КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ РАДІОФІЗИКИ, ЕЛЕКТРОНІКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ

Кафедра комп'ютерної інженерії

До захисту допущено: «На правах рукопису»
Завідувач кафедри _____ Юрій Бойко
« _ » _____ 2023 р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«СИСТЕМА ВИЯВЛЕННЯ ПОРУШЕНЬ ПОЛІТИК КОРИСТУВАННЯ МЕРЕЖЕЮ "NETWORK POLICY VIOLATION DETECTION SYSTEMS"»

Виконав:
студент 2-го курсу магістратури
денної форми навчання
спеціальності 123 Комп'ютерна інженерія
ОНП «_____»
Андрій Козирський _____

Науковий керівник:
кандидат технічних наук, асистент
Мар'яновський Віталій Анатолійович _____

Рецензент:
_____

Засвідчую, що у цій магістерській роботі
немає запозичень з праць інших авторів без
відповідних посилань
Студент _____

Робота допущена до захисту в ЕК рішенням кафедри _____
від «__»_____ 2023 р., протокол № __.

Завідувач кафедри _____,
кандидат фізико-математичних наук, доцент
Бойко Юрій Володимирович

(підпис)

Київ – 2023

РЕФЕРАТ

Звіт з науково-виробничої практики за об’ємом складає 97 сторінок, містить 76 рисунків, 8 додатків, використано 25 інформаційних джерел.

Об’єктом роботи є створення засобу, для визначення порушень загальних політик користування мережею, а саме визначення використання VPN сервісів, Майнингу, Torrent сервісів, TOR сервісу та протоколів шифрування DNS. Предметом роботи є розроблення засобу, для пасивного аналізу мережевої активності у реальному часі, що дозволяє визначити порушення перелічених вище політик користування мережею.

Метою роботи є дослідження особливостей мережевого трафіку, що порушує політики користування мережею. Визначення особливостей роботи сервісів порушників політики мережі, що дозволяють виявити їх. Створення засобу для аналізу та визначення використання заборонених сервісів.

Інструменти розробки: засіб аналізу мережевої активності з відкритим вихідним кодом Suricata, безкоштовне середовище розробки Visual Studio Code, скриптова мова програмування Lua.

Актуальність роботи полягає у тому, що відбувається аналіз мережевої активності користувачів використовуючи інформацію, що передається у відкритому вигляді.

Результатом роботи: розглянуто особливості мережевого трафіку порушників політик користування мережі, розробка засобу, що дозволяє визначати порушення політик користування мережею шляхом пасивного аналізу мережевого трафіку.

ЗМІСТ

РЕФЕРАТ 2

ЗМІСТ 3

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 6

ВСТУП 7

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА 9

1.1. Мережевий трафік 9

1.1.1. Загальний огляд моделі TCP/IP 9

1.1.2. Особливості роботи протоколу TLS 10

1.2. Порушення політик користування мережею 12

1.2.1. Майнинг трафік 12

1.2.2. Torrent трафік 15

1.2.3. Трафік VPN сервісів 16

1.2.4. TOR трафік 21

1.2.5. Протоколи шифрування DNS 22

1.3. Suricata 23

1.3.1. Переваги Suricata 23

1.3.2. Suricata конфігурація 24

1.3.3. Suricata датасети 24

1.3.4. Suricata IP Репутаційні файли 25

1.3.5. Suricata правила 26

1.3.5.4. Lua скрипти 31

1.3.6. Suricata output logging 32

РОЗДІЛ 2. ПРАКТИЧНА ЧАСТИНА 35

2.1. Розробка правил виявлення доступу до відомих сервісів 35

2.1.1. Виявлення за доменними іменами DNS запитів 35

2.1.2. Виявлення за відомими IP адресами 46

2.1.3. Результати виявлення доступу до відомих сервісів 57

2.2. Детектування VPN трафіку 57

2.2.1. Детектування за загальновідомими IP адресами і DNS іменами 58

2.2.2. Детектування за сигнатурами протоколу 58

2.3. Детектування шифрованого DNS трафіку 65

2.4. Детектування TOR трафіку 67

2.5. Детектування Майнинг трафіку 67

2.5.1. Детектування за відомими IP адресами та DNS іменами 67

2.5.2. Детектування Bitcoin протоколу 68

2.6. Детектування Torrent трафіку 71

2.7. Зберігання мережевої активності TLS протоколу 73

РОЗДІЛ 3. ПРАКТИЧНЕ ВПРОВАДЖЕННЯ 76

3.1. Конфігурація та запуск Suricata 76
3.2. Розгляд результату логування 76
3.3. Розгляд продуктивності 77
3.3.1. Статистика завантаженості мережі 77
3.3.2. Статистика завантаженості процесору 78
3.3.3. Статистика використання оперативної пам’яті 79
3.4. Висновки практичного впровадження 80
ВИСНОВКИ 81
ПЕРЕЛІК ПОСИЛАНЬ 82
ДОДАТКИ 85
Додаток А 85
Додаток Б 87
Додаток В 88
Додаток Г 89
Додаток Д 91
Додаток Е 92
Додаток Ж 94
Додаток З 97

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IDS – (Intrusion Detection System) системи виявлення вторгнень, програмне і/або апаратне забезпечення, що використовується для виявлення вторгнень до мережі, шляхом визначення підозрілого трафіку.

NSM – (Network Security Monitor) системи забезпечення безпеки мережі, програмне і/або апаратне забезпечення, для моніторингу безпеки мережі шляхом відстеження мережевої активності.

IPS – (Intrusion Prevention System) системи запобігання вторгнень, програмне і/або апаратне забезпечення, що використовується для виявлення вторгнень до мережі та блокування джерел вторгнення, шляхом визначення підозрілого трафіку.

CPU – (Central Processing Unit) центральний процесор комп’ютера.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Зі створенням алгоритму Діффі-Геллмана з’явилася можливість забезпечення зашифрованої передачі даних без попередніх домовленостей. Розвиток даних ідей знайшов себе для забезпечення шифрування передачі даних в мережі, у 1995 році був опублікованим стандарт SSL 2.0, а вже у 1999 році вигляді відкритого стандарту вийшов протокол TLS 1.0.

На сьогоднішній час спостерігається тенденція розвитку інтернет мережі у напрямку забезпечення анонімності та конфіденційності, що спостерігається наприклад на прикладі збільшенні частки зашифрованого веб трафіку, та збільшенні запитів підтвердження дійсності мережевих сертифікатів збіраних сервісом Lets Encrypt [1.].

В той же час, спостерігається постійне покращення мережевих протоколів, та поведінки мережевих сервісів. Для прикладу протокол Wireguard, що був впроваджений у 2020 році, впровадження протоколу QUIC у якості стандарту RFC 9000 у травні 2021 [2.], та подальше впровадження даного протоколу, у вигляді HTTP/3, DNS over QUIC.

Актуальність роботи та підстави для її виконання. Ідея вільного користування мерею, що покраюється шляхом покращення анонімності в мережі може бути більш привабливою для кінцевих користувачів мережі, однак постає проблемою для власників корпоративних мереж та приватних мереж. Користувачі даних мереж можуть використовувати мережу для завантаження заборонених/піратських файлів, використовувати мережу, або навіть корпоративне обладнання, задля корисних цілей, як у прикладі з майнингом трафіку, а використання засобів анонімізації, такі як використання шифрованого DNS,

мережі TOR або сервісів VPN, можуть допомогти користувачем обійти ці та інші політики мережі, що можуть бути з тих та інших причин власниками мережі.

Враховуючи збільшення частки анонімізації та конфіденційності мережі, та впровадження нових протоколів, потребує постійного їх дослідження, визначення нових методів детектування трафіку та найактуальніших засобів задля виявлення порушень політик мережі.

Мета й завдання роботи. Дослідження особливостей мережевого трафіку на відповідність загальним політикам користування мережею. Створення засобу, що визначає та зберігає порушення політик користування мережею використовуючи пасивний аналіз мережевого трафіку.

Об’єкт, методи й засоби розроблення. Об’єктом розроблення засобу, що дозволяє визначати порушення політик у мережі.

Перед розробкою засобу є необхідним проаналізувати особливості мережевого трафіку, що відповідає порушенням політик користування мережею.

В якості засобу, для аналізу мережевого трафіку було обрано IDS/NSM Suricata, що є безкоштовним програмним забезпеченням, що поширюється з відкритим вихідним кодом, та є розроблена компанією OISF.

Можливі сфери застосування. Джерело матеріалів, що містить посилання на офіційні джерела інформації, та деталі роботи протоколів огляд програмного забезпечення, що використовується. Практична частина частково або повністю може використовуватися у приватних та корпоративних мережах, задля визначення порушень політик мережі.

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА

1.1. Мережевий трафік

Для передачі даних в мережі, до них необхідно додати додаткову інформацію, що є необхідною для пошуку оптимального маршруту надсилання даних, забезпечення цілісності переданих даних, їх шифрування, для можливості ідентифікації отримувачем джерела даних та їх природу.

Оскільки в комп'ютерній мережі спілкуються безліч різних машин, виникла проблема стандартизації, передачі даних при якій різні машини, повинні мати можливість приймати та надсилати дані в одному форматі. Було вирішено, що для забезпечення кожної з функцій необхідно додавати додаткові дані в чітко визначеному форматі визначеному протоколом. Також для можливості правильної інкапсуляції протоколи повинні додаватися в визначеному порядку.

1.1.1. Загальний огляд моделі TCP/IP

TCP/IP - це еталонна модель, що описує принцип роботи сучасної мережі виступає синонімом для "Інтернет технологій".[3.]

TCP/IP об'єднує в собі всі мережеві протоколи та розподіляє їх на 4 рівні. Кожен з рівнів забезпечує за виконання певної функції у процесі передачі даних. Розглянемо призначення кожного із рівнів детальніше:

1. Канальний рівень (PDU – фрейм) – протоколи даного рівня відповідають за передачу даних між пристроями в локальній мережі. Забезпечують надійну передачу даних шляхом детектування/запобігання помилок при передачі. Визначає середовище передачі даних. До протоколів даного рівня належать протоколи Ethernet, PPP, HDLC та інші.

2. Мережевий рівень (PDU – пакет) – протоколи даного рівня відповідають за знаходження ефективного маршруту передачі даних у мережі, та доставку пакетів в мережі, але не забезпечує надійність переданих даних. До протоколів даного рівня належать протоколи IPv4, IPv6 та інші.
3. Транспортний рівень (PDU – сегмент) – протоколи даного рівня відповідають за визначення додатку локального комп'ютера, що отримує/надсилає дані (шляхом присвоєння додаткам певного унікального числа - порту), сегментацію (процес розбиття даних на частини з певною максимальною величиною), десегментацію (процес об'єднання частин даних), перевірку коректності переданих даних. Протоколами даного рівня є TCP, UDP.
4. Прикладний рівень (PDU – дані) – протоколи даного рівня забезпечують обробку та представлення даних, взаємодію з кінцевим користувачем та всі інші застосунки, що використовують Транспортний рівень для надсилання даних.[4.] Наприклад:
    a. Протоколи передачі даних: Http, Ftp, Tftp та інші
    b. Протоколи встановлення сесій та шифрування трафіку Tls, Ssl
    c. Протоколи роботи з поштою Imap, Pop3, Smtp та інші

Для надсилання пакетів – дані інкапсулюються протоколами від верхнього до нижнього рівнів.

1.1.2. Особливості роботи протоколу TLS

TLS – це протокол прикладного рівня моделі TCP/IP. Використовується для забезпечення шифрування (це перетворення даних, з метою приховування інформації), аутентифікацію (перевірка авторства переданої інформації), цілісність (виявлення підміни інформації підробкою).

1.1.2.1. Принцип роботи TLS

Першим етапом роботи протоколу є процес TLS рукостискання за допомогою якого встановлюються усі необхідні коректної роботи параметри, такі як версія протоколу, що буде використовуватися, криптографічні алгоритми, що будуть використовуватися для шифрування.

Під час TLS рукостискання отримавши ланцюжок сертифікатів користувач впевнюється, що звернення відбувається власне до необхідного нам сервісу, тим самим забезпечується аутентифікаця.

Після завершення TLS рукостискання, розпочинається передача шифрованих даних. До шифрованих даних, що передаються додається їхнє хеш значення (незворотне, однозначне перетворення даних довільної довжини в рядок фіксованої довжини), за допомогою якого забезпечується цілісність переданих даних (при зміні даних хеш значення не відповідає дійсності).

1.1.2.2. TLS Client Hello

Першим повідомленням TLS рукостисканнє є звернення клієнта до сервера. В даному повідомленні клієнт повинен вказати початкові дані необхідні для встановлення захищеного з’єднання. Клієнт повідомляє версію протоколу, що використовується, список підтримуваних алгоритмів шифрування, підтримуванні методи стиснення даних, ідентифікатор сесії та інші.

Однак, оскільки на одному сервері можуть бути декілька сервісів одночасно, кожен з яких може мати власні налаштування клієнт повинен вказати назву сервісу до якого він звертається. Тому в Client Hello передається поле server name identifier, що зберігає в собі ідентифікатор сервісу, що зберігається у форматі схожому на доменне ім’я сервісу.

1.1.2.3. TLS висновки по протоколу

TLS протокол використовується для шифрування трафіку, однак в процесі обміну інформацією необхідною для шифрування є можливість визначення сервіс до якого відбувається звернення.

1.2. Порушення політик користування мережею

1.2.1. Майнинг трафік

Протягом останніх років значно зріс попит на криптовалюти, особливо це стосується майнингу криптовалют трафік якого зріс за останні роки. Майнінг не використовує клієнт – серверну архітектуру натомість будується peer-to-peer мережа вузлів, що спілкується між собою протоколами, що зачасту є унікальними для криптовалюти або групи криптовалют, що зазвичай інкапсулюються протоколом транспортного рівня TCP або UDP. До того ж в peer-to-peer мережах кожен вузол виступає в ролі і клієнта і сервера одночасно, тобто кожен вузол може ініціювати з’єднання з кожним вузлом в мережі, що унеможливлює визначення за зверненням до сервісу.

1.2.1.1. Bitcoin протокол

Розглянемо характерні риси майнінг трафіку на прикладі роботи Bitcoin протоколу. Bitcoin протокол інкапсулюється протоколом TCP для отримання повідмолень зазвичай використовується порт 8333 дещо рідше 3333, однак порт можна і змінити в налаштуваннях системи.

Розглянемо загальний формат Bitcoin повідомлення складається з 5 полів, як зображено на зображенні 1.2.1.1.1. [5.]

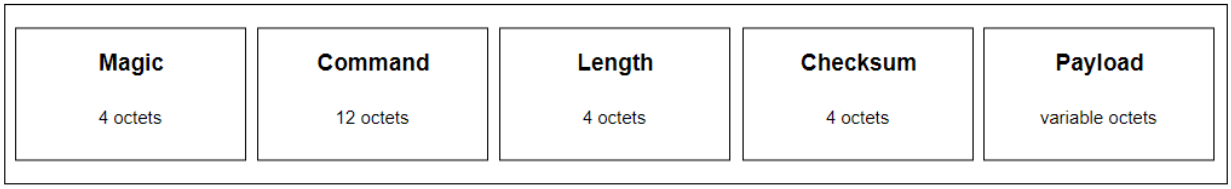

Рис. 1.2.1.1.1 Структура Bitcoin повідомлення

Розглянемо детальніше кожне з полів:

1. Magic (розмір поля 4 октети) – магічне значення, що залежить від джерела мережі, дане поле може приймати 5 значень:
   - F9 BE B4 FE
   - 0A 03 CF 40
   - 0B 11 09 07
   - FA BF B5 DA
   - F9 BE B4 D9
2. Command (розмір поля 12 октетів) – вказує на тип Bitcoin повідомлення що надсилається.

   Актуальні версії команд можна переглянути в вихідному коді [6.]: version, verack, addr, addrv2, sendaddrv2, inv, getdata, merkleblock, getblocks, getheaders, tx, headers, block, getaddr, mempool, ping, pong, notfound, filterload, filteradd, filterclear, sendheaders, feefilter, sendcmpct, cmpctblock, getblocktxn, blocktxn, getcfilters, cfilter, getcfheaders, cfheaders, getcfcheckpt, cfcheckpt, wtxidrelay, sendtxrcncl.

   В полі command зберігається назви команди в кодуванні utf-8, при цьому вільні байти будуть обнулені, наприклад для команди tx значення поля буде – 74 78 00 00 00 00 00 00 00 00 00 00.

3. Length (розмір поля 4 октети) – зберігає довжину поля Payload
4. Checksum (розмір поля 4 октети) – зберігає хеш суму поля Payload, для перевірки цілісності

5. Payload (розмір визначається значенням поля Length) – містить в собі дані, визначені типом повідомлення, записаним в полі Command

Розглянемо первинний пошук вузлів в мережі, що використовує протокол Bitcoin. Всього існує 3 варіанти отримання адрес інших вузлів[6.] в Bitcoin мережі:

- Використання протоколу Bitcoin типу addr, що повертає адреси інших вузлів в мережі. Якщо це перший запуск вузла він може звернутися до завчасно визначених вузлів, IP адреси яких є загальновідомими такі вузли називаються “seed nodes”. Або можливо власноруч добавити IP адресу вузла вказавши параметром “addnode”
- Використання протоколу DNS для отримання потенційних вузлів, що будуть використовуватися як початкові. Даний метод є методом пошуку початкових вузлів за замовчуванням починаючи з версії протоколу v0.6.X
- Використання протоколу IRC, однак даний протокол не вважається протоколом для пошуку початкового вузла починаючи з версії 0.6.X, а починаючи з версії 0.8.2 підтримка протоколу IRC взагалі припинилася

1.2.1.2. Майнинг висновки

Розглянувши характерні риси на прикладі роботи протоколу Bitcoin можна виявити загальні характеристики за якими можна спробувати ідентифікувати даний трафік:

- Виявлення DNS запитів з доменними іменами загальновідомих вузлів
- Виявлення надсилання повідомлень на сервери з IP адресами загальновідомих вузлів
- Спроба ідентифікації мережевого трафіку за полями протоколів

1.2.2. Torrent трафік

В основі технологій Torrent є використання P2P мережі задля поширення файлів в мережі, сама по собі ідея не є шкідливою. Однак на превеликий жаль, застосування технології знайшло себе в пошрені заборонених файлів, та файлів на які поширюються авторські права. Тим самим використання Torrent технологій часно є небажаним, та вважається за порушення политики користування мережею.

Першим та найпопулярнішим Torrent протоколом є протокол BitTorrent. Розглянувши його специфікацію [7.], розглянемо детальніше Handshake повідомленні.

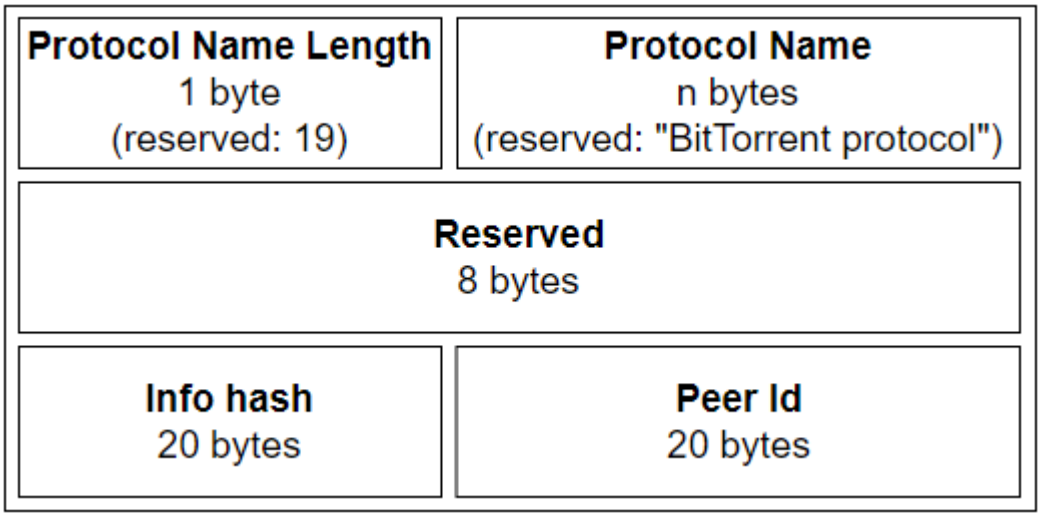

Рис. 1.2.2.1. Структура BitTorrent Handshake повідомлення

Handshake повідомлення BitTorrent протоколу містить такі поля:

- Protocol Name Length (1 байт) – містить у собі довжину наступного поля, за замовчуванням дане поле має значення 19, протокол був створений у 2001 році. Дане поле мало забезпечувати гнучкість протоколу у випадку зміни його логіки, роботи, однак поле залишилося незмінним

- Protocol Name (розмір зазначений в попередньому полі) – містить у собі значення імені протоколу, значення поля за замовчуванням utf-8 закодована строка “BitTorrent protocol”, та не змінилося з часу створення протоколу
- Reserved (8 байтів) – на момент створення даного поля, воно було пустим, однак на даний момент використовується для передачі додаткових параметрів що були додані в процесі історичного розвитку протоколу
- Info hash (20 байтів) – значення SHA1 хеш функції, файлу значення ключа даних файлу.
- Peer Id (20 байтів) – строка, що використовується для ідентифікації вузлів в мережі.

Підсумки протоколу BitTorrent:

- Протокол передається у незашифрованому вигляді, тому найкращим рішенням детектування BitTorrent трафіку є детектування за сигнатурою протоколу.
- В якості забезпечення шифрування Torrent трафіку, зазвичай використовується VPN сервіси.

