119, 123, 124, 136], формуються знання, які можуть бути представлені у вигляді наступних правил:

$$\begin{gathered}
if\ (t_{\mathrm{mod}} \in [0{,}900]\ and\ \ t_{med} \in [0{,}900]\ and\ t_{aver} \in [0{,}900]\ )\ and \\
and\ \ ((n_A \in (5,\infty)\ and\ \ s_A \in (65535,\infty))or(n_{UA} \in (8,\infty)\ and\ \ s_{UA} \in (65535,\infty))) \Rightarrow\ fast_flux \\
if\ t_{\mathrm{mod}} \in [0{,}900]\ and\ \ t_{med} \in [0{,}900]\ and\ t_{aver} \in [0{,}900]\ and \\
and\ \ f_S = 0\ and\ n_D \in [8;\infty] \Rightarrow domain_\ flux \\
if\ t_{\mathrm{mod}} \in [0{,}900]\ and\ \ t_{med} \in [0{,}900]\ and\ t_{aver} \in [0{,}900]\ and \\
and\ n_{IP} \in (5,\infty)\ and\ s_{IP} \in (65535,\infty) \Rightarrow cycling\ \ of\ IP\ mappings \\
if\ ((l_N \in [75{,}255]\ and\ \ n_U \in (27{,}37])\ or\ (e_N \geq f_{E_{B32}}\ or \\
or\ \ (e_R \geq f_{E_{B64}}\ or\ \ e_R \geq f_{E_{B256}})or\ \ f_{UR} = 1\ ))and\ l_P > 300 \Rightarrow DNS_tunneling\ .
\end{gathered} \tag{3.18}$$

На основі знань щодо ознак вхідних DNS-повідомлень до ботів, які застосовують технології ухилення від виявлення на основі DNS, формується промаркована вибірка.

На основі утвореної вибірки здійснюється часткове навчання кластеризатора. Кожен вектор ознак з промаркованої вибірки даних належить

одному з множини наперед визначених кластерів $X = \{x_i\}_{i=1}^{5}$; приналежність вектора ознак кластеру x_1 свідчить про застосування технології ухилення «cycling of IP mapping», x_2 – «domain flux», x_3 – «fast flux», x_4 – «DNS-tunneling», x_5 – кластер, який містить нормальні DNS-запити.

Вхідний DNS-трафік збирається за допомогою мережних давачів, підключених до дзеркалюючих портів комутаторів.

На наступному етапі здійснюється аналіз полів TTL вхідних DNS-повідомлень щодо певного доменного імені. Враховуючи, що для багатьох видів бот-мереж характерною особливістю поведінки є ігнорування TTL-періоду, тому на основі значень полів TTL опрацьовуються такі вхідні DNS-повідомлення:

1) кожне перше зафіксоване DNS-повідомлення щодо певного доменного імені в межах TTL-періоду;

2) кожне DNS-повідомлення, отримане КС повторно в межах TTL-періоду, якщо джерелом повідомлення є нелокальний DNS-сервер, і TTL-період, зазначений в цьому повідомленні, відрізняється від залишка TTL-періоду, в межах якого було отримане це повідомлення.

Решта вхідних DNS-повідомлень відкидаються.

Якщо політика корпоративної мережі дозволяє здійснювати запити поза локальними DNS-серверами, і КС здійснюватимуть запити щодо одного й того ж доменного імені до різних DNS-серверів, то в DNS-відгуках можуть міститись різні значення TTL-періодів. Проте, така особливість функціонування DNS не впливатиме на рівень виявлення, оскільки вона не береться до уваги для ботів однієї бот-мережі.

Наступним етапом є вилучення ознак з вхідних DNS-повідомлень щодо певного доменного імені та побудова вектору ознак $\overline{W_e}$. Відповідні ознаки вхідних DNS-повідомлень щодо певного доменного імені вилучаються та формуються протягом визначеного періоду моніторингу на основі всіх вхідних DNS-повідомлень, які не було відкинуто на попередньому етапі.

На наступному етапі з векторів ознак вхідних DNS-повідомлень формується матриця даних V, кожен рядок якої є вектором ознак вхідних DNS-повідомлень $\overline{W_e}$ щодо певного доменного імені, $V = (v_{ij})_{i=1,j=1}^{N_D,N_S}$, $V(i,) = \overline{W_e}$, де N_S – загальна кількість ознак вхідних DNS-повідомлень, які вказують на використання технологій ухилення від виявлення на основі DNS.

Наступним етапом є здійснення нечіткої кластеризації c-means з частковим навчанням з метою виявлення запитів, які можуть свідчити про функціонування ботів, що належать до бот-мереж, які використовують технології ухилення від виявлення на основі DNS. Перевагою застосування нечіткої кластеризації є послаблення вимоги здійснення однозначної кластеризації об’єктів, що уможливлюється за рахунок введення функцій приналежності до нечітких кластерів, які приймають значення в інтервалі $[0,1]$. Це дозволяє підвищити точність та інформативність результатів кластеризації у випадках, якщо об’єкти кластеризації розташовані на межах кластерів.

Об’єктами кластеризації є вектори ознак $\overline{W_e}$, які отримано з корисного навантаження вхідних DNS-повідомлень щодо певного доменного імені. В якості відстані між об’єктом кластеризації та центром кластера застосовано норму Махаланобіса, оскільки вона дає кращі результати, ніж Евклідова норма [134], яка використовується в базовому алгоритмі c-means.

Результатом кластеризації є матриця нечіткого розбиття U, де кожен елемент матриці u_{ij} визначає ступінь приналежності і-го елемента множини об’єктів кластеризації до j-го кластера: $U = [u_{ij}], u_{ij} \in [0,1], i = \overline{1,N_D}, j = \overline{1,5}$, $\sum_{j=1,5} u_{ij} = 1$.

Таким чином, кожен вектор ознак з певним ступенем приналежності відноситься до кожного з п’яти кластерів.

Приймемо λ та $\lambda`$ як порогові значення приналежності об’єкта кластеризації до кластера, за яких доменне ім’я вважається шкідливим або підозрілим відповідно. Якщо $u_{ij} \geq \lambda$, $j = \overline{1,4}$, то об’єкт відноситься до кластера, який відповідає одній з технологій ухилення. Належність вектора ознак до п’ятого кластера свідчить про виконання запитів до легітимних ресурсів. У

випадку, якщо $\lambda` \le u_{ij} < \lambda$, $j = \overline{1,4}$, то об'єкт може належати кільком кластерам, і має місце певна невизначеність результатів.

З метою усунення невизначеності частини результатів на наступному етапі здійснюється подальший аналіз отриманих результатів кластеризації, а саме аналіз тих DNS-запитів щодо доменних імен, що потрапили на перетин кластерів, і їх не можна однозначно віднести до шкідливих чи легітимних.

Використані ознаки отримані на основі пасивного моніторингу DNS-трафіка, перевагою якого є прихованість та відсутність необхідності ініціювати специфічні дії, які імітують шкідливу активність, з метою збору необхідної інформації щодо доменного імені. Однак в ситуації, коли вказані засоби не можуть дати результату, доцільним є залучення засобів активного DNS-зондування.

Такий підхід дозволить отримати додаткові ознаки, які можуть вказувати на шкідливість доменного імені, на основі здійснення запитів NS-записів, A-записів, SOA-записів, PTR-записів. Підхід застосовуваний для уточнення результатів для виявлення застосування технологій ухилення «domain flux», «fast flux», періодичної зміни IP-відображення.

Також, здійснюються запити з метою визначення номерів автономних систем (ASN), до яких належать IP-адреси, пов'язані з підозрілими доменними іменами та їх серверами імен (з метою уточнення результатів для виявлення застосування технологій «fast flux» та періодичної зміни IP-відображення).

Таким чином, з метою уточнення частини результатів кластеризації використовуються наступні ознаки [115], отримані засобами активного DNS-зондування:

- n_{NS} – кількість NS-записів у DNS-відповіді;

- s_{NS} – середня дистанція між IP-адресами для множини NS-записів щодо доменного імені;

- v_{retry} – значення поля retry, отримане у DNS-відповіді на SOA-запит;

- n_{ASN} – кількість різних номерів автономних систем (ASN), до яких належать IP-адреси, пов'язані з серверами імен;

- n_{ASA} – кількість різних номерів автономних систем, до яких належать IP-адреси, пов'язані з доменним іменем.

Висновок щодо наявності застосування технологій ухилення на основі DNS здійснюється із застосуванням знань, які можуть бути представлені у вигляді правил:

$$\begin{gathered} if\ t_{\mathrm{mod}} \in [0{,}900]\ and\ \ t_{med} \in [0{,}900]\ and\ t_{aver} \in [0{,}900]\ and \\ and\ \ n_{IP} \in (5,\infty)\ and\ s_{IP} \in (65535,\infty)\ and\ \ n_{ASA} > 2 \Rightarrow cycling\ \ of\ IP\ mappings \\ if\ \ (t_{\mathrm{mod}} \in [0{,}900]\ and\ \ t_{med} \in [0{,}900]\ and\ t_{aver} \in [0{,}900]\ )\ and \\ and\ \ ((n_A \in (5,\infty)\ and\ \ s_A \in (65535,\infty))or(n_{UA} \in (8,\infty)\ and\ \ s_{UA} \in (65535,\infty))or\ n_{AS} > 2)\ and \\ and\ \ (s_{NS} \in (65535,\infty)or\ n_{ASN} > 2\ and\ n_{NS} > 3\ and\ v_{retry} \in [0{,}900]) \Rightarrow\ fast\_flux \\ if\ \ n_D \in [8;\infty] \Rightarrow domain\_\ flux \end{gathered} \quad (3.19)$$

На наступному етапі здійснюється локалізація КС, інфікованих ботами, що належать до бот-мереж, які використовують технології ухилення від виявлення на основі DNS, та блокування дій ботів. На основі приналежності векторів $\overline{W_e}$ до кластерів здійснюється визначення доменних імен, до яких зверталися боти бот-мереж. З метою блокування дій ботів, які здійснювали шкідливі запити, локалізація КС мережі здійснюється за допомогою ведення файлів журналювання MAC-адрес КС, що здійснювали DNS-запити, та запитаних ними доменних імен.

Отже, розроблено новий метод виявлення бот-мереж, що використовують технології ухилення від виявлення на основі DNS. Розроблений метод базується на кластерному аналізі векторів ознак, отриманих шляхом пасивного моніторингу вхідного DNS-трафіка, та використовує активне DNS-зондування з метою отримання додаткових ознак, які вказують на використання технологій ухилення від виявлення бот-мереж на основі DNS.

3.4 Приклад визначення можливої наявності бот-мереж в корпоративній мережі на основі аналізу DNS-трафіка

Для визначення оптимальних параметрів методу ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку було досліджено основні його параметри, а саме час синхронізації запитів ts та порогове значення подібності δ. З цією метою було проведено ряд експериментів (в загальній кількості 30), кожен з яких тривав 8 годин.

Для проведення експериментів було згенеровано програмне забезпечення з функційним навантаженням ботів, з яких утворено бот-мережу з централізованою архітектурою. З цією метою було використано університетську мережу з 100 КС, які були інфіковані згенерованими ботами. DNS-трафік локальної мережі збирався засобами утиліти tcpdump (рис.3.10).

Для визначення оптимального значення часу синхронізації запитів ts (в секундах) було проведено експерименти на інтервалі $ts = \overline{0,30}$ з кроком в 2 секунди. Для визначення оптимальних порогових значень подібності δ та δ' було проведено експерименти для пар значень $\delta' = 0.6,\ \delta = 0.7$ та $\delta' = 0.7,\ \delta = 0.8$.

Також, було досліджено ефективність методу для різної кількості інфікованих КС (для цього при проведенні різних експериментів було інфіковано ботами 90 КС та лише 10 КС).

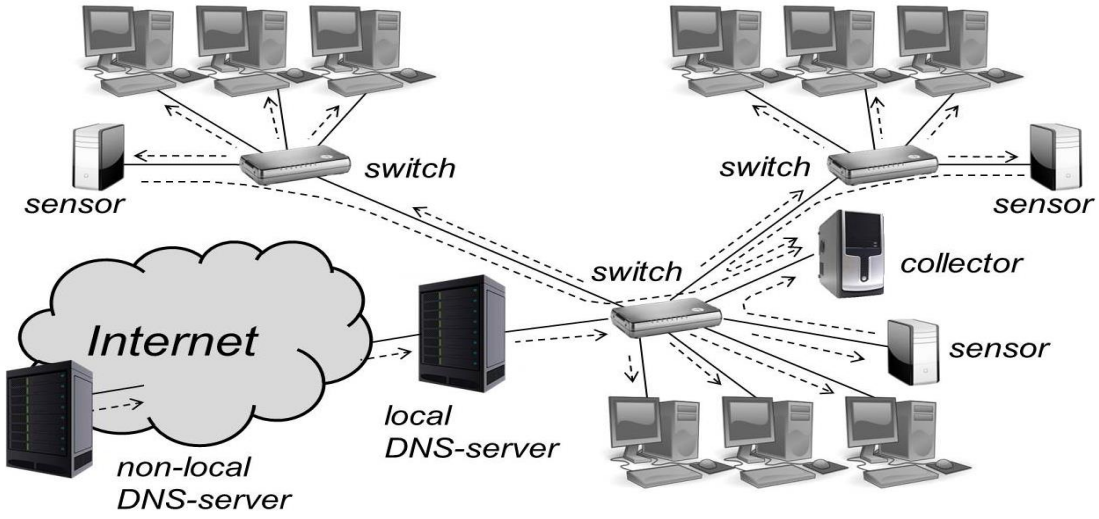


Рисунок 3.10 – Схема мережі, яка використовувалась для виявлення бот-мереж

Результати експериментальних досліджень подано на рис. 3.11, 3.12.

Було встановлено, що зменшення значення ts призводить до зниження рівня виявлення, в той час як його збільшення підвищує рівень виявлення, проте також призводить до збільшення рівня хибних спрацювань. Зменшення порогового значення коефіцієнта подібності δ також призводить до підвищення рівня виявлення, проте підвищує також рівень хибних спрацювань.

Таким чином, експериментально встановлено, що максимальний рівень виявлення з мінімальним рівнем хибних спрацювань досягається при значеннях часу синхронізації ts від 12 до16 секунд та при значеннях порогів подібності $\delta' = 0.7$, $\delta = 0.8$.

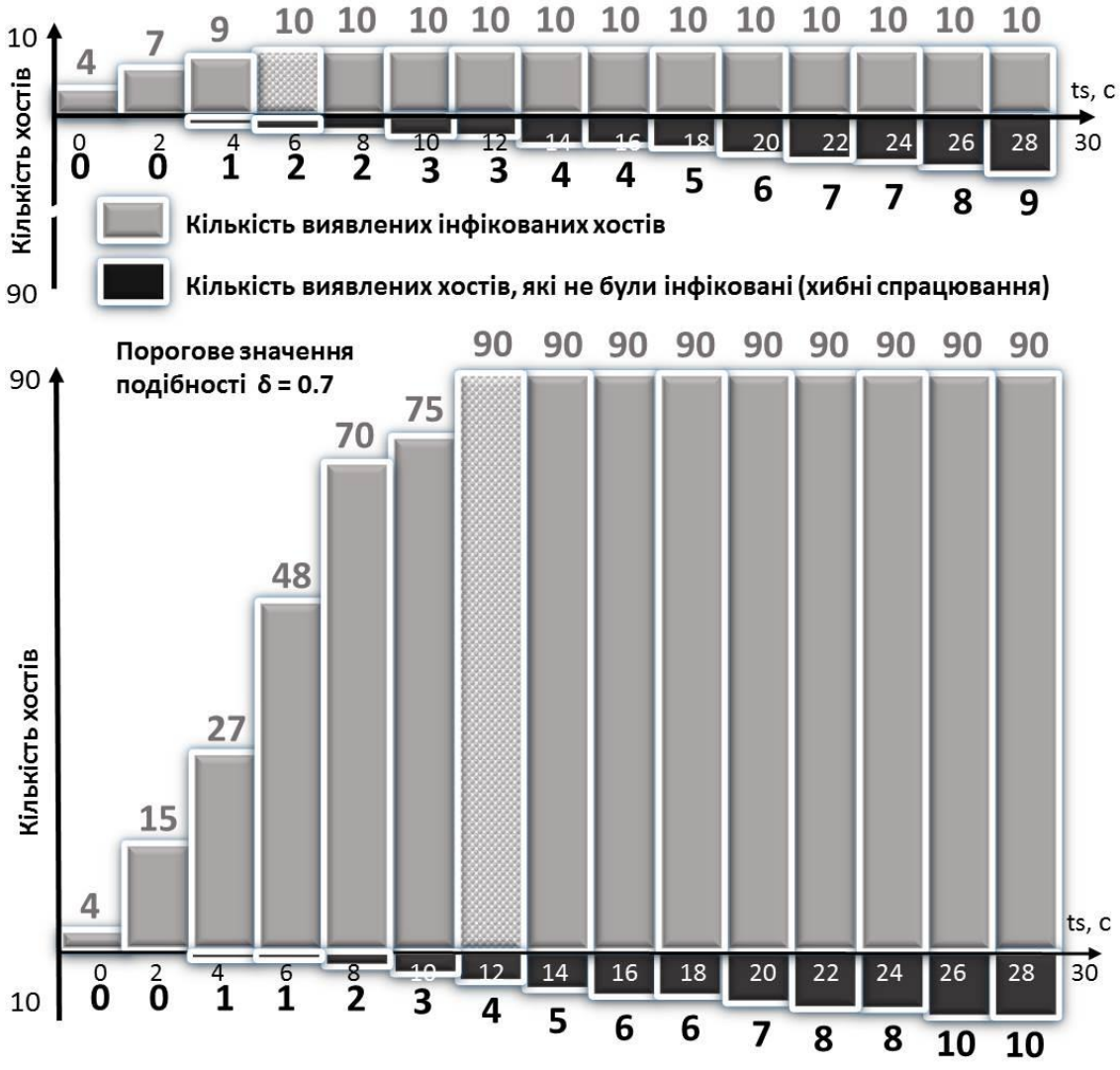


Рисунок 3.11 – Результати експериментальних досліджень:
поріг подібності $\delta' = 0.6$, $\delta = 0.7$

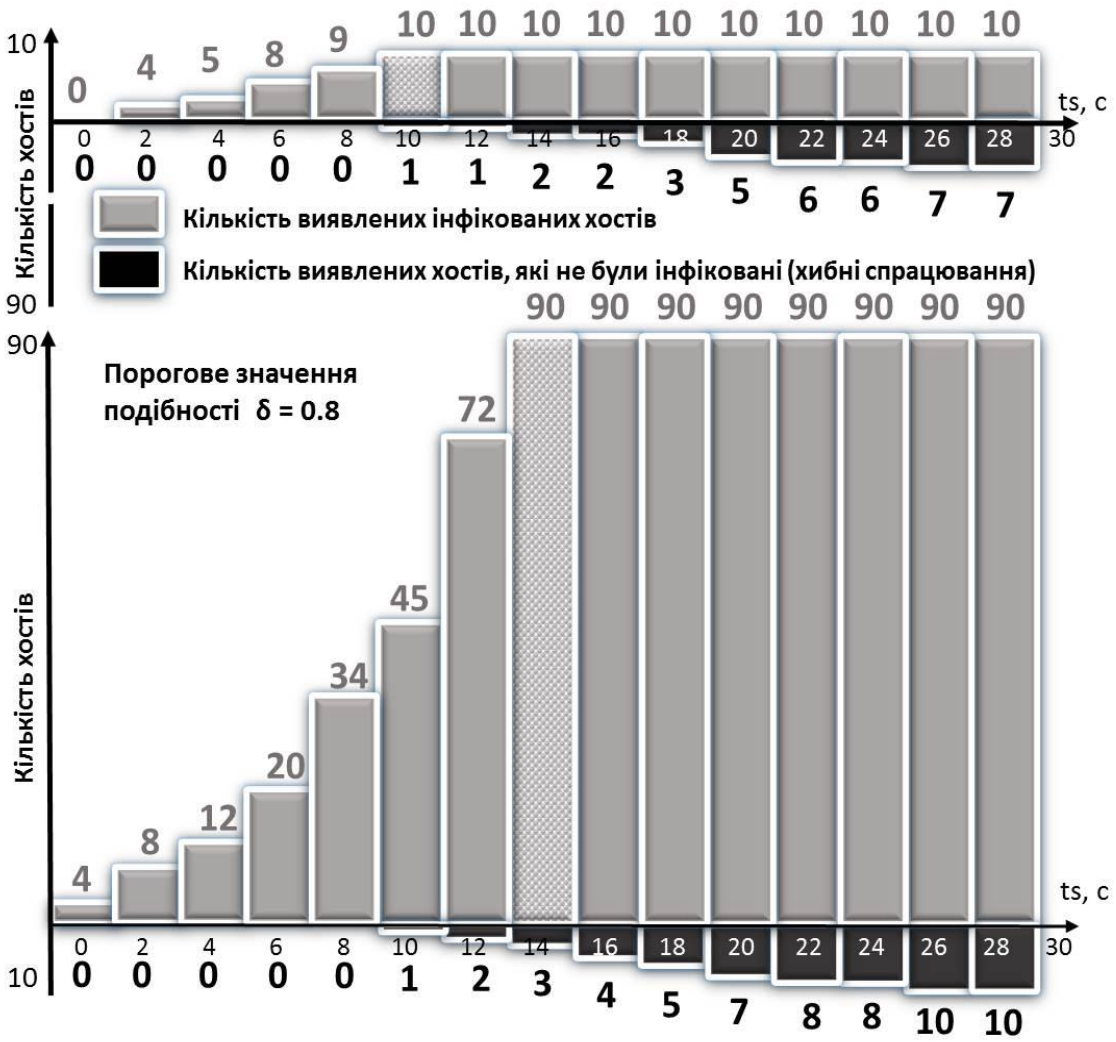


Рисунок 3.12 – Результати експериментальних досліджень:
поріг подібності $\delta' = 0.7$, $\delta = 0.8$

Для прикладу виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, було проведено наступний експеримент. Для проведення експерименту було створено множину спеціального програмного забезпечення з функційним навантаженням ботів бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, в загальній кількості 40 зразків, якими було інфіковано мережу з 100 КС. Також, на час проведення експерименту було зареєстровано множину доменних імен, які було використано в якості доменних імен командно-контролюючих серверів створених тестових бот-мереж, які відтворювали застосування відповідних технологій ухилення від виявлення бот-мереж на основі DNS, і до яких здійснювали відповідні типи DNS-запитів згенеровані боти.

Експеримент тривав 24 години. DNS-трафік локальної мережі збирався засобами утиліти tcpdump. Протягом цього часу шляхом застосування запропонованого методу було проаналізовано та класифіковано 2369 DNS-відповідей.

На основі зібраних даних було побудовано множину векторів ознак, вилучених з вхідних DNS-повідомлень щодо різних доменних імен. Беручи до уваги неоднорідність одиниць виміру та різний порядок величин значень ознак векторів, з метою приведення значень елементів векторів до єдиного діапазону значень було здійснено їх нормування. В якості способу нормування було обрано обчислення стандартизованого вкладу (Z-вкладу), який визначає, скільки стандартних відхилень віддаляє спостереження від середнього значення, за наступною формулою:

$$Z_i = \frac{x_i - \bar{x}}{S}, \tag{3.20}$$

де x_i – елемент вектору ознак;

$\bar{x}$ – середнє арифметичне значення елементів вектору;

S – стандартне відхилення, $S = \sqrt{\frac{n}{n-1}\sigma^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$, де σ^2 – дисперсія, n – кількість елементів вектору.

З нормованих таким чином векторів ознак було побудовано матрицю даних для її подальшої кластеризації. Нечітка кластеризація c-means з частковим навчанням була здійснена засобами пакету прикладних програм для розв’язання задач технічних обчислень Matlab.

Початкові центри кластерів було визначено на основі навчальної вибірки, обсяг якої складав 10% від зібраних для аналізу даних. В якості відстані між вектором ознак та центром кластера застосовано норму Махаланобіса, оскільки вона надає можливість виокремлювати кластери у формі гіпереліпсоїдів з орієнтованими в довільних напрямках осями, що дозволяє врахувати можливу наявність у досліджуваних даних викидів, тобто результатів спостереження, які виділяються із загальної вибірки.

Для прикладу застосування розробленого методу норму Махаланобіса можна задати за допомогою симетричної додатно визначеної матриці B наступним чином:

$$B = R^{-1},\ R = \frac{1}{N_D} \sum_{k=1,N_D} \left(\overline{W_{\text{e,k}}} - \overline{W}\right)^T \cdot (\overline{W_{\text{e,k}}} - \overline{W}) \ , \qquad (3.21)$$

де R – коваріаційна матриця;

$\overline{W}$ – вектор середніх значень даних, $\overline{W} = \frac{1}{N_D} \sum_{k=1,N_D} \overline{W_{\text{e,k}}}$, де $\overline{W_{\text{e,k}}}$ – k-й вектор ознак;

T – операція транспонування.

В результаті кластеризації було одержано матрицю нечіткого розбиття U (2.38). Для визначення приналежності об’єкта кластеризації до кластера було застосовано порогові значення, за яких доменне ім’я вважається підозрілим або шкідливим відповідно $\lambda` = 0.7$ та $\lambda = 0.8$.

На рис. 3.12 відображено проекцію на площину множини векторів ознак DNS-повідомлень, розподілених на кластери. Початок координат на графіку є центроїдом кластера, який містить множину вхідних DNS-повідомлень щодо легітимних ресурсів до КС мережі. Кожен маркер (крапка), що належить кластеру, представляє множину вхідних DNS-повідомлень до інфікованих та неінфікованих КС мережі щодо певного доменного імені. Кожен маркер свідчить про те, що визначена група КС отримала вхідні DNS-повідомлення щодо доменного імені.

Рис. 3.12 демонструє, що частина результатів кластеризації не визначена. Залучення додаткових ознак, одержаних засобами активного DNS-зондування, призвело до зменшення частки невизначених результатів, що підтверджується експериментально. Дослідження показують, що частка об’єктів кластеризації, які не можуть бути точно класифіковані як шкідливі або легітимні, зменшена на 40-50% (рис. 3.13).

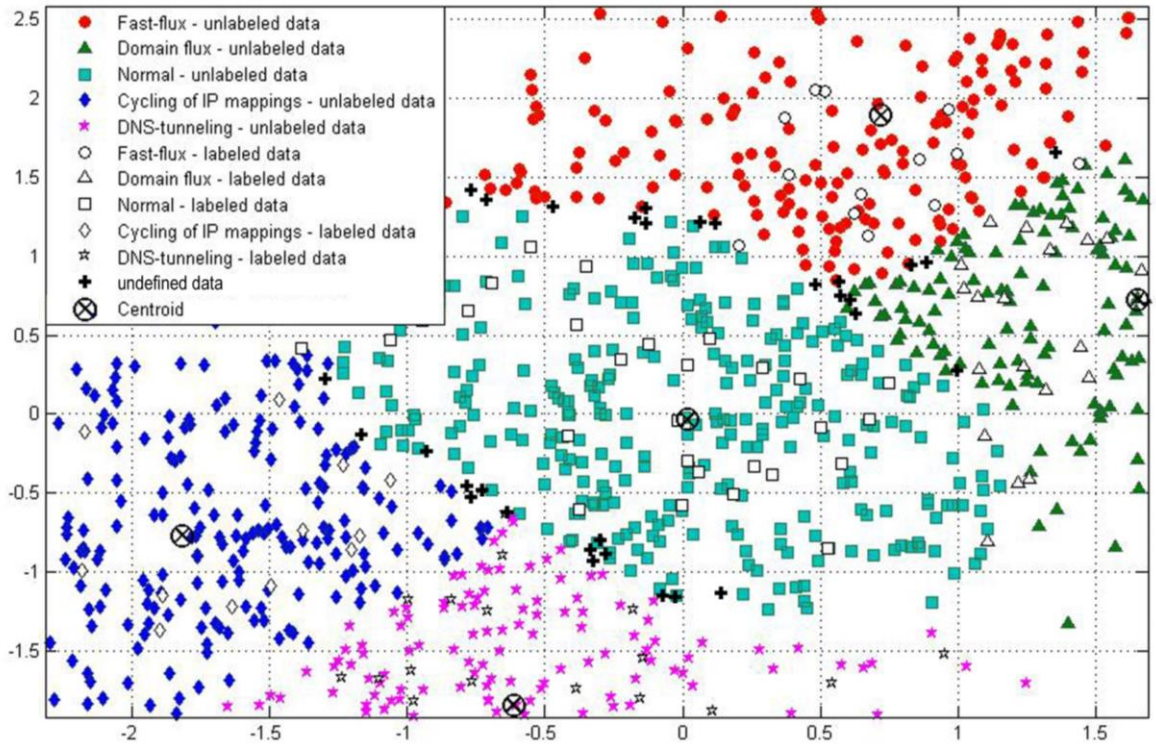


Рисунок 3.12 – Результати кластеризації

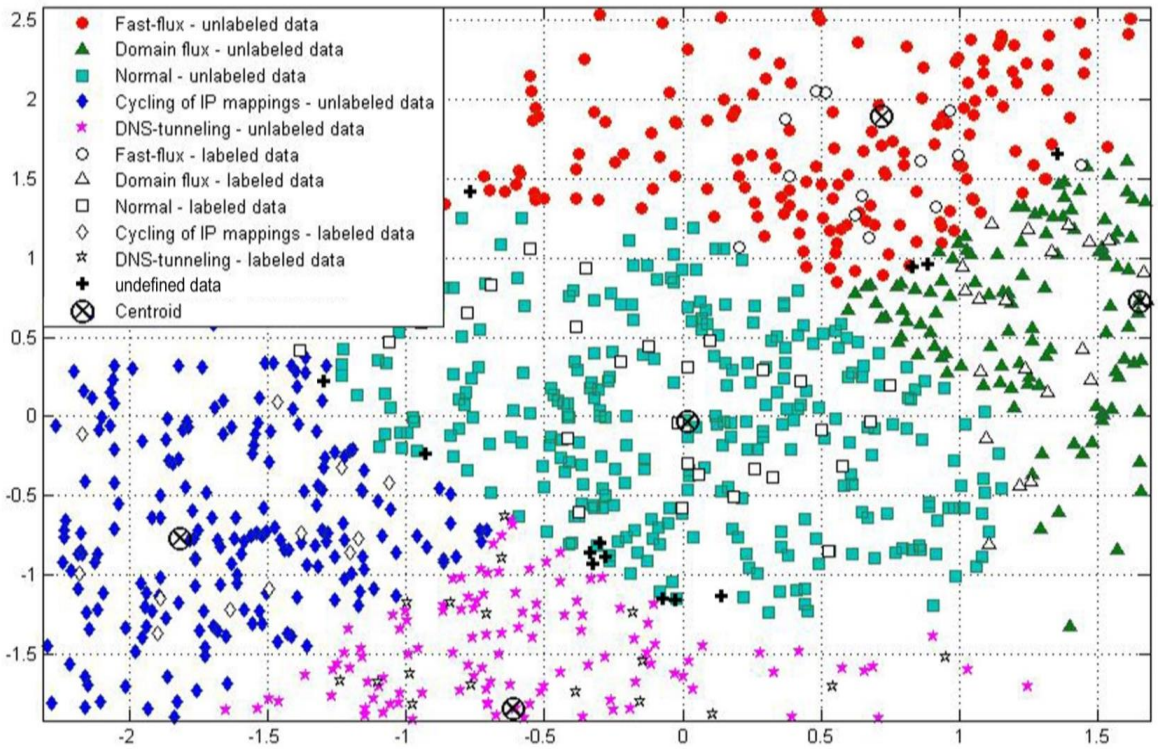


Рисунок 3.13 – Усунення невизначеності частини результатів кластеризації

Локалізація інфікованих ботами КС мережі була здійснена шляхом використання файлів журналювання, в яких зберігаються MAC-адреси КС, що здійснювали DNS-запити, та запитані ними шкідливі доменні імена.

В таблиці 3.1 представлено результати проведених експериментів, а саме: кількість виявлених DNS-відповідей щодо шкідливих доменів на основі лише пасивного моніторингу DNS-трафіка та на основі пасивного моніторингу DNS-трафіка із залученням засобів активного DNS-зондування. Як видно з наведених результатів експерименту, залучення засобів активного DNS-зондування дозволяє зменшити рівень хибних спрацювань. Таким чином, результати застосування розробленого методу демонструють здатність виявлення бот-мереж на рівні до 96%, в той час як рівень хибних спрацювань становить близько 4 %.

Таблиця 3.1

Результати експериментів: кількість проаналізованих DNS-відповідей, одержаних ботами, виявлені DNS-відповіді ботів та хибні спрацювання

Назва технології ухилення	Кількість проаналізованих DNS-відповідей, одержаних ботами	Виявлені DNS-відповіді	
		на основі пасивного моніторингу DNS-трафіка	на основі пасивного моніторингу DNS-трафіка та активного DNS-зондування
		Виявлені DNS-відповіді / Хибні спрацювання, %	
Періодична зміна IP-відображення	308	301 / 2	301 / 1
«Потік доменів»	1432	1406 / 3	1406 / 1
«Швидкозмінні» мережі	485	425 / 3	425 / 2
DNS-тунелювання	144	142 / 0	142 / 0
Всього	2369	2274 (96%) / 8	2274 (96%) / 4

Таким чином, розроблені метод ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку та метод виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, є основою

нової інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка.

3.5 Розрахунок достовірності та ефективності інформаційної технології виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка

З метою визначення загальної ефективності ІТ виявлення бот-мереж на основі аналізу DNS-трафіка необхідно визначити характеристики та показники, які визначають її ефективність. Під ефективністю розробленої ІТ матимемо на увазі наступні властивості антивірусного засобу:

- спроможність надавати своєчасну інформацію щодо інфікованості мережі бот-мережами;
- ефективність витрат часу на здійснення процесу виявлення бот-мереж;
- ефективність ресурсоспоживання;
- достовірність виявлення бот-мереж.

Своєчасність надання інформації щодо інфікованості мережі бот-мережами визначається часом, який необхідний для здійснення збору вхідних даних, передачі зібраних даних по каналах зв`язку та аналізу цих даних.

Тривалість передачі даних по каналах зв`язку та тривалість аналізу даних антивірусним засобом визначаються обчислювальними потужностями вузлів мережі, які виконують функції мережних давачів, пропускною здатністю каналів передачі даних та обчислювальними потужностями комп'ютерної системи, на якій здійснюється аналіз вхідних даних, а також залежать від обсягів зібраних вхідних даних.

Визначимо час, який витрачається на здійснення однієї ітерації процесу виявлення бот-мереж в мережах наступним чином:

$$T_f = T_O + \max(T_{T_i}) + T_P = T_O + \max(\frac{V_{T_i}}{a_T}) + \frac{V_P}{a_P}, i = \overline{1, N_E}, \quad (3.22)$$

де T_O – тривалість ітерації спостереження; T_{T_i} – тривалість передачі даних, зібраних і-м мережним давачем за час ітерації спостереження; T_P – тривалість обробки даних на ітерації аналізу; V_{T_i} – обсяг даних, зібраних і-м мережним давачем за час ітерації спостереження; a_T – середній обсяг даних, який може бути переданий каналом зв'язку за одиницю часу; V_P – обсяг даних, які опрацьовуються на ітерації аналізу; a_P – середній обсяг даних, які можуть бути опрацьовані за одиницю часу; N_E – загальна кількість мережних давачів.

Оскільки $T_O >> \max(T_{T_i})$ та $T_O >> T_P$, то загальна тривалість часу, необхідного для здійснення виявлення бот-мереж, визначається передусім тривалістю та кількістю ітерацій спостереження T_O:

$$T = \sum_{k=1}^{N_I} \left(T_O + \max(T_{T_i})\right)_k + T_{P,l} = \sum_{k=1}^{N_I} \left(T_O + \max(\frac{V_{T_i}}{a_T}) \right)_k + \frac{V_{P,l}}{a_P}, i = \overline{1, N_E}, \quad (3.23)$$

де N_I – загальна кількість ітерацій спостереження та аналізу; $T_{P,l}$ – тривалість обробки даних на останній ітерації аналізу; $V_{P,l}$ – обсяг даних, які обробляються на останній ітерації аналізу.

Визначимо показник ефективності витрат часу на здійснення процесу виявлення бот-мереж в мережах наступним чином:

$$T_E = 1 - \frac{T_{FN} + T_{FP}}{T}, \quad (3.24)$$

де T_{FN} – час, який витрачено на обробку даних, в результаті якої виникли помилки першого роду (хибно-негативні результати); T_{FP} – час, який витрачено на обробку даних, в результаті якої виникли помилки другого роду (хибні спрацювання).

Час, який витрачено на помилки першого та другого роду визначимо наступним чином:

$$T_{FN} = \sum_{k=1}^{N_I} \left(\max\left(\frac{V_{T,FN_i}}{a_T}\right) \right)_k + \frac{V_{P,l,FN}}{a_P}, \tag{3.25}$$

$$T_{FP} = \sum_{k=1}^{N_I} \left(\max\left(\frac{V_{T,FP_i}}{a_T}\right) \right)_k + \frac{V_{P,l,FP}}{a_P}, i = \overline{1, N_E}, \tag{3.26}$$

де V_{T,FN_i}, V_{T,FP_i} – обсяги даних, зібраних і-м мережним давачем за час k-ї ітерації спостереження, в результаті опрацювання яких виникли помилки першого роду та помилки другого роду відповідно; $V_{P,l,FN}$, $V_{P,l,FP}$ – обсяги даних, які було опрацьовано на останній ітерації аналізу, в результаті обробки яких виникли помилки першого та другого роду відповідно.

Визначимо показник ефективності ресурсоспоживання на здійснення процесу виявлення бот-мереж в мережах наступним чином:

$$C_E = 1 - \frac{C_{FN} + C_{FP}}{C}, \tag{3.27}$$

де C – загальне ресурсоспоживання на здійснення процесу виявлення бот-мереж; C_{FN} – ресурсоспоживання, витрачене на помилки першого роду; C_{FP} – ресурсоспоживання, витрачене на помилки другого роду.

Ресурсоспоживання визначимо як сумарну тривалість часу, протягом якого для здійснення процесу виявлення бот-мереж займаються певні обсяги обчислювальних ресурсів (процесорний час, оперативна та віртуальна пам’ять, канали передачі даних):

$$C = \sum_{k=1}^{N_I} \left(\sum_{i=1}^{N_E} \left(T_O + \frac{V_{T_i}}{a_T} \right) + \frac{V_P}{a_P} \right)_k. \tag{3.28}$$

Тоді, ресурсоспоживання, витрачене на помилки першого роду, C_{FN}, та ресурсоспоживання, витрачене на помилки другого роду, C_{FP}, визначимо наступним чином:

$$C_{FN} = \sum_{k=1}^{N_I} \left(\sum_{i=1}^{N_E} \left(\frac{V_{T,FN_i}}{a_T} \right) + \frac{V_{P,FN}}{a_P} \right)_k , \tag{3.29}$$

$$C_{FP} = \sum_{k=1}^{N_I} \left(\sum_{i=1}^{N_E} \left(\frac{V_{T,FP_i}}{a_T} \right) + \frac{V_{P,FP}}{a_P} \right)_k , \tag{3.30}$$

де $V_{P,FN}$, $V_{P,FP}$ – обсяги даних, які було опрацьовано на k-й ітерації аналізу, в результаті обробки яких виникли помилки першого та другого роду відповідно.

Достовірність виявлення бот-мереж визначатимемо за формулою:

$$D_E = \frac{N_N - N_{FP}}{N_M} = \frac{\left| \bigcup_{i=1}^{N_C} \bigcup_{j=1}^{N_{N,i}} H_{N\,i,j} \right| - \left| \bigcup_{i=1}^{N_C} \bigcup_{j=1}^{N_{FP,i}} H_{FP\,i,j} \right|}{N_M}, \tag{3.31}$$

де N_N – кількість КС мережі, визначених антивірусним засобом як інфіковані; N_{FP} – з них кількість помилок другого роду; N_M – кількість інфікованих КС мережі; N_C – загальна кількість типів бот-мереж, виявлених антивірусним засобом; $N_{N,i}$ – кількість КС, визначених як інфіковані і-м типом бот-мереж; $H_{N\,i}$ – підмножини КС, визначені як інфіковані і-м типом бот-мереж; $N_{FP,i}$ – кількість КС, визначених як інфіковані і-м типом бот-мереж в результаті виникнення помилок другого роду; $H_{FP\,i}$ – підмножини КС, які були визначені як інфіковані і-м типом бот-мереж в результаті виникнення помилок другого роду.

Загальну ефективність роботи антивірусного засобу з врахуванням показників ефективності витрат часу, ефективності ресурсоспоживання та достовірності виявлення бот-мереж визначатимемо за формулою:

$$E = T_E C_E D_E . \tag{3.32}$$

З метою визначення загальної ефективності роботи антивірусного засобу визначимо цільові значення параметрів, які впливають на підвищення ефективності, цільове значення $E \to \max$.

Спроможність надавати своєчасну інформацію щодо інфікованості мережі бот-мережами визначається тривалістю часу, необхідного для здійснення збору вхідних даних та їх аналізу. Оскільки $T_O >> \max(T_{T_i})$ та $T_O >> T_P$, то мінімальна тривалість часу, необхідного для здійснення виявлення бот-мереж, визначається передусім тривалістю першої ітерації спостереження та досягається шляхом мінімізації її значення $T_f \to \min$, а в загальному випадку при $N_I > 1$ – мінімізацією значення $T_O \to \min$, а також мінімізацією значень $T_{T_i} \to \min$, $T_P \to \min$. Цільове значення параметра $T \to \min$.

Ефективність використання часу, який витрачається на здійснення виявлення бот-мереж, визначається мінімізацією значень $V_{T,FP_i} \to \min$, $V_{P,l,FP} \to \min$, $V_{T,FN_i} \to \min$, $V_{P,l,FN} \to \min$. Цільове значення показника ефективності використання часу $T_E \to \max$.

Ефективність ресурсоспоживання визначається мінімізацією значень $V_{T,FP_i} \to \min$, $V_{P,FP} \to \min$, $V_{T,FN_i} \to \min$, $V_{P,FN} \to \min$. Цільове значення показника ефективності ресурсоспоживання $C_E \to \max$.

Достовірність виявлення бот-мереж визначається мінімізацією значень $N_{FP} \to \min$, $N_{FN} \to \min$, де N_{FN} – кількість помилок першого роду, $N_{FN} = N_M - (N_D - N_{FP})$. Цільове значення достовірності виявлення $D_E \to \max$.

Отже, дослідження характеристик системи виявлення бот-мереж надало можливість здійснити визначення достовірності та ефективності інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка.

Висновки до розділу 3

Розроблено метод ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку, що надало можливість ідентифікувати як відомі, так

і ще невідомі бот-мережі, а також здійснювати раннє виявлення – на початковій стадії поширення інфекції в мережі.

Розроблено метод виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, що дозволяє виявляти боти відомих та нових бот-мереж, які використовують такі технології ухилення, вже на початковій стадії поширення інфекції в мережі.

Запропоновано методику визначення ефективності та достовірності інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка.

РОЗДІЛ 4
ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ВИЯВЛЕННЯ БОТ-МЕРЕЖ В КОРПОРАТИВНИХ МЕРЕЖАХ НА ОСНОВІ АНАЛІЗУ DNS-ТРАФІКА

4.1 Алгоритми виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка

Інформаційна технологія виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка, заснована на розроблених моделі та методах виявлення бот-мереж на основі аналізу DNS-трафіка, може бути подана у вигляді множини наступних алгоритмів.

Функціонування підсистеми ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку базується на використанні алгоритмів 4.1-4.6. Першочерговими є задача виявлення ігнорування TTL-періодів DNS і задача побудови вектору щільності розподілу DNS-запитів в часі та перевірка їх синхронності за алгоритмами 4.1 та 4.2 відповідно.

Алгоритм 4.1

4.1.1 Поки не завершився TTL-період $TTL_{d^{m'}}$, в межах якого здійснюється пошук, будувати рядок матриці V_{MAC}.

4.1.2 Якщо зафіксовано повторний DNS-запит КС, то додати MAC-адресу КС в наступний рядок матриці V_{MAC}.

4.1.3 Якщо $TTL^{r}_{d^{m'}} < TTL_{d^{m'+1}}$, де $TTL^{r}_{d^{m'}} = TTL_{d^{m'}} - t_{d^{m'+1}}$ – залишок часу до видалення DNS-записів з локальних DNS-кешів на КС, що запитували d, згідно попереднім відгукам, $t_{d^{m'+1}}$ – час надходження наступного DNS-відгуку відносно початку зворотного відліку періоду $TTL_{d^{m'}}$, $TTL_{d^{m'+1}}$ – TTL-період, отриманий в наступному DNS-відгуку, то $TTL_{d^{m'}} := TTL_{d^{m'+1}}$ та виконувати кроки 4.1.1- 4.1.3.

4.1.4 Якщо $\delta \cdot N_G > N_{G_{rep}}$, де N_G та $N_{G_{rep}}$ – розміри груп для попереднього та повторного групових DNS-запитів, то видалити рядок матриці V_{MAC} для

повторного запиту, інакше об’єднати рядки матриці та зафіксувати для групи КС наявність ознаки очищення локальних кешів DNS $F=1$.

4.1.5 Кінець алгоритму

Алгоритм 4.2

4.2.1 Якщо $\Delta t_q \le t_s$, де Δt_q – інтервал часу між першим та останнім DNS-відгуками для групового DNS-запиту щодо доменного імені d, то запити синхронні, інакше виконувати кроки 4.2.2-4.2.3.

4.2.2 Побудувати вектор щільності розподілу DNS-запитів в часі $\overline{W_d}$ згідно формули (2.17).

4.2.3 Здійснити аналіз вектора $\overline{W_d}$ за визначеними правилами (розділ 3, с.80-81).

4.2.4 Кінець алгоритму.

З метою подальшого аналізу вхідного DNS-трафіка за алгоритмом 4.3 будується матриця спостереження M_m та згідно алгоритму 4.4 здійснюється аналіз групових DNS-запитів щодо однакових доменних імен.

Алгоритм 4.3

4.3.1 Якщо DNS-запити групи КС синхронні, то перенести MAC-адреси групи з матриці V_{MAC} в матрицю спостереження M_m.

4.3.2 Занести розмір групи КС в комірку матриці $M_m(d, N_G)$.

4.3.3 Заповнити для групи КС комірку матриці $M_m(d, S)$ за формулою (3.3).

4.3.4 Заповнити для групи КС комірку матриці $M_m(d, F)$ за формулою (3.2).

4.3.5 Заповнити для групи КС комірку матриці $M_m(d, R)$ за формулою (3.4).

4.3.6 Заповнити для групи КС комірки матриці $M_m(d, M)$, $M_m(d, N)$ нулями.

4.3.7 Кінець алгоритму.

Алгоритм 4.4

4.4.1 Виконати оцінку подібності груп КС, які запитували однакові доменні імена d: якщо кількість порівнюваних груп дорівнює двом, то обчислити коефіцієнт Браун-Бланке за формулою (3.5); якщо кількість

порівнюваних груп більша двох, то обчислити індекс дисперсності Коха за формулою (3.6).

4.4.2 Виконати порівняння груп КС з матриці спостереження M_m, що запитували однакові доменні імена, за формулою (2.22). Якщо групи КС визначені інфікованими, то d занести до сірого списку.

4.4.3 Об'єднати групи КС згідно правил (3.7-3.11) та перебудувати матрицю M_m.

4.4.4 Кінець алгоритму.

Для порівняння груп КС, які здійснювали DNS-запити щодо різних доменних імен, за алгоритмом 4.5 будується нижньотрикутна матриця мір Браун-Бланке та здійснюється формування векторів ознак для пар групових DNS-запитів та їх аналіз.

Алгоритм 4.5

4.5.1 Сортувати таблицю M_m за зростанням кількості MAC-адрес КС в групах.

4.5.2 Якщо виконується умова: $N_{Gi} / N_{Gi+1} \geq \delta$`, то обчислювати коефіцієнти Браун-Бланке для пар груп КС та будувати нижньотрикутну матрицю мір Браун-Бланке B_m.

4.5.3 Для кожної пари груп КС, якщо виконується умова $K_B \geq \delta'$, то сформувати вектор ознак $\overline{W_{G_i,G_j}}$ за визначеними правилами (3.12-3.15).

4.5.4 Аналізувати вектори ознак за визначеними правилами (3.16).

4.5.5 Вивести результати.

4.5.6 Кінець алгоритму.

Отже, етапи процесу ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку можуть бути представлені алгоритмом 4.6 у вигляді блок-схеми, поданої на рис. 4.1.

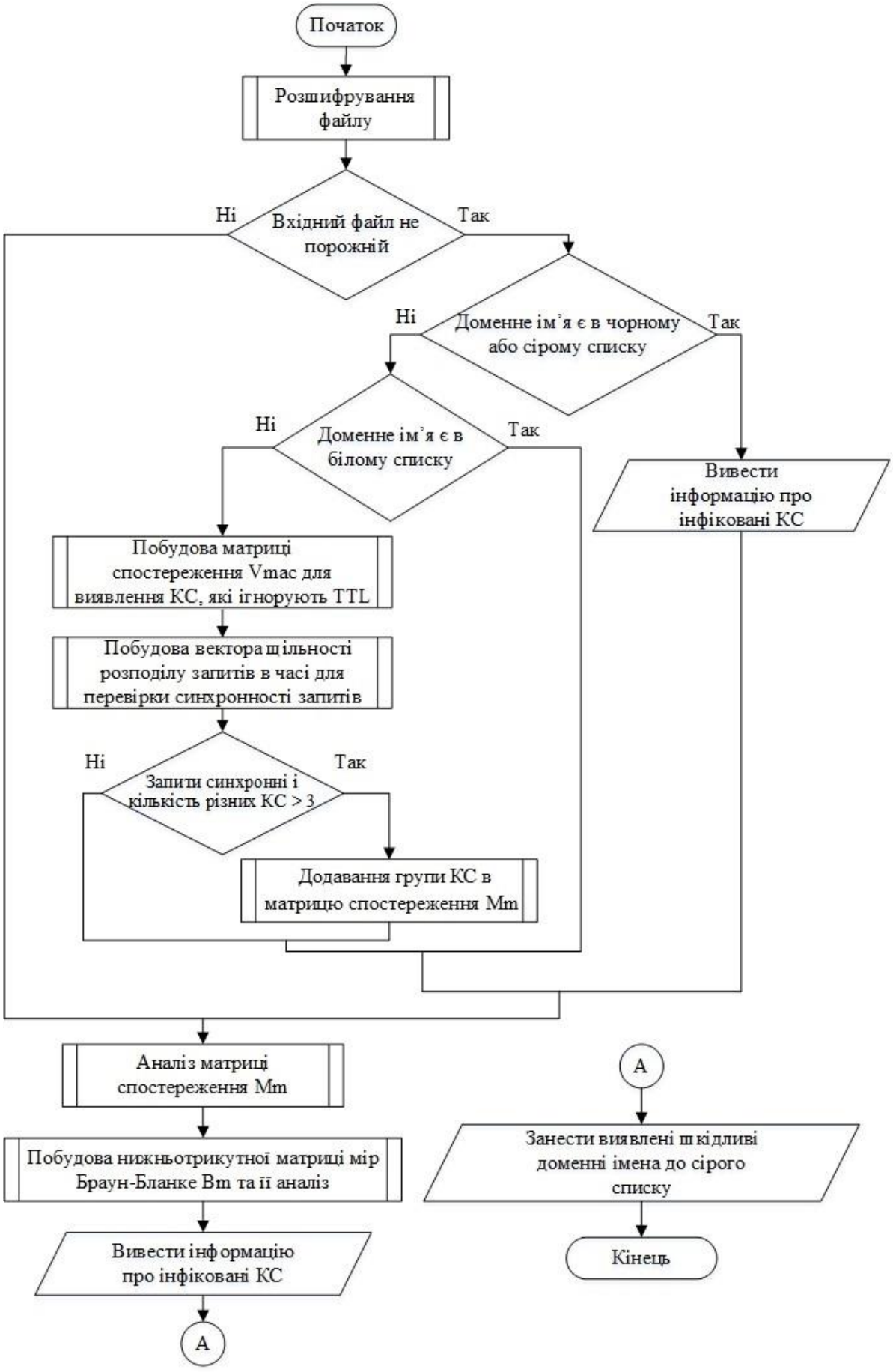


Рисунок 4.1 – Алгоритм функціонування підсистеми ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку

Алгоритм 4.6

4.6.1 Розшифрувати вхідний файл.

4.6.2 Якщо доменне ім’я є в «чорному» або «сірому» списку, то вивести інформацію про інфіковані КС.

4.6.3 Виконувати пункти 4.1.1-4.1.4.

4.6.4 Виконувати пункти 4.2.1-4.2.3.

4.6.5 Виконувати пункти 4.3.1-4.3.6.

4.6.6 Виконувати пункти 4.4.1-4.4.4.

4.6.7 Виконувати пункти 4.5.1-4.5.5.

4.6.8 Кінець алгоритму.

Функціонування підсистеми виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, базується на використанні основних алгоритмів 4.7-4.12. Для побудови вектора ознак, отриманих з вхідних DNS-повідомлень щодо певного доменного імені d, за алгоритмом 4.10 початковою є задача відбору вхідних DNS-повідомлень для подальшого аналізу та формування вибірки TTL-періодів для d за алгоритмом 4.7; визначення середньої дистанції між IP-адресами в множині A-записів для d за алгоритмом 4.8; визначення середньої дистанції між IP-адресами в множинах A-записів, що відповідають доменному імені d, за алгоритмом 4.9. Аналіз результатів кластеризації матриці даних, побудованої на основі векторів ознак вхідних DNS-повідомлень щодо доменних імен, здійснюється за алгоритмом 4.11.

Алгоритм 4.7.

4.7.1 Поки не кінець файлу, виконувати пункти 4.7.2-4.7.5.

4.7.2 Якщо знайдено DNS-відповідь щодо доменного імені d, то виконувати кроки 4.7.3-4.7.6.

4.7.3 Запам’ятати часовий штамп t_1 та TTL-період TTL_1. Занести TTL-період у вибірку S_{TTL}.

4.7.4 Якщо знайдено наступну DNS-відповідь щодо d, то, якщо виконується умова: $|(t_k - t_1) - (TTL_1 - TTL_k)| > 1$, то DNS-відповідь щодо d підлягає

аналізу, де t_k – час отримання наступної DNS-відповіді, TTL_k – TLL-період, отриманий в наступній DNS-відповіді.

4.7.5 Якщо для DNS-відповіді щодо d виконується умова: $(t_k - t_1) > TTL_1$, то перевизначити часовий штамп $t_1 := t_k$, перевизначити TTL-період $TTL_1 := TTL_k$, виконувати крок 4.7.2.

4.7.6 Обчислити моду t_{mod} для S_{TTL}.

4.7.7 Обчислити медіану t_{med} для S_{TTL}.

4.7.8 Обчислити середнє арифметичне значення t_{aver} для S_{TTL}.

4.7.9 Кінець алгоритму.

Алгоритм 4.8.

4.8.1. Якщо кількість записів типу A $n_A > 1$ в DNS-відповіді щодо d, то виконувати кроки 4.8.2-4.8.4.

4.8.2. Здійснити пошук по всім A-записам в DNS-відповіді та занести IP-адреси у вибірку X_1.

4.8.3. Знайти дистанції між парами IP-адрес у вибірці X_1 та занести їх у вибірку X_2.

4.8.4. Відсортувати вибірку X_2 та знайти середню дистанцію s_A.

4.8.5. Кінець алгоритму.

Алгоритм 4.9.

4.9.1. Якщо $n_A > 1$ в DNS-відповіді щодо d, то здійснити пошук всіх DNS-відповідей щодо d та виконувати кроки 4.9.2-4.9.4.

4.9.2 Знайти унікальні IP-адреси в A-записах DNS-повідомлень, підрахувати їх кількість n_{UA} та занести у вибірку X_3.

4.9.3. Знайти дистанції між парами IP-адрес у вибірці X_3 і занести їх у вибірку X_4.

4.9.4. Відсортувати вибірку X_4 та знайти середню дистанцію s_{UA}.

4.9.5. Кінець алгоритму.

Алгоритм 4.10.

4.10.1. Поки файл не порожній, виконувати кроки 4.10.2-4.10.17

4.10.2. Якщо поточне DNS-повідомлення не перше в списку, то перевірити, чи від поточного DNS-повідомлення до першого зустрічалось доменне ім’я d. Якщо зустрічалось, то перейти на крок 4.10.17.

4.10.3. Визначити довжину l_N доменного імені d.

4.10.4. Обчислити кількість унікальних символів n_U в d.

4.10.5. Обчислити ентропію e_N доменного імені d за формулою Шеннона: $H(X) = \sum_{i=1}^{n} p(x_i) \log_b \frac{1}{p(x_i)}$.

4.10.6. Виконувати пункти 4.7.1-4.7.8.

4.10.7. Підрахувати кількість A-записів n_A, що відповідають d, у вихідному DNS-повідомленні.

4.10.8. Якщо $n_A = 1$, то підрахувати кількість IP-адрес, пов’язаних з доменним іменем, n_{IP}, та обчислити середню дистанцію s_{IP} між IP-адресами.

4.10.9. Виконувати пункти 4.8.1-4.8.4.

4.10.10. Виконувати пункти 4.9.1-4.9.4.

4.10.11. Підрахувати кількість доменних імен, які спільно використовують IP-адресу, n_D.

4.10.12. Визначити бінарну ознаку f_{UR} використання в DNS-повідомленні рідковживаних типів записів.

4.10.13. Якщо $f_{UR} = 1$, то обчислити максимальну ентропію вмісту рідковживаних типів записів e_R за формулою Шеннона: $H(X) = \sum_{i=1}^{n} p(x_i) \log_b \frac{1}{p(x_i)}$.

4.10.14. Знайти середній розмір DNS-повідомлення для d, l_P.

4.10.15. Визначити бінарну ознаку успішності DNS-запиту, f_S.

4.10.16. Додати вектор $\overline{W_e}$ (3.17) до матриці даних.

4.10.17. Перейти до кроку 4.10.1.

4.10.18. Кінець алгоритму.

Алгоритм 4.11

4.11.1 Якщо $u_{ij} \geq \lambda$, $j = \overline{1,4}$, то доменне ім'я належить до бот-мереж, які використовують технології ухилення від виявлення на основі DNS, занести d до «сірого» списку.

4.11.2 Інакше, якщо $\lambda \leq u_{ij} < \lambda$, $j = 1$, то здійснити DNS-запити A-записів, визначити ASN, до яких належать IP-адреси, пов'язані з d.

4.11.3 Інакше, якщо $\lambda \leq u_{ij} < \lambda$, $j = 2$, то здійснити DNS-запити PTR-записів.

4.11.4 Інакше, якщо $\lambda \leq u_{ij} < \lambda$, $j = 3$, то здійснити DNS-запити A-записів, NS-записів, SOA-записів, визначити ASN, до яких належать IP-адреси, пов'язані з d та серверами імен для d.

4.11.5 Аналізувати вилучені ознаки за правилами (3.19).

4.11.6 Кінець алгоритму.

Отже, етапи процесу виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, можуть бути представлені алгоритмом 4.12, поданим у вигляді блок-схеми на рис. 4.2-4.4.

Алгоритм 4.12.

4.12.1 Виконувати пункти 4.10.1-4.10.17.

4.12.2 Здійснити кластерний аналіз матриці даних.

4.12.3 Виконувати пункти 4.11.1-4.11.5.

4.12.4. Кінець алгоритму.

Отже, розроблено алгоритми функціонування підсистеми ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку та підсистеми виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, які є основою для програмної реалізації системи виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка.

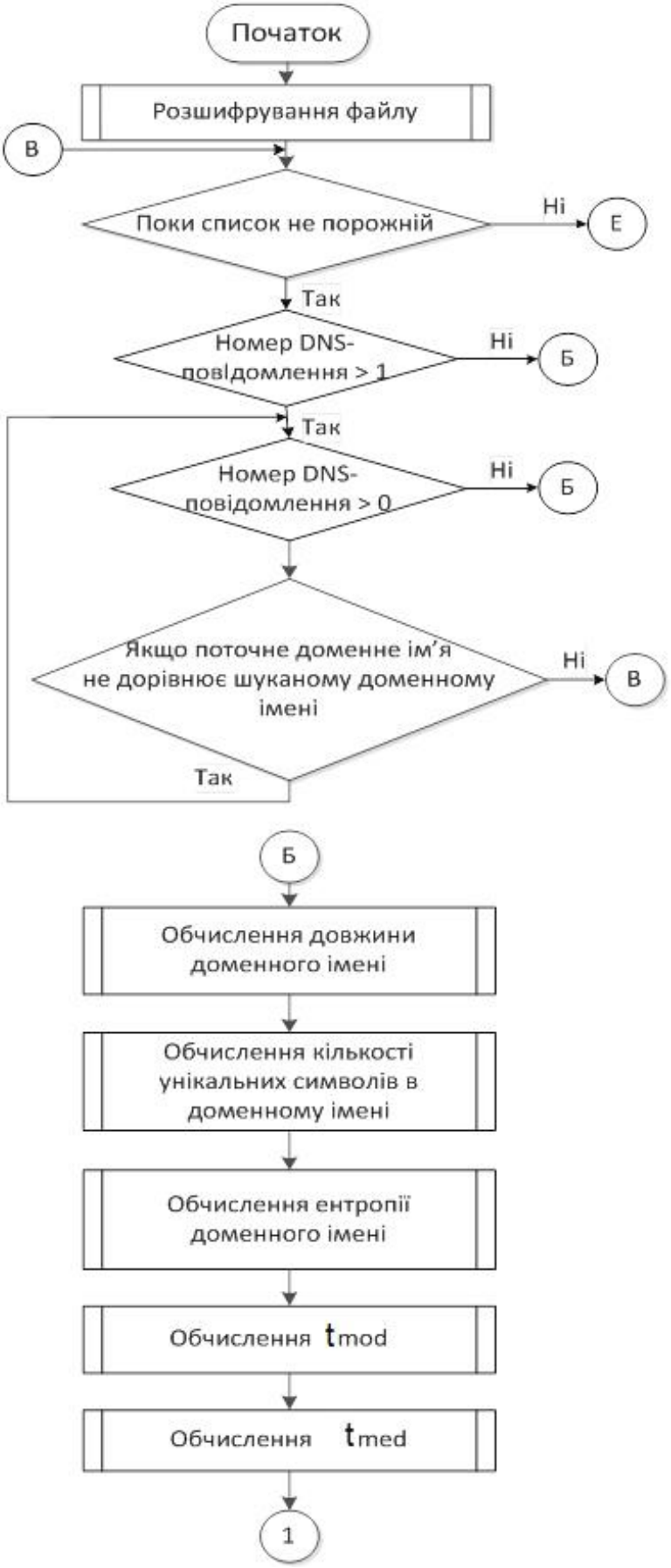


Рисунок 4.2 – Алгоритм функціонування підсистеми виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS

Рисунок 4.3 – Продовження алгоритму функціонування підсистеми виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS

Рисунок 4.4 – Кінець алгоритму функціонування підсистеми виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS

Проведемо дослідження складності алгоритму функціонування підсистеми ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку.