1.2.3. Трафік VPN сервісів

VPN (скорочення від англ. virtual private network – віртуальна приватна мережа) – загальна назва для технологій, що використовуються для створення захищеного каналу між вузлами, що поєднує вузли в одну внутрішню мережу. VPN працює використовуючи клієнт серверну архітектуру.

Використання VPN дозволяє в надійному зашифрованому вигляді надсилати трафік з приватної мережі, що знаходиться в іншому географічно віддаленому місці. Хоч одним з основних призначень VPN забезпечення надійного, зашифрованого доступу до локальної мережі (наприклад, доступ до захищеної

корпоративної мережі при роботі з дому), VPN отримав поширення застосування для приховування мережевої активності та розташування користувачів. Що може використовуватися для порушення політик користування мережею.

Задля функціонування VPN є необхідним відкриття безпечного тунелю передачі, для цього використовуються протоколи:

1. Протокол PPTP

PPTP (Point-to-Point Tunneling Protocol) – протокол, що дозволяє створити тунель. Протокол використовує протокол TCP для передачі даних. Для ініціалізації з'єднання використовується порт 1723. Даний проткол інкапсулюється в TCP. Розроблений в 1999 році компанією Cisco зі специфікацією RFC 2637 [8.]. На даний момент даний протокол вважається вразливим оскільки є потенційно вразливим до підміни трафіку.

2. Протокол IPSec

IPSec (IP Security) – це протокол, що забезпечує аутентифікації на надійної передачі пакетів мережевого рівня, тому протокол інкапсулюється протоколом IP. Протокол використовує 2 протоколи для забезпечення надійної передачі AH (Authentication Header) та ESP (Encapsulating Security Payload).

3. Протокол L2TP

L2TP – це протокол, що забезпечує тунелювання, був розроблений спільно компаніями Microsoft та Cisco як заміна протоколу PPP. Даний протокол інкапсулюється протоколом UDP з портом 1701. L2TP протокол використовується для тунелювання, та не шифрує трафік, зазвичай у комбінації з даним протоколом використовується протокол IPSec для забезпечення шифрування.

4. Протокол IKEv2

IKEv2 – це протокол, що забезпечує встановлення SA для протоколу IPSec, а саме процес обміну сертифікатів. Для обміну даних використовується протокол UPD, зазвичай порт 500. Відповідно тунелювання та шифрування відбувається протоколом IPSec. Це один з нових протоколів був створений в 2005 році, з того часу протокол декілька разів покращився, остання правка була зроблена в 2014 році.

5. Протокол OpenVPN

OpenVPN – це протокол та безкоштовне програмне забезпечення, що реалізує функції VPN. OpenVPN використовує TCP і UDP як протоколи нижнього рівня, а також є підтримка використання протоколу TLS. OpenVPN як проект з відкритим вихідним кодом був заснований в 2001 році та є проектом, що розвивається до сьогоднішнього часу.

Розглянемо загальну структуру протоколу OpenVPN [9.], а саме скоцентруємо нашу увагу на структурі повідомлення за допомогою якого встановлюється з'єднання:

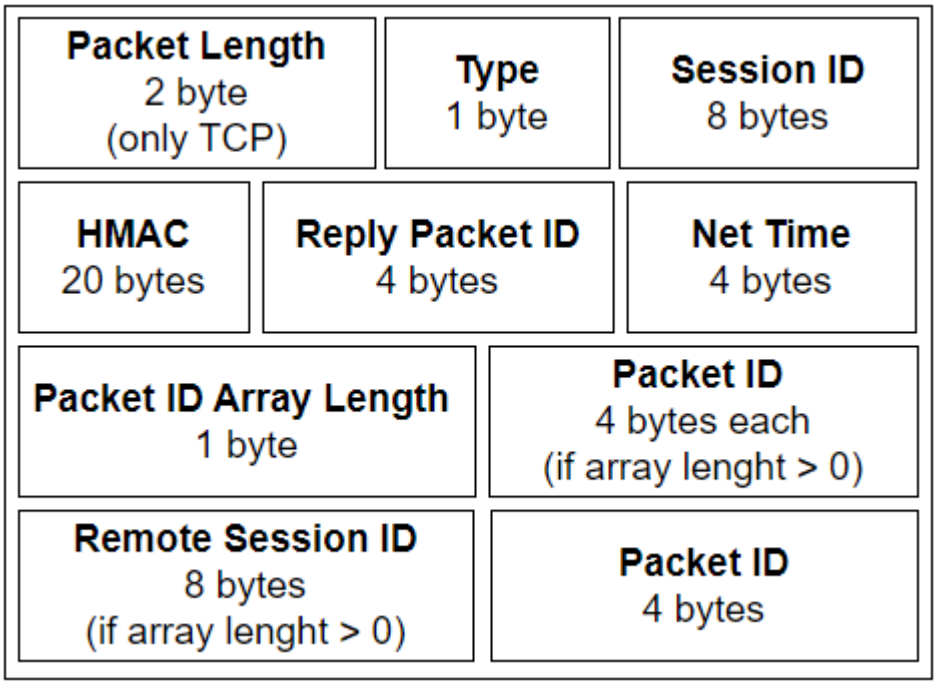

Рис. 1.2.3.1. Структура OpenVPN Handshake повідомлення

Поля даного протоколу:

- Packet Length (2 байти) – вказує на довжину пакету, використовується лише при інкапсуляції протоколом TCP, оскільки протокол UDP не гарантує відновлення порядку відновлення даних, тому для UDP довжина OpenVPN повідомлення рівна довжині UDP даних
- Type (1 байт) – тип OpenVPN повідомлення (перші 5 біт даного байту), тип повідомлення визначається в діапазоні від 1 до 8
- Session ID (8 байти) – ідентифікатор сесії, випадкове число
- HMAC (20 байти) – параметри необхідні для шифрування
- Packet Reply ID (4 байти) – номер повідомлення відповіді
- Net Time (4 байти) – час надсилання
- Packet ID Array Length (1 байт) – довжина список номерів пакетів іншої сесії, використовується для підтвердження у випадку повторного відкриття сесії
- Packet ID (4 байти кожен) – елементи списку кожен з них це номер пакету іншої сесії, використовується для підтвердження у випадку повторного відкриття сесії. Дане поле відсутнє якщо Packet ID Array Length рівне 0
- Remote Session ID (8 байтів) – ідентифікатор сесії, до якої належать пакети зі списку. Дане поле відсутнє якщо Packet ID Array Length рівне 0
- Packet ID – номер даного пакету, розпочинається з 0 і з кожним пакетом збільшується на 1.

6. Протокол WireGuard

WireGuard — це протокол та безкоштовне програмне забезпечення, що реалізує функції VPN. Для передачі даних протокол використовує протокол UDP. Протокол був розроблений Джейсон А. Доненфельд, перша стабільна версія протоколу була випущена в 2020 році.

Розглянувши структури WireGuard повідомлень [10.], визначимо загальну структуру WireGuard повідомлення:

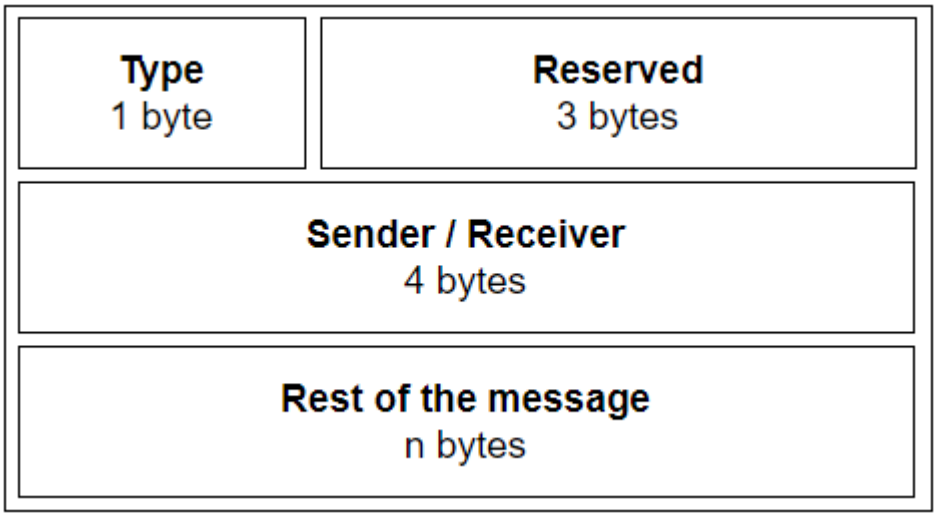

Рис. 1.2.3.2. Структура WireGuard повідомлення

- Type (1 байт) – тип повідомлення, усього є 4 типи повідомлення зі значеннями 1, 2, 3 та 4
- Reserved (3 байти) – зарезервоване значення, заповнене 0. У майбутньому можливо буде змінене
- Sender / Receiver (4 байти) – випадково згенерований ідентифікатор джерела/отримувача (в залежності від типу повідомлення)
- Rest of the message – інші поля, що є специфічними для кожного з типів повідомлення

7. Протокол Lightway

Lightway це протокол розроблений компанією ExpressVPN, для власного продукту в 2020 році. Lightway використовує протокол UPD для передачі даних, і протокол DTLS для встановлення з’єднання. Однак протокол може працювати поверх протоколів TCP та TLS. Даний протокол на даний момент є проектом з відкритим вихідним кодом та розповсюджується з ліцензією GBLv2.

Розглянувши документацію вихідного коду ядра протоколу [11.], можемо відзначити, що даний протокол має безліч типів, в той же час структура пакету, сильно відрізняється в залежності від типу повідомлення. До того ж поля протоколу є такими, що не є наперед визначеними, крім 1 байтового поля, з якого розпочинається пакет, яке ідентифікує тип повідомлення.

1.2.3.1. VPN висновки

Розглянувши характерні риси роботи VPN сервісів, можна виявити загальні характеристики за якими можна спробувати ідентифікувати даний трафік:

- використання клієнт-сервер архітектури, дає можливість нам можливість визначати VPN трафік:
  - Виявляючи DNS запити, з доменами іменами відповідних серверів
  - Виявляючи трафік, що звертається до ресурсів з IP адресами VPN серверів
- використання сигнатур протоколів можливе, однак:
  - використання великої кількості протоколів, що досі розробляються, а також створення нових ускладнює цей процес
  - велика кількість протоколів використовують TLS, IPSec для шифрування/створення каналу, оскільки ці протоколи є широко використовуваними це унеможливлює ідентифікацію за ними власне VPN трафіку

1.2.4. TOR трафік

Використання TOR мережі дозволяє приховати мережеву активність користувача, що може бути використовуватися для приховування порушень користування мережею.

Анонімізація забезпечується шляхом побудови TOR мережі у якій є вихідні вузли, та внутрішні вузли. Основна ідея анонімізації забезпечується тим, що зашифроване повідомлення надсилається декілька разів, та порядок надсилання повідомлення між вузлами визначається клієнтом, що надсилає повідомлення. До того ж, клієнт шифрує повідомлення декілька разів, таким чином, щоб кожен вузол проводив проміжне розшифрування, та не міг побачити оригінального повідомлення, та лише отримувач міг повністю розшифрувати повідомлення.

Для шифрування повідомлення TOR мережа використовує протокол TLS, з портом отримувача 443, що робить його схожим на веб трафік. Однак, користування TOR мережею потребує звернення до вихідних вузлів TOR мережі. Тим самим визначення користування TOR мережею можливе шляхом визначення звернень до вихідним вузлів TOR мережі.

1.2.5. Протоколи шифрування DNS

На перший погляд використання протоколів шифрування DNS складно назвати шкідливим трафіком, що варто додати до тих що порушують політики користування. Однак часто, важливою вимогою користування корпоративною мережею є використання корпоративного DNS, за допомогою якого можуть надаватися IP адреси корпоративних сервісів, блокуватися небажані сервіси шляхом повернення результату неіснуючого доменного імені DNS сервером. Тим самим існує необхідність виявлення DNS запитів, що адресовані зовнішнім DNS серверам, а відповідно протоколи шифрування DNS ускладнюють виявлення користування зовнішніми DNS серверами.

Розглянемо найпоширеніші протоколи шифрування DNS:

- DoT (DNS over TLS) – це протокол, що використовує протокол TLS, для шифрованої передачі DNS повідомлень. Для отримання повідомлень сервер використовує кінцеву точку з портом 853
- DoH (DNS over HTTPS) – це протокол, що використовує протокол HTTPS для визначення IP адрес доменних імен. Враховуючи те, що сервер використовує кінцеву точку з портом 443 даний трафік сигнатурно не відрізняється від веб трафіку. Традиційно протокол HTTPS використовує протокол TLS для шифрування, однак для даного завдання популярним є використання HTTPS 3, що використовує протокол QUIC для шифрування
- DoQ (DNS over QUIC) – це протокол, що використовує протокол QUIC, для шифрованої передачі DNS повідомлень. Для отримання повідомлень сервер використовує кінцеву точку з портом 853. Варто зазначити, хоч визначено стандартом використання протоколу QUIC на практиці є реалізації, що використовують DTLS натомість.

1.3. Suricata

Suricata – програмний засіб для реалізації IDS, IPS та NSM. Продукт був розроблений організацією Open Information Security Foundation (OISF) як альтернатива Snort. Сумісна з багатьма операційними системами, такими як Windows, Linux (Debian, CentOS, Ubuntu, Red Hat), FreeBSD, MacOS X. Suricata може працювати як самостійна система, так і в якості додатка до деяких систем.

1.3.1. Переваги Suricata

Розглянемо чому в нашій роботі була використана Suricata.

- Suricata – це безкоштовний проект, що поширюється з відкритим вихідним кодом

- Розроблений з орієнтацію на підтримку з програмою Snort. Snort – це IDS, що також є безкоштовною для використання, і була однією з перших IDS, за рахунок чого здобула велику популярність та поширення. Тим самим багато налаштувань, правил зроблених для Snort працюють і для Suricata
- На відміну від Snort має вбудовану ідентифікацію багатьох мережевих протоколів 4 рівня TCP/IP, таких як TLS, DNS, FTP, HTTP та інші.
- На відміну від Snort, що є однопоточним. Suricata є багато потоковою.

1.3.2. Suricata конфігурація

Центральною частиною роботи Suricata є файл конфігурації, в ньому задаються всі параметри необхідні для коректної роботи сервісу. Хоч основну кількість параметрів, що задаються в файлі конфігурації можна вказати при запуску програми в більшості випадків це не є зручним, до того ж в файлі конфігурації можна задати параметри за замовчуванням, що за необхідності будуть перевизначатися при запуску.

Конфігурація Suricata зберігається в файлі формату YAML. За замовчуванням в UNIX подібних системах файл конфігурації зберігається за шляхом /etc/suricata/suricata.yaml.

1.3.3. Suricata датасети

Датасети (datasets) дозволяють зберігати велику кількість бінарних значень, і за необхідності порівнювати елементи пакетів з даними збереженими в датасеті (перевіряти, що в датасеті зберігається значення).

Датасет це простий текстовий файл, кожна строка якого містить дані, що зберігаються в одному з 3 типів:

- string (текстовий формат) – байтові дані закодовані в стандарті base64
- md5 – md5 хеш байтових даних, записаний як hex строка
- sha256 – sha256 хеш байтових даних, записаний як hex строка

Задавати датасети можна в файлі конфігурації, в блоці datasets, створити блок з власною назвою, що задає назву для датасету, та містить параметри load – назва файлу датасету, і type – тип файлу, приклад дображено на зображенні 1.2.5.1.

```
datasets:
  defaults:
    memcap: 100mb
    hashsize: 2048

  mining-dns-md5:
    load: /home/userx/suricata/datasets/mining_domains_md5.list
    type: md5
```

Рис. 1.3.3.1. Приклад конфігурації датасету mining-dns-md5

1.3.4. Suricata IP Репутаційні файли

IP Репутаційні файли – це механізм, що надає можливість зберігати групу IP адрес та мереж, з відповідним рівнем довіри та згрупувати їх, за категоріями. Далі перевіримо чи IP адреси пакету належать до певної категорії.

Для налаштування репутаційних файлів, використовуються 2 файли:

- IP Reputation Config – це CSV файл, що містить інформацію про всі IP групи, структура його рядка “<id>,<short name>,<description>”, даний файл є єдиним, задається в файлі конфігурації, за полем reputation-categories-file
- IP Reputation file – це CSV файл, що містить інформацію про IP адреси та мережі, і вказує до якої групи IP адрес він належить і його репутація,

зберігається у форматі “<ip>,<category>,<reputation score>”, даних файлів може бути безліч, задаються в файлі конфігурації в полях:

- default-reputation-path – вказує папку за замовчуванням для репутаційних файлів
- reputation-files – список репутаційних файлів

1.3.5. Suricata правила

```
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:”ET TROJAN Likely Bot Nick in IRC (USA +..)”;
flow:established,to_server; flowbits:isset,is_proto_irc; content:”NICK “; pcre:”/NICK .*USA.*[0-9]
{3,}/i”; reference:url,doc.emergingthreats.net/2008124; classtype:trojan-activity; sid:2008124;
rev:2;)
```

Рис. 1.3.5.1. Структура Suricata правила [13.]

В Suricata правила складаються з 3 частин, розглянемо на зображенні 1.2.7.1. частини з яких складається правило:

1. Action (червоний колір), вказує на дію, що потрібно виконати, якщо правило спрацювало для даного пакету
2. Header (зелений колір), вказує на основні параметри, що потрібні для фільтрування кожного з правил
3. Options (синій колір), вказує на додаткові фільтри та параметри до правила

Прийнято зберігати правила в файлі з розширенням “.rules”. За замовчуванням в UNIX подібних системах правила зберігаються в папці /etc/suricata/rules. Однак є можливість визначити папку для правил за замовчуванням в файлі конфігурації, задавши відповідне значення для поля default-rule-path.

Для того, щоб правило враховувалося необхідно добавити назву його файлу до масиву rule-files в файлі конфігурації.

1.3.5.1. Формат правил – Action

Дія складається з 1 слова, що вказує на дію, що потрібно виконати, якщо правило спрацювало для даного пакету, можливі варіанти:

- alert – згенерувати сповіщення
- pass – завершити подальший аналіз пакету
- drop – відкинути пакет з згенерувати сповіщення
- reject – надіслати RST/ICMP unreach error для джерела пакету
- rejectsrc – аналогічне до reject
- rejectdst – надсилає RST/ICMP error packet для одержувача пакету
- rejectboth – надсилає RST/ICMP error packet для одержувача і джерела пакету

Оскільки в ході роботи буде використовуватися Suricata для аналізу мережевого трафіку відповідно буде використовуватися alert action, що буде генерувати сповіщення, що буде записуватися до лог файлу.

1.3.5.2. Формат правил – Header

Header складається з 6 частин:

1. Protocol

```
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET TROJAN Likely Bot Nick in IRC (USA +..)";
```

Рис. 1.3.5.2.1. Header – Протокол [13.]

Вказує на мережевий протокол, що повинен містити пакет, до них належать:

- ip – означає, що правило для всіх пакетів
- icmp
- tcp

- udp
- dns
- tls

З повним переліком протоколів, що підтримує Suricata можна ознайомитися в документації. [12.]

2. Source IP

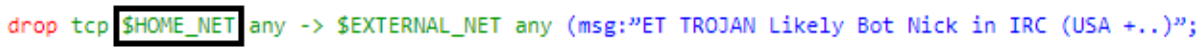

Рис. 1.3.5.2.2. Header – Source IP [13.]

Вказує шаблон, що фільтрує пакети по IP адресі джерела. Можливі варіанти задання IP правила:

- any – вказує, що для даного правила підходить будь яка IP адреса
- $HOME_NET – вказує на домашню мережу, що задана в файлі конфігурації
- $EXTERNAL_NET – вказує на зовнішню мережу, що задана в файлі конфігурації
- {IP адреса} – вказує, що правило підходить для заданої IP адреси
- {IP адреса}/{маска} – символ "/" використовується для задання маски для мережі адрес, що підходять під правило
- !{IP правило} – вказує, що підходять адреси, що не підпадають під правило (може бути як і мережа, так і група адрес)
- [{IP правило}, {IP правило}, …] – вказує на групу правил, якщо хоч одне з них виконується IP адреса вважається підходящою

3. Source Port

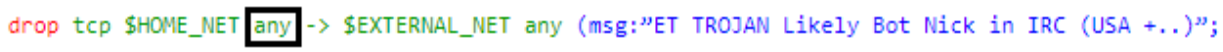

Рис. 1.3.5.2.3. Header – Source Port [13.]

Вказує шаблон, що фільтрує пакети по порту джерела. Можливі варіанти задання правила для порта:

- any – вказує, що підходить будь який порт або його відсутність (стосується пакетів, до яких не входять на UDP ні TCP заголовки)
- {Port} – вказує, що підходить вказаний порт
- {Port1}:{Port2} – вказує, що підходить діапазон портів починаючи з Port1 і завершуючи Port2 включно
- !{Port Rule} – вказує на заперечення заданого правила
- [{Port Rule}, {Port Rule}, …] – вказує на групу правил, якщо хоч одне з них виконується порт вважається підходящим

4. Direction

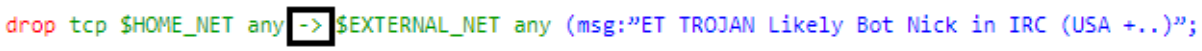

Рис. 1.3.5.2.4. Header – Direction [13.]