Процес ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку складається з шести основних складових: розшифрування вхідного файлу, побудова матриці спостереження v_{MAC}, побудова вектору щільності розподілу DNS-запитів в часі для перевірки синхронності запитів, побудова

матриці спостереження M_m, аналіз матриці спостереження M_m, побудова нижньотрикутної матриці мір Браун-Бланке B_m та формування і аналіз векторів ознак для пар групових DNS-запитів.

Задача розшифрування файлу з вхідними даними зводиться до читання потрібної інформації з файлу. Складність алгоритму розшифрування вхідного файлу становить:

$$O(N), \tag{4.1}$$

де N – кількість записів у файлі.

Складність алгоритму побудови матриці спостереження v_{MAC} становить:

$$O(N_k) * O(N_m), \tag{4.2}$$

де N_k – кількість DNS-відповідей щодо доменного імені в межах TTL-періоду; N_m – кількість MAC-адрес різних КС в матриці спостереження v_{MAC}.

Складність алгоритму побудови вектору щільності розподілу DNS-запитів в часі для перевірки синхронності запитів становить:

$$O(N_r) * (O(2N_c) + O(2N_z)), \tag{4.3}$$

де N_r – кількість рядків в матриці v_{MAC};

N_c – кількість стовпців в матриці v_{MAC};

N_z – кількість інтервалів, на які розбивається вектор.

Складність алгоритму побудови матриці спостереження M_m становить:

$$O(2N_{cm}) + O(2N_{rm}) + O(N_{cm}) * (O(2N_{rm}) + O(N_m)), \tag{4.4}$$

де N_{rm} – кількість рядків в матриці M_m;

N_{cm} – кількість стовпців в матриці M_m;

N_m – кількість MAC-адрес КС, які потрібно додати в матрицю M_m.

Складність алгоритму аналізу матриці спостереження M_m становить:

$$O(N_{rm})*(O(N_{rm})+O(2N_n))+O(2N_{cm})+O(2N_{rm})+O(2N_{rm}N_{cm}), \quad (4.5)$$

де N_n – кількість рядків, які необхідно об'єднати.

Складність алгоритму побудови матриці B_m становить:

$$O(N_{cm}^2)*O(N_{rm})+O(N_{rm}^2)+O(N_{rm}\log_2 N_{rm}). \quad (4.6)$$

Складність алгоритму формування і аналізу векторів ознак для пар групових DNS-запитів становить:

$$O(N_{rb}\log_2 N_{cb}), \quad (4.7)$$

де N_{rb} – кількість рядків в матриці B_m;

N_{cb} – кількість стовпців в матриці B_m.

Проведемо дослідження складності алгоритму функціонування підсистеми виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS.

Процес виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, складається з наступних основних складових: розшифрування вхідного файлу, формування векторів ознак вхідних DNS-повідомлень щодо доменних імен, здійснення кластеризації c-means, аналіз результатів кластеризації, вилучення додаткових ознак, які вказують на використання технологій ухилення від виявлення бот-мереж на основі DNS, одержаних шляхом застосування активного DNS-зондування, та їх аналіз.

Задача розшифрування файлу зводиться до відбору потрібних даних з файлу з метою формування векторів ознак вхідних DNS-повідомлень щодо доменних імен. Складність алгоритму розшифрування вхідного файлу становить:

$$O(N_k) + O(N_a{}^2), \tag{4.8}$$

де N_k – кількість записів в файлі;

N_a – кількість ресурсних записів в кожній секції відгуків.

Складність алгоритму формування вектору ознак становить:

$$O(3N_d) + O(N \log_2 N_f) + (O(3N_d) + O(3N_d) + O(N \log_2 N_f)), \tag{4.9}$$

де N_d – кількість DNS-відповідей щодо d;

N_f – кількість викликаних функцій, кожна складова вказує на складність виконання певної функції чи циклу в алгоритмі.

Складність алгоритму кластеризації c-means становить:

$$O(N_v KI), \tag{4.10}$$

де N_v – кількість векторів ознак;

K – кількість кластерів;

I – кількість ітерацій.

Складність алгоритму аналізу результатів кластеризації становить:

$$O(\log_2 N_v). \tag{4.11}$$

Складність алгоритму вилучення додаткових ознак, які вказують на використання технологій ухилення від виявлення бот-мереж на основі DNS,

одержаних шляхом застосування активного DNS-зондування, та їх аналізу становить:

$$O(2N_v) + O(N \log_2 N_f). \quad (4.12)$$

Отже, дослідження складності алгоритмів інформаційної технології виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка підтверджує можливість її програмної реалізації.

4.2 Програмні засоби виявлення бот-мереж на основі аналізу DNS-трафіка

4.2.1 Програмне забезпечення виявлення бот-мереж

На базі розроблених алгоритмів виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка було розроблено програмне забезпечення, яке надає можливість підвищити ефективність виявлення бот-мереж та зменшити обсяг необхідних для здійснення процесу виявлення бот-мереж ресурсів.

Розроблене ПЗ надає можливість здійснювати виявлення бот-мереж в мережі виключно на основі спостереження за роботою мережі, що надає переваги прихованості діагностування та можливості підвищення достовірності виявлення, оскільки підхід дозволяє здійснювати аналіз DNS-запитів як окремих комп'ютерних систем, так і поведінки в DNS-трафіку груп КС.

При розробці програмного забезпечення було використано мову програмування C++. В якості середовища розробки було обрано Microsoft Visual Studio. Реалізація функції нечіткої кластеризації c-means з частковим навчанням здійснюється засобами пакету прикладних програм для розв'язання задач технічних обчислень Matlab.

Програмне забезпечення, що реалізує ІТ виявлення бот-мереж, дозволяє виконувати наступні задачі: виявлення відомих та ще невідомих ботів бот-

мереж на основі пасивного моніторингу DNS-трафіка та активного DNS-зондування; локалізація інфікованих ботами комп'ютерних систем мережі.

Адміністратор мережі має можливість здійснити конфігурацію програмного забезпечення, задавши налаштування основних параметрів, які використовуються системою, IP-адреси локальних DNS-серверів мережі, а також включити або відключити функцію здійснення активного DNS-зондування. Адміністратор може вказати шлях до файлу з вхідними даними, завантажити файл, здійснити аналіз вхідних даних та вивести звіт з результатами аналізу, а також редагувати «білий», «чорний» та «сірий» списки доменних імен. Діаграму варіантів використання для проектованої системи виявлення бот-мереж на основі аналізу DNS-трафіка подано на рис. 4.5.

Принцип функціонування системи може бути описаний наступним чином. Вхідний DNS-трафік мережі збирається за допомогою множини мережних давачів, підключених до дзеркалюючих портів керованих комутаторів. В якості мережних давачів застосовуються вузли з встановленою на них утилітою tcpdump або Wireshark. Файли із зібраним вхідним DNS-трафіком надходять в систему, розшифровуються та аналізуються на наявність групової активності ботів в DNS-трафіку та застосування технологій ухилення від виявлення бот-мереж на основі DNS.

З метою виявлення інфікованих ботами КС здійснюється формування груп КС, які ініціюють DNS-запити, з врахуванням можливого ігнорування TTL-періодів, та перевірка синхронності цих DNS-запитів. У випадку, якщо було знайдено групи КС, які ініціювали синхронні DNS-запити, проводиться пошук повторних синхронних DNS-запитів подібних груп КС до того самого або інших доменних імен. В разі, якщо повторні групові DNS-запити виявлено, група вважається інфікованою. У випадках, коли не досягнуто визначеного порогового значення подібності для двох або більше порівнюваних груп КС, які здійснювали синхронні DNS-запити, здійснюється також аналіз поведінкових ознак для групових DNS-запитів, властивих для бот-мереж.

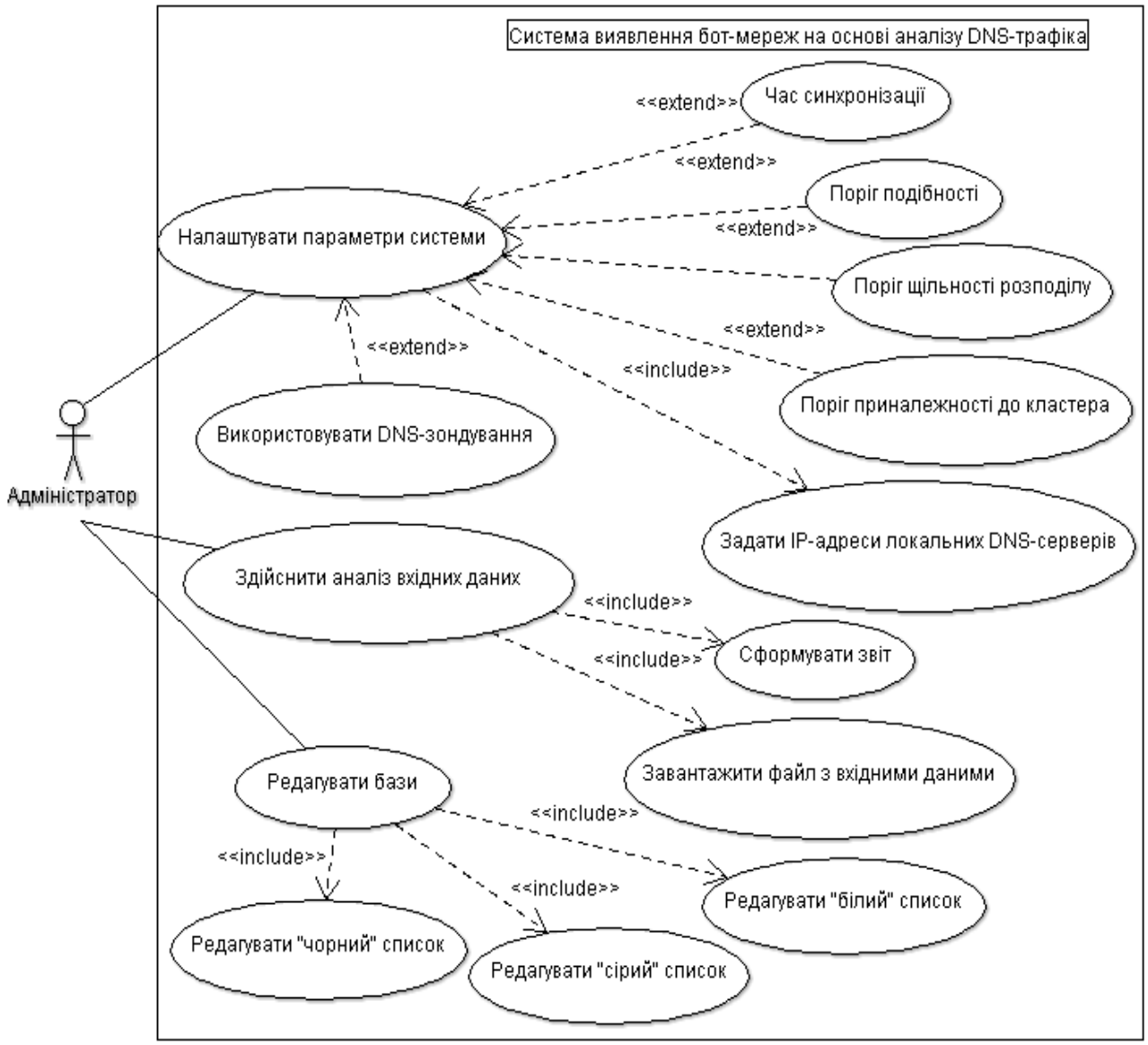


Рисунок 4.5 – Діаграма варіантів використання системи виявлення бот-мереж на основі аналізу DNS-трафіка

Виявлення КС, інфікованих бот-мережами, які застосовують технології ухилення від виявлення на основі DNS, здійснюється на основі кластерного аналізу векторів ознак вхідних DNS-повідомлень щодо запитаних КС мережі доменних імен. У разі перевищення визначеного порогового значення ступеню приналежності вектору ознак до кластерів, що відповідають шкідливим DNS-запитам, комп'ютерні системи, які здійснювали DNS-запити щодо доменного імені, вважаються інфікованими. У разі, якщо ступінь приналежності лежить в межах, які не дозволяють однозначно зробити висновок щодо приналежності вектору ознак до кластера, який відповідає певній технології ухилення від

виявлення бот-мереж на основі DNS, застосовується функція активного DNS-зондування з метою збору та подальшого аналізу додаткових ознак запитаних доменних імен, які можуть вказувати на застосування технологій ухилення бот-мереж на основі DNS.

При виявленні КС, які здійснюють групову активність в DNS-трафіку або ініціюють DNS-запити, які свідчать про використання технологій ухилення від виявлення бот-мереж на основі DNS, за допомогою застосування файлів журналювання здійснюється локалізація інфікованих комп'ютерних систем.

Опис основних модулів, які входять до складу системи виявлення бот-мереж на основі аналізу DNS-трафіка, подано в таблиці 4.1.

Таблиця 4.1

Основні модулі, які входять до складу системи виявлення бот-мереж на основі аналізу DNS-трафіка

Назва файлу	Призначення
clusteringAnalysis.h	Файл заголовків, в якому описані функції, що реалізують аналіз результатів кластеризації
dnsProbe.h	Файл заголовків, в якому описані функції вилучення додаткових ознак шляхом застосування активного DNS-зондування та їх аналіз
generateVectors.cpp	Програма, в якій описано функції, оголошені в файлі generateVectors.h
generateVectors.h	Файл заголовків, в якому оголошено основні функції, призначені для формування векторів ознак вхідних DNS-повідомлень щодо доменних імен
main.cpp	Головна програма, що виконує виклик функцій з вищеописаних файлів

Кінець таблиці 4.1

Основні модулі, які входять до складу системи виявлення бот-мереж на основі аналізу DNS-трафіка

Назва файлу	Призначення
MatLabClustering.dll	Бібліотека, в якій реалізовано функцію здійснення нечіткої кластеризації c-means з частковим навчанням
matrix.cpp	Файл з описом методів, оголошених в файлі заголовків matrix.h.
matrix.h	Файл заголовків з оголошенням методів, необхідних для створення та редагування двовимірних динамічних матриць
parser.h	Файл заголовків з описом структур та функцій, необхідних для розшифрування вхідного файлу
pcap.h	Файл заголовків, який використовується для розшифрування вхідного файлу
searchGroup.cpp	Програма, в якій реалізовано алгоритми методу ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку
searchEvasion.cpp	Програма, в якій реалізовано алгоритми виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS
wpcap.dll	Бібліотека, необхідна для роботи з файлами, створеними за допомогою утиліти tcpdump або Wireshark

Для реалізації системи виявлення бот-мереж на основі аналізу DNS-трафіка також визначено клас matrix та відповідні методи, що слугують для створення та редагування динамічних двовимірних матриць.

4.2.2 Інтерфейс та результати роботи програмного забезпечення

Головне вікно розробленого програмного забезпечення виявлення бот-мереж на основі аналізу DNS-трафіка містить чотири вкладки: Settings (налаштування), Lists Management (керування списками), Analysis (Аналіз), Info (довідка). Інтерфейс розробленого ПЗ подано на рис. 4.6-4.8.

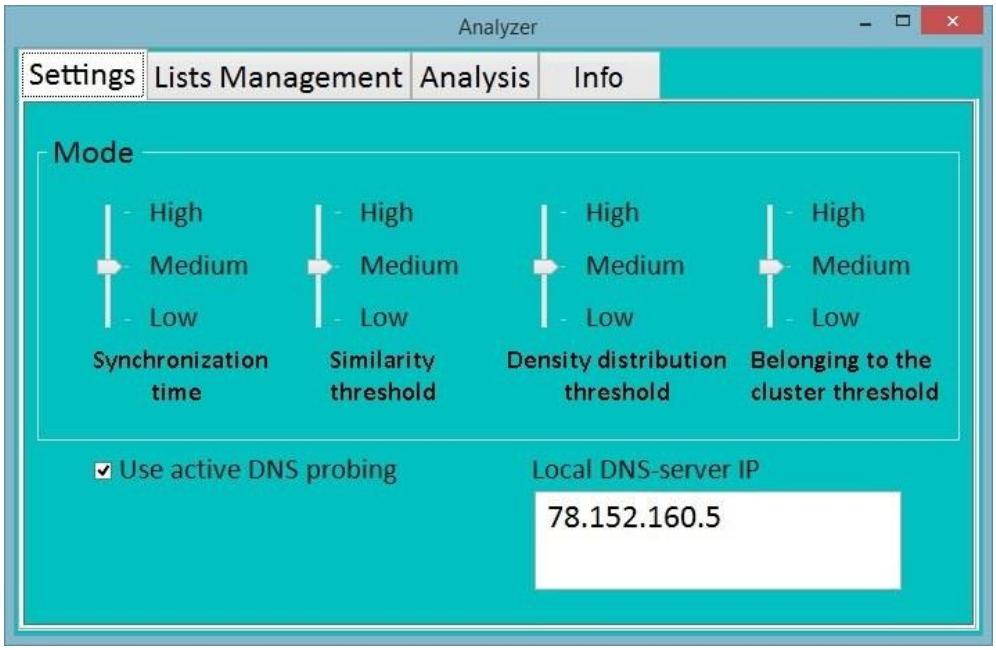


Рисунок 4.6 – Вкладка Settings

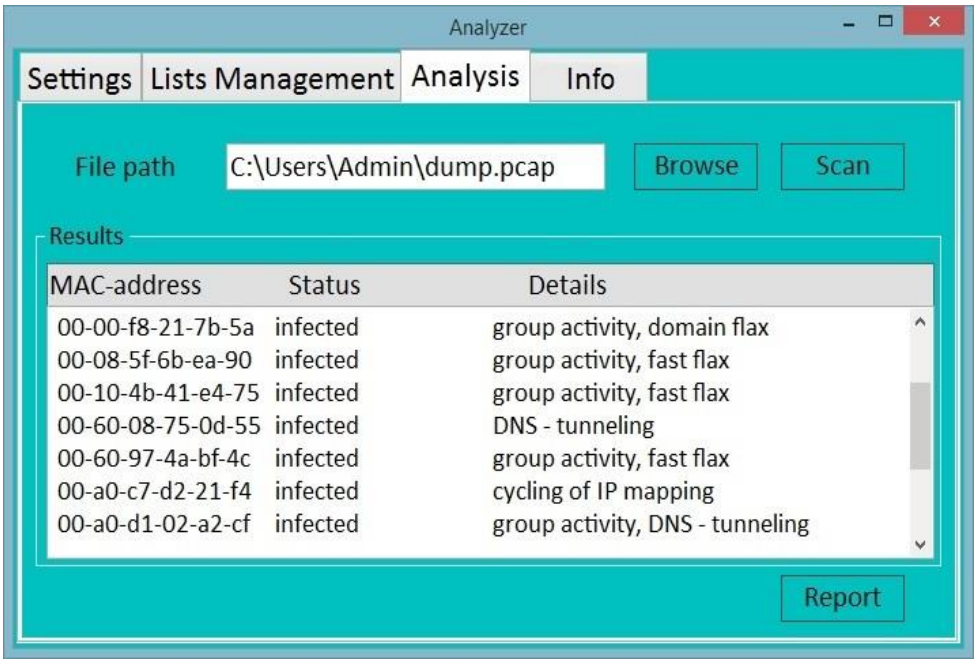


Рисунок 4.7 – Вкладка Analysis

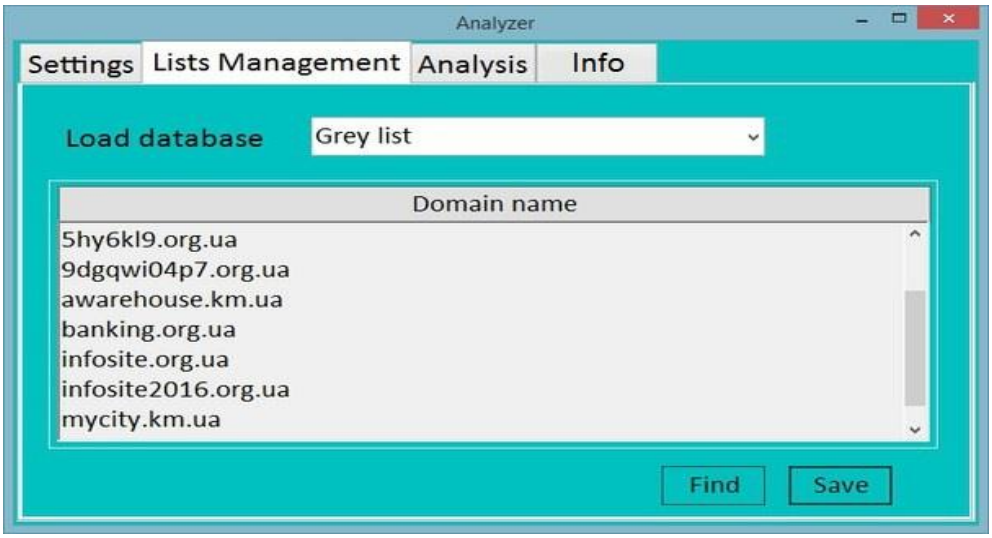


Рисунок 4.8 – Вкладка Lists Management

Вкладка Settings надає можливість адміністратору мережі здійснити налаштування системи. Блок Mode дозволяє встановити рівні основних параметрів, які використовуються системою. Кожен параметр має три рівні: Low (мінімальний), Medium (оптимальний), High (посилений). Блок Local DNS-server IP надає можливість вказати IP-адреси локальних DNS-серверів мережі. Також, передбачена можливість увімкнення або вимкнення функції здійснення активного DNS-зондування, яка застосовується з метою уточнення результатів діагностування.

Вкладка Analysis надає можливість завантажити файл з вхідними даними з метою проведення його аналізу. Блок Result відображає результати роботи ПЗ, а саме: MAC-адреси інфікованих та підозрілих КС мережі, а також додаткову інформацію, яка містить підстави, на яких прийнято відповідне рішення щодо інфікованості або підозрілості КС (виявлення групової активності та/або виявлення застосування технологій ухилення бот-мереж на основі DNS). Також, забезпечена можливість формування, перегляду та друку звіту з результатами діагностування мережі. Вкладка Info дозволяє отримати довідкову інформацію стосовно розробленого ПЗ.

Вкладка Lists Management надає можливість завантаження та редагування бази даних, що містить «білий» список відомих легітимних доменних імен, «чорний» список відомих шкідливих доменних імен, «сірий» список доменних імен бот-мереж, виявлених із застосуванням розробленого ПЗ.

Розроблене програмне забезпечення виявлення бот-мереж на основі аналізу DNS-трафіка надало можливість провести дослідження з метою визначення достовірності інформаційної технології виявлення бот-мереж в корпоративних мережах.

4.3 Інформаційна технологія виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка

З метою усунення недоліків відомих ІТ та підвищення достовірності виявлення бот-мереж в корпоративних мережах було розроблено інформаційну технологію виявлення бот-мереж на основі аналізу DNS-трафіка.

Інформаційна технологія побудована на базі моделей: бот-мереж, DNS-трафіка та процесу виявлення бот-мереж в мережах на основі аналізу DNS-трафіка з врахуванням використання ними DNS в процесі функціонування, особливостей поведінки ботів в DNS-трафіку, а також використання бот-мережами технологій ухилення від виявлення на основі DNS. ІТ надає можливість здійснювати виявлення КС в корпоративній мережі, інфікованих як відомими, так і невідомими бот-мережами.

ІТ виявлення бот-мереж на основі аналізу DNS-трафіка може бути представлена укрупненою схемою, поданою на рис. 4.9.

Інформаційна технологія включає два методи виявлення бот-мереж: метод ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку та метод виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS. ІТ використовує базу даних, що містить: «білий» список популярних легітимних доменних імен, «чорний» список відомих доменних імен бот-мереж, «сірий» список доменних імен бот-мереж, виявлених розробленою ІТ, ґрунтується на знаннях щодо ознак, які можуть бути отримані з вхідних DNS-повідомлень до ботів бот-мереж, які використовують технології ухилення від виявлення на основі DNS, та ознак доменних імен таких бот-мереж, а також використовує множини правил для формування та аналізу векторів ознак групових DNS-запитів.

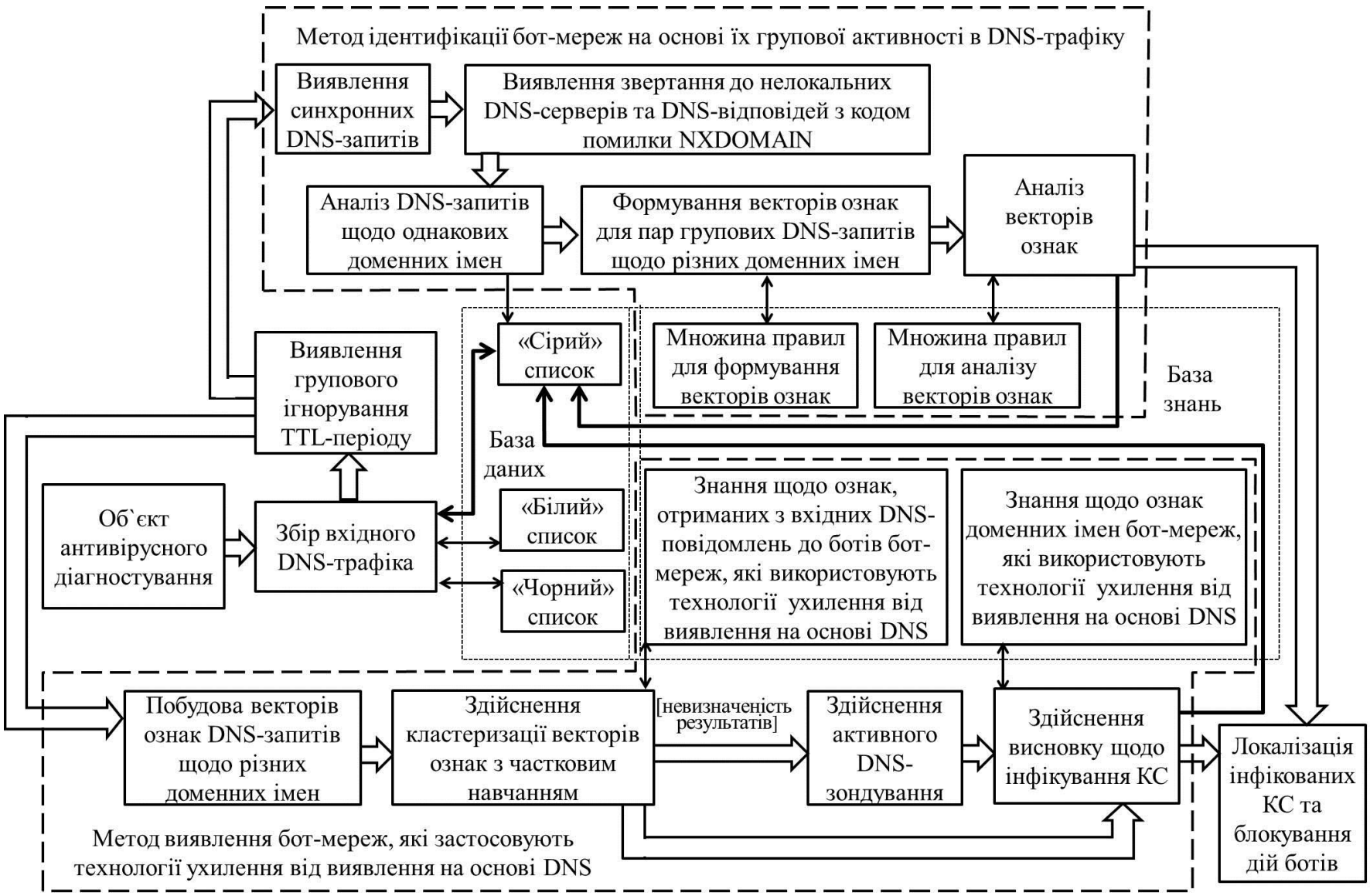


Рисунок 4.9 – Укрупнена схема функціонування ІТ виявлення бот-мереж на основі аналізу DNS-трафіка

Деталізована схема функціонування ІТ подана на рис. 4.10.