Вказує на напрямок, що вказує на те, як IP адреси та порти будуть співставлятися:

- "→" – вказує, що співставляється лише надсилання пакету в прямому порядку
- "<>" – вказує, що правило перевіряє пакет, на надсилання в обох напрямках

5. Destination IP

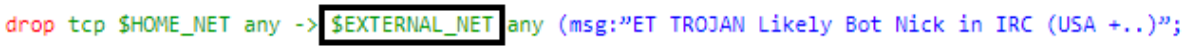

Рис. 1.3.5.2.5. Header – Destination IP [13.]

Вказує шаблон, що фільтрує пакети по IP адресі отримувача. Можливі варіанти задання IP правила, аналогічні для Source IP.

6. Destination Port

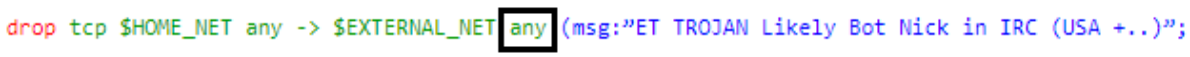

Рис. 1.3.5.2.6. Header – Destination Port [13.]

Вказує шаблон, що фільтрує пакети по порту джерела. Можливі варіанти задання правила для порта аналогічні для Source Port

1.3.5.3. Формат правил – Options

Rule options – задає додаткові фільтри та параметри до правила. Suricata надає можливість задавати велику кількість опцій з повним списком яких можна ознайомитися в документації [13.], в даному розділі згадаємо основні опції, що будуть зустрічатися в нашій практичній частині:

- msg – дозволяє задати сповіщення, що слугує для задання додаткової інформації для правила, та виводиться у разі сповіщення
- sid – дозволяє задати унікальний ідентифікатор для правила
- rev – дозволяє задати версію даного правила (зазвичай, якщо додаються поправки до правила для підтримання версійності необхідно збільшити число версії)
- gid – дозволяє задати число, що ідентифікує групу правил (зазвичай якщо декілька правил фільтрують одну вразливість добрим тоном вважається задати для них єдину групу)
- dns.query – дозволяє зафіксувати, що наступні дані, що будуть порівнюватися це значення dns запиту

- datasets – вказує, що необхідно порівняти байтові дані з даними вказаного датасету
- iprep – вказує, що потрібно співставити вказану IP адресу з IP адресами вказаними в репутаційному файлі

1.3.5.4. Lua скрипти

Lua скрипти це механізм, що дозволяє впроваджувати власну логіку до правил, шляхом написання скрипту, що буде виконуватися при виконанні правила. Приклад lua скрипту можна проглянути на рисунку 1.3.5.4.1:

```lua
function init(args)
    local needs = {}
    needs["payload"] = tostring(true)
    return needs
end

function match(args)
    payload = args["payload"]

    if #payload < 16 then
        return 0
    end

    if CheckMagic(payload) == 0 then
        return 0
    end

    return CheckCommand(payload)
end
```

Рис. 1.3.5.4.1. Lua скрипт

Lua скрипт складається з 2 частин:

- Функція init – це функція, що дозволяє вказати, які саме дані необхідні передаватися скрипту при виконанні

- Функція match – це функція, що виконує логіку відбору при перевірці правила, ця функція може повертати 2 значення – 0 якщо пакет не підходить під критерій перевірки, та 1 якщо пакет проходить критерій перевірки

  Для виконання lua скрипт задається опцією lua:[!]<scriptfilename> в правилі.

1.3.6. Suricata output logging

Зберігання мережевої активності налаштовується в блоці output в файлі конфігурації. В даному блоці можна вказати один із підтримуваних варіантів зберігання вихідних даних, до них належать:

- fast logging – зберігає сповіщення згенеровані правилами в однострочному вигляді. В основному використовується при тестуванні
- http logging – зберігає інформацію про всі HTTP пакети, що були перехоплені в однострочному вигляді
- tls logging – зберігає інформацію про всі TLS сесії, що були перехоплені в однострочному режимі
- serilog – дозволяє вказати використання Serilog логування
- lua logs – дозволяє запрограмувати логування власноручно
- eve logs – дозволяє налаштувати зберігання даних в json форматі, використання даного логування є найбільш функціональним, було вирішено використовувати дане логування, тому розглянемо його детальніше

1.3.6.1. Eve logs

Даний тип логування зберігає вихідні дані у json форматі. Eve логування налаштовується в файлі конфігурації, як секція eve-log всередині секції output. В блоці eve-log, вказується налаштування логу, а також вказати типи подій, що будуть зберігатися в лог файлі.

Дане логування має найбільш повний список вбудованих можливостей, що вирізняє його серед інших типів логування, а саме:

- Дозволяє обрати типи івентів, що зберігаються в лозі
- Дозволяє задати автоматичне розбиття логування, на окремі файли по часовим признакам
- Має існуючий переглядач логу – evebox [14.], що розроблений командою Suricata, що має вбудоване збереження записів у локальній базі даних
- Автоматично зберігає значення полів пакетів, що логуються

Розглянемо налаштування eve логу, а саме шляхом конфігурації полів:

- enabled – вказує, чи використовувати даних лог файл. Дане поле може приймати 2 значення “yes”, “no”
- filename – вказує, назву файлу в якому зберігати логи. У назві файлу можуть використовуватися додаткові символи, що дозволяють додати до назви файлу час створення лог файлу, що є особливо необхідним, якщо налаштовуємо дефрагментацію лог файлу
    - %Y – рік створення лог файлу
    - %m – місяць створення лог файлу
    - %d – день створення лог файлу
    - %H – година створення лог файлу
    - %M – хвилина створення лог файлу
    - %S – секунда створення лог файлу
- rotate-interval – вказує на період перестворення лог файлу. Дане поле є числовим, однак містить модифікатори s – секунди, m – минути, h – години, d – дні і w – тижні. Тобто якщо вказати в даному полі “25s” новий лог файл буде створюватися кожні 25 секунд
- types – це масив блоків, що вказує які події будуть записувати в лог. До можливих типів належать:

- alert – вказує, що до даного лог файлу будуть записуватися сповіщення, що викликаються в правилах
- anomaly – вказує, що до даного лог файлу будуть записуватися виявлені аномалії (аномальний пакет, це такий пакет у якому є поля, що неправильно заповнені у відповідності до протоколів, наприклад вказана неправильна версія протоколу)
- tls – вказує, що необхідно зберігати записи для кожного встановленого tls з’єднання. Додатково можна вказати у якому форматі, які поля є бажаними для зберігання в лог файлі
- dns – вказує, що необхідно зберігати записи для кожного перехопленого dns запису.
- Та інші протоколи 5 рівня, з повним списком яких ви можете ознайомитися в документації [15.]

РОЗДІЛ 2. ПРАКТИЧНА ЧАСТИНА

2.1. Розробка правил виявлення доступу до відомих сервісів

Беручи до уваги теорію, було визначено, що поширеними методами детектування звертання до сервісів є визначення звернень до загальновідомих IP адрес, а також визначення DNS запитів зі значеннями загальновідомих доменних імен. Розглянемо створення та тестування загальних правил, що визначають звернення до заданих IP адрес, та DNS запити з заданими доменними іменами.

2.1.1. Виявлення за доменними іменами DNS запитів

На сьогоднішній день визначення IP адрес ресурсів використовуючи DNS сервери є однозначним світовим стандартом, навіть peer-2-peer такі як Bitcoin, що колись використовували протокол IRC для визначення IP адрес її вузлів, перейшли на використання DNS.

Навіть якщо сам трафік зашифровано, якщо було перехоплено його DNS пакет є можливість визначення до якого ресурсу користувач звертається.

2.1.1.1. Написання програми для створення датасету

Як було визначено в теоретичній частині, в IDS Suricata датасет зберігає у собі байтові дані, що лише зберігаються у одному з строкових виглядів. Оскільки доменні ім’я це строкові данні, до того ж кількість доменних імен може бути довільною, логічним є створення утиліти, що дозволяє перетворити список доменних імен до необхідного нам датасету.

Код розробленої утиліти знаходиться у Додатку А. Розглянемо основний функціонал і засоби взаємодії з утилітою. Утиліта дозволяє створити датасет у

всіх підтримуваних форматах, а саме як base64 (за замовчуванням), sha256, md5. Оскільки для передачі в мережі текст кодується шифром UTF-8 саме він використовується при створенні датасету.

```
E:\Study\Univ\MasterDegree\PythonRuleAuto>python list-to-dataset.py --help
usage: list-to-dataset.py [-h] [--sha256 | --md5 | --string] [-o OUTPUT] [-f FILE]

options:
  -h, --help            show this help message and exit
  --sha256              Create sha256 dataset
  --md5                 Create md5 dataset
  --string              Create string dataset. Behavior by default
  -o OUTPUT, --output OUTPUT
                        Specify output file.
  -f FILE, --file FILE  Specify input file.
```

Рис. 2.1.1.1.1. Опис використання утиліти

Утиліта дозволяє працювати не лише з файлами, вона використовує stdin, stdout консолі, якщо вхідний/вихідний файли невказані. Тим самим вона може бути частиною пайплайна:

```
E:\Study\Univ\MasterDegree\PythonRuleAuto>echo 'Line of dataset' | python list-to-dataset.py --md5
29ec4c25e83c1a48d79763b799c4c77d
```

Рис. 2.1.1.1.2. Використання утиліти в пайплайні

або можемо вказати вихідний файл, і друкувати текстові дані в консолі:

```
E:\Study\Univ\MasterDegree\PythonRuleAuto>python list-to-dataset.py --sha256 -o dataset.out
google.com
testbed.knu.ua
univ.net.ua
^Z

E:\Study\Univ\MasterDegree\PythonRuleAuto>type dataset.out
d4c9d9027326271a89ce51fcaf328ed673f17be33469ff979e8ab8dd501e664f
66d715437e1eeff1d275c465aa8a8417137410e2bb65b82cd6bd14fcf5ee9ea6
f7dc42e38439edd5525482099a91aae7bebf0b0583f9ba4e640c0427fb3a3dae
```

Рис. 2.1.1.1.3. Використання утиліти в режимі вводу даних з консолі

2.1.1.2. Підготовка тестових даних

Для початку зробимо тестову вибірку пакетів, скористаємося програмою Wireshark. Збережемо випадковий трафік, що був зібраний протягом сесії. Розглянемо перехоплений трафік, він складається з 24677 пакетів, з них 669 пакетів, то запити до DNS:

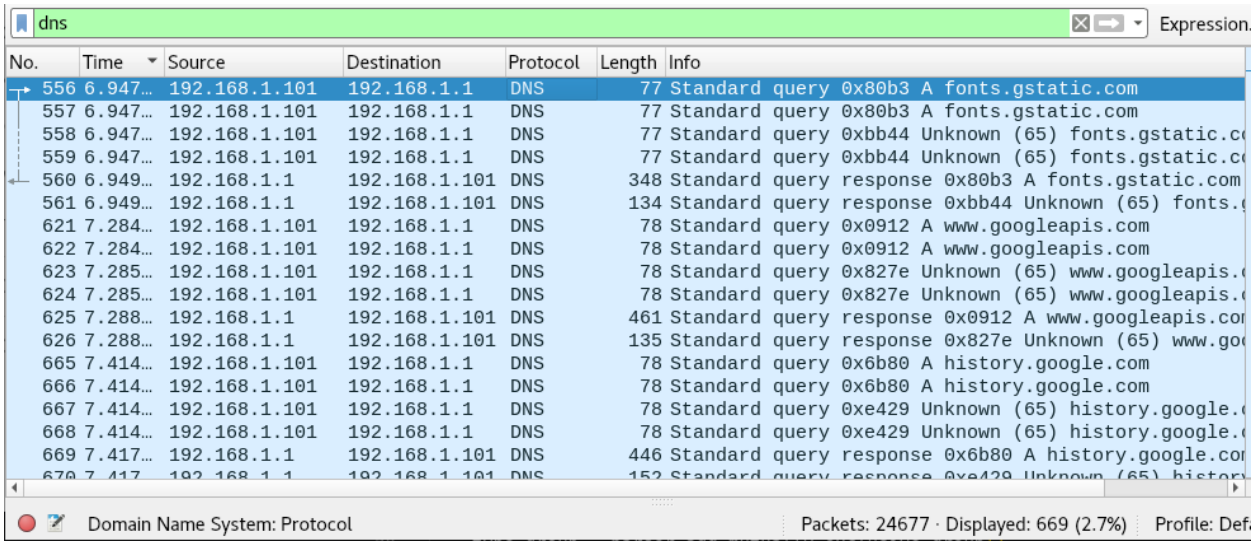

Рис. 2.1.1.2.1. Дані тестового Pcap файлу

Виберемо декілька довільних адрес з NDS запитів, при цьому будемо брати їх з довільних місць, як з початку так і з кінця перехоплених пакетів.

```
[andrii@sandbox suricata]$ cat datasets/domains.list
history.google.com
www.gstatic.com
get.geojs.io
wordwall.net
cdnjs.cloudflare.com
```

Рис. 2.1.1.2.2. Список DNS імен для тесту

Розглянемо детальніше, пакети для кожного з доменних адресів.

Для доменів “history.google.com”, “www.gstatic.com”, “get.geojs.io”, “wordwall.net” кожен з них містить 6 пакетів: 2 пакети запити типу A, 2 пакети запити типу HTTPS, 2 пакети відповіді від DNS серверу.

dns.qry.name=="history.google.com"

No.	Time	Source	Destination	Proto	Lengt	Info
665	7.414549	192.168.1.101	192.168.1.1	DNS	78	Standard query 0x6b80 A history.google.com
666	7.414571	192.168.1.101	192.168.1.1	DNS	78	Standard query 0x6b80 A history.google.com
667	7.414975	192.168.1.101	192.168.1.1	DNS	78	Standard query 0xe429 HTTPS history.google.com
668	7.414987	192.168.1.101	192.168.1.1	DNS	78	Standard query 0xe429 HTTPS history.google.com
669	7.417279	192.168.1.1	192.168.1.101	DNS	446	Standard query response 0x6b80 A history.google
670	7.417279	192.168.1.1	192.168.1.101	DNS	152	Standard query response 0xe429 HTTPS history.go

Рис. 2.1.1.2.3. DNS пакети для домену history.google.com

dns.qry.name=="www.gstatic.com"

No.	Time	Source	Destination	Proto	Lengt	Info
464	6.440061	192.168.1.101	192.168.1.1	DNS	75	Standard query 0x8bb7 HTTPS www.gstatic.com
465	6.440077	192.168.1.101	192.168.1.1	DNS	75	Standard query 0x8bb7 HTTPS www.gstatic.com
461	6.439421	192.168.1.101	192.168.1.1	DNS	75	Standard query 0x9e6f A www.gstatic.com
462	6.439441	192.168.1.101	192.168.1.1	DNS	75	Standard query 0x9e6f A www.gstatic.com
471	6.442538	192.168.1.1	192.168.1.101	DNS	132	Standard query response 0x8bb7 HTTPS www.gst
470	6.442538	192.168.1.1	192.168.1.101	DNS	346	Standard query response 0x9e6f A www.gstatic

Рис. 2.1.1.2.4. DNS пакети для домену www.gstatic.com

dns.qry.name=="get.geojs.io"

No.	Time	Source	Destination	Proto	Lengt	Info
40…	17.8155…	192.168.1.101	192.168.1.1	DNS	72	Standard query 0x2a61 A get.geojs.io
40…	17.8155…	192.168.1.101	192.168.1.1	DNS	72	Standard query 0x2a61 A get.geojs.io
40…	17.8159…	192.168.1.101	192.168.1.1	DNS	72	Standard query 0x5902 HTTPS get.geojs.io
40…	17.8159…	192.168.1.101	192.168.1.1	DNS	72	Standard query 0x5902 HTTPS get.geojs.io
41…	17.8328…	192.168.1.1	192.168.1.101	DNS	120	Standard query response 0x2a61 A get.geojs
42…	17.8465…	192.168.1.1	192.168.1.101	DNS	171	Standard query response 0x5902 HTTPS get.g

Рис. 2.1.1.2.5. DNS пакети для домену get.geojs.io

dns.qry.name=="wordwall.net"

No.	Time	Source	Destination	Proto	Lengt	Info
52…	19.2385…	192.168.1.101	192.168.1.1	DNS	72	Standard query 0x4d43 HTTPS wordwall.net
52…	19.2386…	192.168.1.101	192.168.1.1	DNS	72	Standard query 0x4d43 HTTPS wordwall.net
52…	19.2380…	192.168.1.101	192.168.1.1	DNS	72	Standard query 0xa006 A wordwall.net
52…	19.2381…	192.168.1.101	192.168.1.1	DNS	72	Standard query 0xa006 A wordwall.net
52…	19.2689…	192.168.1.1	192.168.1.101	DNS	165	Standard query response 0x4d43 HTTPS wordwa
52…	19.2551…	192.168.1.1	192.168.1.101	DNS	88	Standard query response 0xa006 A wordwall.n

Рис. 2.1.1.2.6. DNS пакети для домену wordwall.net

Для домену “cdnjs.cloudflare.com” усього 12 пакетів: 4 пакети запити типу A, 4 пакети запити типу HTTPS, 4 пакети відповіді від DNS серверу.

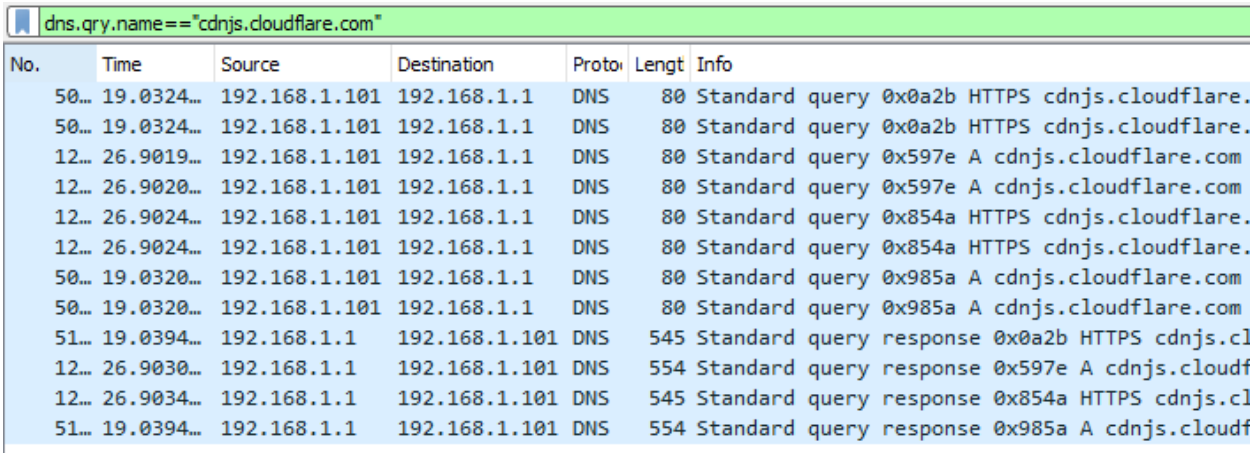

Рис. 2.1.1.2.7. DNS пакети для домену cdnjs.cloudflare.com

Отриманий результат, усього запитів було 24 DNS запитів, 12 з яких звертаються за адресою типу A, і 12 які звертаються за адресою типу HTTPS.

2.1.1.3. Налаштування Suricata

Для початку створимо датасети, для перевірки коректності роботи розробленої утиліти для створення датасетів, створимо датасети всіх можливих типів, для списку доменних імен.

- Тип string:

```
[andrii@sandbox datasets]$ python3 ../list-to-dataset.py -f domains.list
 -o domains_base64.list
[andrii@sandbox datasets]$ cat domains_base64.list
aGlzdG9yeS5nb29nbGUuY29t
d3d3LmdzdGF0aWMuY29t
Z2V0Lmdlb2pzLmlv
d29yZHdhbGwubmV0
Y2RuanMuY2xvdWRmbGFyZS5jb20=
```

Рис. 2.1.1.3.1. Створення датасету типу string

- Тип md5:

```
[andrii@sandbox datasets]$ python3 ../list-to-dataset.py -f domains.list
 -o domains_md5.list --md5
[andrii@sandbox datasets]$ cat domains_md5.list
79afd0ac63778344babfed130634a988
3ac992bd8a8bfcdd4b09a10c26e6b220
f3f8ccaa5b8bc07345dba7887ad939da
14554b902910d79b66cc0c59a7831ff1
c5f1651768b9f419c8c69bbc605d99be
```

Рис. 2.1.1.3.2. Створення датасету типу md5

- Тип sha256:

```
[andrii@sandbox datasets]$ python3 ../list-to-dataset.py -f domains.list
 -o domains_sha256.list --sha256
[andrii@sandbox datasets]$ cat domains_sha256.list
c350913878bceb2f16ac42de2e3ad6f7f82cdfc3cad81ea480c2212ba39db2b1
80255870a6218fe0ce08d8b9c67623991cf2ae1687348ab3b274b4583a71faee
a5d815014377a6f4aaf2eb8bf0a605481f080edd83f86680febccc2ea0b18b55
4667b120e14633819a4a2ef08062466bc649b5e6b173380c1983c7b0d15d9d7d
88bb79fa6efc5d55c70ed478e2e48f058c387b77c4e99ad1f73b9a5bfa237d2c
```

Рис. 2.1.1.3.3. Створення датасету типу sha256

Добавимо створені датасети до файлу конфігурації Suricata.

```
datasets:
    dns-base64:
        load: /home/andrii/suricata/datasets/domains_base64.list
        type: string
    dns-sha256:
        load: /home/andrii/suricata/datasets/domains_sha256.list
        type: sha256
    dns-md5:
        load: /home/andrii/suricata/datasets/domains_md5.list
        type: md5
```

Рис. 2.1.1.3.4. Конфігурація створених датасетів в suricata.yaml

Для можливості отримання результатів, увімкнемо логування сповіщень, а саме fast-log та eve-log для отримання більш детальної інформації.

```
outputs:
  # a line based alerts log similar to Snort's
  - fast:
      enabled: yes
      filename: fast.log
      append: no
  - eve-log:
      enabled: yes
      filetype: regular
      filename: dns-eve.json

      types:
        - alert:
```

Рис. 2.1.1.3.5. Конфігурація Log файлів в suricata.yaml

Створимо 3 файли правил, що будуть сповіщати про dns запити, що містять у собі доменне ім’я, що є у нашому списку. Для тестування всіх типів створених датасетів, створимо окремо 3 файли, з правилом, що використовує свій датасет.

- Тип string:

```
[andrii@sandbox suricata]$ cat rules/alter_by_domain_base64.rules
alert dns any any -> any 53 (msg: "This domain name was in domain_base64";
dns.query; dataset:isset,dns-base64; sid:123256;)
```

Рис. 2.1.1.3.6. Правило, що використовує датасет типу string

- Тип md5, оскільки дані датасету, це хеш значення його не можливо порівнювати напряму з доменним іменем. Нам необхідно порівнювати дані датасету з хеш значенням доменного імені пакету, для цього перед порівнянням використаємо опцію to_md5:

```
[andrii@sandbox suricata]$ cat rules/alter_by_domain_md5.rules
alert dns any any -> any 53 (msg: "This domain name was in domain_md5";
dns.query; to_md5; dataset:isset,dns-md5; sid:123258;)
```

Рис. 2.1.1.3.7. Правило, що використовує датасет типу md5

- Тип sha256, як і у випадку md5, в датасеті зберігається хеш значення, тому для необхідно обчислити sha256 хеш доменного імені пакету за допомогою опції to_sha256:

```
[andrii@sandbox suricata]$ cat rules/alter_by_domain_sha256.rules
alert dns any any -> any 53 (msg: "This domain name was in domain_sha256";
dns.query; to_sha256; dataset:isset,dns-sha256; sid:123257;)
```

Рис. 2.1.1.3.8. Правило, що використовує датасет типу sha256

2.1.1.4. Тестування з датасетом типу string

Запустимо Suricata, з правилом, що використовує string датасет, у офлайн режимі (використовуючи перехоплені нами пакети у pcap форматі). Та розглянемо детальніше вміст fast.log файлу.

```
[andrii@sandbox suricata]$ suricata -r test.pcap -S rules/alter_by_domain_base64.rules
5/12/2022 -- 15:54:22 - <Notice> - This is Suricata version 6.0.8 RELEASE running in USER mode
5/12/2022 -- 15:54:22 - <Notice> - Ring buffer initialized with 0 files.
5/12/2022 -- 15:54:22 - <Notice> - all 3 packet processing threads, 4 management threads initialized, engine start
ed.
5/12/2022 -- 15:54:24 - <Notice> - Signal Received.  Stopping engine.
5/12/2022 -- 15:54:37 - <Notice> - Pcap-file module read 1 files, 24677 packets, 17113630 bytes
[andrii@sandbox suricata]$ cat fast.log | wc -l
24
[andrii@sandbox suricata]$ cat fast.log
11/30/2022-04:31:41.281374  [**] [1:123256:0] This domain name was in domain_base64 [**] [Classification: (null)]
[Priority: 3] {UDP} 192.168.1.101:62228 -> 192.168.1.1:53
11/30/2022-04:31:41.281394  [**] [1:123256:0] This domain name was in domain_base64 [**] [Classification: (null)]
[Priority: 3] {UDP} 192.168.1.101:62228 -> 192.168.1.1:53
11/30/2022-04:31:42.256502  [**] [1:123256:0] This domain name was in domain_base64 [**] [Classification: (null)]
[Priority: 3] {UDP} 192.168.1.101:53497 -> 192.168.1.1:53
11/30/2022-04:31:42.256524  [**] [1:123256:0] This domain name was in domain_base64 [**] [Classification: (null)]
[Priority: 3] {UDP} 192.168.1.101:53497 -> 192.168.1.1:53
```

Рис. 2.1.1.4.1. Запуск suricata та читання результату fast-log

Як можемо спостерігати fast-log містить однотипну інформацію, та не містить додаткову інформацію про пакети. Усього записів 24, як і було очікувано.

Спробуємо отримати з eve-json файлу інформацію, про кількість сповіщень для кожного з доменних імен. Для цього створимо простий скрип, що підраховує кількість строк, у яких зустрічається дане доменне ім’я. Оскільки за замовчуванням увесь json пакується в одній строці, кількість строк де

зустрічається доменне ім’я повинне вказати нам на кількість спрацювань правила для кожного з доменів.

```
[andrii@sandbox suricata]$ cat domain-name-in-file.sh
#!/bin/bash

eve-json=$(cat "$1")

historyCount=$(echo "$eve-json" | grep history.google.com | wc -l)
gstaticCount=$(echo "$eve-json" | grep www.gstatic.com | wc -l)
geojsCount=$(echo "$eve-json" | grep get.geojs.io | wc -l)
wordwallCount=$(echo "$eve-json" | grep wordwall.net | wc -l)
cloudflareCount=$(echo "$eve-json" | grep cdnjs.cloudflare.com | wc -l)

echo "history.google.com   $historyCount"
echo "www.gstatic.com      $gstaticCount"
echo "get.geojs.io         $geojsCount"
echo "wordwall.net         $wordwallCount"
echo "cdnjs.cloudflare.com $cloudflareCount"
```

Рис. 2.1.1.4.2. Скрипт, визначення кількості доменних імен в eve-log файлі

Хоч і розмір лог файлу – 24 записи, було виявлено лише, 12 записів з доменними іменами, розглянемо лог файл детальніше.

```
[andrii@sandbox suricata]$ ./domain-name-in-file.sh dns-eve.json
history.google.com   2
www.gstatic.com      2
get.geojs.io         2
wordwall.net         2
cdnjs.cloudflare.com 4
```

Рис. 2.1.1.4.3. Отримання кількості доменних імен в eve-log файлі

Для початку можемо помітити сповіщення про DNS запит, що звертався за адресою “www.gstatic.com” типу A Додаток Б. При детальному розгляді лог файлу, можемо спостерігати, що є 12 сповіщень про DNS запит, що не містять у собі звернень за доменною адресою, один з даних записів зображено в Додатку В. З нього можемо побачити, що номер даного пакету – 464. Розглянемо його в Wireshark.

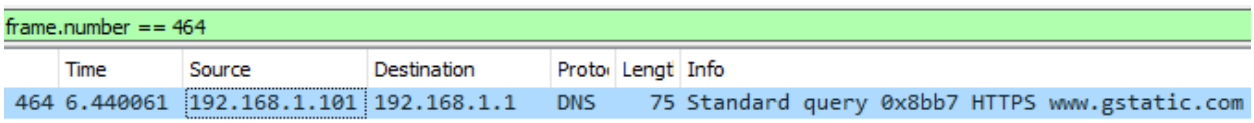

Рис. 2.1.1.4.4. Пакет №464 розглянутий в Wireshark

Як можемо спостерігати, пакет, що викликав сповіщення є DNS запитом, що просить адресу “www.gstatic.com” типу HTTPS. При ознайомлені з іншими записами в лог файлі, що не містять доменних адрес можемо зробити висновок, що за замовчуванням Suricata не включає до логу вміст DNS запитів, що просять запис типу HTTPS, дана поведінка потребує детальнішого вивчення.

Результат: було отримано сповіщення про кожен пакет, що вказує на правильність роботи правила та датасету, однак виявлено неповене збереження бажаної інформації до лог файлу.

2.1.1.5. Тестування з датасетом типу md5

По аналогії з першим тестом, запустимо Suricata, у офлайн режимі (використовуючи перехоплені нами пакети у pcap форматі), однак використаємо правило, що використовує md5 датасет. Та розглянемо детальніше вміст fast.log файлу.

```
[andrii@sandbox suricata]$ suricata -r test.pcap -S rules/alter_by_domain_md5.rules
5/12/2022 -- 16:30:11 - <Notice> - This is Suricata version 6.0.8 RELEASE running in U
SER mode
5/12/2022 -- 16:30:11 - <Notice> - Ring buffer initialized with 1 files.
5/12/2022 -- 16:30:11 - <Notice> - all 3 packet processing threads, 4 management threa
ds initialized, engine started.
5/12/2022 -- 16:30:13 - <Notice> - Signal Received.  Stopping engine.
5/12/2022 -- 16:30:29 - <Notice> - Pcap-file module read 1 files, 24677 packets, 17113
630 bytes
[andrii@sandbox suricata]$ cat fast.log | wc -l
24
[andrii@sandbox suricata]$ cat fast.log
11/30/2022-04:31:42.256502  [**] [1:123258:0] This domain name was in domain_md5 [**]
[Classification: (null)] [Priority: 3] {UDP} 192.168.1.101:53497 -> 192.168.1.1:53
11/30/2022-04:31:42.256524  [**] [1:123258:0] This domain name was in domain_md5 [**]
[Classification: (null)] [Priority: 3] {UDP} 192.168.1.101:53497 -> 192.168.1.1:53
11/30/2022-04:31:42.256928  [**] [1:123258:0] This domain name was in domain_md5 [**]
[Classification: (null)] [Priority: 3] {UDP} 192.168.1.101:52570 -> 192.168.1.1:53
11/30/2022-04:31:42.256940  [**] [1:123258:0] This domain name was in domain_md5 [**]
[Classification: (null)] [Priority: 3] {UDP} 192.168.1.101:52570 -> 192.168.1.1:53
11/30/2022-04:31:41.281374  [**] [1:123258:0] This domain name was in domain_md5 [**]
[Classification: (null)] [Priority: 3] {UDP} 192.168.1.101:62228 -> 192.168.1.1:53
```

Рис. 2.1.1.5.1. Запуск suricata та читання результату fast-log

Можемо спостерігати, що відбулося, 24 сповіщення. Даний лог є аналогічним логу для датасету string, за винятком зміненого вихідного повідомлення.

```
[andrii@sandbox suricata]$ ./domain-name-in-file.sh dns-eve.json
history.google.com   2
www.gstatic.com      2
get.geojs.io         2
wordwall.net         2
cdnjs.cloudflare.com 4
```

Рис. 2.1.1.5.2. Отримання кількості доменних імен в eve-log файлі

Застосувавши напередодні написаний нами скрипт, можемо спостерігати, що результат записаний до eve-log є аналогічним до результату отриманому при виконанні правила до датасету string. Тест вважається успішним.

2.1.1.6. Тестування з датасетом типу sha256

По аналогії з двома попередніми тестами, запустимо Suricata, у офлайн режимі (використовуючи перехоплені нами пакети у pcap форматі), однак використаємо правило, що використовує sha256 датасет. Та розглянемо детальніше вміст fast.log файлу.

```
[andrii@sandbox suricata]$ suricata -r test.pcap -S rules/alter_by_domain_sha256.rules

5/12/2022 -- 16:35:47 - <Notice> - This is Suricata version 6.0.8 RELEASE running in U
SER mode
5/12/2022 -- 16:35:47 - <Notice> - Ring buffer initialized with 0 files.
5/12/2022 -- 16:35:47 - <Notice> - all 3 packet processing threads, 4 management threa
ds initialized, engine started.
5/12/2022 -- 16:35:48 - <Notice> - Signal Received.  Stopping engine.
5/12/2022 -- 16:36:05 - <Notice> - Pcap-file module read 1 files, 24677 packets, 17113
630 bytes
[andrii@sandbox suricata]$ cat fast.log | wc -l
24
[andrii@sandbox suricata]$ cat fast.log
11/30/2022-04:31:42.256502  [**] [1:123257:0] This domain name was in domain_sha256 [*
*] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.101:53497 -> 192.168.1.1:53
11/30/2022-04:31:42.256524  [**] [1:123257:0] This domain name was in domain_sha256 [*
*] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.101:53497 -> 192.168.1.1:53
11/30/2022-04:31:41.281374  [**] [1:123257:0] This domain name was in domain_sha256 [*
*] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.101:62228 -> 192.168.1.1:53
11/30/2022-04:31:41.281394  [**] [1:123257:0] This domain name was in domain_sha256 [*
*] [Classification: (null)] [Priority: 3] {UDP} 192.168.1.101:62228 -> 192.168.1.1:53
11/30/2022-04:31:41.282014  [**] [1:123257:0] This domain name was in domain_sha256 [*
```

Рис. 2.1.1.6.1. Запуск suricata та читання результату fast-log

Можемо спостерігати, що відбулося, 24 сповіщення. Даний лог є аналогічним до логів двох попередніх датасетів, за винятком зміненого вихідного повідомлення.

```
[andrii@sandbox suricata]$ ./domain-name-in-file.sh dns-eve.json
history.google.com   2
www.gstatic.com      2
get.geojs.io         2
wordwall.net         2
cdnjs.cloudflare.com 4
```

Рис. 2.1.1.6.2. Отримання кількості доменних імен в eve-log файлі

Застосувавши напередодні написаний нами скрипт, можемо спостерігати, що результат записаний до eve-log є аналогічним до результату отриманому при виконанні попередніх правил. Тест вважається успішним.

2.1.2. Виявлення за відомими IP адресами

Як один з останніх способів ідентифікації трафіку є співставлення використання списків завчасно IP адрес та мереж доступ до яких вказує на використання певного ресурсу. Дана перевірка є затратною, оскільки зазвичай

списки містять велику кількість IP адрес та мереж. Реалізуємо дану перевірку використовуючи інструмент Suricata.

2.1.2.1. Написання програми для створення репутаційного списку

Звичайно, IP репутаційний список, це текстовий файл, що зберігається в простому для розуміння форматі, тому може виникнути сумнів в необхідності створення утиліти, що створює простий репутаційний список на основі списку IP адрес, однак, бувають випадки коли нам необхідно створити репутаційний список для тисяч IP адрес (приклади будуть надаватися в висновку до даного розділу), в таких випадках автоматизація процесу є необхідною. Вихідний код у Додатку Г.

```
E:\Study\Univ\MasterDegree\PythonRuleAuto>python ip-list-to-reputation.py --help
usage: ip-list-to-reputation.py [-h] [--category CATEGORY] [--reputation REPUTATION] [-o OUTPUT] [-f FILE]

options:
  -h, --help            show this help message and exit
  --category CATEGORY   Specify category id for all ip. Default value is 1.
  --reputation REPUTATION
                        Specify reputation score for all ip. Default value is 64.
  -o OUTPUT, --output OUTPUT
                        Specify output file.
  -f FILE, --file FILE  Specify input file.
```

Рис. 2.1.2.1.1. Опис використання утиліти

Утиліта дозволяє працювати не лише з файлами, вона використовує stdin, stdout консолі, якщо вхідний/вихідний файли невказані. Тим самим вона може бути частиною пайплайна:

```
E:\Study\Univ\MasterDegree\PythonRuleAuto>echo 192.168.0.1/24 |  python ip-list-to-reputation.py --reputation 3
 --category 2
192.168.0.1/24,2,3
```

Ри.2.1.2.1.2. Використання утиліти в пайплайні

або можемо вказати вихідний файл, і друкувати текстові дані в консолі:

```
E:\Study\Univ\MasterDegree\PythonRuleAuto>python ip-list-to-reputation.py -o ip-test.csv
192.168.0.1
10.20.30.0/31
23.1.1.0/24
^Z

E:\Study\Univ\MasterDegree\PythonRuleAuto>type ip-test.csv
192.168.0.1,1,64
10.20.30.0/31,1,64
23.1.1.0/24,1,64
```

Ри.2.1.2.1.3. Використання утиліти у режимі вводу даних з консолі

2.1.2.2. Підготовка тестових даних

Для початку зробимо тестову вибірку пакетів, скористаємося програмою Wireshark. Збережемо випадковий трафік, що був зібраний протягом сесії. Розглянемо перехоплений трафік, він складається з 24677 пакетів:

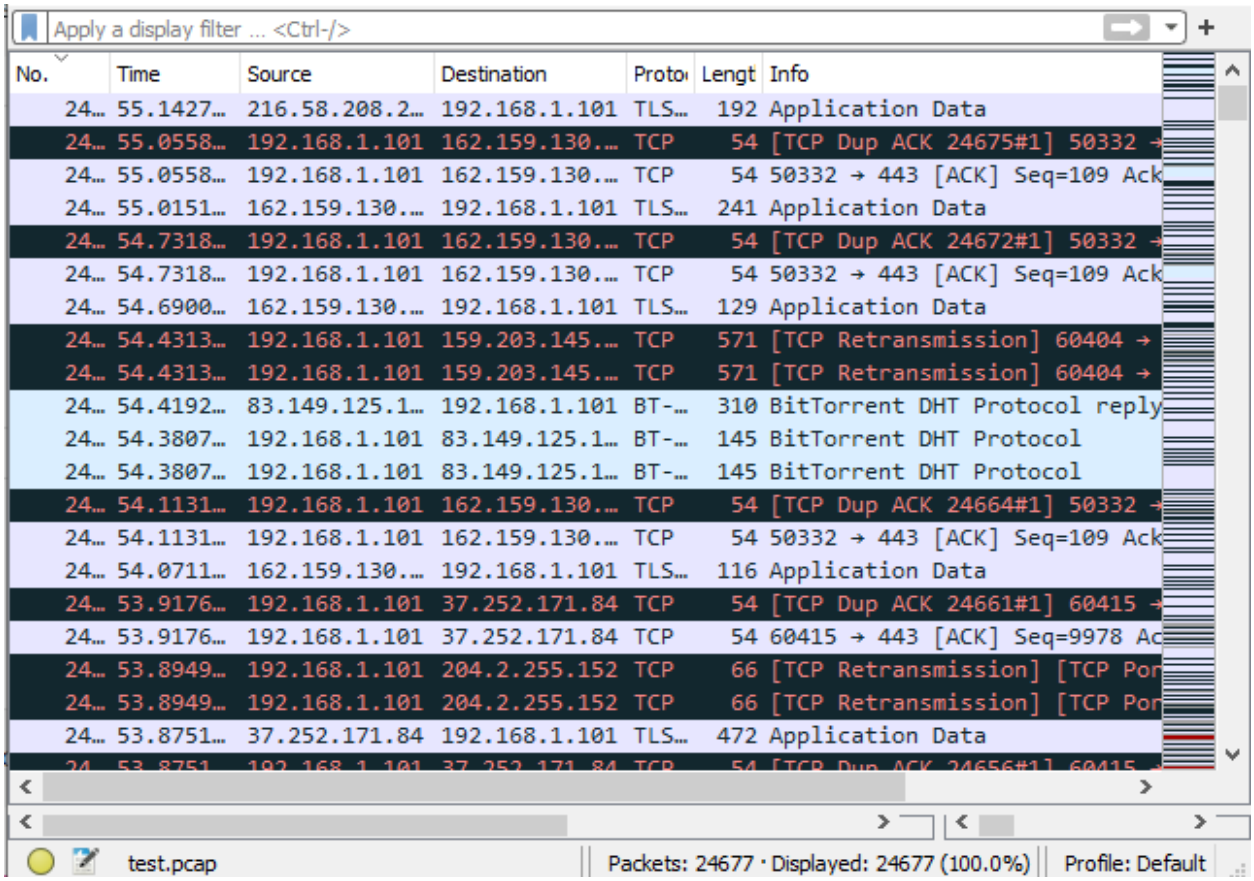

Рис. 2.1.2.2.1. Тестові дані Pcap файлу

Виберемо декілька довільних IP адрес: “2.17.156.123” і “18.66.15.106”, що з’являються протягом файлу, а також добавимо IP мережу “216.58.209.0/24” в тестовій вибірці є звертання до таких IP адрес даної мережі: “216.58.209.3”, “216.58.209.4”, “216.58.209.10” і “216.58.209.14”.

```
[andrii@sandbox suricata]$ cat iptest.list
216.58.209.0/24
2.17.156.123
18.66.15.106
```

Рис. 2.1.2.2.2. Список IP адрес для тестування

Розглянемо скільки всього сесій з IP адресами з даного списку було відкрито протягом тестової вибірки. Як можемо спостерігати, з даними IP серверами було відчинено 21 сесію.