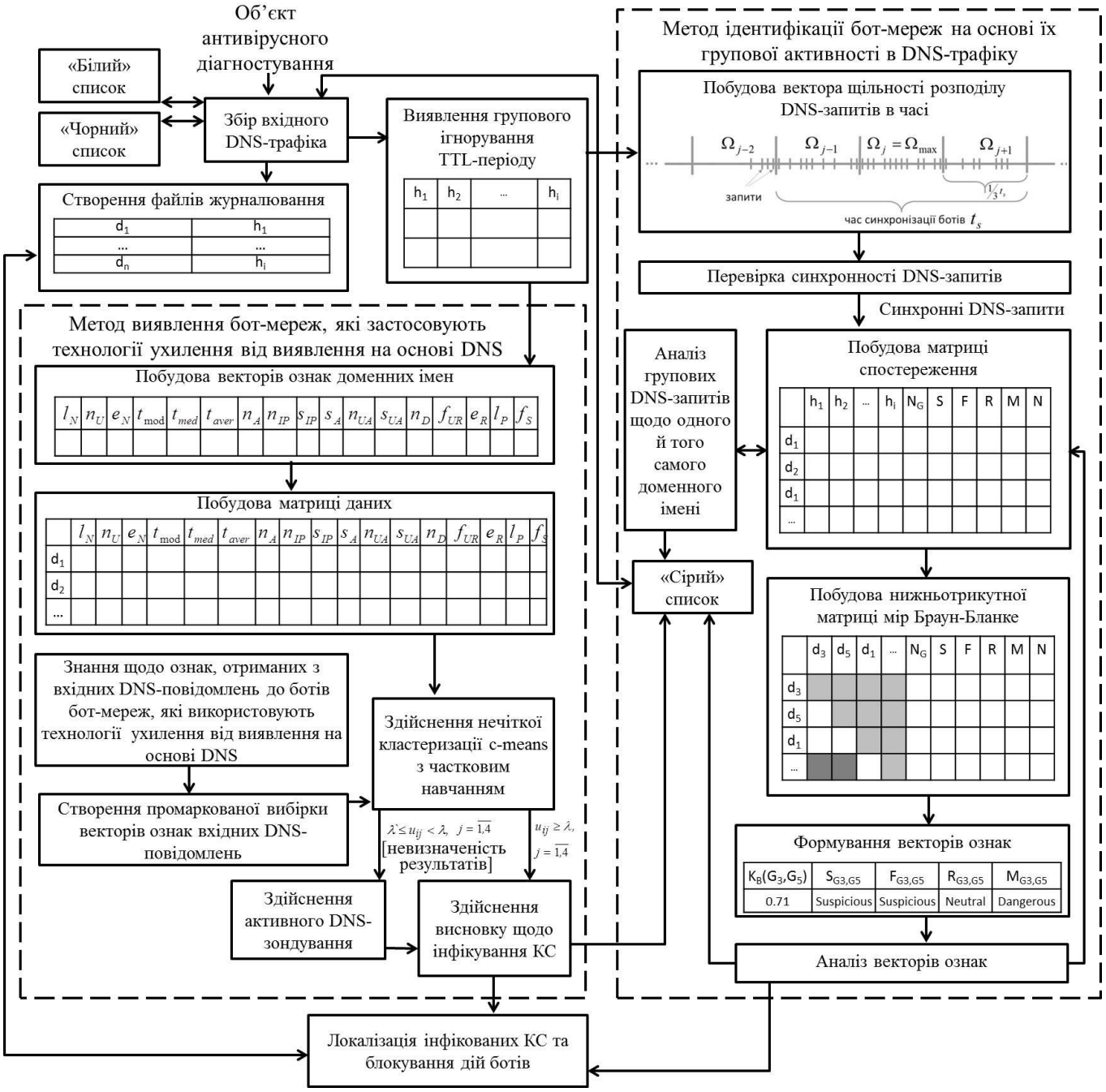


Рисунок 4.10 – Деталізована схема функціонування ІТ виявлення бот-мереж на основі аналізу DNS-трафіка

Виявлення бот-мереж в корпоративних мережах здійснюється на основі аналізу вхідного DNS-трафіка.

Для виявлення групової активності ботів в DNS-трафіку здійснюється поділ КС мережі на групи, які, імовірно, здійснюють шкідливу активність, і

порівняння подібності груп КС за MAC-адресами та аналіз наявності у груп КС особливостей поведінки в DNS-трафіку, притаманних для бот-мереж.

З метою виявлення використання технологій ухилення бот-мереж на основі DNS застосовується кластерний аналіз ознак, одержаних шляхом здійснення пасивного моніторингу вхідного DNS-трафіка. На основі знань щодо ознак, які можуть бути вилучені з вхідних DNS-повідомлень до ботів бот-мереж, які використовують технології ухилення на основі DNS, здійснюється часткове навчання кластеризатора. Знання можуть бути представлені у вигляді правил, описаних алгоритмом, поданим на рис. 4.11.

for $all\ DNS\_messages\_or\_training\_data$ **do**
  if $(t_{mod} \in [0, 900]\ and\ t_{med} \in [0, 900]\ and\ t_{aver} \in [0, 900])$ **then**
    if $((n_A \in (5, \infty)\ and\ s_A \in (65535, \infty))\ or\ (n_{UA} \in (8, \infty)\ and\ s_{UA} \in (65535, \infty))$ **then**
      $evasion\_technique \leftarrow fast\_flux$
    end
    if $(f_S = 0\ and\ n_D \in [8, \infty])$ **then**
      $evasion\_technique \leftarrow domain\_flux$
    end
    if $(n_{IP} \in (5, \infty)\ and\ s_{IP} \in (65535, \infty))$ **then**
      $evasion\_technique \leftarrow cycling\_of\_IP\_mappings$
    end
    if $((l_N \in [75, 255]\ and\ n_U \in (27, 37])\ or\ (e_N \geq f_{Eb32}\ or\ (e_R \geq f_{Eb64}\ or\ e_R \geq f_{Eb256})\ or\ f_{UR} = 1))\ and\ l_P > 300)$ **then**
      $evasion\_technique \leftarrow DNS\_tunneling$
    end
  end
end

Рисунок 4.11 – Правила, на основі яких здійснюється часткове навчання кластеризатора

В разі необхідності усунення невизначеності частини результатів кластерного аналізу застосовується активне DNS-зондування, тобто здійснення DNS-запитів відповідних типів, які дозволять одержати додаткові ознаки доменних імен, на підставі яких може бути прийнято рішення щодо шкідливості доменного імені. Алгоритм виявлення застосування технологій ухилення від виявлення бот-мереж на основі DNS на базі пасивного моніторингу DNS-трафіка та активного DNS-зондування подано на рис. 4.12.

Function passive_analysis
for *all gathered_incoming_DNS_messages* **do**
  for *all first_DNS_messages_within_TTL or all repeated_DNS_messages_within_TTL_from_non_local_DNS_server* **do**
    form $feature\_vector\_\overline{W_e}$
  end
  form $data\_matrix\_of\_feature\_vectors\_V$
  $data\_matrix\_of\_feature\_vectors\_V \rightarrow set\_of\_clusters\_X$ where $V(i,) = \overline{W_e}$
  form $fuzzy\_splitting\_matrix\_U$
  if $(u_{ij} \geq \lambda)$ **then**
    block
    else
      if $(\lambda' \leq u_{ij} < \lambda)$ **then**
        execute *active_analysis*
      end
  end
end
Function active_analysis
for *all* $\overline{W_e}$ **do**
  if $(\lambda' \leq u_{ij} < \lambda)$ **then**
    if $((\overline{W_e} \in X_{cycling\_of\_IP\_mapping})$ *and* $(\overline{W_e} \in X_{legitimate}))$ **then**
      if $((t_{mod} \in [0, 900]$ *and* $t_{med} \in [0, 900]$ *and* $t_{aver} \in [0, 900])$ *and* $(n_{IP} \in (5, \infty)$ *and* $s_{IP} \in (65535, \infty)$ *and* $n_{ASA} > 2))$ **then**
        block
      end
    end
    if $((\overline{W_e} \in X_{fast\_flux})$ *and* $(\overline{W_e} \in X_{legitimate}))$ **then**
      if $((t_{mod} \in [0, 900]$ *and* $t_{med} \in [0, 900]$ *and* $t_{aver} \in [0, 900])$ *and* $((n_A \in (5, \infty)$ *and* $s_A \in (65535, \infty))$ *or* $(n_{UA} \in (8, \infty)$ *and* $(s_{UA} \in (65535, \infty))$ *or* $n_{AS} > 2)$ *and* $(s_{NS} \in (65535, \infty)$ *or* $n_{ASN} > 2$ *and* $n_{NS} > 3$ *and* $v_{retry} \in [0, 900])))$ **then**
        block
      end
    end
    if $((\overline{W_e} \in X_{domain\_flux})$ *and* $(\overline{W_e} \in X_{legitimate}))$ **then**
      if $(n_D \in [8, \infty])$ **then**
        block
      end
    end
    else
      block
  end
end

Рисунок 4.12 – Алгоритм виявлення застосування технологій ухилення від виявлення бот-мереж на основі DNS на базі пасивного моніторингу DNS-трафіка та активного DNS-зондування

Моделі та методи, на яких базується ІТ виявлення бот-мереж на основі аналізу DNS-трафіка, а також її програмні засоби дозволяють виконувати наступні задачі: ідентифікація бот-мереж на основі їх групової активності в DNS-трафіку; виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, на базі пасивного моніторингу DNS-трафіка та активного DNS-зондування; локалізація інфікованих КС мережі. По аналогії з [137, 138] розроблена інформаційна технологія може бути подана у вигляді схеми, представленої на рис 4.13.

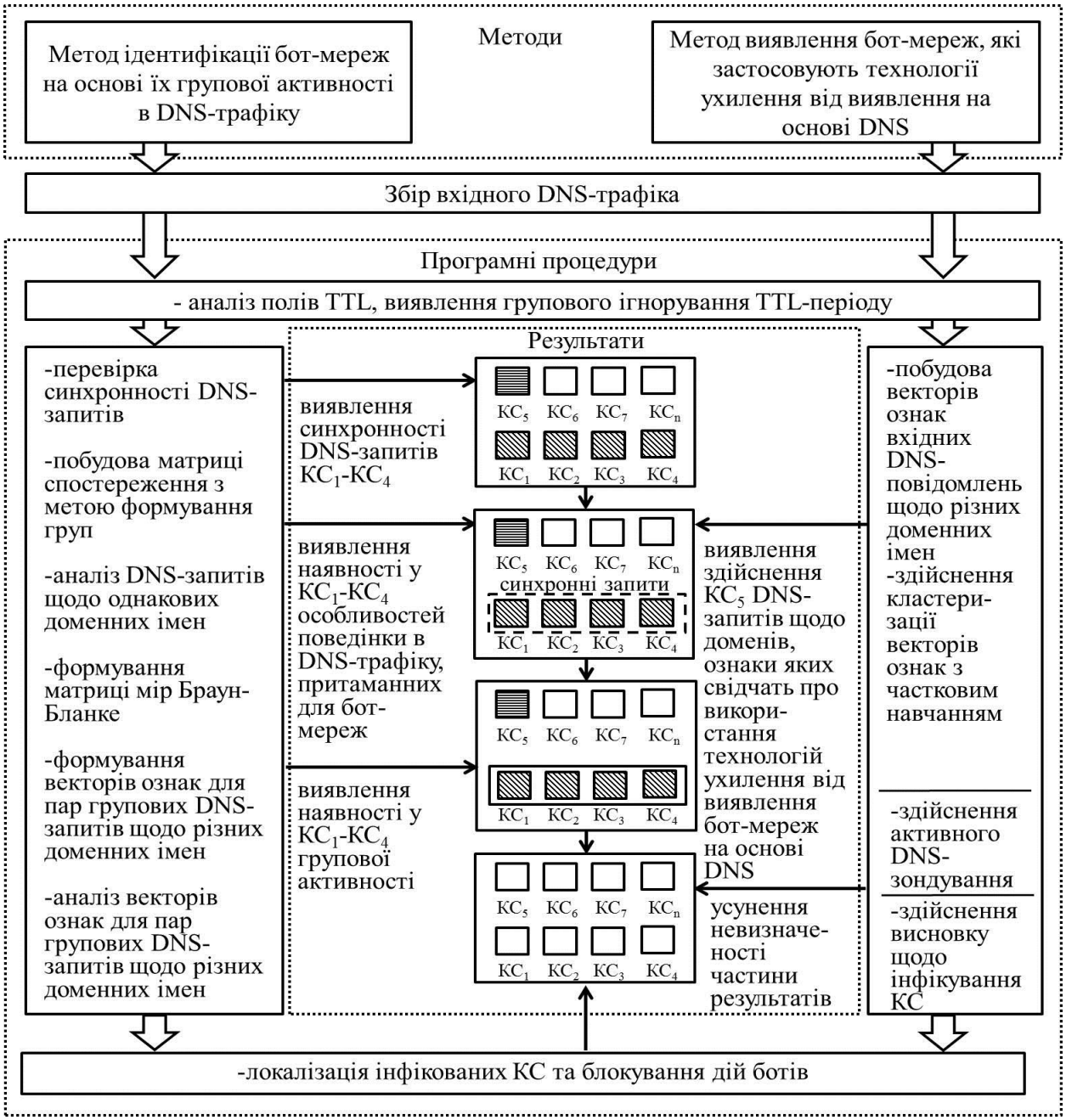


Рисунок 4.13 – Схема застосування ІТ виявлення бот-мереж на основі аналізу DNS-трафіка

На рис. 4.13 КС1-КС4 позначено комп'ютерні системи мережі, інфіковані ботами бот-мережі, що здійснюють групову активність в DNS-трафіку, КС5 – комп'ютерна система, інфікована ботом бот-мережі, що застосовує технології ухилення від виявлення на основі DNS, КС6-КСn – неінфіковані комп'ютерні системи.

З метою розрахунку достовірності виявлення бот-мереж шляхом застосування розробленої інформаційної технології в порівнянні з відомими антивірусними засобами було проведено ряд експериментів (в загальній кількості 90, кожен тривалістю 24 години).

Для проведення порівняльного аналізу було обрано наступні відомі антивірусні засоби: Panda Endpoint Protection, ESET Endpoint Security, Kaspersky Endpoint Security, Avira Small Business Security Suite, Microsoft System Center Endpoint Protection, Avast Endpoint Protection Suite, McAfee Endpoint Protection Suite, Dr.Web CureNet!.

Для проведення експериментів було створено спеціальне програмне забезпечення, яке дозволяло здійснювати DNS-запити і мало властивості ботів бот-мереж з функціоналом, невідомим для антивірусних засобів. Множина створеного програмного забезпечення була пропорційно розподілена на групи за їх функціональними властивостями, кожна з яких відповідала одній з чотирьох технологій ухилення бот-мереж – «потік доменів», «швидкозмінні» мережі, DNS-тунелювання, періодична зміна IP-відображення, які, в залежності від їх функціональності, здійснювали відповідні типи DNS-запитів.

З метою імітації C&C-серверів бот-мереж на період проведення експериментів було зареєстровано множину доменних імен, які розглядались як шкідливі. C&C-сервери мали можливість імітувати застосування технологій ухилення на основі DNS (здійснювали такі дії, як періодична зміна IP-відображення доменних імен, зміна доменних імен, циклічна зміна DNS A-записів та NS-записів для доменного імені за алгоритмом round robin, передача трафіка командування та контролю за допомогою технології DNS-тунелювання тощо).

Створені боти мали різні множини властивостей та виконували DNS-запити щодо шкідливих доменних імен, які не були попередньо відомі та не використовувались для навчання. Також, було імітовано активність користувачів, для чого КС здійснювали DNS-запити щодо легітимних ресурсів. Кожна бот-мережа відтворювала різні сценарії здійснення DNS-запитів в різний час. Певна частина ботів здійснювали групову активність, тобто їх DNS-запити були синхронними. DNS-трафік локальної мережі збирався засобами утиліти Wireshark.

Таким чином, для оцінки достовірності діагностування кожного антивірусного засобу мережа з 100 КС інфікувалась 60 створеними зразками вірусних програм з функційним навантаженням ботів.

Усереднені результати виявлення бот-мереж із застосуванням розробленого програмного забезпечення, обчислені на основі результатів множини проведених експериментів, наведено в табл. 4.2.

Таблиця 4.2

Усереднені результати виявлення бот-мереж із застосуванням розробленого програмного забезпечення

Назва технології ухилення	Кількість проаналізованих DNS-відповідей ботів	Виявлені DNS-відповіді / з них групові	Хибні спрацювання, %
Періодична зміна IP-відображення	481 / 321	476 / 321	1
«Потік доменів»	1796 / 1520	1753 / 1509	1
«Швидкозмінні» мережі	615 / 429	558 / 421	2
DNS-тунелювання	183 / 47	180 / 44	0
Всього	3075 / 2317	2967 / 2295 (96% / 99%)	4

Результати оцінки достовірності виявлення бот-мереж програмним забезпеченням розробленої інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка в порівнянні з відомими антивірусними засобами представлено в табл. 4.3.

Таблиця 4.3

Результати експериментальних досліджень: оцінка достовірності виявлення бот-мереж розробленою інформаційною технологією в порівнянні з відомими антивірусними засобами

№ зп	Засіб антивірусного діагностування	Середня достовірність виявлення, %	Середнє значення помилки 1-го роду, %	Середнє значення помилки 2-го роду, %	Середня тривалість часу, затраченого на виявлення, хв.
1.	Розроблений АЗ	96,22	3,78	3,44	30
2.	Avast Endpoint Protection	84,80	15,20	11,66	25
3.	Avira Small Business Security Suite	86,38	13,62	9,52	42
4.	Dr.Web CureNet!	86,18	13,82	10,28	38
5.	ESET Endpoint Security	86,92	13,08	7,48	24
6.	Kaspersky Endpoint Security	88,86	11,14	8,30	31
7.	McAfee Endpoint Protection Suite	85,92	14,08	9,24	33
8.	Microsoft System Center Endpoint Protection	78,58	21,42	3,94	21
9.	Panda Endpoint Protection	83,56	16,44	5,84	29

Результати проведених експериментальних досліджень показують, що рівень достовірності виявлення бот-мереж при застосуванні розробленого антивірусного засобу складає близько 96%, що на 8-22% вище в порівнянні з відомими антивірусними програмними засобами (рис.4.14, 4.15).

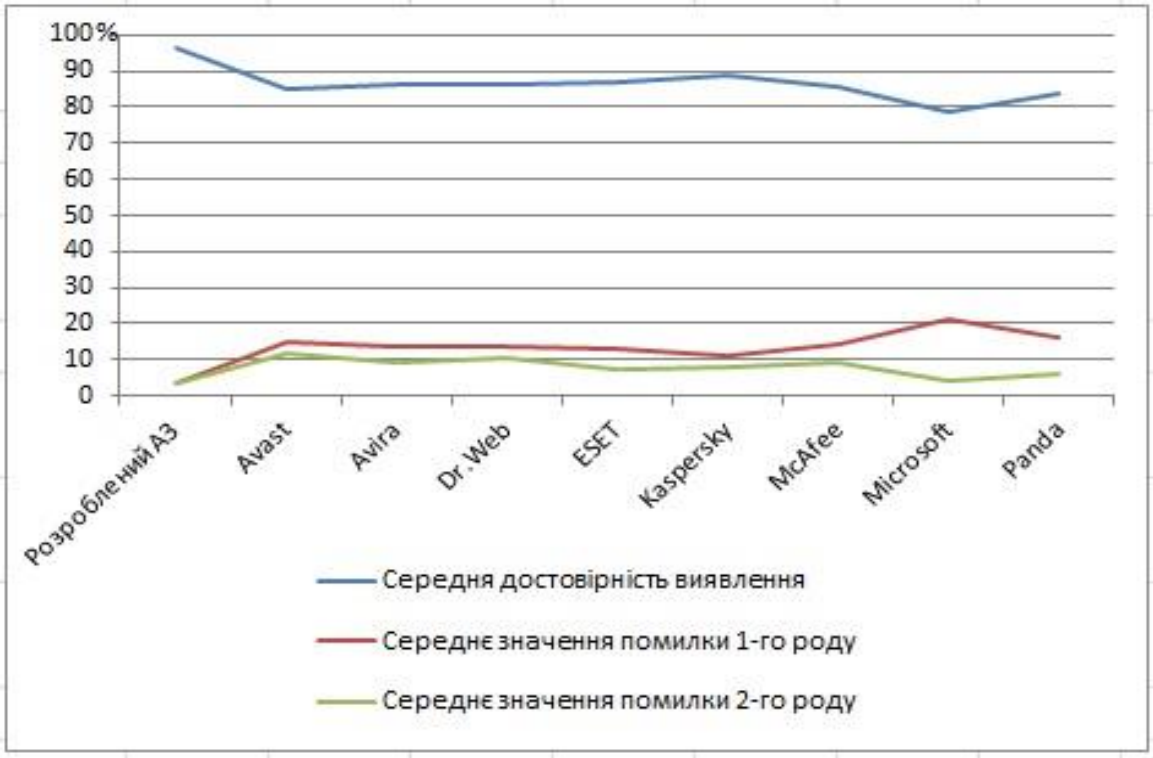


Рисунок 4.14 – Порівняльний аналіз розробленої інформаційної технології з відомими

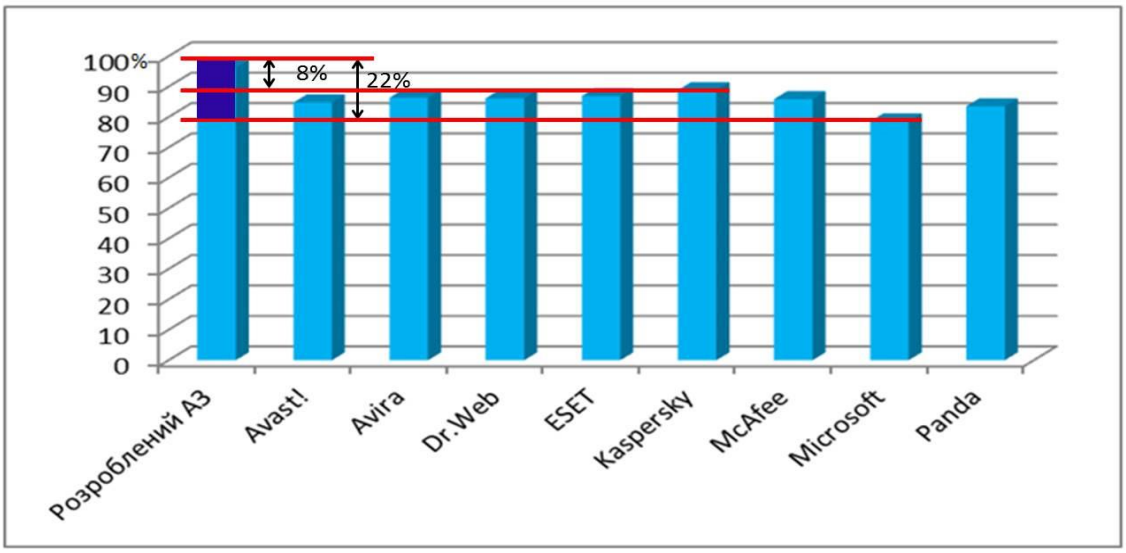


Рисунок 4.15 – Порівняльний аналіз розробленої інформаційної технології з відомими (достовірність виявлення, %)

Застосування розробленого антивірусного засобу дозволяє досягти зниження рівня помилок другого роду до 4%, що на 13-70% нижче в порівнянні з відомими антивірусними програмними засобами (рис. 4,14, рис. 4.16).

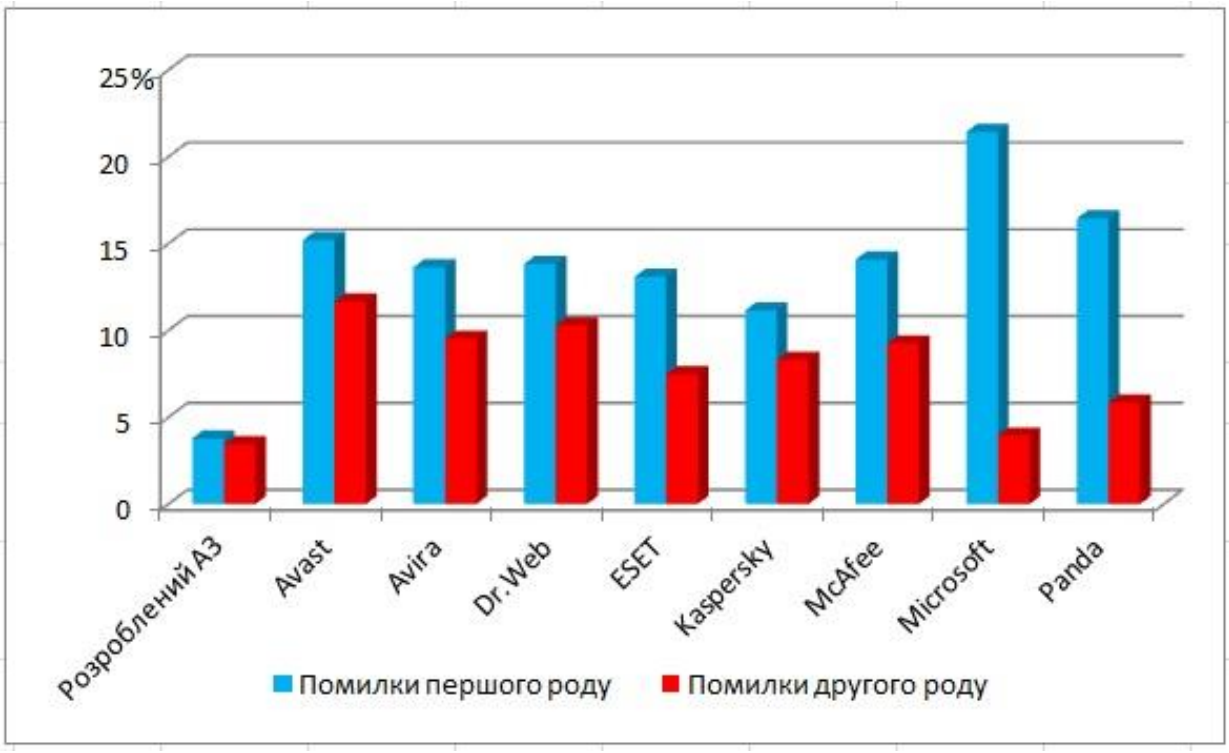


Рисунок 4.16 – Порівняльний аналіз розробленої інформаційної технології з відомими: помилки першого роду (хибно-негативні результати), %, та помилки другого роду (хибні спрацювання), %

Дослідження характеристик розробленої інформаційної технології надало можливість визначити загальну ефективність виявлення бот-мереж в мережах. В ході проведених експериментів було отримано наступні показники:

- показник ефективності витрат часу $T_E = 0{,}997$;
- показник ефективності ресурсоспоживання $C_E = 0{,}954$;
- достовірність виявлення $D_E = 0{,}962$.

Таким чином, загальна ефективність роботи розробленого антивірусного засобу складає $E \approx 0{,}997 \times 0{,}954 \times 0{,}962 \approx 0{,}91$.

Отже, результати дослідження достовірності розробленої інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка показують, що використання розробленого програмного забезпечення дозволяє підвищити рівень достовірності виявлення бот-мереж на 8-22% в порівнянні з відомими антивірусними програмними засобами.

Висновки до розділу 4

1. Розроблено алгоритми виявлення бот-мереж та програмне забезпечення, яке реалізує інформаційну технологію виявлення бот-мереж на основі аналізу DNS-трафіка.

2. Розроблено інформаційну технологію виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка, яка дозволяє ідентифікувати боти, що здійснюють групову активність в DNS-трафіку, а також виявляти боти бот-мереж, які застосовують технології ухилення від виявлення на основі DNS. Застосування розробленої інформаційної технології дозволяє підвищити достовірність процесу виявлення бот-мереж в порівнянні з відомими інформаційними технологіями та виявляти нові бот-мережі. Розроблена інформаційна технологія, на відміну від відомих, уможливлює уточнений поділ періоду моніторингу на інтервали, в межах яких здійснюється пошук груп інфікованих комп'ютерних систем, що ґрунтується на основі аналізу значень TTL, які містяться в DNS-повідомленнях, враховує особливості поведінки груп інфікованих комп'ютерних систем, характерні для багатьох видів бот-мереж; а також ґрунтується на залученні кластерного аналізу множини ознак, одержаних з корисного навантаження DNS-повідомлень, які вказують на використання технологій ухилення бот-мереж на основі DNS.

3. Розроблено програмне забезпечення, яке дозволяє виявляти боти відомих та невідомих бот-мереж в корпоративній мережі на основі аналізу DNS-трафіка. Застосування ПЗ розробленої інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка дозволяє підвищити

оперативність роботи інформаційних систем корпоративної мережі та рівень безпеки корпоративної мережі, що підтверджується відповідними актами впровадження. Одержані результати досліджень показали підвищення достовірності виявлення на 8-22% в порівнянні з відомими засобами виявлення бот-мереж.

ВИСНОВКИ

В дисертаційній роботі вирішена актуальна науково-технічна задача виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка. Обмеження областю корпоративних мереж та володіння апріорними знаннями стосовно системи доменних імен надало можливість підвищити достовірність виявлення нових та вже відомих бот-мереж. Враховуючи широке поширення КС в переважній більшості галузей, вирішення цієї задачі має важливе значення.

Основні наукові і практичні результати роботи полягають у наступному:

1. Проведено аналіз сучасних інформаційних технологій виявлення бот-мереж, який показав не високу достовірність виявлення ними невідомих нових ботів бот-мереж.

2. Досліджено особливості функціонування бот-мереж з врахуванням системи доменних імен та застосування технологій ухилення від виявлення бот-мереж на основі DNS.

3. Розроблено модель бот-мереж з врахуванням системи доменних імен, а також використання бот-мережами технологій ухилення від виявлення на основі DNS, що надає можливість підвищити достовірність виявлення бот-мереж.

4. Розроблено модель DNS-трафіка та модель процесу виявлення бот-мереж в мережах на основі аналізу DNS-трафіка, яка відрізняється від відомих моделей тим, що дозволяє здійснювати виявлення вже відомих та невідомих бот-мереж.

5. Розроблено та досліджено метод ідентифікації бот-мереж на основі їх групової активності в DNS-трафіку, який, на відміну від відомих, уможливлює уточнений поділ періоду моніторингу на інтервали, в межах яких здійснюється пошук груп інфікованих комп’ютерних систем, що ґрунтується на основі аналізу значень TTL, які містяться в DNS-повідомленнях, використовує нову ознаку синхронності DNS-запитів, а також враховує особливості поведінки груп інфікованих комп’ютерних систем,

характерні для багатьох видів бот-мереж, що дозволило підвищити достовірність виявлення нових бот-мереж.

6. Розроблено та досліджено метод виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, який ґрунтується на залученні кластерного аналізу множини ознак, одержаних з корисного навантаження DNS-повідомлень, які вказують на використання таких технологій ухилення, що дозволило підвищити достовірність виявлення нових бот-мереж.

7. Розроблені методи інтегровано в інформаційну технологію виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка. Інформаційну технологію реалізовано у вигляді програмного забезпечення, яке дозволяє ідентифікувати боти, що здійснюють групову активність в DNS-трафіку, а також виявляти боти бот-мереж, які застосовують технології ухилення від виявлення на основі DNS. Застосування розробленої інформаційної технології дозволяє підвищити достовірність процесу виявлення бот-мереж в порівнянні з відомими інформаційними технологіями на 8-22% та виявляти як відомі, так і нові бот-мережі.

Література

1. DAMBALLA. Botnet detection for communications service providers [Electronic resource]. – Mode of access: https://www.damballa.com/downloads/r_pubs/WP_Botnet_Detection _for_CSPs.pdf. – Title from the screen.

2. OpenDNS. Security Whitepaper. The Role of DNS in Botnet Command & Control [Electronic resource]. – Mode of access: http://info.opendns.com/rs/opendns/images/OpenDNS_SecurityWhitepaper-DNSRoleInBotnets.pdf. – Title from the screen.