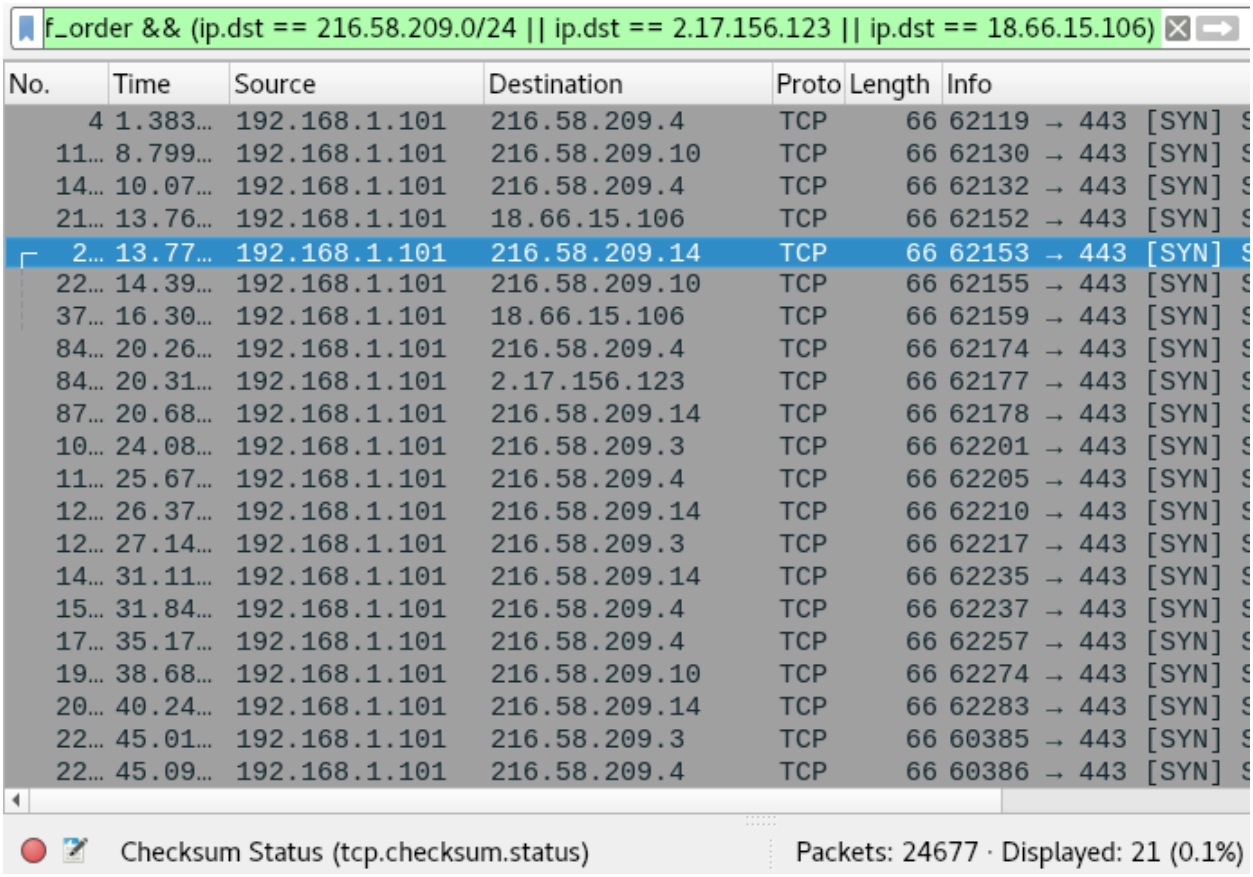

Рис. 2.1.2.2.3. Відкриті сесії для для всіх тестових IP адрес

2.1.2.3. Налаштування Suricata

Для початку створимо репутаційний файл використовуючи нашу створену утиліту, на основі списку IP адрес створеному в попередньому параграфі, задамо категорію 1 і значення репутації 64.

```
[andrii@sandbox suricata]$ python3 ip-list-to-reputation.py -f iptest.list
-o reputation/reputation-test.list --category 1 --reputation 64
[andrii@sandbox suricata]$ cat reputation/reputation-test.list
216.58.209.0/24,1,64
2.17.156.123,1,64
18.66.15.106,1,64
```

Рис. 2.1.2.3.1. Створення тестового репутаційного файлу

Створимо категорійний файл, в якій визначається категорія 1, з назвою testHosts:

```
[andrii@sandbox suricata]$ cat reputation/categories.txt
1,testHosts,Ip reputation test
```

Рис. 2.1.2.3.2. Створення тестового катеріонального файлу

Додамо репутаційний файл і категорійний файл до файлу конфігурації Suricata:

```
reputation-categories-file: /home/andrii/suricata/reputation/categories.txt
default-reputation-path: /home/andrii/suricata/reputation
reputation-files:
 - reputation-test.list
```

Рис. 2.1.2.3.3. Задання тестових даних в suricata.yaml

Створимо правило, що спрацьовує для пакетів у яких відбувається звернення до IP адрес з репутаційного файлу

```
[andrii@sandbox suricata]$ cat rules/alter_by_reputation.rules
alert ip any any -> any any (msg:"Dst IP in testHosts reputation file";
 iprep:dst,testHosts,>,30; sid:12378;)
```

Рис. 2.1.2.3.4. Створення правила, що виявляє з'єдання з репутаційного файлу

2.1.2.4. Тестування виявлення за IP адресами

Запустимо Suricata, у офлайн режимі (використовуючи тестову вибірку в pcap форматі), а також застосуємо правило визначене в попередньому параграфі, а також розглянемо кількість записів в fast-log, і його вміст:

```
[andrii@sandbox suricata]$ suricata -r test.pcap -S rules/alter_by_reputation.rules
5/12/2022 -- 19:02:55 - <Notice> - This is Suricata version 6.0.8 RELEASE running in U
SER mode
5/12/2022 -- 19:02:55 - <Notice> - Ring buffer initialized with 1 files.
5/12/2022 -- 19:02:56 - <Notice> - all 3 packet processing threads, 4 management threa
ds initialized, engine started.
5/12/2022 -- 19:02:59 - <Notice> - Signal Received.  Stopping engine.
5/12/2022 -- 19:03:12 - <Notice> - Pcap-file module read 1 files, 24677 packets, 17113
630 bytes
[andrii@sandbox suricata]$ cat fast.log | wc -l
21
[andrii@sandbox suricata]$ cat fast.log
11/30/2022-04:31:36.225301  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62119 -> 216.58.209.4:443
11/30/2022-04:31:44.914306  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62132 -> 216.58.209.4:443
11/30/2022-04:31:43.641902  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62130 -> 216.58.209.10:443
11/30/2022-04:31:48.612132  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62153 -> 216.58.209.14:443
11/30/2022-04:31:48.611285  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
```

Рис. 2.1.2.4.1. Запуск suricata та читання результату fast-log

Ми можемо спостерігати, що як і очікувалося в даному файлі збереглося 21 записи, для кожної відкритої сесії з репутаційного списку.

Для підтвердження пройдемося по кожній з IP адрес по окремості аби підтвердити правильність виконання:

- “2.17.156.123” – можемо спостерігати, усього 1 сесія встановлена була до даного сервісу і вона була успішно записана в fast-log

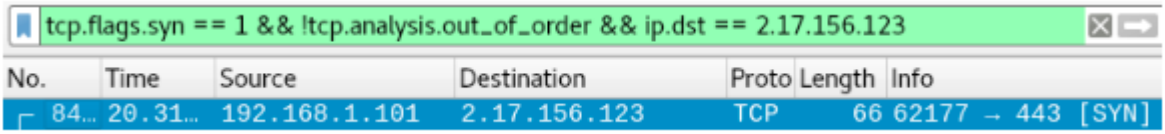

Рис. 2.1.2.4.2. З’єднання для IP адерси 2.17.156.123

```
[andrii@sandbox suricata]$ cat fast.log | grep 2.17.156.123 | wc -l
1
[andrii@sandbox suricata]$ cat fast.log | grep 2.17.156.123
11/30/2022-04:31:55.152618  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62177 -> 2.17.156.123:443
```

Рис. 2.1.2.4.3. Виявлені з’єднання для IP адерси 2.17.156.123

- “18.66.15.106” – можемо спостерігати, до даного сервісу з’єднання було встановлено двічі, що було успішно отримано в fast-log

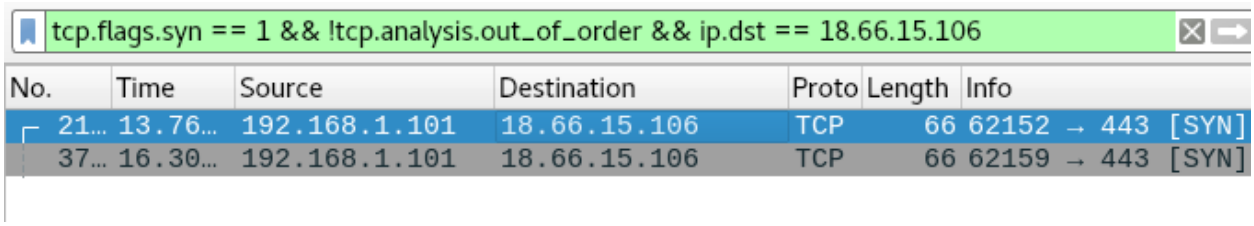

Рис. 2.1.2.4.4. З’єднання для IP адерси 18.66.15.106

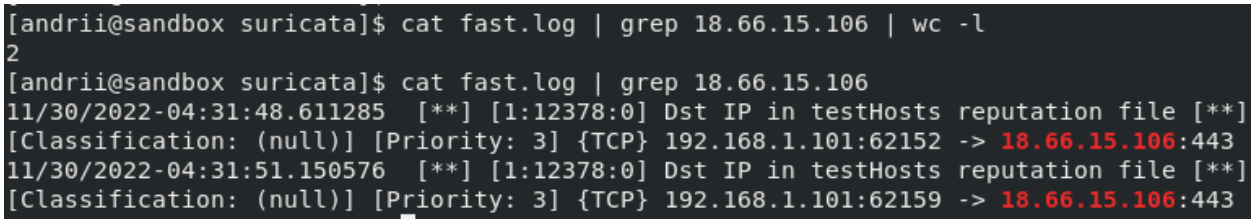

Рис. 2.1.2.4.5. Виявлені з’єднання для IP адерси 18.66.15.106

- “216.58.209.0/24” – можемо спостерігати для даної мережі було встановлено з’єднання 18 разів, що і було отримано в fast-log. Для пересвідчення достовірності збереження даних про доступ до ресурсів даної IP мережі, розглянемо запис для кожної IP адреси даної мережі поокремості:

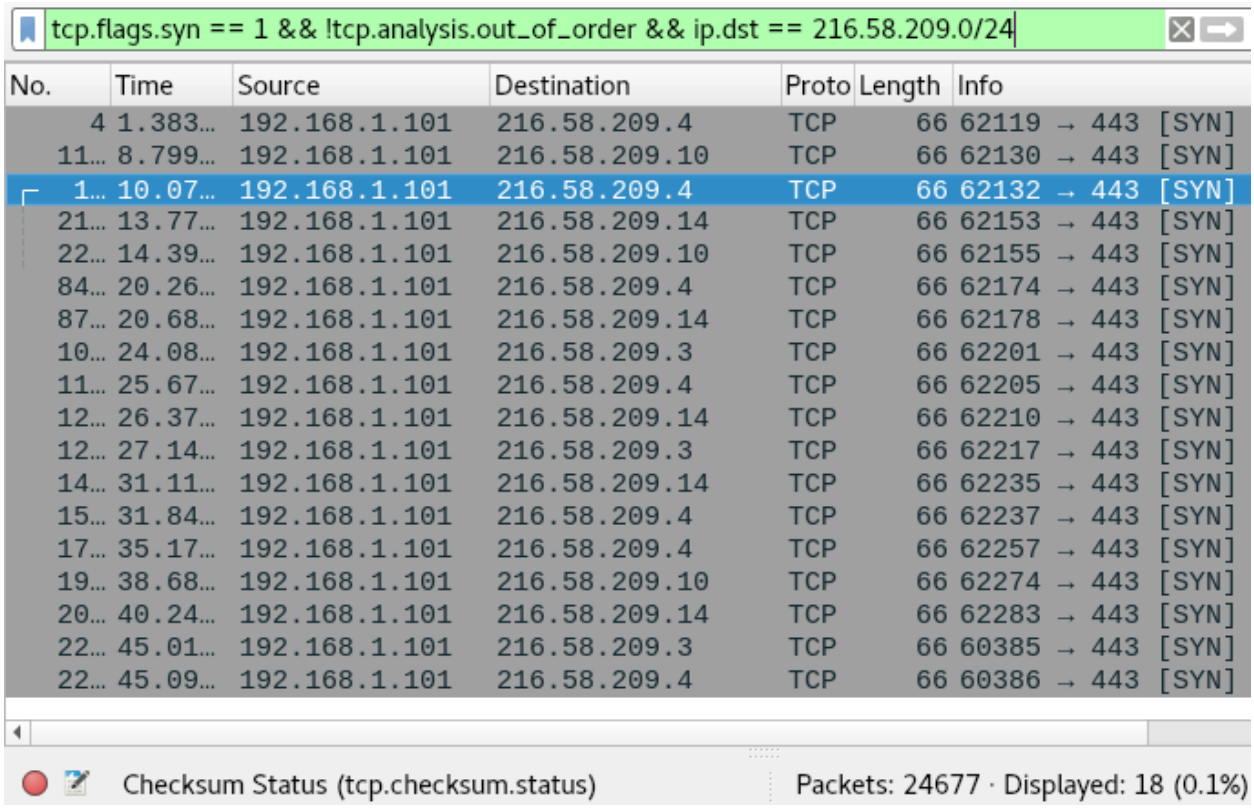

Рис. 2.1.2.4.6. З'єднання для IP мережі 216.58.209.0/24

```
[andrii@sandbox suricata]$ cat fast.log | grep 216.58.209 | wc -l
18
```

Рис. 2.1.2.4.7. Кількість виявлених з'єднання для IP мережі 216.58.209.0/24

- "216.58.209.3" – можемо спостерігати усього було 3 з'єднання, і всі вони були збережені до fast-log

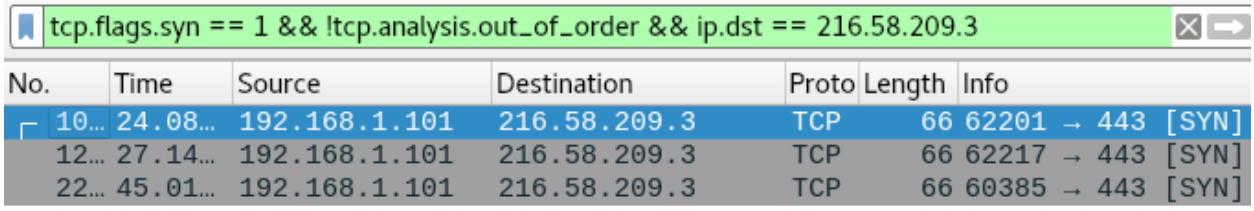

Рис. 2.1.2.4.8. З'єднання для IP адерси 216.58.209.3

```
[andrii@sandbox suricata]$ cat fast.log | grep 216.58.209.3 | wc -l
3
[andrii@sandbox suricata]$ cat fast.log | grep 216.58.209.3
11/30/2022-04:31:58.926923  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62201 -> 216.58.209.3:443
11/30/2022-04:32:01.984694  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62217 -> 216.58.209.3:443
11/30/2022-04:32:19.857818  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:60385 -> 216.58.209.3:443
```

Рис. 2.1.2.4.9. Виявлені з'єднання для IP адерси 216.58.209.3

- "216.58.209.4" – можемо спостерігати усього було 7 з'єднання, і всі вони були збережені до fast-log

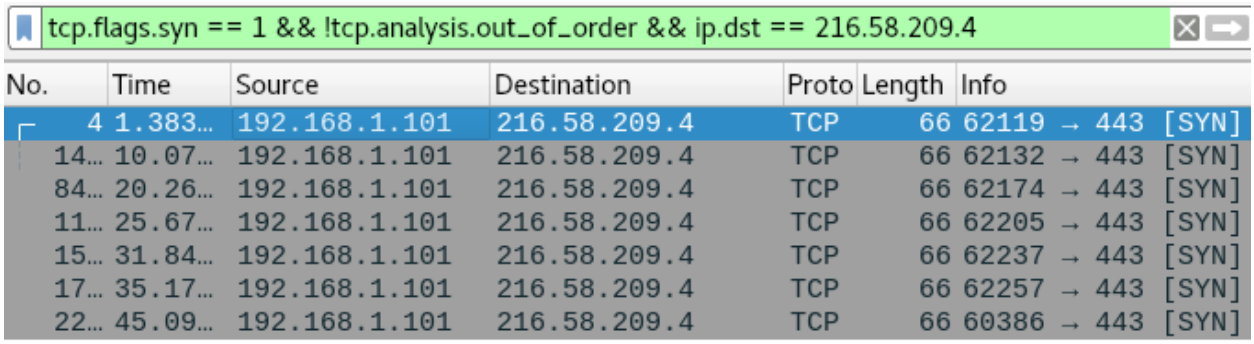

Рис. 2.1.2.4.10. З'єднання для IP адерси 216.58.209.4

```
[andrii@sandbox suricata]$ cat fast.log | grep 216.58.209.4 | wc -l
7
[andrii@sandbox suricata]$ cat fast.log | grep 216.58.209.4
11/30/2022-04:31:36.225301  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62119 -> 216.58.209.4:443
11/30/2022-04:31:44.914306  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62132 -> 216.58.209.4:443
11/30/2022-04:32:00.520063  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62205 -> 216.58.209.4:443
11/30/2022-04:31:55.105907  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62174 -> 216.58.209.4:443
11/30/2022-04:32:06.687613  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62237 -> 216.58.209.4:443
11/30/2022-04:32:10.019878  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:62257 -> 216.58.209.4:443
11/30/2022-04:32:19.940443  [**] [1:12378:0] Dst IP in testHosts reputation file [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.101:60386 -> 216.58.209.4:443
```

Рис. 2.1.2.4.11. Виявлені з'єднання для IP адерси 216.58.209.4

- "216.58.209.10" – можемо спостерігати усього було 3 з'єднання, і всі вони були збережені до fast-log

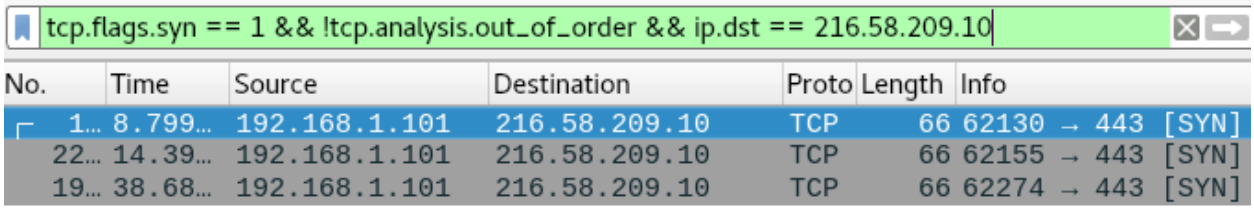

Рис. 2.1.2.4.12. З'єднання для IP адерси 216.58.209.10

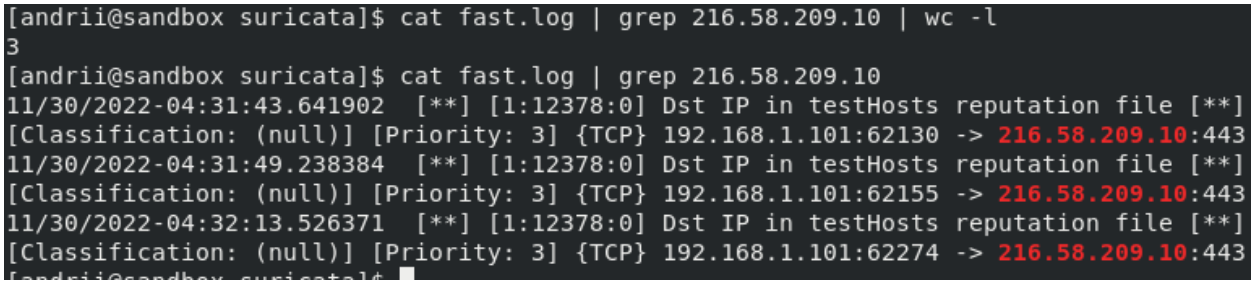

Рис. 2.1.2.4.13. Виявлені з'єднання для IP адерси 216.58.209.10

- "216.58.209.14" – можемо спостерігати усього було 5 з'єднання, і всі вони були збережені до fast-log

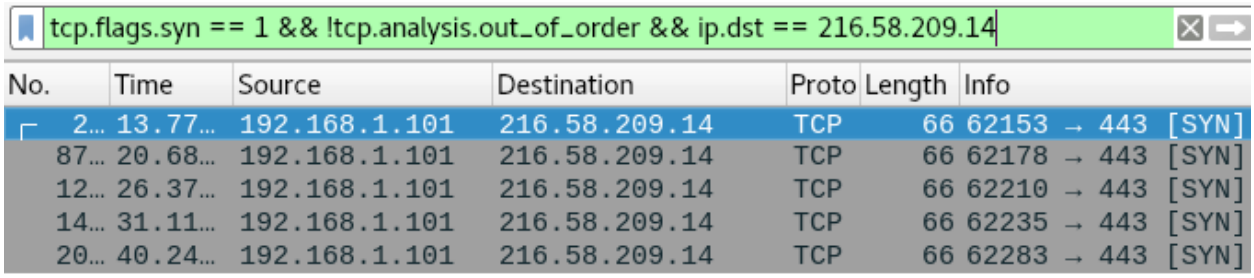

Рис. 2.1.2.4.14. З'єднання для IP адерси 216.58.209.14

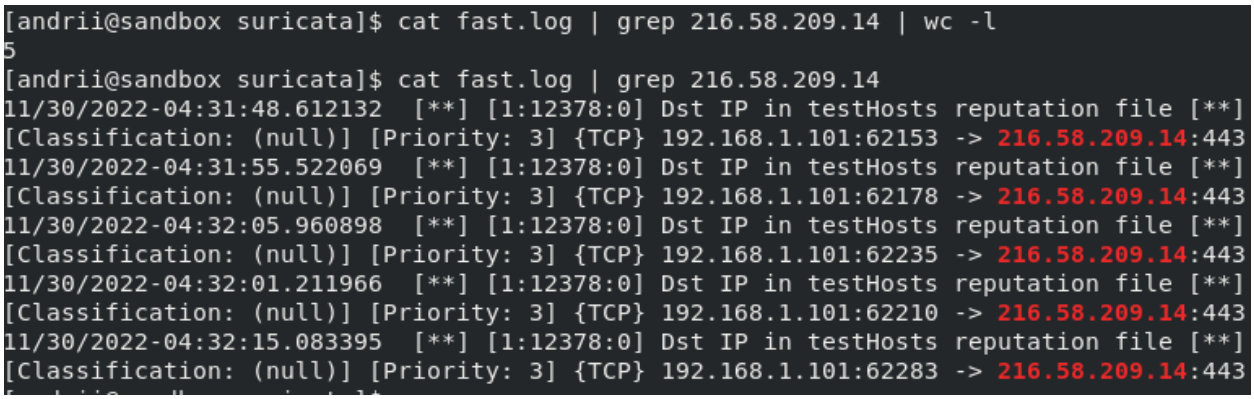

Рис. 2.1.2.4.15. Виявлені з'єднання для IP адерси 216.58.209.14

2.1.2.5. Результат тестування

Можемо спостерігати, що всі записи були збережені в fast-log файлі, тому тест вважається успішним.

2.1.3. Результати виявлення доступу до відомих сервісів

Дані методи можна вважати в загальному універсальним підходом до виявлення мережевої активності, особливо це стосується тих випадків коли можуть використовуватися безліч різних протоколів, навіть можуть розроблятися нові протоколи. Одним з прикладів де відслідкування DNS запитів і використання IP репутаційних списків може використовуватися на практиці це відслідкування VPN та Mining трафіків.

Дана задача для цих областей не є новою, на даний момент, існує безліч відкритих збірників списків IP адрес, та DNS імен, що допомагають у відслідкуванні VPN та Mining трафіків, розглянемо деякі з них.

2.2. Детектування VPN трафіку

Розглянувши теоретичну частину, було вирішено, використовувати підходити детектування:

- за IP адресами та DNS іменами загальновідомих VPN серверів
- за сигнатурами протоколів специфічних для VPN технологій.

Розглянемо детальніше реалізацію кожного з підходів в розділах нижче.

2.2.1. Детектування за загальновідомими IP адресами і DNS іменами

Розглянемо декілька публічних проектів, що зберігають в собі дані про IP адреси VPN серверів:

- X4BNet/lists_vpn [18.] – публічний репозиторій, що містить IP адреси загальновідомих VPN серверів

2.2.2. Детектування за сигнатурами протоколу

Було вирішено використовуючи сигнатури протоколів для визначення протоколів WireGuard та OpenVPN. Оскільки протоколи TLS і IPSec використовуються для шифрування трафіку, та не ідентифікують VPN трафік. А протокол Lightweight немає загальної структури повідомлення, а дані що зберігаються в його полях зберігають дані, що не дозволяють ідентифікувати його за сигнатурами.

2.2.2.1. Детектування протоколу WireGuard

Нагадаємо структуру протоколу WireGuard:

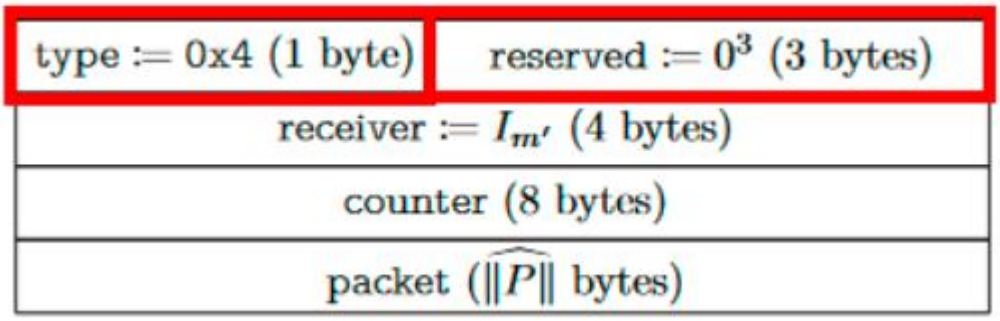

Рис. 2.2.2.1.1. Структура WireGuard повідомлення, з виділеними статичними полями

Розглянувши структуру WireGuard протоколу можемо спостерігати, що всі повідомлення даного протоколу розпочинаються з 2 полів, що мають поля, що

мають фіксовані значення, тому було вирішено виявляти трафік, що розпочинається з даних значень.

Даний підхід було обрано оскільки він не потребує зберігання сесії, а також потребує лише операцій порівняння, що забезпечує високу швидкодію.

Недоліком даного підходу є те, що є можливим хибне спрацювання у випадках коли дані випадкового пакету співпали з тими, що перевіряється. Однак, оскільки виявляється кожен пакет протоколу WireGuard вважається за виявлення використання даного протоколу при не однократному спрацювання правила в межах однієї сесії, а одиночні спрацювання вважати помилковими.

Lua скрипт, що використовується для виявлення даного протоколу знаходиться в Додатку Д.

Тестування:

Для тестування скористаємося прикладом мережевого трафіку, опублікований в документації Wireshark: [18.], файл wireguard-ping-tcp.pcap, заданий файл містить 22 wireguard пакети:

No.	Time	Source	Destination	Proto	Lengt	Info
1	0.000000	10.9.0.1	10.9.0.2	Wir…	190	Handshake Initiation, sender=0x30D037D8
2	0.002518	10.9.0.2	10.9.0.1	Wir…	134	Handshake Response, sender=0xAB7DF406, receiver=0x30D037D8
3	0.002850	10.9.0.1	10.9.0.2	Wir…	170	Transport Data, receiver=0xAB7DF406, counter=0, datalen=96
4	0.003071	10.9.0.2	10.9.0.1	Wir…	170	Transport Data, receiver=0x30D037D8, counter=0, datalen=96
5	1.007113	10.9.0.1	10.9.0.2	Wir…	170	Transport Data, receiver=0xAB7DF406, counter=1, datalen=96
6	1.007446	10.9.0.2	10.9.0.1	Wir…	170	Transport Data, receiver=0x30D037D8, counter=1, datalen=96
7	2.010695	10.9.0.1	10.9.0.2	Wir…	170	Transport Data, receiver=0xAB7DF406, counter=2, datalen=96
8	2.011037	10.9.0.2	10.9.0.1	Wir…	170	Transport Data, receiver=0x30D037D8, counter=2, datalen=96
9	12.1870…	10.9.0.1	10.9.0.2	Wir…	74	Keepalive, receiver=0xAB7DF406, counter=3
10	140.092…	10.9.0.2	10.9.0.1	Wir…	138	Transport Data, receiver=0x30D037D8, counter=3, datalen=64
11	140.093…	10.9.0.2	10.9.0.1	Wir…	138	Transport Data, receiver=0x30D037D8, counter=4, datalen=64
12	140.133…	10.9.0.1	10.9.0.2	Wir…	154	Transport Data, receiver=0xAB7DF406, counter=4, datalen=80
13	140.133…	10.9.0.1	10.9.0.2	Wir…	190	Handshake Initiation, sender=0xBFFD41C5
14	140.139…	10.9.0.2	10.9.0.1	Wir…	134	Handshake Response, sender=0x26DE9EB9, receiver=0xBFFD41C5
15	140.139…	10.9.0.1	10.9.0.2	Wir…	170	Transport Data, receiver=0xAB7DF406, counter=5, datalen=96
16	140.139…	10.9.0.1	10.9.0.2	Wir…	74	Keepalive, receiver=0x26DE9EB9, counter=0
17	140.190…	10.9.0.2	10.9.0.1	Wir…	138	Transport Data, receiver=0xBFFD41C5, counter=0, datalen=64
18	140.229…	10.9.0.1	10.9.0.2	Wir…	122	Transport Data, receiver=0x26DE9EB9, counter=1, datalen=48
19	140.231…	10.9.0.2	10.9.0.1	Wir…	122	Transport Data, receiver=0xBFFD41C5, counter=1, datalen=48
20	140.231…	10.9.0.2	10.9.0.1	Wir…	330	Transport Data, receiver=0xBFFD41C5, counter=2, datalen=256
21	140.231…	10.9.0.1	10.9.0.2	Wir…	122	Transport Data, receiver=0x26DE9EB9, counter=2, datalen=48
22	140.277…	10.9.0.1	10.9.0.2	Wir…	1494	Transport Data, receiver=0x26DE9EB9, counter=3, datalen=1420

Рис. 2.2.2.1.2. Тестові дані

Запустимо дане правило для даних тестових даних, та переглянемо результат:

```
E:\Study\Univ\MasterDegree\Suricata>suricata -c suricata.yaml -S testRule\wireguard.rules
-r Templates\wireguard-ping-tcp.pcap
9/4/2023 -- 22:08:34 - <Info> - Running as service: no
9/4/2023 -- 22:08:34 - <Notice> - This is Suricata version 6.0.9 RELEASE running in USER m
ode
9/4/2023 -- 22:08:34 - <Notice> - all 5 packet processing threads, 4 management threads in
itialized, engine started.
9/4/2023 -- 22:08:34 - <Notice> - Signal Received.  Stopping engine.
9/4/2023 -- 22:08:35 - <Notice> - Pcap-file module read 1 files, 22 packets, 4744 bytes
```

Рис. 2.2.2.1.3. Тестовий запуск

Розглянувши результат, можемо спостерігати, що було виявлено усі 22 пакети, що були згруповані по сесіям у обидві сторони.

	#	Timestamp	Src / Dst	Signature
▶ ☆	12	2018-07-21 01:41:01 5 years ago	S: 10.9.0.1 D: 10.9.0.2	VPN: Signature of Wireguard protocol.
☆	10	2018-07-21 01:41:01 5 years ago	S: 10.9.0.2 D: 10.9.0.1	VPN: Signature of Wireguard protocol.

Рис. 2.2.2.1.4. Результат тестування 1 етапу

Тестування 2 етап:

Запустимо правило, для підготовлених завчасно тестових даних, що не містять порушень правил:

```
E:\Study\Univ\MasterDegree\Suricata>suricata -c suricata.yaml -S testRule\wireguard.rules
-r Templates\original.pcapng
9/4/2023 -- 22:11:08 - <Info> - Running as service: no
9/4/2023 -- 22:11:08 - <Notice> - This is Suricata version 6.0.9 RELEASE running in USER m
ode
9/4/2023 -- 22:11:08 - <Notice> - all 5 packet processing threads, 4 management threads in
itialized, engine started.
9/4/2023 -- 22:11:40 - <Notice> - Signal Received.  Stopping engine.
9/4/2023 -- 22:11:42 - <Notice> - Pcap-file module read 1 files, 787160 packets, 867648140
 bytes

E:\Study\Univ\MasterDegree\Suricata>type fast.log

E:\Study\Univ\MasterDegree\Suricata>
```

Рис. 2.2.2.1.5. Запуск та перевірка відсутності спрацювання тестування 2 етапу

Оскільки fast.log файл є пустим, це означає, що задане правило не спрацювало, отже тест пройдено успішно.

2.2.2.2. Детектування протоколу OpenVPN

Нагадаємо структуру протоколу OpenVPN:

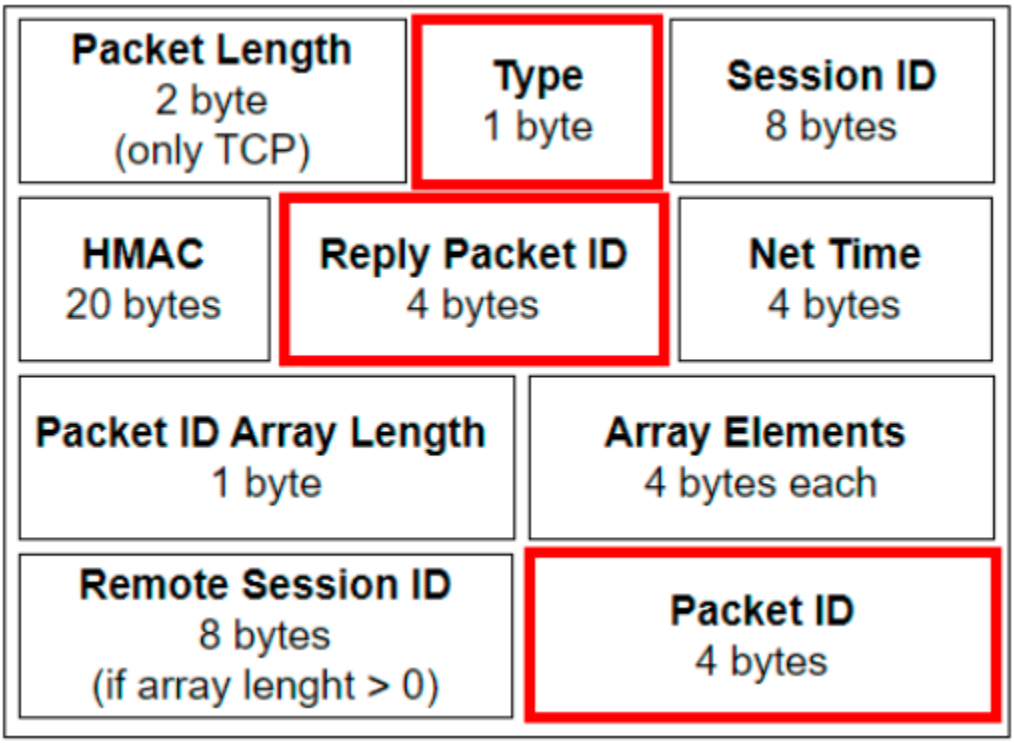

Рис. 2.2.2.2.1. Структура OpenVPN повідомлення, з виділеними статичними полями

Можемо спостерігати, єдиним полем даного протоколу, що містить фіксовані значення це поле Type, оскільки дане поле має розмір 1 байт це унеможливлює фільтрування пакетів лише з значеннями даного поля. Оскільки відслідкування кожної сесії є затратним за часом та ресурсами, було вирішено

детектувати 1 пакет OpenVPN протоколу, у якому: поле Reply Packet ID має значення 1, поле Packet ID має значення 0.

Даний підхід має недолік, а саме можливе хибне спрацювання у випадках коли дані випадково співпали з правилом. Оскільки даний підхід спрацьовує лише для першого пакету встановленої сесії враховуючи можливе спрацювання для випадкового трафіку, воно може ідентифікувати даний протокол у випадках систематичного спрацювання, у інших випадках воно може вказувати на необхідність власноручного перегляду мережевої активності даного користувача.

Lau скрипт даного правила знаходиться в Додатку Е.

Тестування: Тестування для TCP

Для тестування скористаємося прикладом мережевого трафіку, опублікований в документації Wireshark: [19.], файл OpenVPN_TCP_tls-auth.pcapng, заданий файл містить 438 пакетів, серед яких 4 пакети, що використовуються для відкриття OpenVPN сесії:

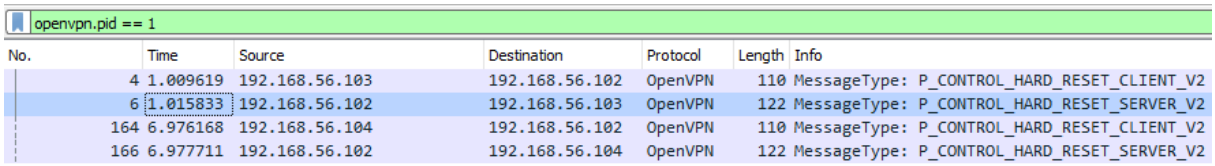

Рис. 2.2.2.2.2. Тестові дані для TCP

Запустимо задане правило для даних тестових даних:

```
E:\Study\Univ\MasterDegree\Suricata>suricata -c suricata.yaml -S testRule\openvpn.rules -r
 Templates\OpenVPN_TCP_tls-auth.pcapng
9/4/2023 -- 22:40:29 - <Info> - Running as service: no
9/4/2023 -- 22:40:29 - <Notice> - This is Suricata version 6.0.9 RELEASE running in USER m
ode
9/4/2023 -- 22:40:30 - <Notice> - all 5 packet processing threads, 4 management threads in
itialized, engine started.
9/4/2023 -- 22:40:30 - <Notice> - Signal Received.  Stopping engine.
9/4/2023 -- 22:40:30 - <Notice> - Pcap-file module read 1 files, 438 packets, 72149 bytes
```

Рис. 2.2.2.2.3. Запуск тесту 1 етапу для TCP

Розглянемо результат:

	#	Timestamp	Src / Dst	Signature
☆	1	2013-01-23 00:30:11 10 years ago	S: 192.168.56.102 D: 192.168.56.104	VPN: Signature of OpenVpn protocol.
☆	1	2013-01-23 00:30:11 10 years ago	S: 192.168.56.104 D: 192.168.56.102	VPN: Signature of OpenVpn protocol.
☆	1	2013-01-23 00:30:05 10 years ago	S: 192.168.56.102 D: 192.168.56.103	VPN: Signature of OpenVpn protocol.
☆	1	2013-01-23 00:30:05 10 years ago	S: 192.168.56.103 D: 192.168.56.102	VPN: Signature of OpenVpn protocol.

Рис. 2.2.2.2.4. Результат тесту 1 етапу для TCP

Розглянувши результат, можемо спостерігати, що було виявлено усі 4 пакети.

Тестування: Тестування для UDP

Для тестування скористаємося прикладом мережевого трафіку, опублікований в документації Wireshark: [19.], файл OpenVPN_UDP_tls-auth.pcapng, заданий файл містить 440 пакетів, серед яких 4 пакети, що використовуються для відкриття OpenVPN сесії:

openvpn.pid == 1

No.	Time	Source	Destination	Protocol	Lengt	Info
1	0.000000	192.168.56.103	192.168.56.102	OpenVPN	84	MessageType: P_CONTROL_HARD_RESET_CLIENT_V2
2	0.003189	192.168.56.102	192.168.56.103	OpenVPN	96	MessageType: P_CONTROL_HARD_RESET_SERVER_V2
183	7.446031	192.168.56.104	192.168.56.102	OpenVPN	84	MessageType: P_CONTROL_HARD_RESET_CLIENT_V2
184	7.447372	192.168.56.102	192.168.56.104	OpenVPN	96	MessageType: P_CONTROL_HARD_RESET_SERVER_V2

Рис. 2.2.2.2.5. Тестові дані для UDP

Запустимо задане правило для даних тестових даних:

```
E:\Study\Univ\MasterDegree\Suricata>suricata -c suricata.yaml -S testRule\openvpn.rules
-r Templates\OpenVPN_UDP_tls-auth.pcapng
9/4/2023 -- 23:01:55 - <Info> - Running as service: no
9/4/2023 -- 23:01:55 - <Notice> - This is Suricata version 6.0.9 RELEASE running in USER
 mode
9/4/2023 -- 23:01:55 - <Notice> - all 5 packet processing threads, 4 management threads
initialized, engine started.
00

9/4/2023 -- 23:01:55 - <Notice> - Signal Received.  Stopping engine.
9/4/2023 -- 23:01:55 - <Notice> - Pcap-file module read 1 files, 440 packets, 61544 byte
s
```

Рис. 2.2.2.2.6. Запуск тесту 1 етапу для UDP

Розглянемо результат:

	#	Timestamp	Src / Dst	Signature
☆	1	2013-01-23 00:52:19 10 years ago	S: 192.168.56.102 D: 192.168.56.104	VPN: Signature of OpenVpn protocol.
☆	1	2013-01-23 00:52:19 10 years ago	S: 192.168.56.104 D: 192.168.56.102	VPN: Signature of OpenVpn protocol.
☆	1	2013-01-23 00:52:12 10 years ago	S: 192.168.56.102 D: 192.168.56.103	VPN: Signature of OpenVpn protocol.
☆	1	2013-01-23 00:52:12 10 years ago	S: 192.168.56.103 D: 192.168.56.102	VPN: Signature of OpenVpn protocol.

Рис. 2.2.2.2.7. Результат тесту 2 етапу для UDP

Розглянувши результат, можемо спостерігати, що було виявлено усі 4 пакети.

Тестування 2 етап:

Запустимо правила для UDP та TCP одночасно, для підготовлених завчасно тестових даних, що не містять порушень правил:

```
E:\Study\Univ\MasterDegree\Suricata>suricata -c suricata.yaml -S testRule\openvpn.rules
-r Templates\original.pcapng
9/4/2023 -- 23:06:55 - <Info> - Running as service: no
9/4/2023 -- 23:06:55 - <Notice> - This is Suricata version 6.0.9 RELEASE running in USER
 mode
9/4/2023 -- 23:06:55 - <Notice> - all 5 packet processing threads, 4 management threads
initialized, engine started.
9/4/2023 -- 23:07:29 - <Notice> - Signal Received.  Stopping engine.
9/4/2023 -- 23:07:31 - <Notice> - Pcap-file module read 1 files, 787160 packets, 8676481
40 bytes

E:\Study\Univ\MasterDegree\Suricata>type fast.log

E:\Study\Univ\MasterDegree\Suricata>
```

Рис. 2.2.2.2.8. Запуск та перевірка відсутності спрацювання тестування 2 етапу

Оскільки fast.log файл є пустим, це означає, що задане правило не спрацювало, отже тест пройдено успішно.

2.3. Детектування шифрованого DNS трафіку

Для детектування DNS over TLS, скористаємося тим, що даний протокол звертається до серверу за портом 853 використовуючи протокол TLS.

Тим самим відповідне правило:

```
alert tls any any -> any 853 ( \
    msg: "Potential usage of DNS over TLS (DoT)"; \
    sid:218763; \
)
```

Рис. 2.3.1. Правило для виокремлення DNS over TLS

Для детектування DNS over QUIC, скористаємося тим, що даний протокол звертається до серверу за протоколом 853 з протоколом транспортного рівня UDP.