3. Salusky W., Danford R. Know your enemy: Fast-flux service networks. The Honeynet Project, 2007 [Electronic resource]. – Mode of access: http://www.honeynet.org/book/export/html/130. – Title from the screen.

4. Farnham G. Detecting DNS tunneling [Text] / G. Farnham, A. Atlasis // SANS Institute InfoSec Reading Room, 2013. – PP. 1-32.

5. The Honeynet Project. Know your enemy: Tracking botnets [Electronic resource] / P. Bacher, T. Holz, M. Kotter [et al.]. – Mode of access: http://www.honeynet.org/papers/bots. – Title from the screen.

6. Holz T. Measuring and Detecting Fast-Flux Service Networks [Text] / T. Holz, C. Gorecki, K. Rieck, F. C. Freiling // Proceedings of the Network and Distributed System Security Symposium (NDSS), 2008. – P. 12.

7. Malan D.J. Rapid Detection of Botnets through Collaborative Networks of Peers / D.J.Malan // Ph.D. Thesis. Harvard University, School of Engineering and Applied Sciences, Cambridge, Massachusetts, 2007. – P.104.

8. Antonakakis M. Building a Dynamic Reputation System for DNS [Text] / M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, N. Feamster // 19th Usenix Security Symposium, 2010. – PP. 273-290.

9. Antonakakis M. Detecting Malware Domains at the Upper DNS Hierarchy [Text] / M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, D. Dagon // Proceedings of the 20th USENIX Conference on Security, SEC'11, Berkeley, CA, USA, 2011. USENIX Association. – PP. 1-16.

10. Antonakakis M. From throw-away traffic to bots: Detecting the rise of dga-based malware [Text] / M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, D. Dagon // Proceedings of the 21st USENIX security symposium, 2012. – PP. 491-506.

11. Dagon D. Botnet detection and response. The network is the infection [Text] / D. Dagon // OARC Workshop, 2005. – P. 45.

12. Dagon D. Modeling botnet propagation using time zones [Text] / D. Dagon, C. Zou, W. Lee // Proceedings of the 13th Annual Network and Distributed System Security Symposium, 2006. – PP. 2-13.

13. Dagon D. A Taxonomy of Botnet Structures [Text] / D. Dagon, G. Gu, C.P. Lee // Botnet Detection. Countering the Largest Security Threat: Springer US, Springer Science+Business Media, LLC, 2008. – PP. 143-164. – ISBN: 978-0-387-68766-7, ISBN: 978-0-387-68768-1.

14. Choi H. Botnet Detection by Monitoring Group Activities in DNS Traffic [Text] / H. Choi, H. Lee, H. Lee, H. Kim // Seventh IEEE International Conference on Computer and Information Technology (CIT 2007), 2007. – PP. 715-720.

15. Choi H. Identifying botnets by capturing group activities in DNS traffic [Text] / H. Choi, H. Lee // Computer Networks, 56, 2012. – PP. 20-33.

16. Dietrich C.J. On Botnets that use DNS for Command and Control [Text] / C.J. Dietrich, C. Rossow, F.C. Freiling, H. Bos, M. van Steen, N. Pohlmann // Proceedings of European Conference on Computer Network Defense, 2011. – PP. 9-16.

17. Ichise H. Detection Method of DNS-based Botnet Communication Using Obtained NS Record History [Text] / H. Ichise, J. Yong, K. Iida // Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual, vol.3, 2015. – PP. 676-677.

18. Ichise H. Analysis of via-resolver DNS TXT queries and detection possibility of botnet communications [Text] / H. Ichise, J. Yong, K. Iida // Communications, Computers and Signal Processing (PACRIM), 2015 IEEE Pacific Rim Conference on, 2015. – PP.216-221.

19. Погребенник В.Д. Пасивні методи виявлення ботнет-мереж [Текст] / В.Д. Погребенник, П.Т. Хромчак // Вісник Національного університету «Львівська політехніка». – № 741, 2012. – С. 97-104.

20. Погребенник В.Д. Активні методи виявлення ботнет-мереж [Текст] / В. Д. Погребенник, П. Т. Хромчак // Вісник Національного університету "Львівська політехніка". Автоматика, вимірювання та керування. – 2013. – № 774. – С. 3-9.

21. Погребенник В.Д. Розроблення моделі системи виявлення центрів управління ботнет-мережами [Текст] / В.Д. Погребенник, П.Т. Хромчак // Вісник Національного університету «Львівська політехніка». Автоматика, вимірювання та керування № 639. - Л. : Вид-во Нац. ун-ту "Львів. політехніка", 2009. – С. 117-123.

22. Kotenko I. Simulation of Protection Mechanisms against Botnets on The Basis of “Nervous Network” Framework [Text] / I. Kotenko, A. Shorov // Proceedings of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2012), SciTePress, 2012, – P.164-169.

23. Kotenko I. Simulation of botnets and protection mechanisms against them: software environment and experiments [Text] / I. Kotenko, A. Konovalov, A. Shorov // 16th Nordic Conference on Secure IT-Systems. October 26th-28th, 2011. Tallinn, Estonia, Preproceedings, Cybernetica, 2011, – P. 119-126.

24. Котенко И.В. Исследовательское моделирование бот-сетей и механизмов защиты от них [Текст] / И.В. Котенко, А.М. Коновалов, А.В. Шоров // Приложение к журналу «Информационные технологии». – Москва: Издательство Новые технологии, 2012, № 1, 32 с. – ISSN 1684-6400.

25. Рувинская В.М. Эвристические методы детектирования вредоносных программ на основе сценариев [Текст] / В.М. Рувинская, Е.Л. Беркович, А.А. Лотоцкий // Штучний інтелект. – 2008. – № 3. – С. 197-207.

26. Сиротинський О.І. Захист інформаційних систем від мережевих DDOS атак на основі марківської моделі поведінки ботнету [Текст] / Сиротинський О.І., Беляєв І.С., Максимюк Т.А., Олексін М.І. // Вісник

Національного університету «Львівська політехніка». – 2012. – № 738. – С. 192-197.

27. Касперський Е.В. Компьютерное зловредство, 1-е издание [Текст] / Е.В. Касперський. – СПб. : Питер. – 2009. – 208 с.

28. Касперски К. Записки исследователя компьютерных вирусов [Текст] / Касперски К. – СПб. : Питер. – 2005. – 320 с.

29. Собейкис В.Г. Азбука хакера. Компьютерная вирусология [Текст] / В.Г. Собейкис. – М.: Майор. – 2006. – 512 с.

30. Бобровнікова К.Ю. Модель інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка [Текст] / К.Ю. Бобровнікова // Вісник Хмельницького національного університету. – 2015. – № 6 (231). – С.164-172 (індексується в наукометричній базі Index Copernicus).

31. Бобровнікова К.Ю. Методи та програмне забезпечення інформаційної технології виявлення бот-мереж на основі аналізу DNS–трафіка [Текст] / К.Ю. Бобровнікова // Вісник Хмельницького національного університету. – 2016. – № 2. – С.53-57 (індексується в наукометричній базі Index Copernicus).

32. Бобровнікова К.Ю. Методи виявлення бот-мереж, що використовують технології ухилення на основі DNS [Текст] / К.Ю.Бобровнікова, Д.В.Муравський, О.О.Павлова // Збірник наукових праць IV Всеукраїнської науково-практичної конференції молодих учених та студентів «Інтелектуальні технології в системному програмуванні», Хмельницький, 2015. – С. 25-29.

33. Бобровнікова К.Ю. Аналіз DNS-методів виявлення бот-мереж [Текст] / К.Ю.Бобровнікова, А.І.Наконечний, М.А.Репушанська // Збірник наукових праць IV Всеукраїнської науково-практичної конференції молодих учених та студентів «Інтелектуальні технології в системному програмуванні», Хмельницький, 2015. – С. 29-33.

34. Савенко О.С. Дослідження DNS-методів виявлення бот-мереж [Текст] / О. С. Савенко, С. М. Лисенко, К. Ю. Бобровнікова // Контроль і управління в складних системах (КУСС-2014). XII Міжнародна конференція.

Тези доповідей. Вінниця, 14-16 жовтня 2014 року. – Вінниця: ВНТУ. – 2014. – С. 87.

35. Савенко О.С. Модель процесу виявлення бот-мереж на основі аналізу DNS-трафіка [Текст] / О. С. Савенко, К. Ю. Бобровнікова // Матеріали 3-ї Міжнародної конференції з автоматичного управління та інформаційних технологій, ICACIT-2015 . – К.: КПІ. – 2015. – С. 60-67.

36. Савенко О.С. DNS-метод виявлення бот-мереж [Текст] / О. С. Савенко, С. М. Лисенко, К. Ю. Бобровнікова // Інформаційні технології та комп'ютерна інженерія, 2014. – № 3. – С. 39-45.

37. Pomorova O. A Technique for the Botnet Detection Based on DNS-Traffic Analysis [Text] / O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, K. Bobrovnikova // Communications in Computer and Information Science. – 2015. – Vol. 522. – PP. 127-138, ISSN: 1865-0929 (part scientometric Web of Science and SCOPUS).

38. Савенко О.С. Метод виявлення бот-мереж, що використовують технології ухилення на основі DNS [Текст] / О. С. Савенко, С. М. Лисенко, К. Ю. Бобровнікова // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. – 2015. – № 19. – С. 71-78.

39. Савенко О.С. Метод виявлення бот-мереж на основі пасивного моніторингу DNS-трафіка та активного DNS-зондування [Текст] / О. С. Савенко, С. М. Лисенко, К. Ю. Бобровнікова // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. – 2016. – № 22. – С. 136-143.

40. Lysenko S. DNS-based Anti-evasion Technique for Botnets Detection [Text] / S. Lysenko, O. Pomorova, O. Savenko, A. Kryshchuk, K. Bobrovnikova // Proceedings of the 2015 IEEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), IDAACS'2015, Warsaw, Poland, September 24-26, 2015, Vol.1. – PP. 453-458, ISBN: 978-1-4673-8359-2 (Print) (part scientometric Web of Science and SCOPUS).

41. Lysenko S. Anti-evasion Technique for the Botnets Detection Based on the Passive DNS Monitoring and Active DNS Probing / S. Lysenko, O. Pomorova,

O. Savenko, A. Kryshchuk, K. Bobrovnikova [Text] // Communications in Computer and Information Science. – 2016. – Vol. 608. – PP. 83-95, ISSN: 1865-0929 (part scientometric SCOPUS).

42. Бобровнікова К.Ю. Метод виявлення бот-мереж на основі пасивного моніторингу DNS-трафіка та активного DNS-зондування [Текст] / К. Ю. Бобровнікова, О. С. Савенко, С. М. Лисенко // Збірник тез доповідей міжнародного науково-практичного семінару молодих вчених та студентів «Програмовані логічні інтегральні схеми та мікропроцесорна техніка в освіті і виробництві». – м. Луцьк, ЛНТУ. – 2016. – С. 20-23.

43. Савенко О.С. Применение кластерного анализа для решения задачи обнаружения бот-сетей на основе анализа DNS-трафика [Текст] / О.С. Савенко, С. Н. Лысенко, К.Ю. Бобровникова // Сборник трудов XVI Международной научной конференции «Интеллектуальный анализ информации (ИАИ-2016)» им. Т.А.Таран. Сборник трудов. – К.: Просвіта . – 2016. – С.186-192.

44. Савенко О. С. Інформаційна технологія виявлення бот-мереж на основі аналізу DNS-трафіка [Текст] / О. С. Савенко, С. М. Лисенко, К. Ю. Бобровнікова // Радіоелектронні і комп'ютерні системи. – 2016. – № 5. – С. 38-42 (індексується в наукометричній базі Index Copernicus).

45. Мультиагентний спосіб локалізації бот-мереж у корпоративних комп'ютерних мережах [Текст] : пат. 108238 Україна : МПК G06F 21/55 (2013.01) / О.В. Поморова, О.С. Савенко, А.Ф. Крищук, С.М. Лисенко, К.Ю. Бобровнікова, А.О. Нічепорук; заявник та власник Хмельницький національний університет. – № u2016 00127 ; заявл. 04.01.16 ; опубл. 11.07.16, Бюл. № 13.

46. PaySpaceMagazine. С какими киберугрозами столкнулся бизнес в 2015 году? [Електронний ресурс]. – Режим доступу: http://psm7.com/s-kakimi-kiberugrozami-stolknulsya-biznes-v-2015-godu.html. – Назва з екрана.

47. FINANCE.UA. На скільки банки можуть забезпечити безпеку інтернет-платежів (дослідження) [Електронний ресурс]. – Режим доступу:

http://news.finance.ua/ua/news/-/361848/na-skilky-banky-mozhut-zabezpechyty-bezpeku-internet-platezhiv-doslidzhennya. – Назва з екрана.

48. ОБОЗРЕВАТЕЛЬ.UA. Впервые в истории: стало известно, как хакеры отключили свет на западе Украины [Електронний ресурс]. – Режим доступу: http://finance.obozrevatel.com/business-and-finance/67950-vpervyie-v-istorii-stalo-izvestno-kak-hakeryi-otklyuchili-svet-na-zapade-ukrainyi.htm. – Назва з екрана.

49. УНІАН. Експерти розповіли, яку технологію використовували хакери при нападі на українські обленерго та ЗМІ [Електронний ресурс]. – Режим доступу: http://www.unian.ua/society/1234268-eksperti-rozpovili-yaku-tehnologiyu-vikoristovuvali-hakeri-pri-napadi-na-ukrajinski-oblenergo-ta-zmi.html. – Назва з екрана.

50. HiTech-News.ru. Кіберзлочинці почали впроваджувати шкідливе ПЗ в відео [Електронний ресурс]. – Режим доступу: http://hitech-news.ru/bezopasnost/kiberzlochinci-pochali-vprovadzhuvati-shkidlive-pz-v-video. – Назва з екрана.

51. HiTech Expert. Как хакеры взламывают Smart TV [Електронний ресурс]. – Режим доступу: http://expert.com.ua/99655-kak-xakery-vzlamyvayut-smart-tv.html. – Назва з екрана.

52. xakep.ru. Неизвестные атаковали корневые DNS-серверы [Електронний ресурс]. – Режим доступу: https://xakep.ru/2015/12/11/root-dns-ddos/. – Назва з екрана.

53. Українська правда. Економічна правда. Новий вірус атакує користувачів bitcoin [Електронний ресурс]. – Режим доступу: http://www.epravda.com.ua/news/2013/11/26/405406/. – Назва з екрана.

54. Савенко О.С. Методи та засоби антивірусного комбінованого діагностування персональних комп'ютерів [Текст] : автореф. дис. на здобуття наук. ступеня канд. техн. наук : спец. 05.13.13 – обчислювальні машини, системи і мережі / Олег Станіславович Савенко. – Вінниця : Вінницький державний технічний університет, 1999. – 20 с.

55. Лисенко С.М. Адаптивна інформаційна технологія діагностування комп'ютерних систем на наявність троянських програм [Текст] : автореф. дис. на здобуття наук. ступеня канд. техн. наук: спец. 05.13.06 - інформаційні технології / Сергій Миколайович Лисенко. – Тернопіль : ТНЕУ, 2010. – 20 с.

56. Крищук А.Ф. Мультиагентна інформаційна технологія діагностування комп'ютерних систем на наявність бот-мереж у корпоративних мережах [Текст] : автореф. дис. на здобуття наук. ступеня канд. техн. наук : спец. 05.13.06 - інформаційні технології / Андрій Федорович Крищук. – Тернопіль : ТНЕУ, 2015. – 20 с.

57. Загородна Н.В. Аналіз анти-бот стратегій, що грунтуються на інтерактивному підтвердженні участі людини [Текст] / Н.Загородна, О.Маєвський, Р.Козак // Матеріали ІІІ-ої міжнародної науково-технічної конференції "Захист інформації і безпека інформаційних систем". - Львів 05-06 червня 2014р., НУ: Львівська Політехніка, 2014. – С.42-43.

58. Козак Р.О. Аналіз засобів забезпечення анонімності в мережі Інтернет [Текст] / Р.О. Козак // Вимірювальна та обчислювальна техніка в технологічних процесах. – Х.: Хмельницький національний університет. – 2014. – №1(46). – С. 100-105.

59. Securelist. Kaspersky Security Bulletin 2015. Основная статистика за 2015 год [Електронний ресурс]. – Режим доступу: https://securelist.ru/analysis/ksb/27543/kaspersky-security-bulletin-2015-osnovnaya-statistika-za-2015-god/. – Назва з екрана.

60. Symantec. Internet Security Threat Report, april 2016 [Electronic resource]. – Mode of access: https://www.symantec.com/content/dam/symantec /docs/reports/istr-21-2016-en.pdf. – Title from the screen.

61. Demarest J. Statement before the Senate Judiciary Committee, Subcommittee on Crime and Terrorism, Washington, D.C., 2014, July 15 [Electronic resource] / The Federal Bureau of Investigation. Official Website. – Mode of access: http://www.fbi.gov/news/testimony/taking-down-botnets. – Title from the screen.

62. LB.UA. Украина – один из главных источников DDoS-атак в мире [Електронний ресурс]. – Режим доступу: http://economics.lb.ua /. – Назва з екрана.

63. Українська правда. Економічна правда. В Україні викрили потужну бот-мережу Mumblehard [Електронний ресурс]. – Режим доступу: http://www.epravda.com.ua/news/2016/04/8/588839/. – Назва з екрана.

64. ТЕКСТИ.ORG.UA. Полювання на ботів розпочато! “Авто для АТО”: Як 2400 проросійських ботів засмічують ефір у “Твіттері” [Електронний ресурс]. – Режим доступу: http://texty.org.ua/pg/article/An-21/read/62788/Poluvanna_na_botiv_rozpochato_Avto_dla_ATO. – Назва з екрана.

65. Украинский Телекоммуникационный Портал. Троян Bookworm использует для работы компоненты антивирусов [Електронний ресурс]. – Режим доступу: http://portaltele.com.ua/news/internet/troyan-bookworm-ispolzuet-dlya-raboty-kompone.html. – Назва з екрана.

66. SecurityLab.ru. Backdoor [Електронний ресурс]. – Режим доступу: http://www.securitylab.ru/news/tags/%E1%FD%EA%E4%EE%F0/. – Назва з екрана.

67. Dr. Web Антивирус. Информационный бюллетень. Ботнеты [Електронний ресурс]. – Режим доступу: https://st.drweb.com/static/new-www/files/bulletin/DrWeb_bulletin_n1.pdf. – Назва з екрана.

68. Mockapetris P. RFC-1034. Domain names – concepts and facilities [Electronic resource] // ISI, 1987. – Mode of access: http://www.ietf.org/rfc/rfc1034.txt?number=1034. – Title from the screen.

69. Mockapetris P. RFC-1035. Domain names – implementation and specification [Electronic resource] // ISI, 1987. – Mode of access: http://www.ietf.org/rfc/rfc1035.txt?number=1035. – Title from the screen.

70. Schiller C. Botnets: The Killer Web Application [Text] / Craig A. Schiller, Binkley J., Harley D. etc. // Syngress Publishing, Inc. 2007. – 446 p. ISBN-10: 1-59749-135-7, ISBN-13: 978-1-59749-135-8.

71. Eastlake D., Panitz A. RFC-2606. Reserved Top Level DNS Names [Electronic resource] // 1999. – Mode of access: http://www.faqs.org/rfcs/rfc2606.html. – Title from the screen.

72. DAMBALLA. Botnet communication topologies. Understanding the intricacies of botnet command-and-control [Electronic resource]. – Mode of access: https://www.damballa.com/downloads/r_pubs/WP_Botnet_Communications_Primer.pdf. – Title from the screen.

73. Rekhter Y. RFC-1918. Address Allocation for Private Internets [Electronic resource] / Rekhter Y., Moskowitz B., Karrenberg D., de Groot G. J., Lear E. // 1996. – Mode of access: https://tools.ietf.org/html/rfc1918. – Title from the screen.

74. Barr D. RFC 1912. Common DNS Operational and Configuration Errors [Electronic resource] // The Pennsylvania State University, 1996. – Mode of access: http://www.faqs.org/rfcs/rfc1912.html. – Title from the screen.

75. SecurityLab.ru. Conficker [Електронний ресурс]. – Режим доступу: http://www.securitylab.ru/news/tags/Conficker/. – Назва з екрана.

76. Porras P. An Analysis of Conficker's Logic and Rendezvous Points [Electronic resource] / Porras P., Saidi H., Yegneswaran V. // Technical report, Computer science laboratory SRI international 333 ravenswood avenue menlo park ca 94025 USA, 2009. – Mode of access: http://mtc.sri.com/Conficker/. – Title from the screen.

77. Kaspersky lab. «Лаборатория Касперского» анализирует новую версию вредоносной программы Kido (Conficker) [Електронний ресурс]. – Режим доступу: http://www.kaspersky.ru/about/news/virus/2009/Laboratoriya_Kasperskogo_analiziruet_novuyu_versiyu_vredonosnoi_programmy_Kido_Conficker. – Назва з екрана.

78. Wu Chwan-Hwa (John). Introduction to Computer Networks and Cybersecurity [Text] / Wu Chwan-Hwa (John), Irwin J. David // – CRC Press, 2015 p. – P. 1336.

79. DAMBALLA. On the Kraken and Bobax Botnets [Electronic resource] / Royal P. // – Mode of access: https://www.damballa.com/downloads /r_pubs/Kraken_Response.pdf, 2008. – Title from the screen.

80. Stone-Gross B. Your Botnet is My Botnet: Analysis of a Botnet Takeover [Text] / B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, G. Vigna // Proceedings of the ACM CCS, November 2009. – PP. 635-647.

81. Taking over the Torpig botnet – My botnet is your botnet [Electronic resource]: Project Website, Computer Security Group, UC Santa Barbara. – Mode of access: http://www.cs.ucsb.edu/~seclab/projects/torpig/. – Title from the screen.

82. Dr. Web. «Доктор Веб» обнаружил ботнет из более чем 550 000 «маков» [Електронний ресурс]. – Режим доступу: http://news.drweb.ru/?i=2341&c=5&lng=ru&p=0. – Назва з екрана.

83. Securelist. FAQ: Disabling the new Hlux/Kelihos Botnet [Electronic resource]. – Mode of access: https://securelist.com/blog/research/32634/faq-disabling-the-new-hluxkelihos-botnet-13/. – Title from the screen.

84. Symantec Official Blog. Morto worm sets a (DNS) record [Electronic resource]. – Mode of access: http://www.symantec.com/connect /blogs/morto-worm-sets-dns-record. – Title from the screen.

85. CH Mag. Low Profile Botnets [Electronic resource]. – Mode of access: http://www.chmag.in/low-profile-botnets/. – Title from the screen.

86. COMSS1 [Electronic resource]. – Mode of access: http://www.comss.ru/page.php?id=2758. – Title from the screen.

87. Avira [Electronic resource]. – Mode of access: http://www.avira.com. – Title from the screen.

88. Avast! [Electronic resource]. – Mode of access: https://www.avast.com/index. – Title from the screen.

89. AVG [Electronic resource]. – Mode of access: http://www.avg.com. – Title from the screen.

90. Bitdefender [Electronic resource]. – Mode of access: http://www.bitdefender.com/. – Title from the screen.

91. Dr. WEB Anti-virus [Electronic resource]. – Mode of access: http://www.drweb.com/. – Title from the screen.

92. Kaspersky Lab [Electronic resource]. – Mode of access: http://www.kaspersky.ru. – Title from the screen.

93. ESET [Electronic resource]. – Mode of access: http://www.eset.com/. – Title from the screen.

94. Symantec [Electronic resource]. – Mode of access: http://www.symantec.com. – Title from the screen.

95. Panda Security [Electronic resource]. – Mode of access: http://www.pandasecurity.com. – Title from the screen.

96. Virus Bulletin [Electronic resource]. – Mode of access: http://www.virusbtn.com. – Title from the screen.

97. The Independent IT-Security Institute AV-TEST [Electronic resource]. – Mode of access: http://www.av-test.org/. – Title from the screen.

98. Ramachandran A. Revealing botnet membership using DNSBL counter-intelligence [Text] / A. Ramachandran, N. Feamster, D. Dagon // 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI 2006), 2006. – PP. 8-13.

99. Morales J.A. Analyzing DNS activities of bot processes [Text] / J.A. Morales, A. Al-Bataineh, S. Xu, R. Sandhu // Proceedings of the 4th International Conference on Malicious and Unwanted Software (MALWARE), 2009. – PP. 98-103.

100. Akiyama M. A proposal of metrics for botnet detection based on its cooperative behavior [Text] / M. Akiyama, T. Kawamoto, M. Shimamura, T. Yokoyama, Y. Kadobayashi, S. Yamaguchi // International Symposium on Applications and the Internet Workshops (SAINTW'07), 2007. – PP. 82-85.

101. Manasrah A.M. Detecting Botnet Activities Based on Abnormal DNS traffic [Text] / A.M. Manasrah, A. Hasan, O.A. Abouabdalla, S. Ramadass // International Journal of Computer Science and Information Security (IJCSIS), Vol. 6, No.1, 2009. – PP. 97-104.

102. Roshna R.S. Botnet Detection Using Adaptive Neuro Fuzzy Inference System [Text] / R.S. Roshna, E. Vinodh // International Journal of Engineering Research and Applications (IJERA), Vol. 3, Issue 2, March-April 2013. – PP.1440-1445.

103. Villamarın-Salomon R. Identifying Botnets Using Anomaly Detection Techniques Applied to DNS Traffic [Text] / R. Villamarın-Salomon, J.C. Brustoloni // Consumer Communications and Networking Conference, 2008. – PP. 476-481.

104. Basheer N.AI-Duwairi. Fast Flux Watch: A mechanism for online detection of fast flux networks [Text] / Basheer N.AI-Duwairi, Ahmad T. AI-Hammouri // Joural of Advanced Research, May 2014. – PP. 473-479.

105. Caglayan A. Real-time detection of fast flux service networks [Text] / A. Caglayan, M. Toothaker, D. Drapeau, D. Burke, G. Eaton // Proceedings of the Cybersecurity Applications & Technology Conference for Homeland Security, 2009. – PP. 285-292.

106. Celik Z.B. Detection of Fast-Flux Networks using various DNS feature sets [Text] / Z.B. Celik, S. Oktug // Computers and Communications (ISCC), 2013 IEEE Symposium, PP. 868-873.

107. Hsu C.-H. Fast-Flux Bot Detection in Real Time [Text] / C.-H. Hsu, C.-Y. Huang, K.-T. Chen // S. Jha, R. Sommer, and C. Kreibich, editors, Recent Advances in Intrusion Detection, volume 6307 of Lecture Notes in Computer Science : Springer Berlin / Heidelberg, 2010. – PP. 464-483.

108. Knysz M. Good guys vs. Bot Guise: Mimicry attacks against fast-flux detection systems [Text] / M. Knysz, X. Hu, K. Shin // INFOCOM, 2011 Proceedings IEEE, 2011. – PP. 1844-1852.

109. Martinez-Bea S. Real-time malicious fast-flux detection using DNS and bot related features [Text] / S. Martinez-Bea, S. Castillo-Perez, J. Grcia-Alfaro // 2013 Eleventh annual international conference on privacy, security and trust (PST), 2013. – PP. 369-372.

110. Zhou C. A Self-Healing, Self-Protecting Collaborative Intrusion Detection Architecture to Trace-Back Fast-Flux Phishing Domains [Text] / C. Zhou,

C. Leckie, S. Karunasekera, T. Peng // Network Operations and Management Symposium Workshops, NOMS Workshops 2008, IEEE, 2008. – PP. 321-327.

111. Xu Y. The Availability of Fast-Flux Service Networks [Text] / Y. Xu, Y. Lu, Z. Guo // Mobile and Wireless Networking (iCOST), International Conference on Selected Topics in Mobile & Wireless Networking, 2011. – PP.89-93.

112. Wang H. Real-time fast-flux identification via localized spatial geolocation detection [Text] / H. Wang, C. Mao, K. Wu, H. Lee // Proc. Computer Software and Applications Conference(COMPSAC), 2012. – PP. 244-252.

113. Arbor Summary Report on Global Fast Flux [Electronic resource]. – Mode of access: http://atlas.arbor.net/summary/fastflux. – Title from the screen.

114. Weimer F. Passive DNS replication [Text] / F. Weimer // 17th Annual FIRST Conference on Computer Security Incident Handling (FIRST 2005), 2005. – P. 98.

115. Nazario J. As the net churns: fast-flux botnet observations [Text] / J. Nazario, T. Holz // Conference on Malicious and Unwanted Software (Malware'08), 2008. – PP. 24-31.

116. Bilge L. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis [Text] / L. Bilge, E. Kirda, C. Kruegel, M. Balduzzi // NDSS, 2011. – PP. 1-17.

117. Yong Jin. Design of Detecting Botnet Communication by Monitoring Direct Outbound DNS Queries [Text] / Jin Yong, H. Ichise, K. Iida // Cyber Security and Cloud Computing (CSCloud), IEEE 2nd International Conference on, 2015. – PP. 37-41.

118. Zhao Y. Quickly Identifying FFSN Domain and CDN Domain with Little Dataset [Text] / Y. Zhao, Z. Jin // 4th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering (ICMMCCE 2015), 2015. – PP. 1999-2004.

119. Yadav S. Winning with DNS failures: Strategies for faster botnet detection [Text] / S.Yadav, A.L.N. Reddy // Proc. of the 7th International ICST

Conference on Security and Privacy in Communication Networks, 2011. – PP. 446-459.

120. Guerid H. Privacy-preserving domain-flux botnet detection in a large scale network [Text] / H.Guerid, K. Mittig, A. Serhrouchni // Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on, IEEE, 2013. – PP. 1-9.

121. Perdisci R. Early detection of malicious flux networks via large-scale passive DNS analysis [Text] / R. Perdisci, I. Corona, G. Giacinto // Dependable and Secure Computing, IEEE Transactions, Vol. 9, Issue 5, 2012. – PP. 714-726.

122. Butler P. Quantitatively analyzing stealthy communication channels [Text] / P. Butler, K. Xu, D. Yao // Proc. Ninth Int'l Conf. Applied Cryptography and Network Security (ACNS '11), 2011. – PP. 238-254.

123. Guy J. A study of DNS [Electronic resource]. – Mode of access: http://armatum.com/blog/2009/a-study-of-dns/. – Title from the screen.

124. Guy J. Dns part ii: visualization [Electronic resource]. – Mode of access: http://armatum.com/blog/2009/dns-part-ii/. – Title from the screen.

125. Gao H. An empirical reexamination of global dns behavior [Text] / H. Gao, V. Yegneswaran, Y. Chen, P. Porras, S. Ghosh, J. Jiang, H. Duan // Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, ACM, 2013. – PP. 267-278.

126. НД ТЗІ 1.1-003-99 Термінологія в галузі захисту інформації в комп’ютерних системах від несанкціонованого доступу [Електронний ресурс] / Державна служба спеціального зв’язку та захисту інформації України. Офіційний веб-сайт. – Режим доступу: http://www.dstszi.gov.ua/dstszi/control/uk /doccatalog/. – Назва з екрана.

127. ДСТУ 3396.0-96. Захист інформації. Технічний захист інформації. Основні положення. Чинний від 01.01.1997 р. [Електронний ресурс] / Державна служба спеціального зв’язку та захисту інформації України. Офіційний веб-сайт. – Режим доступу : http://www.dstszi.gov.ua/dstszi/control/uk/publish/. – Назва з екрана.

128. НД ТЗІ 1.1-002-99 Загальні положення щодо захисту інформації в комп'ютерних системах від несанкціонованого доступу [Електронний ресурс] / Державна служба спеціального зв'язку та захисту інформації України. Офіційний веб-сайт. – Режим доступу: http://www.dstszi.gov.ua/dstszi/control/uk /doccatalog/. – Назва з екрана.

129. Kaspersky Security Bulletin 2015. Эволюция угроз информационной безопасности в бизнес-среде [Електронний ресурс]. – Режим доступу: https://securelist.ru/analysis/ksb/27519/kaspersky-security-bulletin-2015-evolyuciya-ugroz-informacionnoj-bezopasnosti-v-biznes-srede/. – Назва з екрана.

130. Braun-Blanquet J. Pflanzensoziologie Grundzüge der Vegetationskunde [Text] / J. Braun-Blanquet // Berlin: Verlaq von Julius springer, 1928. – 330 p.

131. Сёмкин Б.И. Дескриптивные множества и их приложения [Текст] / Б.И. Сёмкин // Исследование систем. Т. 1. Анализ сложных систем. – Владивосток. ДВНЦ АН СССР, 1973. – С. 83-94.

132. Семкин Б.И. О связи между средними значениями двух мер включения и мерами сходства [Текст] / Б.И. Сёмкин // Бюллетень Ботанического сада-института ДВО РАН, вып. 3, 2009. – С. 91-101.

133. Koch L. Index of biotal dispersity [Text] / L. Koch // Ecology, V. 38, No. 1, 1957. – PP. 145-148.

134. Lai D.T.C. A comparison of distance-based semi-supervised fuzzy c-means clustering algorithms [Text] / D.T.C. Lai, J.M. Garibaldi // Fuzzy Systems (FUZZ), 2011 IEEE International Conference, 2011. – PP. 1580-1586.

135. Pedrycz W. Algorithms of fuzzy clustering with partial supervision [Text] / W. Pedrycz // Pattern Recognition Letters, Vol. 3, 1985. – PP. 13–20.

136. Karasaridis A. Detection of DNS anomalies using flow data analysis [Text] / A. Karasaridis, K.S. Meier-Hellstern, D.A. Hoeflin // GLOBECOM. IEEE, 2006. – PP. 1-6.

137. Полякова М.В. Сегментация реализаций случайных процессов в автоматизированных системах технической диагностики режущих

инструментов [Текст] : дис. канд. техн. наук: 05.13.06 – информационные технологии / Полякова Марина Вячеславовна. – Одесса: Одесский национальный политехнический университет, 2003. – 206 с.

138. Полякова М. В. Разработка преобразования с обобщенными гребенчатыми масштабными и вейвлетфункциями для сегментации изображений / М. В. Полякова, В. Н. Крылов, А. В. Ищенко // Восточно-Европейский журнал передовых технологий. – 2014. – Т. 5. – №. 2 (71). – С. 33-41.

ДОДАТОК А

Лістинги основних модулів програмного забезпечення інформаційної технології виявлення бот-мереж на основі аналізу DNS-трафіка

Лістинг файлу Parser.h

```
void dispatcher_handler(u_char *, const struct pcap_pkthdr *, const u_char *);
unsigned char* ReadName(u_char*, u_char *, int *);
u_int ret4(unsigned int *);
WORD ret2(WORD *);

/* 14 bytes MAC adress */
typedef struct mac_header{
	u_char destAddr[6]; //48 bit
	u_char sourceAddr[6]; //48 bit
	u_short protocol; //16 bit
}mac_header;

/* 4 bytes IP address */
typedef struct ip_address{
	u_char byte1;
	u_char byte2;
	u_char byte3;
	u_char byte4;
}ip_address;

/* IPv4 header */
typedef struct ip_header{
	u_char  ver_ihl;	// Version (4 bits) + Internet header length (4 bits)
	u_char  tos;		// Type of service
	u_short tlen;		// Total length
	u_short identification; // Identification
	u_short flags_fo;	// Flags (3 bits) + Fragment offset (13 bits)
	u_char  ttl;		// Time to live
	u_char  proto;		// Protocol
	u_short crc;		// Header checksum
	ip_address  saddr;	// Source address
	ip_address  daddr;	// Destination address
	u_int   op_pad;		// Option + Padding
}ip_header;
```

```
/* UDP header*/
typedef struct udp_header{
        u_short sport;        // Source port
        u_short dport;        // Destination port
        u_short len;          // Datagram length
        u_short crc;          // Checksum
}udp_header;

typedef struct DNS_HEADER {
        WORD Xid;
        BYTE RecursionDesired : 1;
        BYTE Truncation : 1;
        BYTE Authoritative : 1;
        BYTE Opcode : 4;
        BYTE IsResponse : 1;
        BYTE ResponseCode : 4;
        BYTE CheckingDisabled : 1;
        BYTE AuthenticatedData : 1;
        BYTE Reserved : 1;
        BYTE RecursionAvailable : 1;
        WORD QuestionCount;
        WORD AnswerCount;
        WORD NameServerCount;
        WORD AdditionalCount;
} DNS_HEADER, *PDNS_HEADER;

struct QUERY
{
        WORD qtype;
        WORD qclass;
};

struct RES_RECORD
{
        WORD type;
        WORD _class;
        u_int ttl;
        WORD data_len;
```

```
};

union
{
        u_int a;
        u_char b[4];
} un4;

union
{
        WORD a;
        u_char b[2];
} un2;

void dispatcher_handler(u_char *temp1, const struct pcap_pkthdr *header, const u_char *pkt_data)
{
        ofstream file("dump.txt", ios_base::app);
        struct tm ltime;
        char timestr[16];
        ip_header *ih;
        udp_header *uh;
        mac_header *mac;
        DNS_HEADER *dns;
        u_int ip_len;
        u_int udp_len;
        time_t local_tv_sec;

        (VOID)temp1;

        // convert the timestamp to readable format
        local_tv_sec = header->ts.tv_sec;
        localtime_s(&ltime, &local_tv_sec);
        strftime(timestr, sizeof timestr, "%H:%M:%S", &ltime);

        mac = (mac_header *)(pkt_data);

        // retireve the position of the ip header
        ih = (ip_header *)(pkt_data + 14); //length of ethernet header
```

```
/* retireve the position of the udp header */
ip_len = (ih->ver_ihl & 0xf) * 4;
uh = (udp_header *)((u_char*)ih + ip_len);

//retireve the position of the dns header*/
dns = (DNS_HEADER *)((u_char*)uh + 8);
dns->QuestionCount = ret2(&dns->QuestionCount);
dns->AnswerCount = ret2(&dns->AnswerCount);
dns->NameServerCount = ret2(&dns->NameServerCount);
dns->AdditionalCount = ret2(&dns->AdditionalCount);

if (dns->IsResponse)
{
        QUERY *query;
        RES_RECORD *answers; //the replies from the DNS server
        u_char *buf;
        buf = (u_char *)dns;
        u_char *reader;
        u_char *qname = NULL, *name = NULL;
        int stop = 0;

        //move ahead of the dns header and the query field1
        reader = (u_char *)((u_char *)dns + sizeof(DNS_HEADER));
        //reading query
        qname = ReadName(reader, buf, &stop);

        query = (QUERY *)((u_char *)dns + sizeof(DNS_HEADER)+(int)strlen((const char*)qname) +
2);
        query->qtype = ret2(&query->qtype);

        //reading and printing answers
        stop = 0;
        reader = ((u_char *)dns + sizeof(DNS_HEADER)+(int)strlen((const char*)qname) + 2 + 4);
        name = ReadName(reader, buf, &stop);
        reader = reader + stop;

        answers = (RES_RECORD *)(reader);
        answers->type = ret2(&answers->type);
        answers->ttl = ret4(&answers->ttl);
```

```
		if ((dns->AnswerCount != 0 && ((int)query->qtype == 1 || (int)query->qtype == 28) ||
((int)dns->ResponseCode == 3 && dns->AnswerCount != 0)))
		{
			file << local_tv_sec << " ";
			for (int i = 0; i < 6; i++)
			{
				if ((int)mac->destAddr[i] <= 0xF)
					file << "0";
				file << hex << (int)mac->destAddr[i];
				if (i < 5) file << ":";
			}
			file << dec;
			file << " ";
			file << (int)ih->saddr.byte1 << "." << (int)ih->saddr.byte2 << "." << (int)ih-
>saddr.byte3 << "." << (int)ih->saddr.byte4 << " ";
			file << (int)dns->ResponseCode << " ";

			if (dns->ResponseCode == 3) file << qname << " "; else
				file << name << " ";
			file << (int)answers->ttl << endl;
		}
	}
	file.close();
}

unsigned char* ReadName(unsigned char* reader, unsigned char* buffer, int* count)
{
	unsigned char *name;
	unsigned int p = 0, jumped = 0, offset;
	int i, j;

	*count = 1;
	name = (unsigned char*)malloc(256);
	name[0] = '\0';

	while (*reader != 0)
	{
		if (*reader >= 192)
		{
```

```
                offset = (*reader) * 256 + *(reader + 1) - 49152;
                reader = buffer + offset - 1;
                jumped = 1;
            }
            else
            {
                name[p++] = *reader;
            }
            reader = reader + 1;
            if (jumped == 0) *count = *count + 1; //if we havent jumped to another location then we
can count up
        }
        name[p] = '\0'; //string complete
        if (jumped == 1)
        {
            *count = *count + 1; //number of steps we actually moved forward in the packet
        }

        //now convert 3www6google3com0 to www.google.com
        for (i = 0; i<(int)strlen((const char*)name); i++)
        {
            p = name[i];
            for (j = 0; j<(int)p; j++)
            {
                name[i] = name[i + 1];
                i = i + 1;
            }
            name[i] = '.';
        }

        name[i - 1] = '\0'; //remove the last dot

        return name;
}

u_int ret4(unsigned int *k)
{
        un4.a = *k;
        swap(un4.b[0], un4.b[3]);
```

```
	swap(un4.b[1], un4.b[2]);
	return un4.a;
}

WORD ret2(WORD *k)
{
	un2.a = *k;
	swap(un2.b[0], un2.b[1]);
	return un2.a;
}
```

Лістинг файлу Matrix.h

```
#pragma once
#include <vector>
#include <pcap/pcap.h>
#include <string>
#include <iostream>

using namespace std;

struct Record
{
	unsigned long long int time;
	string mac;
	string ip;
	int rcode;
	string host;
	u_int ttl;
	double check;
};


class table
{
public:
	Record **mas;
	int rows;
	int cols;
	table();
```

```
        ~table();
        void addCols();
        void addRows();
        void delrow(int );
};
```

Лістинг файлу Matrix.cpp

```
#include "matrix.h"

table::table()
{
        rows = 1;
        cols = 0;
        mas = new Record *[rows];
        for (int i = 0; i < rows; i++)
        {
                mas[i] = new Record[cols];
        }
}

table::~table()
{
        for (int i = 0; i < rows; i++)
                delete [] mas[i];
        delete [] mas;
}

void table::addCols()
{
        Record **tmp = new Record * [rows];
        for (int i = 0; i < rows; i++)
        {
                tmp[i] = new Record[cols];
        }

        for (int i = 0; i < rows; i++)
                for (int j = 0; j < cols; j++)
                {
                        tmp[i][j] = mas[i][j];
```

```
		}

	cols++;

	for (int i = 0; i < rows; i++)
	{
		mas[i] = new Record [cols];
	}


	for (int i = 0; i < rows; i++)
		for (int j = 0; j < cols; j++)
		{
			if (j < cols - 1) mas[i][j] = tmp[i][j]; else
				mas[i][j].check = 0;
		}
}

void table::addRows()
{
	Record **tmp = new Record *[rows];
	for (int i = 0; i < rows; i++)
	{
		tmp[i] = new Record[cols];
	}

	for (int i = 0; i < rows; i++)
	for (int j = 0; j < cols; j++)
	{
		tmp[i][j] = mas[i][j];
	}

	rows ++;
	mas = new Record *[rows];

	for (int i = 0; i < rows; i++)
	{
		mas[i] = new Record[cols];
	}
```

```
	for (int i = 0; i < rows; i++)
	for (int j = 0; j < cols; j++)
	{
		if (i < (rows - 1)) mas[i][j] = tmp[i][j]; else
			mas[i][j].check = 0;
	}
}

void table::delrow(int p)
{
	if (rows > 1)
	{
		rows--;
		Record **tmp = new Record *[rows];
		for (int i = 0; i < rows; i++)
		{
			tmp[i] = new Record[cols];
		}

		for (int i = 0; i < p; i++)
		for (int j = 0; j < cols; j++)
			tmp[i][j] = mas[i][j];

		for (int i = p + 1; i < rows + 1; i++)
		for (int j = 0; j < cols; j++)
			tmp[i - 1][j] = mas[i][j];

		mas = new Record *[rows];
		for (int i = 0; i < rows; i++)
		{
			mas[i] = new Record[cols];
		}

		for (int i = 0; i < rows; i++)
		for (int j = 0; j < cols; j++)
			mas[i][j] = tmp[i][j];
	}
	else
```

```
        {
                for (int i = 0; i < rows; i++)
                        delete[] mas[i];
                delete[] mas;
                rows = 0;
                cols = 0;
        }
}
```

Лістинг файлу searchGroup.cpp

```
#include <fstream>
#include "matrix.h"
#include "parser.h"
#include <msclr\marshal_cppstd.h>
#include <string.h>
using namespace msclr::interop;
using namespace System;
using namespace System::IO;

int parser(char *);
int ignore_ttl(vector <Record> , int, double);
bool check(vector <string>, string);
void readRecord(ifstream &, Record *);
void writeRecord(ofstream &, Record );
bool sync(table *, vector <int>, vector <int>, int, double);
void readff(ifstream &,table*, vector<string> *, vector<string> *,vector <int> *, vector <double> *, vector
<int> *,
        vector <int> *, vector <double> *, vector <int>*);
void writetf(ofstream &, table*, vector<string> , vector<string> , vector <int> , vector <double> , vector
<int> ,
        vector <int> , vector <double> , vector <int>);
void buildmatrix(string, vector<string>, int, double, int, int);
double Braun_Blanquet(vector <int> , table *, vector <int>);
double Koch(vector <int>, table *, vector <int>);
void deleter(vector <int> ,table*, vector<string> *, vector <int> *, vector <double> *, vector <int> *,
        vector <int> *,
        vector <double> *, vector <int>*);
void merger(vector<int>, table *, vector <int>,  vector <int>);
void matrix_Braun_Blanquet(table *, vector<string> , vector <int> , vector <double> , vector <int> ,
        vector <int>, vector <double>, vector <int>, double, vector<string>, vector<string>, int);
```

```
void processmatrix(double, vector <string>, int);

int main()
{
	cout << "Wait a minutes, in process...";
	remove("dump.pcap");
	String^ path = Directory::GetCurrentDirectory();
	int ts;
	double kof;
	double kof2;
	int number;
	string name;

	ifstream conf("conf.txt");
	getline(conf, name);
	conf >> ts >> kof >> kof2 >> number;
	conf.close();

	if (name[name.length() - 1] == 'p')
	{
		path += "/dump.pcap";
		String^ fileName = gcnew System::String(name.c_str());
		File::Copy(fileName, path);
		parser("dump.pcap");
	}
	else
	{
		path += "/dump.txt";
		String^ fileName = gcnew System::String(name.c_str());
		File::Copy(fileName, path);
	}
	name = "dump.txt";

	Record package;
	vector <Record> packages;
	vector <string> inf;
	string domain;

	ifstream blacklist("black list.txt");
```

```
ifstream greylist("grey list.txt");
ifstream whitelist("white list.txt");
vector <string> black;
vector <string> white;
vector <string> grey;

while (!blacklist.eof())
{
	blacklist >> domain;
	black.push_back(domain);
}

while (!whitelist.eof())
{
	whitelist >> domain;
	white.push_back(domain);
}

while (!greylist.eof())
{
	greylist >> domain;
	if (!greylist.eof())
	{
		grey.push_back(domain);
	}
}

blacklist.close();
whitelist.close();
greylist.close();

int counter = 1;
while (counter > 0)
{
	ifstream file(name);
	ofstream tmp("dump-tmp.txt");

	counter = 0;
	if (file.fail()) cout << "File not found" << endl; else
```

```
{
	if (!file.eof()) readRecord(file, &package);
	if (!file.eof())
	{
		packages.push_back(package);
		domain = package.host;
	}
	while (!file.eof())
	{
		readRecord(file, &package);
		if (!(package.host == domain))
		{
			if (!file.eof())
			{
				writeRecord(tmp, package);
				counter++;
			}
		}
		else
		{
			if (!file.eof())
			{
				packages.push_back(package);
			}
		}
	}
	if (packages.size() > 0) {
		if (check(black, packages[0].host) == true || check(grey, packages[0].host)
== true)
		{
			for (int i = 0; i < packages.size(); i++)
			{
				if (!check(inf, packages[i].mac))
				{
					inf.push_back(packages[i].mac);
				}
			}
		}
		else
```

```
                                if ((packages.size() > 0) && (check(white, packages[0].host) == false))
                                        ignore_ttl(packages, ts, kof);

                                packages.clear();
                                file.close();
                                tmp.close();
                                remove("dump.txt");
                                rename("dump-tmp.txt", "dump.txt");
                        }
                }
        }
        processmatrix(kof2,inf,number);
        remove("dump.txt");
        return 0;
}

int parser(char* pcap_file)
{
        pcap_t *fp;
        char errbuf[PCAP_ERRBUF_SIZE];
        char source[PCAP_BUF_SIZE];

        /* Create the source string according to the new WinPcap syntax */
        if (pcap_createsrcstr(source,        // variable that will keep the source string
                PCAP_SRC_FILE,  // we want to open a file
                NULL,         // remote host
                NULL,         // port on the remote host
                pcap_file,       // name of the file we want to open
                errbuf        // error buffer
                ) != 0)
        {
                fprintf(stderr, "\nError creating a source string\n");
                return -1;
        }

        /* Open the capture file */
        if ((fp = pcap_open(source,        // name of the device
                65536,         // portion of the packet to capture
                // 65536 guarantees that the whole packet will be captured on all the link layers
```

```
                PCAP_OPENFLAG_PROMISCUOUS,     // promiscuous mode
                1000,             // read timeout
                NULL,             // authentication on the remote machine
                errbuf        // error buffer
                )) == NULL)
        {
                fprintf(stderr, "\nUnable to open the file %s.\n", source);
                return -1;
        }

        // read and dispatch packets until EOF is reached
        pcap_loop(fp, 0, dispatcher_handler, NULL);
        return 0;
}

int ignore_ttl(vector <Record> list, int ts, double kof)
{
        table Vmac;
        vector <int> first;
        vector <int> last;
        vector <string> mac;
        Record pac;

        first.push_back(-1);
        last.push_back(0);

        ifstream file(list[0].host+".txt");
        if (!file.fail())
        {
                int n, m, time;
                string name,s;
                file >> n >> m;

                for (int i = 0; i < m; i++)
                {
                        file >> name;
                        mac.push_back(name);
                }
```

```
	for (int i = 0; i < n; i++)
	{
		file >> time;
		first.push_back(time);
		for (int j = 0; j < m; j++)
		{
			Vmac.addCols();
			file >> s;
			if (s == "1")
			{
				readRecord(file, &Vmac.mas[i][j]);
				Vmac.mas[i][j].check = 1;
				pac = Vmac.mas[i][j];
			}
			else
			{
				Vmac.mas[i][j].check = 0;
			}
		}
		Vmac.addRows();
		file >> time;
		last.push_back(time);
	}

	if (list.size() > 0)
	{
		if (abs((long long int)(list[0].time - pac.time) - (long long int)(pac.ttl - list[0].ttl)) > 1)
		{
			for (int i = 0; i < Vmac.rows; i++)
				delete[] Vmac.mas[i];
			delete[] Vmac.mas;
			Vmac.rows = 1;
			Vmac.cols = 0;
			mac.clear();
			first.clear();
			last.clear();
		}
	}
}
```

```
		file.clear();
		file.close();
		pac.ttl = -1;

		while (list.size() > 0)
		{
			if (pac.ttl != -1)
			{
				if (abs((long long int)(list[0].time - pac.time) - (long long int)(pac.ttl - list[0].ttl)) > 1)
break;
			}

			if (check(mac, list[0].mac))
			{
				for (int i = 0; i < mac.size(); i++)
				{
					if (list[0].mac == mac[i])
					{
						if ((int)Vmac.mas[Vmac.rows - 1][i].check == 0)
						{
							Vmac.mas[Vmac.rows - 1][i] = list[0];
							Vmac.mas[Vmac.rows - 1][i].check = 1;
							last[Vmac.rows - 1] = list[0].ttl;
							pac = list[0];
							list.erase(list.begin());
						}
						else
						{
							Vmac.addRows();
							first.push_back(list[0].ttl);
							last.push_back(list[0].ttl);
							Vmac.mas[Vmac.rows - 1][i] = list[0];
							Vmac.mas[Vmac.rows - 1][i].check = 1;
							pac = list[0];
							list.erase(list.begin());
						}
						break;
					}
				}
```

```
		}
		else
		{
			Vmac.addCols();
			mac.push_back(list[0].mac);
			if (first[0] == -1) first[0] = list[0].ttl;
			last[Vmac.rows - 1] = list[0].ttl;
			Vmac.mas[Vmac.rows - 1][Vmac.cols - 1] = list[0];
			Vmac.mas[Vmac.rows - 1][Vmac.cols - 1].check = 1;
			pac = list[0];
			list.erase(list.begin());
		}
}

if (Vmac.cols > 3)
{
		if (sync(&Vmac, first, last, ts, kof) == true)
		{
			ifstream local("local-DNS.txt");
			vector <string> loc;
			string st;
			while (!local.eof())
			{
				local >> st;
				loc.push_back(st);
			}
			local.close();

			int ng = 0, f = 0, r = 0;
			double s = 1;
			ng = mac.size();
			if (Vmac.rows > 1) f = 1; else f = 0;

			for (int i = 0; i < Vmac.rows; i++)
			{
				for (int j = 0; j < Vmac.cols; j++)
				{
					if (Vmac.mas[i][j].check == 1 && Vmac.mas[i][j].rcode == 3) r = 1;
					if (check(loc, Vmac.mas[i][j].ip))
```

```
                        {
                                if (s == 1)         s = 0.5;
                        }
                        else
                        {
                                if (s == 0.5)
                                {
                                        s = 0;
                                }
                        }
                }
        }

        buildmatrix(pac.host, mac, ng, s, f, r);
}
else

if (Vmac.cols > 3)
{
        ofstream file1(pac.host + ".txt");
        file1 << Vmac.rows << " " << Vmac.cols << endl;
        for (int i = 0; i < mac.size(); i++)
                file1 << mac[i] << endl;
        for (int i = 0; i < Vmac.rows; i++)
        {
                file1 << first[i] << " ";
                for (int j = 0; j < Vmac.cols; j++)
                {
                        if ((int)Vmac.mas[i][j].check == 1)
                        {
                                file1 << "1 ";
                                writeRecord(file1, Vmac.mas[i][j]);
                        }
                        else
                                file1 << "0 ";
                }
                file1 << last[i] << endl;
        }
        file1.close();
```

```
                }

                if (list.size() > 0) ignore_ttl(list, ts, kof);
                return 0;
        }
}

bool check(vector <string> vec, string str)
{
        for (int i = 0; i < vec.size(); i++)
        {
                if (str == vec[i]) return true;
        }
        return false;
}

void readRecord(ifstream &file, Record * pack)
{
        file >> pack->time;
        file >> pack->mac;
        file >> pack->ip;
        file >> pack->rcode;
        file >> pack->host;
        file >> pack->ttl;
}

void writeRecord(ofstream & file, Record pack)
{
        file << pack.time << " ";
        file << pack.mac << " ";
        file << pack.ip << " ";
        file << pack.rcode << " ";
        file << pack.host << " ";
        file << pack.ttl << endl;
}

bool sync(table *Vmac, vector <int> first, vector <int> last, int ts, double kof)
{
        bool synh = false;
```

```
int z;

for (int i = 0; i < Vmac->rows; i++)
{
	int pp = 0;
	for (int j = 0; j < Vmac->cols; j++)
	if (Vmac->mas[i][j].check == 1) pp++;
	if (pp < 4) break;
	z = (first[i] - last[i]) / ((1.0 / 3.0)*ts);
	int *arr = new int[z];
	for (int l = 0; l < z; l++)
		arr[l] = 0;
	for (int j = 0; j < Vmac->cols; j++)
	{
		if (last[0] - first[0] <= ts)
		{
			synh = true;
			break;
		}
		else
		{
			if (Vmac->mas[i][j].check == 1)
			{
				int k = (Vmac->mas[i][j].ttl - last[i]) / ((1 / 3)*ts);
				if (k == z) k--;
				arr[k]++;
			}
		}
	}
	if (synh == true) break;
	int max = arr[0];
	int p = 0, sum = max;
	for (int j = 1; j < z; j++)
	{
		sum += arr[i];
		if (arr[j] > max)
		{
			max = arr[i];
			p = i;
```

```
                }
            }
            if (p == 0)
            {
                max = arr[0] + arr[1] + arr[2];
            }
            else
            if (p == 1)
            {
                max = arr[0] + arr[1] + arr[2];
                if (max - arr[0] + arr[3] > max) max = max - arr[0] + arr[3];
            }
            else
            if (p == z - 1)
            {
                max = arr[z - 1] + arr[z - 2] + arr[z - 3];
            }
            else
            if (p == z - 2)
            {
                max = arr[z - 1] + arr[z - 2] + arr[z - 3];
                if (max - arr[z - 1] + arr[z - 4] > max) max = max - arr[z - 1] + arr[z - 4];
            }
            else
            if (p > 1)
            {
                max = arr[p - 2] + arr[p - 1] + arr[p];
                if (max - arr[p - 2] + arr[p + 1] > max) max = arr[p - 1] + arr[p] + arr[p + 1];
                if (arr[p] + arr[p + 1] + arr[p + 2] > max) max = arr[p] + arr[p + 1] + arr[p + 2];
            }

            if ((max*1.0) / (sum*1.0) >= kof)
            {
                synh = true;
                break;
            }
        }
        return synh;
}
```

```
void readff(ifstream &file, table* table, vector<string> *host, vector<string> *domain, vector <int> *ng,
vector <double> *ss,
	vector <int> *ff, vector <int> *rr, vector <double> *mm, vector <int> *nn)
{
	int ng1, ff1, rr1, nn1;
	double ss1, mm1;

	int row, col;
	file >> row >> col;
	string str;
	for (int i = 0; i < col; i++)
	{
		file >> str;
		host->push_back(str);
	}
		for (int i = 0; i < row; i++)
	{
		file >> str;
		domain->push_back(str);
		file >> ng1 >> ss1 >> ff1 >> rr1 >> mm1 >> nn1;
		ng->push_back(ng1);
		ss->push_back(ss1);
		ff->push_back(ff1);
		rr->push_back(rr1);
		mm->push_back(mm1);
		nn->push_back(nn1);
	}
	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < col; j++)
		{
			if (i == 0)		table->addCols();
			file >> table->mas[i][j].check;
		}
		table->addRows();
	}
}
```

```
void writetf(ofstream & file, table* table, vector<string> host, vector<string> domain, vector <int> ng,
vector <double> ss,
	vector <int> ff, vector <int> rr, vector <double> mm, vector <int> nn)
{
	file << table->rows << " " << table->cols << endl;

	for (int i = 0; i < host.size(); i++)
	{
		file << host[i] << endl;
	}
	for (int i = 0; i < table->rows; i++)
	{
		file << domain[i] << " " << ng[i] << " " << ss[i] << " " << ff[i] << " " << rr[i] << " " << mm[i] << "
" << nn[i] << endl;
	}

	for (int i = 0; i < table->rows; i++)
	{
		for (int j = 0; j < table->cols; j++)
		{
			file << table->mas[i][j].check << " ";
		}
		file << endl;
	}
}

void buildmatrix(string name, vector<string> mac, int ng1, double s1, int f1, int r1)
{
	table Mk;
	vector <string> host;
	vector <string> domain;
	vector <int> Ng;
	vector <double> S;
	vector <int> F;
	vector <int> R;
	vector <double> M;
	vector <int> N;

	ifstream file("observation matrix.txt");
```

```
if (!file.fail())
{
	readff(file, &Mk, &host, &domain, &Ng, &S, &F, &R, &M, &N);
}

file.close();

domain.push_back(name);
for (int j = 0; j < host.size(); j++)
{
	for (int i = 0; i < mac.size(); i++)
	{
		if (mac[i] == host[j])
		{
			Mk.mas[Mk.rows - 1][j].check = 1;
			mac.erase(mac.begin() + i);
			break;
		}
	}
}
if (mac.size() > 0)
{
	for (int i = 0; i < mac.size(); i++)
	{
		Mk.addCols();
		host.push_back(mac[i]);
		Mk.mas[Mk.rows - 1][Mk.cols - 1].check = 1;
	}
}
mac.clear();
Ng.push_back(ng1);
S.push_back(s1);
F.push_back(f1);
R.push_back(r1);
M.push_back(0);
N.push_back(0);

ofstream out("observation matrix.txt");
writetf(out, &Mk, host, domain, Ng, S, F, R, M, N);
```

```
	out.close();
}

double Braun_Blanquet(vector <int> num, table *table, vector <int> ng)
{
	int N0 = 0;
	int max;

	for (int i = 0; i < table->cols; i++)
	if (table->mas[num[0]][i].check == 1 && table->mas[num[1]][i].check == 1) N0++;
	if (ng[num[0]] > ng[num[1]]) max = ng[num[0]]; else max = ng[num[1]];

	double res;
	res = (N0*1.0) / (max*1.0);
	return res;
}

double Koch(vector <int> num, table *table, vector <int> ng)
{
	int q, C = 0, A = 0;
	q = num.size();
	for (int i = 0; i < q; i++)
		C += ng[i];

	for (int i = 0; i < table->cols; i++)
	{
		for (int j = 0; j < q; j++)
		if (table->mas[num[j]][i].check == 1)
		{
			A++;
			break;
		}
	}

	double res;
	res = ((C-A)*1.0)/((q-1)*A*1.0);
	return res;
}
```

```
void deleter(vector <int> num,table* table, vector<string> *domain, vector <int> *ng, vector <double> *ss,
	vector <int> *ff, vector <int> *rr, vector <double> *mm, vector <int> *nn)
{
	for (int i = num.size() - 1; i >= 0; i--)
	{
		domain->erase(domain->begin() + num[i]);
		ng->erase(ng->begin() + num[i]);
		ss->erase(ss->begin() + num[i]);
		ff->erase(ff->begin() + num[i]);
		rr->erase(rr->begin() + num[i]);
		mm->erase(mm->begin() + num[i]);
		nn->erase(nn->begin() + num[i]);
		table->delrow(num[i]);
	}
}

void merger(vector<int> num, table * table, vector <int> ng, vector <int> rr)
{
	int k = 0;
	for (int i = 0; i < table->cols; i++)
	{
		for (int j = 1; j < num.size(); j++)
		if (table->mas[num[j]][i].check == 1)
		{
			table->mas[num[0]][j].check = 1;
			k++;
			break;
		}
	}
	ng[num[0]] += k;

	for (int i = 0; i < num.size(); i++)
	if (rr[num[i]] == 1)
	{
		rr[num[0]] = 1;
		break;
	}
}
```

```
void matrix_Braun_Blanquet(table * Mk, vector<string> host, vector<string> domain, vector <int> ng,
vector <double> ss, vector <int> ff,
	vector <int> rr, vector <double> mm, vector <int> nn, double kof2, vector <string> inf, vector
<string> greyl, int number)
{
	vector <int> num;
	for (int i = 0; i < Mk->cols; i++)
	{
		for (int j = 0; j < Mk->rows - 1; j++)
		{
			if (ng[j] > ng[j + 1])
			{
				swap(ng[j], ng[j + 1]);
				swap(ss[j], ss[j + 1]);
				swap(ff[j], ff[j + 1]);
				swap(rr[j], rr[j + 1]);
				swap(mm[j], mm[j + 1]);
				swap(nn[j], nn[j + 1]);
				swap(domain[j], domain[j + 1]);
				for (int l = 0; l < Mk->cols; l++)
				{
					swap(Mk->mas[j][l], Mk->mas[j + 1][l]);
				}
			}
		}
	}

	table Bk;
	for (int i = 0; i < Mk->rows; i++)
	{
		for (int j = 0; j < Mk->rows; j++)
			Bk.addCols();
		Bk.addRows();
	}
	Bk.rows--;
	num.clear();
	for (int i = 0; i < Bk.rows; i++)
	{
		num.clear();
```

```
		for (int j = 0; j <= i; j++)
		{
			if ((ng[j] * 1.0) / (ng[i] * 1.0) >= (kof2 - 0.1))
			{
				num.push_back(i);
				num.push_back(j);
				Bk.mas[i][j].check = Braun_Blanquet(num, Mk, ng);
			}
			else break;
			num.clear();
		}
	}

	vector <int> delvec;
	vector <int> infected;

	for (int i = 0; i < Bk.rows; i++)
	{
		for (int j = 0; j <= i; j++)
		{
			if (Bk.mas[i][j].check >= (kof2 - 0.1))
			{
				int vec[4];
				if (ss[i] == ss[j] && ss[i] == 0) vec[0] = 1; else
				if (ss[i] == ss[j] && ss[i] == 0.5) vec[0] = 0; else
				if (ss[i] == ss[j] && ss[i] == 1) vec[0] = 3; else
					vec[0] = 2;

				if (ff[i] == ff[j] && ff[i] == 0) vec[1] = 0; else
				if (ff[i] != ff[j]) vec[1] = 2; else
				if (ff[i] == ff[j] && ff[i] == 1) vec[1] = 3;

				if (rr[i] == rr[j] && rr[i] == 0) vec[2] = 0; else
				if (rr[i] != rr[j]) vec[2] = 2; else
				if (rr[i] == rr[j] && rr[i] == 1) vec[2] = 3;

				if (mm[i] == mm[j] && mm[i] == 0) vec[3] = 0; else
				if (((mm[i] == 0, 5 || mm[j] == 0.5) && mm[i] != 1 && mm[j] != 1) && mm[i]
!= mm[j]) vec[3] = 2; else //suspicious
```

```
                if (mm[i] == 1 || mm[j] == 1 || ((mm[i] == mm[j] && mm[i] == 0.5) && (nn[i]
!= mm[j] || (nn[i] == nn[j] && nn[i] == 0))))
                        vec[3] = 3;


                if (Bk.mas[i][j].check < kof2 && vec[0] == 1 && vec[1] != 2 && vec[2] != 2
&& vec[3] != 2 && vec[1] != 3 &&
                        vec[2] != 3 && vec[3] != 3)
                {
                        delvec.push_back(i);
                        delvec.push_back(j);
                } else
                if (Bk.mas[i][j].check < kof2 && vec[0] != 1 && vec[0] != 2 && vec[1] != 2 &&
vec[2] != 2 && vec[3] != 2 && vec[0] != 3
                        && vec[1] != 3 && vec[2] != 3 && vec[3] != 3)
                {
                        int l = 0;
                        while (l < delvec.size())
                        {
                                if (delvec[l] == i || delvec[l] == j)
                                {
                                        delvec.erase(delvec.begin() + l);
                                }
                                else l++;
                        }
                } else
                if (vec[0] == 3 || vec[1] == 3 || vec[2] == 3 || vec[3] == 3 ||
Bk.mas[i][j].check >= kof2)
                {
                        infected.push_back(i);
                        infected.push_back(j);
                        bool p1 = false, p2 = false;
                        for (int l = 0; l < delvec.size(); l++)
                        {
                                if (delvec[l] == i) p1 = true;
                                if (delvec[l] == j) p2 = true;
                        }
                        if (p1 == true) delvec.push_back(i);
                        if (p2 == true) delvec.push_back(j);
                }
                else
```

```
                {
                        mm[i] = 0,5;
                        mm[j] = 0,5;
                        nn[i] = number;
                        nn[i] = number;
                        int l = 0;
                        while (l < delvec.size())
                        {
                                if (delvec[l] == i || delvec[l] == j)
                                {
                                        delvec.erase(delvec.begin() + l);
                                }
                                else l++;
                        }
                }
        }
}

if (!delvec.empty())
{
        if (!infected.empty())
        {
                for (int i = 0; i < infected.size(); i++)
                if (!check(greyl, domain[i]))
                {
                        greyl.push_back(domain[i]);

                        for (int j = 0; j < Mk->cols; j++)
                        {
                                if (Mk->mas[i][j].check == 1 && !check(inf, host[j]))
                                        inf.push_back(host[j]);
                        }
                }
        }
        {
                deleter(delvec, Mk, &domain, &nn, &ss, &ff, &rr, &mm, &nn);

                ofstream out("observation matrix.txt");
```

```
                        writetf(out, Mk, host, domain, nn, ss, ff, rr, mm, nn);
                        out.close();
                }
        }
        ofstream file("Infected.txt");
        for (int i = 0; i < inf.size(); i++)
                file << inf[i] << endl;
        file.close();

        ofstream file2("new.txt");
        for (int i = 0; i < greyl.size(); i++)
                file2 << greyl[i] << endl;
        file2.close();
}

void processmatrix(double kof2, vector <string> infected, int numb)
{
        table Mk;
        vector <string> host;
        vector <string> domain;
        vector <int> Ng;
        vector <double> S;
        vector <int> F;
        vector <int> R;
        vector <double> M;
        vector <int> N;

        double res = 0;
        ifstream file("observation matrix.txt");

        vector<string> grey;

        if (!file.fail())
        {
                readff(file, &Mk, &host, &domain, &Ng, &S, &F, &R, &M, &N);
                file.close();

                vector <int> num;
                int i = 0;
```

```
while (i < domain.size() - 1)
{
	num.push_back(i);
	for (int j = i + 1; j < domain.size(); j++)
	{
		if (domain[i] == domain[j]) num.push_back(j);
	}
	if (num.size() == 1) num.clear(); else
	if (num.size() == 2) res = Braun_Blanquet(num, &Mk, Ng);
	else res = Koch(num, &Mk, Ng);

	if (res >= kof2) res = 1; else
	if (res >= (kof2 - 0.1) && res < kof2) res = 0.5; else res = 0;

	if (res == 0.5)
	{
		bool f1 = true, f2 = true;
		for (int j = 0; j < num.size(); j++)
		{
			if (F[num[j]] != 1) f1 = false;
			if (S[num[i]] != 1) f2 = false;
		}
		if (f1 == true && f2 == true) res = 1;
	}

	if (res == 1)
	{
		M[num[0]] = 1;
		F[num[0]] = 1;
		S[num[0]] = 1;
		N[num[0]] = numb;
		grey.push_back(domain[num[0]]);
		merger(num, &Mk, Ng, R);
		for (int i = 0; i < host.size(); i++)
		{
			if (Mk.mas[num[0]][i].check == 1)
			if (!check(infected, host[i]))
			{
				infected.push_back(host[i]);
```

```
                        }
                    }
                }
                else
                if (res == 0.5)
                {
                    F[num[0]] = 0;
                    int f2 = 0, f3 = 0;
                    for (int l = 0; l < num.size(); l++)
                    {
                        if (S[num[l]] == 0.5) f2++; else
                        {
                            f3 = num.size();
                            break;
                        }
                    }
                    if (f2 == num.size()) S[num[0]] = 0.5; else S[num[0]] = 0;
                    M[num[0]] = 0.5;
                    N[num[0]] = numb;
                    merger(num, &Mk, Ng, R);
                }

                if (res != 0) num.erase(num.begin());
                if (num.size() > 0)         deleter(num, &Mk, &domain, &Ng, &S, &F, &R, &M, &N);

                num.clear();
                i++;
            }

            ofstream out("observation matrix.txt");
            Mk.rows--;
            writetf(out, &Mk, host, domain, Ng, S, F, R, M, N);
            out.close();
            matrix_Braun_Blanquet(&Mk, host, domain, Ng, S, F, R, M, N, kof2, infected, grey, numb);

        }
}
```