Тим самим відповідне правило:

```
alert quic any any -> any 853 ( \
    msg: "Potential usage of DNS over QUIC (DoQ)"; \
    sid:218764; \
)
```

Рис. 2.3.2. Правило для виокремлення DNS over QUIC

Для детектування DNS over HTTPS, немає засобу гарантованого визначення використовуючи лише сигнатури протоколу оскільки даний протокол побудований на основі використання веб технологій, а отже і його неможливо відрізнити від нього. Тому запропонованим варіантом визначення DNS over HTTPS є детектування звернення до веб серверів загальновідомих провайдерів послуг використовуючи протоколи TLS, QUIC з протком 443. Власне оскільки дане правило не гарантує того, що звернення було до DNS over HTTPS, а не до веб сайту даного провайдеру воно показує на використання DNS over HTTPS лише при систематичному спрацюванні.

Відповідно правила, що визначають звернення до Cloudflare сервісу:

```
alert tls any any -> [1.1.1.1, 1.0.0.1, 1.1.1.2, 1.0.0.2, 1.1.1.3, 1.0.0.3] 443 ( \
    msg: "Secure DNS: DoH: Web traffic to Cloudflare, possible usage of DNS over HTTPS (DoH).
    sid:218765; \
)

alert udp any any -> [1.1.1.1, 1.0.0.1, 1.1.1.2, 1.0.0.2, 1.1.1.3, 1.0.0.3] 443 ( \
    msg: "Secure DNS: DoH: Web traffic to Cloudflare, possible usage of DNS over HTTPS (DoH).
    sid:218766; \
)
```

Рис. 2.3.3. Правило для виокремлення DNS over HTTPS, для Cloudflare

Варто зазначити, що у другому правилі варто використовувати quic замість udp, підтримка використання якого з’явиться у Suricata версії 7 та вище. Однак на момент написання роботи, існує лише бета версія Suricata 7 версії.

Розглянемо список загальновідомих провайдерів DoH:

- Cloudflare – з IP адресами: 1.1.1.1, 1.0.0.1, 1.1.1.2, 1.0.0.2, 1.1.1.3, 1.0.0.3
- Google public DNS – з IP адресами: 8.8.4.4, 8.8.8.8
- CleanBrowsing – з IP адресами: 185.228.169.168, 185.228.168.168
- Adguard – з IP адресами: 176.103.130.130, 176.103.130.131
- Quad9 – з IP адресами: 9.9.9.9, 149.112.112.112
- OpenDNS – з IP адресами: 208.67.222.222, 208.67.220.220

2.4. Детектування TOR трафіку

Для детектування TOR трафіку скористаємося списками IP адрес зовнішніх вузлів мережі, даний список публікується провайдером послуг TOR [20.], відповідно будемо виявляти звернення до вузлів, з IP адресами до даних серверів.

2.5. Детектування Майнинг трафіку

2.5.1. Детектування за відомими IP адресами та DNS іменами

Розглянемо декілька публічних проектів, що зберігають в собі дані про IP адреси і Доменні імена загальновідомих вузлів, що використовуються в майнинг мережах:

- CoinBlockerLists[21.] – публічний репозиторій що зберігає в собі доменні імена загальновідомих вузлів майнинг мереж
- Crypto mining pools aggregator [22.] – публічний репозиторій що зберігає в собі доменні імена і IP адреси загальновідомих вузлів майнинг мереж

2.5.2. Детектування Bitcoin протоколу

Для детектування протоколу Bitcoin використаємо той факт, що значення його перших 2 полів є незмінними, а саме Magic поле має 5 варіантів, а Command поле має 35 значень.

На превеликий жаль, використовуючи використовуючи синтаксис правил в Suricata неможливо вказати перевірювати частину байтового пакету, на приналежність одному з декількох варіантів, можна лише порівняти байтову область використовуючи context та перевіряючи 1 значення. Оскільки перевіряється значення 2 полів, це означає, що нам необхідно написати 165 правил. Замість цього напишемо Lua скрипт, який буде перевіряти чи отриманий payload починається з відповідних значень. Код даного скрипта додано до Додатку Ж.

Напишемо правило, що використовує даний Lua скрипт:

```
[andrii@sandbox suricata]$ cat rules/alter_by_bitcoin_proto.rules
alert tcp any any -> any [8333, 3333] (msg: "This packet could be bitcoin mining";
 lua:/home/andrii/suricata/rules/bitcoin_proto.lua; sid: 1248326;)
```

Рис. 2.5.2.1. Правило, що виявляє bitcoin протокол

Як джерело тестових даних використаємо приклад, мережевого трафіку Bitcoin протоколу, що знаходиться в проекті bitcoin-labs/bitcoin-details [23.]. Відкриємо тестову вибірку, використовуючи програму Wireshark. Можемо спостерігати, фільтруючи пакети по протоколу Bitcoin, отримано усього 355 пакетів.

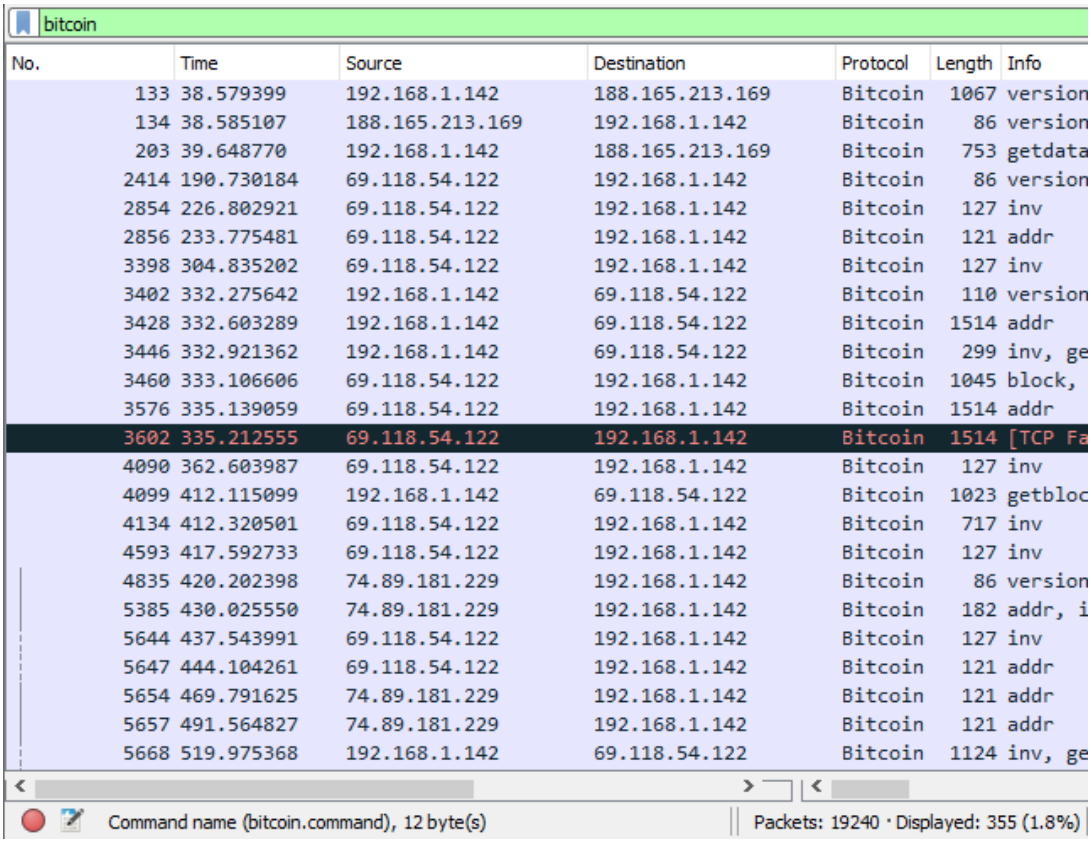

Рис. 2.5.2.2. Тестові дані Pcap файлу

Запустимо Suricata в режимі обробки даної тестової вибірки, та переглянемо результат:

```
[andrii@sandbox suricata]$ suricata -r bitcoin-capture.pcap -S rules/alter_by_bitcoin_pr
oto.rules
7/12/2022 -- 05:46:07 - <Notice> - This is Suricata version 6.0.8 RELEASE running in USE
R mode
7/12/2022 -- 05:46:07 - <Notice> - Ring buffer initialized with 1 files.
7/12/2022 -- 05:46:07 - <Notice> - all 3 packet processing threads, 4 management threads
 initialized, engine started.
7/12/2022 -- 05:46:10 - <Notice> - Signal Received.  Stopping engine.
7/12/2022 -- 05:46:19 - <Notice> - Pcap-file module read 1 files, 19240 packets, 1558176
2 bytes
[andrii@sandbox suricata]$ cat fast.log | wc -l
205
```

Рис. 2.5.2.3. Запуск Suricata та перегляд кількості виявлених даних

```
[andrii@sandbox suricata]$ head fast.log
03/28/2011-11:58:57.725033  [**] [1:1248326:0] This packet could be bitcoin mining [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.142:55317 -> 188.165.213.169:8333
03/28/2011-11:58:57.931550  [**] [1:1248326:0] This packet could be bitcoin mining [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.142:55317 -> 188.165.213.169:8333
03/28/2011-11:58:58.587289  [**] [1:1248326:0] This packet could be bitcoin mining [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.142:55317 -> 188.165.213.169:8333
03/28/2011-12:01:29.970465  [**] [1:1248326:0] This packet could be bitcoin mining [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.142:55328 -> 69.118.54.122:8333
03/28/2011-12:03:51.627793  [**] [1:1248326:0] This packet could be bitcoin mining [**]
[Classification: (null)] [Priority: 3] {TCP} 192.168.1.142:55328 -> 69.118.54.122:8333
```

Рис. 2.5.2.4. Перші 10 строчок файлу fast-log

Можемо спостерігати до fast-log файлу було додано 205 записів. Існує декілька причин чому у нас вийшло менше перехопити пакетів:

- Деякі з пакетів, що передавалися в ході однієї сесії були показані як окремі пакети
- Тестові дані зібрані за 2002 рік, з того часу деякі типи команд, були застарілими, відповідно не підтримується в даний момент

Однак, робота в даному напрямку потребує доопрацювання.

Зробимо перевірку, чи наший метод, не буде видавати звичайний трафік за трафік, що використовує протокол Bitcoin, для цього запустимо Suricata використовуючи тестові дані, що використовувалися для тестування визначення DNS запитів, по списку доменних імен в параграфі 2.1.1.2

```
[andrii@sandbox suricata]$ suricata -r test.pcap -S rules/alter_by_bitcoin_proto.rules
7/12/2022 -- 05:52:20 - <Notice> - This is Suricata version 6.0.8 RELEASE running in USE
R mode
7/12/2022 -- 05:52:20 - <Notice> - Ring buffer initialized with 2 files.
7/12/2022 -- 05:52:20 - <Notice> - all 3 packet processing threads, 4 management threads
 initialized, engine started.
7/12/2022 -- 05:52:23 - <Notice> - Signal Received.  Stopping engine.
7/12/2022 -- 05:52:37 - <Notice> - Pcap-file module read 1 files, 24677 packets, 1711363
0 bytes
[andrii@sandbox suricata]$ cat fast.log | wc -l
0
```

Рис. 2.5.2.5. Запуск Suricata для тестових даних де немає Bitcoin трафіку

Як і очікувалося, не було ніяких спрацювань.

2.6. Детектування Torrent трафіку

В межах детектування Torrent трафіку, забезпечено засіб детектування Bittorrent протоколу. Для початку пригадаємо структуру Handshake повідомлення Bittorrent протоколу:

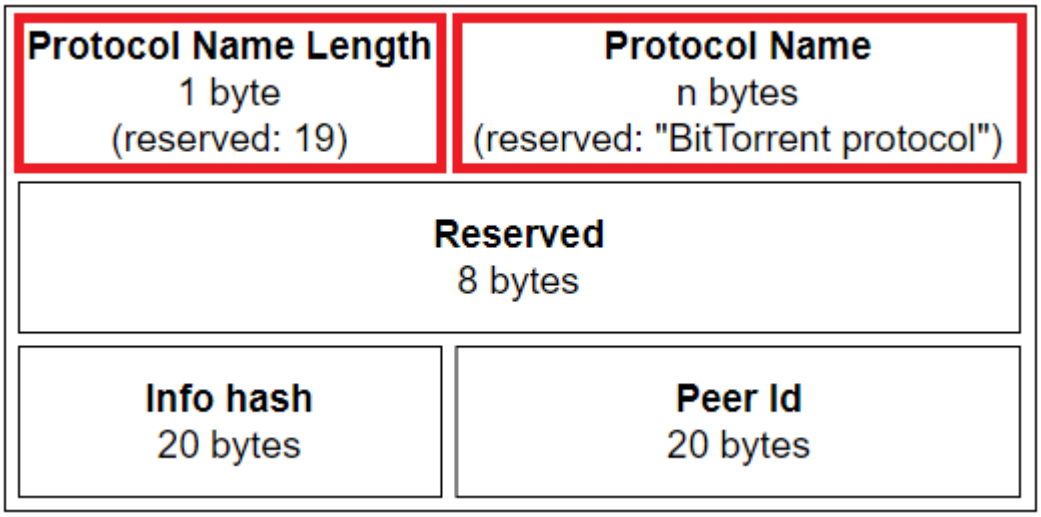

Рис. 2.6.1. Структура BitTorrent Handshake повідомлення, з виділеними статичними полями

Розглянувши його структуру, визначено для детектування протоколу Bittorrent виявляти Handshake повідомлення використання значень за замовчуванням для протоколу BitTorrent, а саме поле Protocol Name Length зі значенням 19, а також поле Protocol Name, що місить строку в кодуванні utf-8 зі значенням “BitTorrent protocol”.

```
alert tcp any any -> any any ( \
    msg: "Signature of BitTorrent protocol"; \
    content: "|13|BitTorrent protocol"; depth:20; \
    sid: 2345163; rev: 1; \
)
```

Рис. 2.6.2. Правило, для визначення BitTorrent Handshake повідомлення

Тестування:

Для тестування скористаємося прикладом мережевого трафіку, опублікований в документації Wireshark: [24.], файл BITTORRENT.pcap, заданий файл містить 53 пакетів, серед яких 2 Handshake пакети:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.10.10.23	10.10.10.22	BitTorrent	122	Handshake
2	0.004682	10.10.10.22	10.10.10.23	BitTorrent	445	Handshake Extended
3	1.186658	10.10.10.23	10.10.10.22	BitTorrent	108	Extended Interested
4	1.528815	10.10.10.22	10.10.10.23	BitTorrent	60	Unchoke
5	1.540399	10.10.10.23	10.10.10.22	BitTorrent	88	Request, Piece (Idx:0

Рис. 2.6.3. Тестові дані

Запустимо задане правило для даних тестових даних:

```
E:\Study\Univ\MasterDegree\Suricata>suricata -c suricata.yaml -S testRule\bittorrent.rules
-r Templates\BITTORRENT.pcap
9/4/2023 -- 23:31:18 - <Info> - Running as service: no
9/4/2023 -- 23:31:18 - <Notice> - This is Suricata version 6.0.9 RELEASE running in USER mo
de
9/4/2023 -- 23:31:18 - <Notice> - all 5 packet processing threads, 4 management threads ini
tialized, engine started.
9/4/2023 -- 23:31:18 - <Notice> - Signal Received.  Stopping engine.
9/4/2023 -- 23:31:18 - <Notice> - Pcap-file module read 1 files, 53 packets, 43120 bytes
```

Рис. 2.6.4. Запуск тесту 1 етапу

Розглянемо результат:

#	Timestamp	Src / Dst	Signature
1	2007-04-11 19:51:36 16 years ago	S: 10.10.10.22 D: 10.10.10.23	Signature of BitTorrent protocol
1	2007-04-11 19:51:36 16 years ago	S: 10.10.10.23 D: 10.10.10.22	Signature of BitTorrent protocol

Рис. 2.6.5. Результати тесту 1 етапу

Розглянувши результат, можемо спостерігати, що було виявлено усі 2 пакети.

Тестування 2 етап:

Запустимо правило, для підготовлених завчасно тестових даних, що не містять порушень правил:

```
E:\Study\Univ\MasterDegree\Suricata>suricata -c suricata.yaml -S testRule\bittorrent.rules
-r Templates\original.pcapng
10/4/2023 -- 00:29:20 - <Info> - Running as service: no
10/4/2023 -- 00:29:20 - <Notice> - This is Suricata version 6.0.9 RELEASE running in USER m
ode
10/4/2023 -- 00:29:20 - <Notice> - all 5 packet processing threads, 4 management threads in
itialized, engine started.
10/4/2023 -- 00:30:09 - <Notice> - Signal Received.  Stopping engine.
10/4/2023 -- 00:30:11 - <Notice> - Pcap-file module read 1 files, 787160 packets, 867648140
 bytes

E:\Study\Univ\MasterDegree\Suricata>type fast.log

E:\Study\Univ\MasterDegree\Suricata>
```

Рис. 2.6.6. Запуск та перевірка відсутності спрацювання тестування 2 етапу

Оскільки fast.log файл є пустим, це означає, що задане правило не спрацювало, отже тест пройдено успішно.

2.7. Зберігання мережевої активності TLS протоколу

У деяких випадках, для прикладу для прикладу, коли дані, що передаються мережею випадково збіглися з сигнатурою протоколу, що намагаємося визначити або коли на загально-відомому сервері звернення до якого визначаємо розташовані інші відмінні ресурси відмінні від того, що порушує політики користування мережею. Для таких випадків варто зберігати інформацію про мережеву активність користувачів, і у випадках спрацювання правил порушення політик, необхідно додатково впевнитися переглянувши активність користувача.

Логування мережевої активності TLS є вбудованою функцією Suricata, усе що необхідне для цього, активувати секцію tls-log, або додати тип tls до eve-log і за потреби можна вказати поля пакету та їх порядок, у якому їх необхідно виводити у результуючому log файлі.

У якості тесту зберігатимемо у вигляді tls логу. Налаштуємо Suricata на зберігання tls sni (server name) в лог файлі, для цього задамо в файлі конфігурації:

```
  - tls-log:
      enabled: yes
      filename: tls-sni.log
      append: yes

      custom: yes
      customformat: "%{%D-%H:%M:%S}t.%z %a:%p -> %A:%P %n"
```

Рис. 2.7.1. Налаштування tls-log в suricata.yaml

Для перевірки роботи tls-log запустимо Suricata в режимі перехоплення трафіку з інтерфейсу та перейдемо на головну сторінку нашого університету, knu.ua:

```
[root@sandbox ~]# suricata -i ens33
7/12/2022 -- 06:22:31 - <Notice> - This is Suricata version 6.0.8 RELEASE running in SYSTEM mode
7/12/2022 -- 06:22:32 - <Warning> - [ERRCODE: SC_ERR_NO_RULES(42)] - No rule files match the pattern /va
r/lib/suricata/rules/suricata.rules
7/12/2022 -- 06:22:32 - <Warning> - [ERRCODE: SC_ERR_NO_RULES_LOADED(43)] - 1 rule files specified, but
no rules were loaded!
7/12/2022 -- 06:22:32 - <Notice> - Ring buffer initialized with 1 files.
7/12/2022 -- 06:22:33 - <Notice> - all 2 packet processing threads, 4 management threads initialized, en
gine started.
^C7/12/2022 -- 06:22:57 - <Notice> - Signal Received.  Stopping engine.
7/12/2022 -- 06:22:59 - <Notice> - Stats for 'ens33':  pkts: 4964, drop: 0 (0.00%), invalid chksum: 0
```

Рис. 2.7.2. Запуск Suricata в режимі перехоплення інтерфейсу ens33

Переглянемо отриманий лог файл:

```
[root@sandbox ~]# cat /home/andrii/suricata/logs/tls.log
12/07/22-06:22:36.449963 192.168.1.102:42430 -> 34.102.187.140:443 firefox.settings.services.mozi
lla.com
12/07/22-06:22:37.251785 192.168.1.102:48740 -> 142.250.203.206:443 www.google-analytics.com
12/07/22-06:22:40.533891 192.168.1.102:42866 -> 91.202.128.77:443 knu.ua
12/07/22-06:22:42.633832 192.168.1.102:50204 -> 54.188.211.138:443 push.services.mozilla.com
12/07/22-06:22:44.542028 192.168.1.102:42890 -> 91.202.128.77:443 knu.ua
12/07/22-06:22:44.613631 192.168.1.102:42878 -> 91.202.128.77:443 knu.ua
12/07/22-06:22:44.608697 192.168.1.102:42898 -> 91.202.128.77:443 knu.ua
12/07/22-06:22:44.909797 192.168.1.102:42902 -> 91.202.128.77:443 knu.ua
12/07/22-06:22:44.908090 192.168.1.102:42914 -> 91.202.128.77:443 knu.ua
12/07/22-06:22:49.066886 192.168.1.102:38948 -> 142.250.203.206:443 www.google-analytics.com
12/07/22-06:22:50.243526 192.168.1.102:49918 -> 34.120.208.123:443 incoming.telemetry.mozilla.org
12/07/22-06:22:50.355622 192.168.1.102:43944 -> 74.125.131.157:443 stats.g.doubleclick.net
12/07/22-06:22:52.613026 192.168.1.102:39094 -> 142.250.203.195:443 www.google.com.ua
12/07/22-06:22:52.638727 192.168.1.102:35042 -> 142.250.203.196:443 www.google.com
12/07/22-06:22:56.851685 192.168.1.102:41616 -> 13.32.110.25:443 services.addons.mozilla.org
12/07/22-06:22:56.862129 192.168.1.102:41598 -> 13.32.110.25:443 services.addons.mozilla.org
12/07/22-06:22:56.863349 192.168.1.102:41586 -> 13.32.110.25:443 services.addons.mozilla.org
12/07/22-06:22:56.863791 192.168.1.102:41584 -> 13.32.110.25:443 services.addons.mozilla.org
12/07/22-06:22:56.860111 192.168.1.102:41602 -> 13.32.110.25:443 services.addons.mozilla.org
12/07/22-06:22:56.862862 192.168.1.102:41588 -> 13.32.110.25:443 services.addons.mozilla.org
```

Рис. 2.7.3. Результат записаний в tls-log

Можемо спостерігати, протягом переходу на головний сайт було відкрито безліч інших TLS сесій серед яких є TLS сесії, що звертаються до ресурсу knu.ua.