Лістинг файлу searchEvasion.cpp

```
// searchEvasion.cpp: определяет точку входа для консольного приложения.
//
#include <string>
#include <iostream>
#include <fstream>
#include <pcap.h>
#include <winsock2.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <windows.h>


#define WINVER 0x0501
#define _WIN32_WINNT 0x0501
#define IPv6 28

#define ETHERMTU    1500
#define ETHER_ADDR_LEN     6
struct  ether_header {
   u_int8_t    ether_dhost[ETHER_ADDR_LEN];
   u_int8_t    ether_shost[ETHER_ADDR_LEN];
   u_int16_t   ether_type;
};
#define ETHER_HDRLEN       14

#ifndef ETHERTYPE_IP
#define ETHERTYPE_IP            0x0800 /* IP protocol */
#endif
#ifndef ETHERTYPE_ARP
#define ETHERTYPE_ARP                   0x0806 /* Addr. resolution protocol */
#endif
#ifndef ETHERTYPE_IPV6
#define ETHERTYPE_IPV6                  0x86dd
#endif

#define SIZE_UDP 8
```

```
#ifndef CLASS_IN
#define CLASS_IN                0x0001 /* Class IN */
#endif

/* 4 bytes IP address */
typedef struct ip_address{
    u_char byte1;
    u_char byte2;
    u_char byte3;
    u_char byte4;
}ip_address;

/* IPv4 header */
typedef struct ip_header{
    u_char  ver_ihl;        // Version (4 bits) + Internet header length (4 bits)
    u_char  tos;            // Type of service
    u_short tlen;           // Total length
    u_short identification; // Identification
    u_short flags_fo;       // Flags (3 bits) + Fragment offset (13 bits)
    u_char  ttl;            // Time to live
    u_char  proto;          // Protocol
    u_short crc;            // Header checksum
    ip_address  saddr;      // Source address
    ip_address  daddr;      // Destination address
    u_int   op_pad;         // Option + Padding
}ip_header;

/* UDP header*/
typedef struct udp_header{
    u_short sport;          // Source port
    u_short dport;          // Destination port
    u_short len;            // Datagram length
    u_short crc;            // Checksum
}udp_header;

typedef struct dns_header {
    WORD Xid;
    BYTE RecursionDesired : 1;
    BYTE Truncation : 1;
```

```
    BYTE Authoritative : 1;
    BYTE Opcode : 4;
    BYTE IsResponse : 1;
    BYTE ResponseCode : 4;
    BYTE CheckingDisabled : 1;
    BYTE AuthenticatedData : 1;
    BYTE Reserved : 1;
    BYTE RecursionAvailable : 1;
    WORD QuestionCount;
    WORD AnswerCount;
    WORD NameServerCount;
    WORD AdditionalCount;
};

//Constant sized fields of query structure
struct query
{
    WORD qtype;
    WORD qclass;
};

struct r_data
{
    unsigned short type;
    unsigned short _class;
            unsigned short ttl[2];
    unsigned short data_len;
};

struct res_record
{
    unsigned char *name;
    struct r_data *resource;
    unsigned char *rdata;
};

struct type_a{
            unsigned char *name;
            unsigned short ttl;
```

```
        struct ip_address ip;
};

struct other_type{
        unsigned char *name;
        unsigned short type;
        unsigned char *rdata;
        unsigned short data_len;
};

struct necessary_data{
        struct timeval ts;
        ip_address  daddr;     // Destination address
        u_int8_t    ether_shost[ETHER_ADDR_LEN];
        BYTE ResponseCode : 4;
        WORD AnswerCount;
        unsigned short count_a_type;
        struct type_a *a_type;
        unsigned short count_other_type;
        struct other_type *arr_other_answer;
        bool first_a_type;
        bool successful;
        unsigned short dns_len;
};

struct vector_signs{
        unsigned char *domain_name;
        ip_address  daddr;
        u_int8_t    ether_shost[ETHER_ADDR_LEN];
        unsigned short domain_len;
        unsigned short domain_count_unical_char;
        float domain_entropy;
        unsigned short ttl_mod;
        unsigned short ttl_med;
        unsigned short ttl_aver;
        unsigned short count_a_type;
        unsigned short count_ip_addr;
        unsigned long ser_distance_ip_addr;
        unsigned long ser_distance_ip_addr_plural;
```

```
        unsigned short count_un_ip_adr_plurals;
        unsigned long ser_distance_ip_addr_plurals;
        unsigned short count_domain_ip_address;
        bool bin_rarery_used_types;
        double max_entropy_rarery_types;
        unsigned int max_len_dns;
        bool succesful;
};

using namespace std;

// Declare helper functions
void PrintPacket(pcap_pkthdr * header, const u_char *data);
void PrintData(u_int startOctet, u_int endOctet, const u_char *data);
u_short get_ethertype_value(const u_char *data);
char * get_ethertype(const u_int16_t ethertype);

// static values
static u_int packetCount = 0;

// Defines
#define SIZE_ETHERNET 14

u_char* ReadName(unsigned char* reader,unsigned char* buffer,int* count);

double Log2(double n){
        return log(n)/log(2);
}

unsigned short CountUnicalChar(unsigned char* str){
        unsigned short count = 0;
        int strlength = strlen((const char*) str);
        for(int i=0; i<strlen((const char*) str); i++)
        {
                int n = 1;
                bool flag = true;
                for(int y=i-1; y>=0 && flag; y--){
                        if(str[i] == str[y])
          flag = false;
```

```
                }
                if(flag)
    {
                        count++;
    }
        }
        return count;
}

float entropy(unsigned char* str){
        int count = 0;
        int *countpovt;
        int strlength = strlen((const char*) str);
        for(int i=0; i<strlen((const char*) str); i++)
        {
                int n = 1;
                bool flag = true;
                for(int y=i-1; y>=0 && flag; y--){
                        if(str[i] == str[y])
         flag = false;
                }
                if(flag)
    {
                        for(int j = i+1; j < strlen((const char*) str); j++)
                                if(str[i] == str[j]) n++;
                        count++;
                        if(count==1) countpovt = new int[count];
                        else{
                                countpovt = (int *)realloc(countpovt , count*sizeof(int));
                        }
                                countpovt[count-1] = n;
    }
        }

        float result_entropy = 0;
        for(int i=0; i<count; i++){
                result_entropy+=(countpovt[i]/(float)strlength) * (Log2(strlength/(float)countpovt[i]));
        }
```

```
	return result_entropy;
}

unsigned short TTLmod(unsigned short *ttl,int count){
	if(count==1) return ttl[0];

	int count_el=0;
	unsigned short **arr;
	for(int i=0; i<count; i++){
		int n = 1;
		bool flag = true;
		for(int y=i-1; y>=0 && flag; y--){
			if(ttl[i] == ttl[y])
				flag = false;
		}
		if(flag)
		{
			for(int j = i+1; j < count; j++)
				if(ttl[i] == ttl[j]) n++;
			count_el++;
			if(count_el==1) {
				arr =  new unsigned short* [count_el];
				arr[0] = new unsigned short[2];
			}
			else{
				*arr = (unsigned short *)realloc(*arr , count_el*sizeof(unsigned short));
				arr[count_el-1] = new unsigned short[2];
			}
			arr[count_el-1][0] = ttl[i];
			arr[count_el-1][1] = n;
		}
	}

	for(int i=0; i<count_el; i++){
		for(int j=0; j<count_el-i-1; j++){
			if(arr[j][0]>arr[j+1][0]){
				unsigned short tmp;
				for(int z=0; z<2; z++){
					tmp = arr[j][z];
```

```
                                        arr[j][z] = arr[j+1][z];
                                        arr[j+1][z] = tmp;
                                }
                        }
                }
        }

        for(int i=0; i<count_el; i++){
                for(int j=0; j<count_el-i-1; j++){
                        if(arr[j][1]>arr[j+1][1]){
                                unsigned short tmp;
                                for(int z=0; z<2; z++){
                                        tmp = arr[j][z];
                                        arr[j][z] = arr[j+1][z];
                                        arr[j+1][z] = tmp;
                                }
                        }
                }
        }

        if(arr[0][1]==1){
                unsigned int summ=0;
                for(int i=0; i<count_el; i++){
                        summ += arr[i][0];
                }
                return summ/count_el;
        }
        return arr[0][0];
}

unsigned short TTLmed(unsigned short *ttl,int count){
        if(count==1) return ttl[0];

        for(int i=0; i<count; i++){
                for(int j=0; j<count-i-1; j++){
                        if(ttl[j]>ttl[j+1]){
                                unsigned short tmp = ttl[j];
                                ttl[j] = ttl[j+1];
                                ttl[j+1] = tmp;
```

```
                        }
                }
        }

        return ttl[count/2];
}

unsigned short TTLser(unsigned short *ttl,int count){
        if(count==1) return ttl[0];

        unsigned int result = 0;

        for(int i=0; i<count; i++){
                result += ttl[i];
        }

        return result/count;
}

unsigned long IPlenght(ip_address ip){
        unsigned long result;
        result = ip.byte4;
        result += ip.byte3*256;
        result += ip.byte2*65536;
        result += ip.byte1*16777216;
        return result;
}

unsigned long SerDistanceIPaddr(ip_address *unic_ip, int count){
        int count_elements_mas = 0;
        for(int i=count-1; i>0; i--){
                count_elements_mas += i;
        }

        unsigned long *mas_distance = new unsigned long[count_elements_mas];
        int counter = 0;
        for(int i=0; i<count-1; i++){
                for(int j=i+1; j<count; j++){
                        unsigned long ip1 = IPlenght(unic_ip[i]);
```

```
                        unsigned long ip2 = IPlenght(unic_ip[j]);
                        if(ip2>ip1) {
                                mas_distance[counter] = ip2 - ip1;
                        } else {
                                mas_distance[counter] = ip1 - ip2;
                        }
                        counter++;
                }
        }

        unsigned long result = 0;
        for(int i=0; i<counter; i++){
                result += mas_distance[i];
        }
        result = result / counter;
        return result;
}

void parseFile(string file, necessary_data* &arr_ness_data, unsigned int &count_ness_data){
        /*
   * Step 2 - Get a file name
   */

    /*
   * Step 3 - Create an char array to hold the error.
   */
   // Note: errbuf in pcap_open functions is assumed to be able to hold at least PCAP_ERRBUF_SIZE chars
   //      PCAP_ERRBUF_SIZE is defined as 256.
   char errbuff[PCAP_ERRBUF_SIZE];

   /*
   * Step 4 - Open the file and store result in pointer to pcap_t
   */
   // Use pcap_open_offline
   pcap_t * pcap = pcap_open_offline(file.c_str(), errbuff);

   /*
   * Step 5 - Create a header and a data object
   */
```

```
// Create a header object:
struct pcap_pkthdr *header;

// Create a character array using a u_char
// u_char is defined here:
// C:\Program Files (x86)\Microsoft SDKs\Windows\v7.0A\Include\WinSock2.h
// typedef unsigned char   u_char;
const u_char *data;

/*
* Step 6 - Loop through packets and print them to screen
*/
while (int returnValue = pcap_next_ex(pcap, &header, &data) >= 0)
{
                count_ness_data++;
                if(count_ness_data==1) arr_ness_data = new necessary_data[1];
                else {
                        arr_ness_data = (necessary_data *)realloc(arr_ness_data ,
count_ness_data*sizeof(necessary_data));
                }
    // I moved the code for printing the packet as hex values
    // to a function
                arr_ness_data[count_ness_data-1].ts = header->ts;

    // Unwrap ether_header (first 14 bytes)
    const struct ether_header *ethernet;
    ethernet = (struct ether_header*)(data);

                ip_header *ih;
                udp_header *uh;
                u_int ip_len;
                u_short sport,dport;

                ih = (ip_header *) (data + 14); //length of ethernet header


                /* retireve the position of the udp header */
                ip_len = (ih->ver_ihl & 0xf) * 4;
                uh = (udp_header *) ((u_char*)ih + ip_len);
```

```
		/* convert from network byte order to host byte order */
		sport = ntohs( uh->sport );
		dport = ntohs( uh->dport );

		arr_ness_data[count_ness_data-1].dns_len = ntohs( uh->len );

		dns_header* dns;

		dns = (dns_header*) ((u_char*)ih + ip_len + SIZE_UDP);


		for(int z=0; z<ETHER_ADDR_LEN; z++){
			arr_ness_data[count_ness_data-1].ether_shost[z] = ethernet->ether_shost[z];
		}

// Print EtherType
u_short ethertype = ntohs(ethernet->ether_type);

// Add two lines between packets
		arr_ness_data[count_ness_data-1].daddr = ih->daddr;

		arr_ness_data[count_ness_data-1].ResponseCode = dns->ResponseCode;

		arr_ness_data[count_ness_data-1].successful=false;
		int rcode = (u_int)dns->ResponseCode;
		if(rcode==0){
			arr_ness_data[count_ness_data-1].successful=true;
		}
		arr_ness_data[count_ness_data-1].AnswerCount = ntohs(dns->AnswerCount);

		query *q;
res_record *answers = new res_record [ntohs(dns->AnswerCount)]; //the replies from the DNS server
u_char *reader;
u_char *name = NULL;
u_char* buf = (u_char*) ((u_char*)ih + ip_len + SIZE_UDP);;
//move ahead of the dns header and the query field
reader = (u_char *) ((u_char *)dns + sizeof(dns_header));
//reading query
```

```
int stop = 0;

name = ReadName(reader, buf, &stop);
reader += strlen((const char*) name) + 1;
//cout << endl << name << endl;
q = (query*) (reader + 1);

//answers
reader = (u_char *) ((u_char *)dns + sizeof(dns_header) + (int) strlen((const char*) name) + 6 );
                delete [] name;
u_char* reader_t = reader;
                stop = 0;

                struct type_a *arr_types_a;
                unsigned short count_a_type=0;

                struct other_type *arr_other_types;
                unsigned short count_other_type=0;

                arr_ness_data[count_ness_data-1].first_a_type = true;

for(int i(0); i < ntohs(dns->AnswerCount); i++)
{
   answers[i].name = ReadName(reader, buf, &stop);
   reader = reader + stop;

                        if(strlen((const char*) answers[i].name)==0) {reader -= 1;
                        answers[i].name = answers[i-1].name;
                        }
                        float entr = entropy(answers[i].name);

                        answers[i].resource = (struct r_data*)(reader);
                        reader = reader + sizeof(struct r_data);
                        reader_t += sizeof(struct r_data);

                        if(ntohs(answers[i].resource->type) == 1) //if its an ipv4 address
                        {
                                count_a_type++;
                                if(count_a_type==1) arr_types_a = new type_a[1];
```

```
                                else {
                                        arr_types_a = (type_a *)realloc(arr_types_a ,
count_a_type*sizeof(type_a));
                                }
                                arr_types_a[count_a_type-1].name = answers[i].name;

                                answers[i].rdata = (unsigned char*)malloc(ntohs(answers[i].resource-
>data_len));

                                for(int j=0 ; j<ntohs(answers[i].resource->data_len) ; j++)
                                {
                                        answers[i].rdata[j]=reader[j];
                                }

                                ip_address *ip_adr =  (ip_address*) (reader);

                                arr_types_a[count_a_type-1].ip.byte1 = ip_adr->byte1;
                                arr_types_a[count_a_type-1].ip.byte2 = ip_adr->byte2;
                                arr_types_a[count_a_type-1].ip.byte3 = ip_adr->byte3;
                                arr_types_a[count_a_type-1].ip.byte4 = ip_adr->byte4;

                                arr_types_a[count_a_type-1].ttl = ntohs(answers[i].resource->ttl[1]);

                                answers[i].rdata[ntohs(answers[i].resource->data_len)] = '\0';

                                reader = reader + ntohs(answers[i].resource->data_len);
                        }
                        else
                        {
                                count_other_type++;
                                if(count_other_type==1) arr_other_types = new other_type[1];
                                else {
                                        arr_other_types = (other_type *)realloc(arr_other_types ,
count_other_type*sizeof(other_type));
                                }
                                arr_other_types[count_other_type-1].name =  answers[i].name;
                                arr_other_types[count_other_type-1].type = ntohs(answers[i].resource-
>type);
                                arr_other_types[count_other_type-1].data_len =
ntohs(answers[i].resource->data_len);
```

```
                        if(ntohs(answers[i].resource->type)!=IPv6){
                                answers[i].rdata = ReadName(reader,buf,&stop);
                                arr_other_types[count_other_type-1].rdata = answers[i].rdata;
                        } else {
                                answers[i].rdata = (unsigned char*)"NONE";
                                arr_other_types[count_other_type-1].rdata = answers[i].rdata;
                        }
                        reader = reader + ntohs(answers[i].resource->data_len);
                        if(i==0) arr_ness_data[count_ness_data-1].first_a_type = false;
                }
}

        arr_ness_data[count_ness_data-1].count_a_type = count_a_type;
        arr_ness_data[count_ness_data-1].count_other_type = count_other_type;

        arr_ness_data[count_ness_data-1].a_type = arr_types_a;
        arr_ness_data[count_ness_data-1].arr_other_answer = arr_other_types;

        //res_record* auth = new res_record[ntohs(dns->NameServerCount)];

        //reader -= 2;
        /*for(int i=0;i<ntohs(dns->NameServerCount);i++)
        {
                auth[i].name=ReadName(reader,buf,&stop);
                reader+=stop;
                cout << auth[i].name << endl;

                auth[i].resource=(struct r_data*)(reader);
                reader+=sizeof(struct r_data);

                auth[i].rdata=ReadName(reader,buf,&stop);
                cout << auth[i].rdata << endl;
                reader+=stop;
        }*/

        //res_record* addit = new res_record[ntohs(dns->AdditionalCount)];

                //read additional
```

```
                /*
                res_record* auth = new res_record[dns->NameServerCount / 256];
    reader_t -= 2;
    reader = reader_t;
    //read authorities
    for(int i=0;i<dns->NameServerCount / 256;i++)
    {
      auth[i].name=ReadName(reader,buf,&stop);
      reader+=stop;

      reader_t += strlen((const char*) auth[i].name);

      cout << auth[i].name << endl;

      auth[i].resource=(struct r_data*)(reader);
      reader+=sizeof(struct r_data);
      reader_t += sizeof(struct r_data);

      auth[i].rdata=ReadName(reader,buf,&stop);
      cout << auth[i].rdata << endl;
      reader_t += strlen((const char*) auth[i].rdata);
      reader+=stop;
    }
    delete [] auth;
                */
  }

}

vector_signs* forming_feature_vector(necessary_data* &arr_ness_data, unsigned int count_ness_data, int
&count_arr_vector){
        struct vector_signs *arr_vector;
        for(int i=0; i<count_ness_data; i++){

                unsigned char *dom_addr;
                if(arr_ness_data[i].count_a_type>0) {
                        dom_addr = arr_ness_data[i].a_type[0].name;
                } else if(arr_ness_data[i].count_other_type>0) {
                        dom_addr = arr_ness_data[i].arr_other_answer[0].name;
```

```
		}

		if(arr_ness_data[i].count_a_type==0) continue;

		bool flag = true;
		for(int y=i-1; y>=0 && flag; y--){
			if(arr_ness_data[y].AnswerCount==0) continue;
			unsigned char *tmp_dom_addr;
			if(arr_ness_data[y].count_a_type>0) {
				tmp_dom_addr = arr_ness_data[y].a_type[0].name;
			} else if(arr_ness_data[y].count_other_type>0) {
				tmp_dom_addr = arr_ness_data[y].arr_other_answer[0].name;
			}

			if(strcmp((const char*)dom_addr,(const char*)tmp_dom_addr) == 0) flag = false;
		}
		if(flag)
	{
			count_arr_vector++;
			if(count_arr_vector==1) arr_vector = new vector_signs[1];
			else {
				arr_vector = (vector_signs *)realloc(arr_vector ,
count_arr_vector*sizeof(vector_signs));
			}


	if((arr_ness_data[i].first_a_type==false)&&(arr_ness_data[i].arr_other_answer[0].type==5)&&(strc
mp((const char*)dom_addr,(const char*)arr_ness_data[i].arr_other_answer[0].rdata)==0)){
				arr_vector[count_arr_vector-1].domain_name =
arr_ness_data[i].arr_other_answer[0].name;
				arr_vector[count_arr_vector-1].daddr = arr_ness_data[i].daddr;
				arr_vector[count_arr_vector-1].domain_len = strlen((const
char*)arr_ness_data[i].arr_other_answer[0].name);
				arr_vector[count_arr_vector-1].domain_count_unical_char =
CountUnicalChar(arr_ness_data[i].arr_other_answer[0].name);
				arr_vector[count_arr_vector-1].domain_entropy =
entropy(arr_ness_data[i].arr_other_answer[0].name);

			} else {
				arr_vector[count_arr_vector-1].domain_name = dom_addr;
				arr_vector[count_arr_vector-1].daddr = arr_ness_data[i].daddr;
```

```
                arr_vector[count_arr_vector-1].domain_len = strlen((const
char*)dom_addr);
                arr_vector[count_arr_vector-1].domain_count_unical_char =
CountUnicalChar(dom_addr);
                arr_vector[count_arr_vector-1].domain_entropy = entropy(dom_addr);
            }

            long time_cur = arr_ness_data[i].ts.tv_sec;
            unsigned short cur_ttl = arr_ness_data[i].a_type[0].ttl;
            unsigned short *tmp_arr_ttl = new unsigned short[1];
            int count_arr_ttl = 1;
            tmp_arr_ttl[0] = cur_ttl;

            unsigned short max_count_a_type = arr_ness_data[i].count_a_type;
            int index_max_count = i;
            for(int z=i+1; z<count_ness_data; z++){
                if(strcmp((const char*)dom_addr,(const
char*)arr_ness_data[z].a_type[0].name)==0){
                    long tmp_time = arr_ness_data[z].ts.tv_sec;
                    if((tmp_time-time_cur)>cur_ttl){
                        time_cur = tmp_time;
                        count_arr_ttl++;
                        tmp_arr_ttl = (unsigned short *)realloc(tmp_arr_ttl ,
count_arr_ttl*sizeof(unsigned short));
                        tmp_arr_ttl[count_arr_ttl-1] =
arr_ness_data[z].a_type[0].ttl;
                    } else {
                        int difference_time_ttl = tmp_time-time_cur;
                        int diff_ttl = cur_ttl - arr_ness_data[z].a_type[0].ttl;
                        if(abs(difference_time_ttl-diff_ttl)>1){
                            count_arr_ttl++;
                            tmp_arr_ttl = (unsigned short *)realloc(tmp_arr_ttl
, count_arr_ttl*sizeof(unsigned short));
                            tmp_arr_ttl[count_arr_ttl-1] =
arr_ness_data[z].a_type[0].ttl;
                        }
                    }
                    if(arr_ness_data[z].count_a_type>max_count_a_type){
                        max_count_a_type = arr_ness_data[z].count_a_type;
                        index_max_count = z;
                    }
                }
```

```
                }

                arr_vector[count_arr_vector-1].ttl_mod = TTLmod(tmp_arr_ttl,count_arr_ttl);
                arr_vector[count_arr_vector-1].ttl_med = TTLmed(tmp_arr_ttl,count_arr_ttl);
                arr_vector[count_arr_vector-1].ttl_aver = TTLser(tmp_arr_ttl,count_arr_ttl);

                if(count_arr_ttl>0) delete [] tmp_arr_ttl;

                arr_vector[count_arr_vector-1].count_a_type = max_count_a_type;

                int count_unical_ip = 1;
                ip_address *unical_ip = new ip_address[1];

                        if(max_count_a_type == 1){
                        unical_ip[0] = arr_ness_data[i].a_type[0].ip;

                        for(int z=i+1; z<count_ness_data; z++){
                                if(strcmp((const char*)dom_addr,(const
char*)arr_ness_data[z].a_type[0].name)==0){
                                        bool add = true;
                                        for(int y=0; y<count_unical_ip; y++){
                                                if(IPlenght(unical_ip[y]) ==
IPlenght(arr_ness_data[z].a_type[0].ip)) {
                                                        add = false;
                                                        break;
                                                }
                                        }
                                        if(add==true){
                                                count_unical_ip++;
                                                unical_ip = (ip_address *)realloc(unical_ip ,
count_unical_ip*sizeof(ip_address));
                                                unical_ip[count_unical_ip-1] =
arr_ness_data[z].a_type[0].ip;
                                        }
                                }
                        }

                        arr_vector[count_arr_vector-1].count_ip_addr = count_unical_ip;
                        if(count_unical_ip==1){
                                arr_vector[count_arr_vector-1].ser_distance_ip_addr = 0;
```

```
                        } else {
                                arr_vector[count_arr_vector-1].ser_distance_ip_addr =
SerDistanceIPaddr(unical_ip, count_unical_ip);
                        }
                        arr_vector[count_arr_vector-1].ser_distance_ip_addr_plural = 0;
                        arr_vector[count_arr_vector-1].count_un_ip_adr_plurals = 0;
                        arr_vector[count_arr_vector-1].ser_distance_ip_addr_plurals = 0;

                } else {
                        arr_vector[count_arr_vector-1].count_ip_addr = 0;
                        arr_vector[count_arr_vector-1].ser_distance_ip_addr = 0;

                        int count_unical_plural_ip = 1;
                        ip_address *unical_plural_ip = new ip_address[1];
                        unical_plural_ip[0] = arr_ness_data[index_max_count].a_type[0].ip;
                        for(int z=1; z<arr_ness_data[index_max_count].count_a_type; z++){
                                bool add = true;
                                for(int y=0; y<count_unical_plural_ip; y++){
                                        if(IPlenght(unical_plural_ip[y]) ==
IPlenght(arr_ness_data[index_max_count].a_type[z].ip)) {
                                                add = false;
                                                break;
                                        }
                                }
                                if(add==true){
                                        count_unical_plural_ip++;
                                        unical_plural_ip = (ip_address *)realloc(unical_plural_ip ,
count_unical_plural_ip*sizeof(ip_address));
                                        unical_plural_ip[count_unical_plural_ip-1] =
arr_ness_data[index_max_count].a_type[z].ip;
                                }
                        }
                        if(count_unical_plural_ip==1){
                                arr_vector[count_arr_vector-1].ser_distance_ip_addr_plural = 0;
                        } else {
                                arr_vector[count_arr_vector-1].ser_distance_ip_addr_plural =
SerDistanceIPaddr(unical_plural_ip, count_unical_plural_ip);
                        }
                        delete [] unical_plural_ip;

                        unical_ip[0] = arr_ness_data[i].a_type[0].ip;
```

```
                    for(int z=i; z<count_ness_data; z++){
                        if(strcmp((const char*)dom_addr,(const
char*)arr_ness_data[z].a_type[0].name)==0){
                            for(int k=0; k<arr_ness_data[z].count_a_type; k++){
                                bool add = true;
                                for(int y=0; y<count_unical_ip; y++){
                                    if(IPlenght(unical_ip[y]) ==
IPlenght(arr_ness_data[z].a_type[k].ip)) {
                                        add=false;
                                        break;
                                    }
                                }
                                if(add==true){
                                    count_unical_ip++;
                                    unical_ip = (ip_address *)realloc(unical_ip ,
count_unical_ip*sizeof(ip_address));
                                    unical_ip[count_unical_ip-1] =
arr_ness_data[z].a_type[k].ip;
                                }
                            }
                        }
                    }
                    arr_vector[count_arr_vector-1].count_un_ip_adr_plurals =
count_unical_ip;

                    if(count_unical_ip==1){
                        arr_vector[count_arr_vector-1].ser_distance_ip_addr_plurals = 0;
                    } else {
                        arr_vector[count_arr_vector-1].ser_distance_ip_addr_plurals =
SerDistanceIPaddr(unical_ip, count_unical_ip);
                    }
                }

                /** count domain uses ip address **/
                unsigned short max_count = 0;

                for(int y=0; y<count_unical_ip; y++){
                    unsigned char **mas_not_domain_addres = new unsigned
char*[count_ness_data];
                    int count_not_domain_addres = 1;
                    mas_not_domain_addres[0] = dom_addr;
```

```
                int count_domain_to_ip = 0;
                for(int z=0; z<count_ness_data; z++){
                        if(arr_ness_data[z].count_a_type==0) continue;
                        bool provirka = true;
                        for(int k=0; k<count_not_domain_addres; k++){
                                if(strcmp((const char*)mas_not_domain_addres[k],(const
char*)arr_ness_data[z].a_type[0].name)==0) {
                                        provirka = false;
                                        break;
                                }
                        }

                        if(provirka==true){
                                for(int m=0; m<arr_ness_data[z].count_a_type; m++){
                                        if(IPlenght(arr_ness_data[z].a_type[m].ip) ==
IPlenght(unical_ip[y])){
                                                count_domain_to_ip++;
count_not_domain_addres++;

        mas_not_domain_addres[count_not_domain_addres-1] = arr_ness_data[z].a_type[0].name;
                                                break;
                                        }
                                }
                        }
                }
                if(max_count<count_domain_to_ip) max_count = count_domain_to_ip;
        }

        arr_vector[count_arr_vector-1].count_domain_ip_address = max_count+1;

        bool rar_used_types = false;
        for(int z=0; z<arr_ness_data[i].count_other_type; z++){
                unsigned short type = arr_ness_data[i].arr_other_answer[z].type;
                if((type==5) || (type==16) || (type==10) || (type==25)){
                        rar_used_types = true;
                        break;
                }
        }
        arr_vector[count_arr_vector-1].bin_rarery_used_types = rar_used_types;
```

```
                        float max_entropy = 0;
                        if(rar_used_types==true){
                                for(int z=0; z<arr_ness_data[i].count_other_type; z++){
                                        unsigned short type = arr_ness_data[i].arr_other_answer[z].type;
                                        if((type==5) || (type==16) || (type==10) || (type==25)){
                                                float entropy_other_type =
entropy(arr_ness_data[i].arr_other_answer[z].rdata);
                                                if(max_entropy<entropy_other_type) max_entropy =
entropy_other_type;
                                        }
                                }
                        }
                        arr_vector[count_arr_vector-1].max_entropy_rarery_types = max_entropy;

                        unsigned short max_data_len = 0;
                        for(int z=i; z<count_ness_data; z++){
                                if(strcmp((const char*)dom_addr,(const
char*)arr_ness_data[z].a_type[0].name)==0){
                                        if(max_data_len<arr_ness_data[z].dns_len) max_data_len =
arr_ness_data[z].dns_len;
                                }
                        }
                        arr_vector[count_arr_vector-1].max_len_dns = max_data_len;
                        arr_vector[count_arr_vector-1].succesful = arr_ness_data[i].successful;

                        for(int v=0; v<ETHER_ADDR_LEN; v++){
                                arr_vector[count_arr_vector-1].ether_shost[v] =
arr_ness_data[i].ether_shost[v];
                        }

    }
        }
        return arr_vector;
}

int calculate_centre_vectors(vector_signs *vector, int count);

void findInCluster(vector_signs *vector, int count);

typedef int (__stdcall  *f_mathLabClastering)();
```

```
int main(int argc, char *argv[])
{
	srand(time(NULL));
	char *file_path;
	/*
	if(argc<2) {
		//file_path = "D:\\Disk E\\Study\\3 course\\SPZ\\Analyser\\Analyser\\dump500.pcap";
		file_path = "gooddump.pcap";
	}
	else {
		file_path = argv[1];
	}
	*/

	string name;

	ifstream conf("conf.txt");
	getline(conf, name);
	conf.close();

	file_path = _strdup(name.c_str());

	struct necessary_data *arr_ness_data;
	unsigned int count_ness_data = 0;

	parseFile(file_path, arr_ness_data, count_ness_data);

	struct vector_signs *arr_vector;
	int count_arr_vector=0;

	arr_vector = forming_feature_vector(arr_ness_data,count_ness_data,count_arr_vector);

	cout<<"Count elements to vector: "<<count_arr_vector<<endl<<"---------------Vector params:----------
----"<<endl;
	for(int i=0; i<count_arr_vector; i++){
		printf("%s - IP: %d.%d.%d.%d - Dom len: %d - Unical char: %d - Dom entropy:
%5.3f\n",arr_vector[i].domain_name, arr_vector[i].daddr.byte1, arr_vector[i].daddr.byte2,
arr_vector[i].daddr.byte3, arr_vector[i].daddr.byte4, arr_vector[i].domain_len,
arr_vector[i].domain_count_unical_char, arr_vector[i].domain_entropy);
```

```
		cout<<"TTL mod: "<<arr_vector[i].ttl_mod<<" TTL med: "<<arr_vector[i].ttl_med<<" TTL
aver: "<<arr_vector[i].ttl_aver<<endl;

		cout<<"count_a_type: "<<arr_vector[i].count_a_type<<" count_ip_addr:
"<<arr_vector[i].count_ip_addr<<" ser_distance_ip_addr: "<<arr_vector[i].ser_distance_ip_addr<<endl;

		cout<<"ser_distance_ip_addr_plural: "<<arr_vector[i].ser_distance_ip_addr_plural<<"
count_un_ip_adr_plurals: "<<arr_vector[i].count_un_ip_adr_plurals<<" ser_distance_ip_addr_plurals:
"<<arr_vector[i].ser_distance_ip_addr_plurals<<endl;

		cout<<"count_domain_ip_address: "<<arr_vector[i].count_domain_ip_address<<"
bin_rarery_used_types: "<<arr_vector[i].bin_rarery_used_types<<endl;

		cout<<"max_entropy_rarery_types: "<<arr_vector[i].max_entropy_rarery_types<<"
max_len_dns: "<<arr_vector[i].max_len_dns<<" successful: "<<arr_vector[i].succesful<<endl;

		cout<<"-----------------------------------------------"<<endl;

	}


	calculate_centre_vectors(arr_vector, count_arr_vector);


	findInCluster(arr_vector,count_arr_vector);


	if(argc<2) {

		system("pause");

	}

}


int calculate_centre_vectors(vector_signs *vector, int count) {

   int i, j, k;

	double cluster_centre[1000][5];
return 0;

   double numerator, denominator;

   double t[1000][5];

	int degree_of_memb[10][10];

   for (i = 0; i < count; i++) {

      for (j = 0; j < 5; j++) {

				t[i][j] = pow(degree_of_memb[i][j], denominator);

      }

   }

   for (j = 0; j < 5; j++) {

      for (k = 0; k < 1; k++) {

         numerator = 0.0;

         denominator = 0.0;

         for (i = 0; i < count; i++) {

					numerator += t[i][j] * degree_of_memb[i][k];

            denominator += t[i][j];
```

```
        }
        cluster_centre[j][k] = numerator / denominator;
      }
   }
   return 0;
}

void findInCluster(vector_signs *vector, int count){
        FILE *file;
        file = fopen("result_vectors.dat","w");
        int arr_clasters[5];
        float arr_res_clasters[5];

        HINSTANCE hGetProcIDDLL = LoadLibrary("MatLabClustering.dll");
        if( ! hGetProcIDDLL )  {
                return EXIT_FAILTURE;
        }
f_mathLabClastering  mathLabClastering  =
(f_mathLabClastering)GetProcAddress(hGetProcIDDLL,"MatLabClustering.dll" );
        if( ! mathLabClastering  )  {
                return EXIT_FAILTURE;
        }

        printf("\n\n----------RESULT----------\n");
        for(int i=0; i<count; i++){
                int cluster_count=0;
                arr_res_clasters = mathLabClastering  (vector[i]);
                fprintf(file,"%s %.2x:%.2x:%.2x:%.2x:%.2x:%.2x %.2f %.2f %.2f %.2f
%.2f\n",vector[i].domain_name,
vector[i].ether_shost[0],vector[i].ether_shost[1],vector[i].ether_shost[2],vector[i].ether_shost[3],vector[i].
ether_shost[4],arr_res_clasters[0],arr_res_clasters[1],arr_res_clasters[2],arr_res_clasters[3],arr_res_claste
rs[4]);
                printf("\n");
        }
}

void PrintPacket(pcap_pkthdr * header, const u_char *data)
{
   // Print using printf. See printf reference:
   // http://www.cplusplus.com/reference/clibrary/cstdio/printf/
```

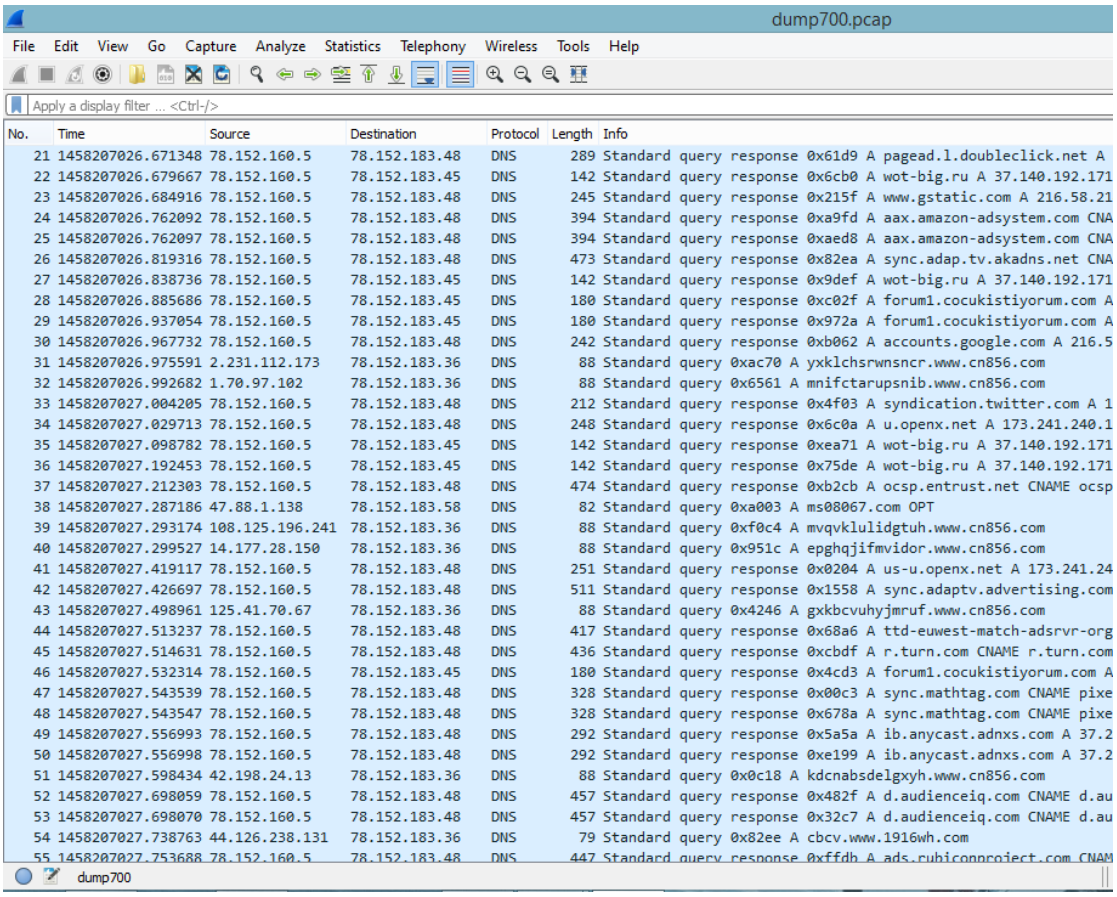
```
    // Show the packet number
    printf("Packet # %i\n", ++packetCount);

    // Show the size in bytes of the packet
    printf("Packet size: %ld bytes\n", header->len);

    // Show a warning if the length captured is different
    if (header->len != header->caplen)
        printf("Warning! Capture size different than packet size: %ld bytes\n", header->len);

    // Show Epoch Time
    printf("Epoch Time: %ld:%ld seconds\n", header->ts.tv_sec, header->ts.tv_usec);

    // loop through the packet and print it as hexidecimal representations of octets
    // We also have a function that does this similarly below: PrintData()
    for (u_int i=0; (i < header->caplen ) ; i++)
    {
        // Start printing on the next after every 16 octets
        if ( (i % 16) == 0) printf("\n");

        // Print each octet as hex (x), make sure there is always two characters (.2).
        printf("%.2x ", data[i]);
    }
    printf("\n\n");
}

void PrintData(u_int startOctet, u_int endOctet, const u_char *data)
{
    for (u_int i = startOctet; i <= endOctet; i++)
    {
        // Print each octet as hex (x), make sure there is always two characters (.2).
        printf("%.2x ", data[i]);
    }
}

u_short get_ethertype_value(const u_char *data)
{
    u_short ethertype = data[12] << 8;
    ethertype += data[13];
```

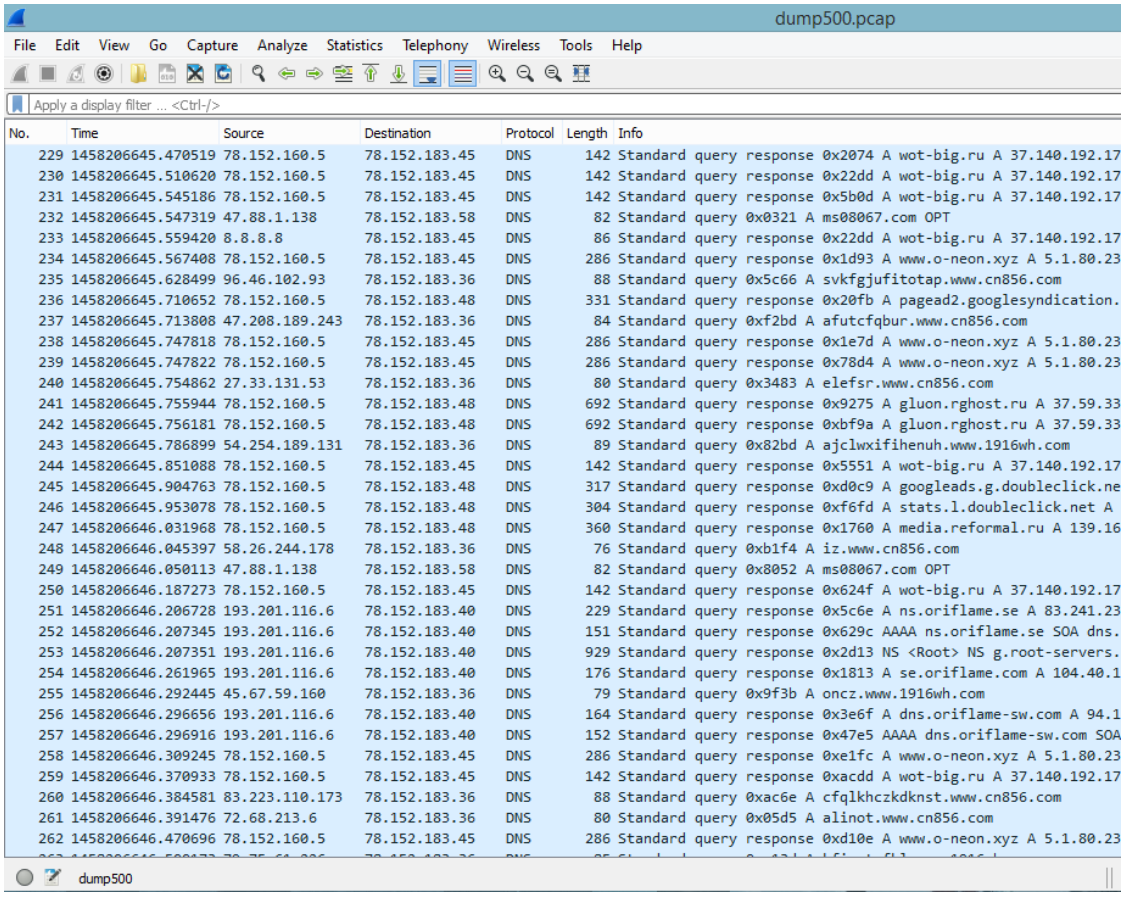
```
    return ethertype;
}

// Returns a string representation of the common Ethertype
char * get_ethertype(const u_int16_t ethertype)
{
    switch(ethertype)
    {
    case ETHERTYPE_IP:
        return "IPv4";
    case ETHERTYPE_IPV6:
        return "IPv6";
    case ETHERTYPE_ARP:
        return "ARP";
    default:
        return "Unknown";
    }
}

/*
 *
 * */
u_char* ReadName(unsigned char* reader,unsigned char* buffer,int* count)
{
    unsigned char *name;
    unsigned int p=0,jumped=0,offset;
    int i , j;

    *count = 1;
    name = (unsigned char*)malloc(256);

    name[0]='\0';

    //read the names in 3www6google3com format
    while(*reader!=0)
    {
        if(*reader>=192)
        {
            offset = (*reader)*256 + *(reader+1) - 49152; //49152 = 11000000 00000000 ;)
```

```
            reader = buffer + offset - 1;
            jumped = 1; //we have jumped to another location so counting wont go up!
        }
        else
        {
            name[p++]=*reader;
        }

        reader = reader+1;

        if(jumped==0)
        {
            *count = *count + 1; //if we havent jumped to another location then we can count up
        }
    }

    name[p]='\0'; //string complete
    if(jumped==1)
    {
        *count = *count + 1; //number of steps we actually moved forward in the packet
    }

    //now convert 3www6google3com0 to www.google.com
    for(i=0;i<(int)strlen((const char*)name);i++)
    {
        p=name[i];
        for(j=0;j<(int)p;j++)
        {
            name[i]=name[i+1];
            i=i+1;
        }
        name[i]='.';
    }
    name[i-1]='\0'; //remove the last dot
    return name;
}
```

ДОДАТОК Б

Приклад візуалізації DNS-трафіка, зібраного утилітою Wireshark





ДОДАТОК В

Акти впровадження

«Затверджую»
Директор ТОВ «ІТТ»
_____ В.С. Сімогук
_____ 2016 р.

АКТ

про впровадження результатів дисертаційної роботи
Бобровнікової Кіри Юліївни
«Інформаційна технологія виявлення бот-мереж у корпоративних мережах
на основі аналізу DNS-трафіка»

Комісія в складі:

Технічний директор	Веремеєнко В.А.
Начальник відділу продажу комп'ютерної техніки	Іванова В.О.
Адміністратор	Ладунець В.І.

склала акт про впровадження результатів дисертаційної роботи асистента кафедри системного програмування Хмельницького національного університету Бобровнікової К.Ю. на «ІТТ», в тому, що вона проводила роботу по впровадженню системи виявлення бот-мереж в корпоративних мережах.

В процесі вирішення науково-практичної задачі підвищення достовірності виявлення нових бот-мереж в корпоративних мережах, яка ґрунтується на використанні апарату системного аналізу, теорій множин та штучного інтелекту, зокрема нечіткої логіки, Бобровніковою К.Ю. було особисто отримано і використано на «ІТТ» такі результати:

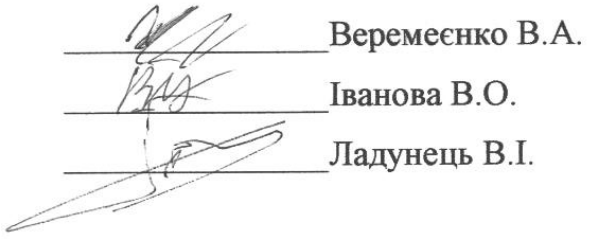
1) метод виявлення бот-мереж в корпоративних мережах на основі їх групової активності в DNS-трафіку;

2) метод виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS, суть якого полягає у здійсненні висновку щодо наявності використання таких технологій ухилення із застосуванням нечіткої кластеризації з частковим навчанням;

3) проведені експериментальні дослідження за участі Бобровнікової К.Ю. надали можливість перевірити ефективність програмного засобу, розробленого на базі методу виявлення бот-мереж в корпоративних мережах на основі їх групової активності в DNS-трафіку та методу виявлення бот-мереж, які застосовують технології ухилення від виявлення на основі DNS;

4) експериментальні дослідження показали, що розроблені методи виявлення бот-мереж в корпоративних мережах дозволяють виявляти існуючі та нові бот-мережі з високим рівнем достовірності.

Отримані результати дозволили підвищити достовірність виявлення бот-мереж на 8-22% в порівнянні з існуючими антивірусними програмними засобами.



Веремеєнко В.А.

Іванова В.О.

Ладунець В.І.

«Затверджую»
Директор технічного управління
ДП «Новатор»
Тітов Ю.О.
«___» _____ 2016 р.

АКТ

про впровадження результатів дисертаційної роботи

Бобровнікової Кіри Юліївни

«Інформаційна технологія виявлення бот-мереж у корпоративних мережах

на основі аналізу DNS-трафіка»

Результати дисертаційної роботи асистента кафедри системного програмування Хмельницького національного університету Бобровнікової К.Ю. впроваджені на Державному підприємстві «Новатор» у відділі автоматизованих систем управління.

В процесі впровадження і розробки інформаційної технології виявлення бот-мереж у корпоративних мережах на основі аналізу DNS-трафіка були використані на ДП «Новатор» такі результати, які одержані Бобровніковою К.Ю. особисто:

1) інформаційна технологія виявлення бот-мереж у корпоративних мережах на основі аналізу DNS-трафіка, яка включає модель процесу та методи виявлення бот-мереж в мережах і ґрунтується на властивості групової активності бот-мереж в DNS-трафіку та застосовує нечітку кластеризацію з частковим навчанням;

2) програмне забезпечення для виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка, яке дозволяє здійснювати пошук відомих та нових бот-мереж та підвищує достовірність виявлення.

Отримані результати дозволили підвищити достовірність виявлення бот-мереж. Результати роботи використано у відділі автоматизованих систем управління для організації захисту інформації в сегменті корпоративної мережі.

Цей акт не є підставою для фінансових розрахунків.

Начальник відділу
автоматизованих систем управління Шилов В.П.

«Затверджую»
Проректор з науково-педагогічної
роботи, д.е.н., професор
_____Войнаренко М.П.
«5» березня 2016 р.

АКТ

про впровадження в навчальний процес Хмельницького національного університету результатів дисертаційної роботи аспірантки кафедри системного програмування Бобровнікової Кіри Юліївни
«Інформаційна технологія виявлення бот-мереж у корпоративних мережах на основі аналізу DNS-трафіка»

Ми, комісія в складі: декана факультету програмування та комп'ютерних і телекомунікаційних систем к.т.н., доцента Савенка О.С., завідувача кафедри системного програмування, д.т.н., професора Поморової О.В., к.т.н., доцента кафедри системного програмування Медзатого Д.М. склала акт про те, що результати дисертаційної роботи Бобровнікової К.Ю. впроваджені та використовуються в навчальному процесі на кафедрі системного програмування для спеціальності 8.05010202 «Системне програмування» напряму підготовки 6.050102 «Комп'ютерна інженерія», зокрема в курсах «Програмування комп'ютерних мереж», «Системи штучного інтелекту», «Технічна діагностика і надійність комп'ютерних пристроїв та систем» та «Інженерія програмного забезпечення».

При викладанні цих дисциплін автором використовувалися наступні матеріали досліджень, отримані ним особисто:

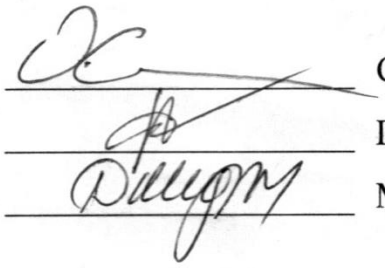
1) модель процесу виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка, яка ґрунтується на використанні апарату системного аналізу, теорій множин та штучного інтелекту, зокрема нечіткої логіки. На основі цієї моделі було розроблено інформаційну технологію, яка

надала можливість здійснювати виявлення нових невідомих ботів бот-мереж в корпоративних мережах;

2) кластерний аналіз, а саме нечітка кластеризація векторів ознак, вилучених з корисного навантаження DNS-повідомлень, з частковим навчанням, що дозволяє здійснити висновок про використання бот-мережами технологій ухилення від виявлення на основі DNS;

3) програмне забезпечення для виявлення бот-мереж в корпоративних мережах на основі аналізу DNS-трафіка, яке дозволило підвищити достовірність діагностування.

Отримані матеріали досліджень дозволили розробити лабораторні практикуми з використанням систем нечіткої логіки для виявлення бот-мереж в мережах.



Савенко О.С.

Поморова О.В.

Медзатий Д.М.