РОЗДІЛ 3. ПРАКТИЧНЕ ВПРОВАДЖЕННЯ

3.1. Конфігурація та запуск Suricata

Розглянемо розгортання розробленого нами рішення, на віртуальній машині, на яку перенаправляється трафік університетської мережі. Встановивши останню стабільну версію suricata 6.0.1 для операційної системи debian.

Провівши початкову конфігурацію запустимо систему з розробленими правилами, у вигляді окремого процесу, що є від'єднаний від ssh сесії, за допомогою якої є з'єднання до віртуальної машини.

```
userx@AddDetect:~/suricata$ sudo setsid suricata -c suricata.yaml -i ens224 >/dev/null 2>&1
 < /dev/null &
[1] 17880
userx@AddDetect:~/suricata$
```

Рис. 3.1.1. Запуск Suricata на віртуальній машині

Проведено дослідження протягом доби, а саме починаючи від 1:15 та завершуючи 22:45.

3.2. Розгляд результату логування

Оскільки університетська мережа, містить велику кількість користувачів, велика частина яких немає юридичних зобов'язань мережа немає системи слідкування порушень політик користування мережею, як результат протягом часу роботи Suricata виявила 4 352 343 порушень.

```
userx@AddDetect:~$ sudo wc -l suricata/logs/fast.log
[sudo] password for userx:
4352343 suricata/logs/fast.log
```

Рис. 3.2.1. Отримання кількості порушень

Оскільки усі результати роботи неможливо розглянути в межах роботи розглянемо детальніше, результат зібраний за перші 15 хвилин роботи,

зображення логу додано до Додатку З. За перші 15 хвилин, після запуску було зафіксовано 33 спрацювання правил, 7 спрацювань було спричинено користувачем мережі з IP адресою 10.33.4.232, що користується DoH використовуючи провайдер Google public DNS. Решта 25 порушень, були спричинені 8-ма користувачами з IP адресами, що додані до загальновідомих IP адрес для майнингу, однак визначення чи відбулося порушення політики потребує детальнішого розгляду систематичного спрацювання правила для даних користувачів.

3.3. Розгляд продуктивності

Для збору статистики навантаженості віртуальної машини скористаємося утилітою Monitorix [25.], що зберігає основні характеристики навантаженості серверу, та дозволяє переглянути її у вигляді графіків.

Характеристики віртуальної машини, на якій було проведено тестування:

- Операційна система – Debian 10
- Процессор – 8 CPUs x Intel(R) Xeon(R) CPU E5606 @ 2.13GHz, з виділеним 1 ядром під віртуальну машину
- Оперативна пам’ять – 2 ГБ, DDR3

3.3.1. Статистика завантаженості мережі

Розглянемо статистику. Середня завантаженість мережі за весь період збору статистики рівна 28 Мб/с (3,5 МБ/с), однак варто зазначити, що протягом нічного часу – до 7 години ранку, ті іншої частини практичної частини навантаженість мережі сильно відрізняється. Протягом ночі середня навантаженість мережі рівна 2,4 Мб/с (0,3 МБ/с), а протягом іншого часу практичної частини (надалі в денний період задля простоти) середня швидкість – 56 Мб/с (7 МБ/с).

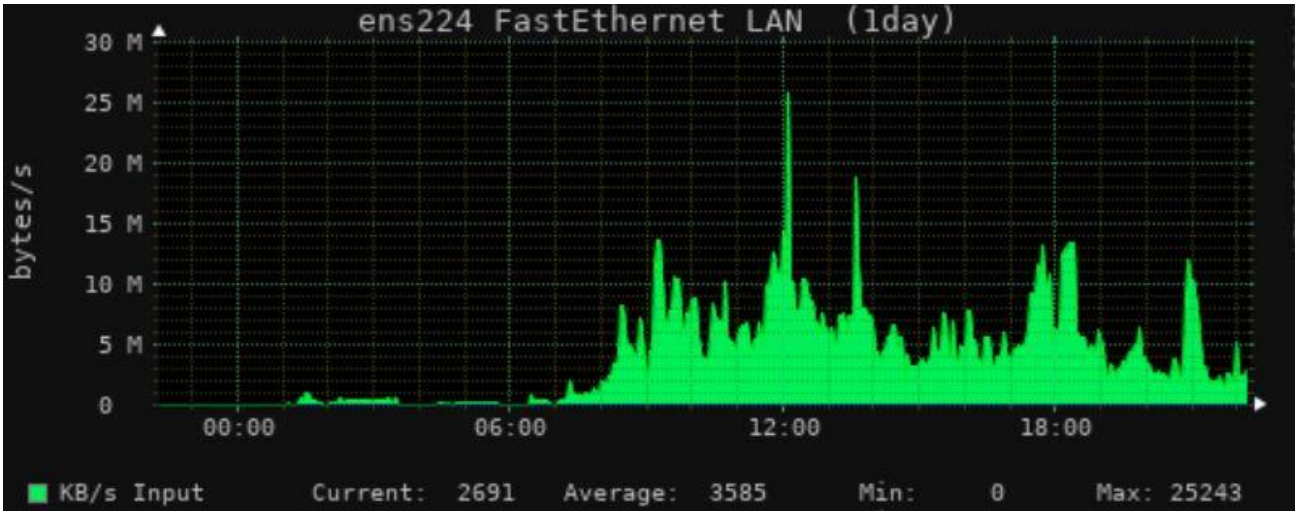

Рис. 3.3.1.1. Графік завантаженості мережі

Також окремо варто зазначити на різкі підвищення завантаженості мережі, а саме:

- 10:05 - 10:15 – підвищення завантаженості мережі до 120 Мб/с (15 МБ/с)
- 12:01 - 12:06 – різке підвищення завантаженості мережі до 180 Мб/с (25 МБ/с)
- 13:35 - 13:40 – різке підвищення завантаженості мережі до 152 Мб/с (19 МБ/с)
- 17:30 - 18:30 – підвищення завантаженості мережі до 108 Мб/с (13,5 МБ/с)
- 20:50 - 21:05 – підвищення завантаженості мережі до 96 Мб/с (12 МБ/с)

3.3.2. Статистика завантаженості процесору

Розглянемо навантаження процесору протягом проведення практичної частини, для початку пригадаємо характеристики процесору – 8 CPUs x Intel(R) Xeon(R) CPU E5606 @ 2.13GHz.

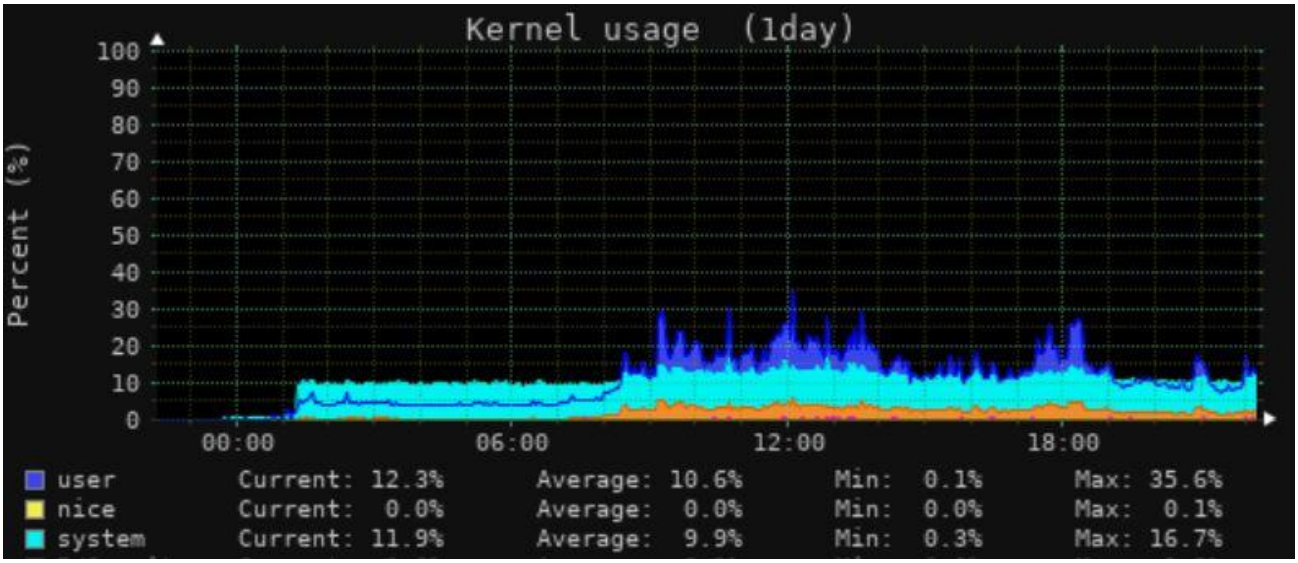

Рис. 3.3.2.1. Графік завантаженості CPU

Розглянемо детальніше графік навантаження процесору у залежності від часу. Можемо спостерігати, що протягом нічного періоду середня навантаженість процесору складала 5%, протягом даного періоду – середня навантаженість складала 17%, з піками навантаженості до 35,6% у пісках мережевого трафіку. Після вечірнього піку об 18, навантаженість процесору зменшилася до середньої навантаженості в 10 %.

3.3.3. Статистика використання оперативної пам’яті

Розглянемо використання оперативної пам’яті протягом практичної частини, для початку пригадаємо, що для віртуальної машини виділено 2 ГБ оперативної пам’яті. На початок практичної частини на віртуальній машині було зайнято 1,1 ГБ, по завершенню системні процеси займали 1,2 ГБ оперативної пам’яті. Протягом нічного часу споживання пам’яті віртуальною машиною збільшилося до 1,2 ГБ у порівнянні з 1,1 ГБ, на початок практичної частини.

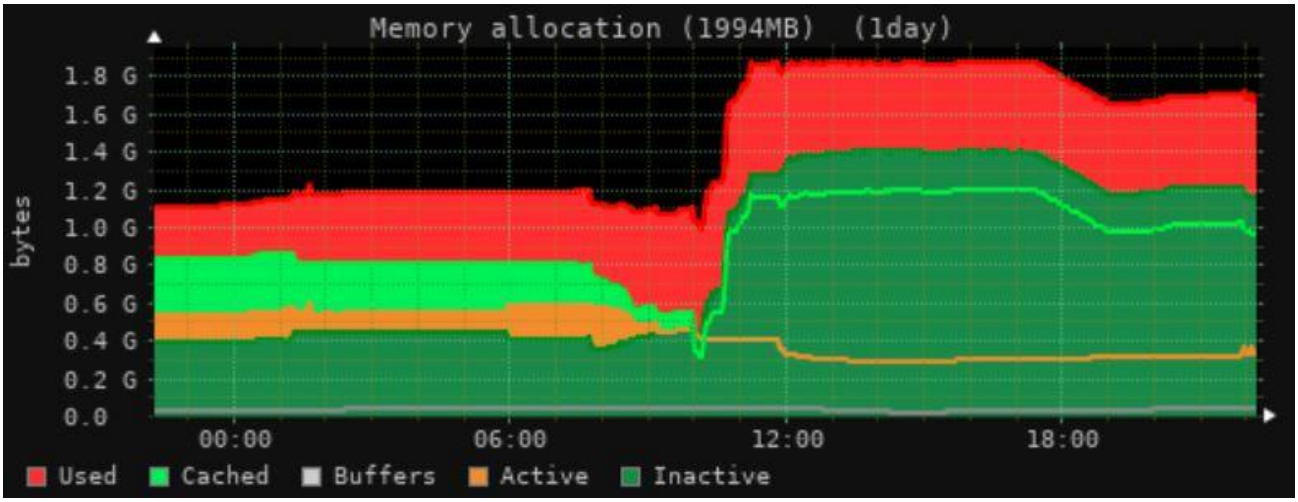

Рис. 3.3.3.1. Графік завантаженості CPU

Протягом денного часу споживання пам’яті віртуальної машини підвищилося до 1,85 ГБ, а після вечірнього піку об 18, використання оперативної пам’яті зменшилося до 1,7 ГБ.

3.4. Висновки практичного впровадження

Розроблене рішення було успішно запущено для реального трафіку університетської мережі з піками завантаженості мережі до 180 Мб/с. Протягом доби було зафіксовано 4 352 343 порушень, розглянувши споживання системних ресурсів можемо спостерігати, що розроблене рішення має стабільне споживання пам’яті до 800 МБ, а також стабільне навантаження CPU не було отримано різких стрибків навантаження, максимальним навантаження CPU було 35,6%.

ВИСНОВКИ

Виявлення мережевого трафіку використовуючи його сигнатури це найшвидший та найнадійніший засіб виявлення типу мережевого трафіку, однак використання протоколів шифрування ускладнює виявлення мережевої активності, в таких випадках можливе визначення мережевої активності шляхом виявлення звернень за загально-відомими IP адресами та DNS іменами.

Проведений теоретичний аналіз, забезпечив можливість створення та тестування правил, що за сигнатурами визначають протоколи: Bitcoin, WireGurad, OpenVPN, BitTorrent, DNS over TLS, DNS over QUIC.

Розроблено та протестовано загальні правила, визначення звернень до загальновідомих IP адрес та звернень за загально-відомими доменними іменами, а також засоби створення датасетів, та репутаційних файлів на основі списків IP адрес/доменних імен.

В якості підтвердження порушень політик користування мережею користувачем було визначено та протестовано використання зберігання TLS активності користувачів.

Продемонстровано доцільність використання IDS системи Suricata для перевірки відповідності політикам мережі, що дозволяє виконувати поставлене завдання при невеликих потужностях комп’ютеру.

ПЕРЕЛІК ПОСИЛАНЬ

1. Статистика Let's Encrypt [Електронний ресурс]. URL: https://letsencrypt.org/stats/
2. QUIC: A UDP-Based Multiplexed and Secure Transport [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc9000.txt
3. A TCP/IP Tutorial [Електронний ресурс]. URL: https://datatracker.ietf.org/doc/html/rfc1180
4. Understanding TCP/IP [Електронний ресурс]. URL: https://www.cisco.com/E-Learning/bulk/public/tac/cim/cib/using_cisco_ios_software/linked/tcpip.htm
5. Bitcoin Wiki. Protocol documentation [Електронний ресурс]. URL: https://en.bitcoin.it/wiki/Protocol_documentation#Scripting
6. bitcoin/bitcoin/src/protocol.cpp [Електронний ресурс]. URL: https://github.com/bitcoin/bitcoin/blob/a97791d9fb977cf2a0d19268253238b0fee173f6/src/protocol.cpp
7. Bittorrent Protocol Specification v1.0 [Електронний ресурс]. URL: https://wiki.theory.org/BitTorrentSpecification
8. Point-to-Point Tunneling Protocol (PPTP) [Електронний ресурс]. URL: https://www.rfc-editor.org/rfc/rfc2637
9. OpenVPN's network protocol [Електронний ресурс]. URL: https://build.openvpn.net/doxygen/network_protocol.html
10. WireGuard: Next Generation Kernel Network Tunnel [Електронний ресурс]. URL: https://www.wireguard.com/papers/wireguard.pdf

11. Helium: Data Structures [Електронний ресурс]. URL: https://lightway.com/doxygen/annotated.html

12. Suricata User Guide 6.1.2. Protocol [Електронний ресурс]. URL: https://suricata.readthedocs.io/en/suricata-6.0.9/rules/intro.html#protocol

13. Suricata User Guide 6. Suricata Rules [Електронний ресурс]. URL: https://suricata.readthedocs.io/en/suricata-6.0.9/rules/index.html

14. EveBox [Електронний ресурс]. URL: https://evebox.org/

15. Suricata User Guide 15.1.1. Eve JSON Output [Електронний ресурс]. URL: https://suricata.readthedocs.io/en/suricata-6.0.9/output/eve/eve-json-output.html#

16. Репозиторій VPN & datacenter IPs [Електронний ресурс]. URL: https://github.com/X4BNet/lists_vpn

17. Репозиторій bitcoin-labs/bitcoin-details [Електронний ресурс]. URL: https://github.com/bitcoin-labs/bitcoin-details

18. WireGuard [Електронний ресурс]. URL: https://wiki.wireshark.org/WireGuard

19. OpenVPN Protocol (OpenVPN) [Електронний ресурс]. URL: https://wiki.wireshark.org/OpenVPN.md

20. TOR Network Endpoints [Електронний ресурс]. URL: https://check.torproject.org/torbulkexitlist

21. Репозиторій CoinBlockerLists [Електронний ресурс]. URL: https://gitlab.com/ZeroDot1/CoinBlockerLists

22. Репозиторій Crypto mining pools aggregator [Електронний ресурс]. URL: https://github.com/ilmoi/mining-pools-aggregator

23. bitcoin-labs/bitcoin-details [Електронний ресурс]. URL: https://github.com/bitcoin-labs/bitcoin-details

24. BitTorrent [Електронний ресурс]. URL: https://wiki.wireshark.org/BitTorrent

25. Monitorix [Електронний ресурс]. URL: https://www.monitorix.org/

ДОДАТКИ

Додаток А

```
from hashlib import sha256, md5
from argparse import Namespace, ArgumentParser, FileType
from typing import TextIO
import base64
import sys

class __DatasetsCoder:
    def encode(self, line : str) -> str:
        raise NotImplemented('Classes derived from __DatasetsCoder should implement "encode" method.\n'
            + f'Class {type(self)} is not implement "write_body" method.')

class __DatasetsStringCoder(__DatasetsCoder):
    def encode(self, line : str) -> str:
        base64_bytes = base64.b64encode(bytes(line, 'utf-8'))
        return base64_bytes.decode('utf-8')

class __DatasetsSha256Coder(__DatasetsCoder):
    def encode(self, line : str) -> str:
        return sha256(bytes(line, 'utf-8')).hexdigest()

class __DatasetsMd5Coder(__DatasetsCoder):
    def encode(self, line : str) -> str:
        return md5(bytes(line, 'utf-8')).hexdigest()


def __initialize_cmd_args() -> Namespace:
    parser = ArgumentParser()

    type_group = parser.add_mutually_exclusive_group()
    type_group.add_argument('--sha256', action='store_true',
        help='Create sha256 dataset')
    type_group.add_argument('--md5', action='store_true',
        help='Create md5 dataset')
    type_group.add_argument('--string', action='store_true',
        help='Create string dataset. Behavior by default')

    parser.add_argument('-o', '--output', type=FileType('w'), help='Specify output file.')

    parser.add_argument('-f', '--file', type=FileType('r'), help='Specify input file.')

    return parser.parse_args()
```

```
def __get_input_stream(args : Namespace) -> TextIO:
    if args.file:
        return args.file

    return sys.stdin

def __get_output_stream(args: Namespace) -> TextIO:
    if args.output:
        return args.output

    return sys.stdout

def __get_dataset_encoder(args : Namespace) -> __DatasetsCoder:
    if args.sha256:
        return __DatasetsSha256Coder()

    if args.md5:
        return __DatasetsMd5Coder()

    if args.string:
        return __DatasetsStringCoder()

    return __DatasetsStringCoder()


def __main():
    args = __initialize_cmd_args()

    dataset_coder = __get_dataset_encoder(args)

    with __get_input_stream(args) as input:
        with __get_output_stream(args) as output:
            for line in input:
                if line and not line.isspace():
                    output.write(dataset_coder.encode(line.strip()))
                    output.write('\n')

if __name__ == '__main__':
    __main()
```

Додаток Б

```
[andrii@sandbox suricata]$ head -3 dns-eve.json | tail -1 | python3 -m json.tool
{
    "timestamp": "2022-11-30T04:31:41.281374-0500",
    "flow_id": 1671900358855454,
    "pcap_cnt": 461,
    "event_type": "alert",
    "src_ip": "192.168.1.101",
    "src_port": 62228,
    "dest_ip": "192.168.1.1",
    "dest_port": 53,
    "proto": "UDP",
    "tx_id": 0,
    "alert": {
        "action": "allowed",
        "gid": 1,
        "signature_id": 123256,
        "rev": 0,
        "signature": "This domain name was in domain_base64",
        "category": "",
        "severity": 3
    },
    "dns": {
        "query": [
            {
                "type": "query",
                "id": 40559,
                "rrname": "www.gstatic.com",
                "rrtype": "A",
                "tx_id": 0
            }
        ]
    },
    "app_proto": "dns",
    "flow": {
        "pkts_toserver": 1,
        "pkts_toclient": 0,
        "bytes_toserver": 75,
        "bytes_toclient": 0,
        "start": "2022-11-30T04:31:41.281374-0500"
    }
}
```

Додаток В

```
[andrii@sandbox suricata]$ head -1 dns-eve.json | python3 -m json.tool
{
    "timestamp": "2022-11-30T04:31:41.282014-0500",
    "flow_id": 812093003353502,
    "pcap_cnt": 464,
    "event_type": "alert",
    "src_ip": "192.168.1.101",
    "src_port": 61601,
    "dest_ip": "192.168.1.1",
    "dest_port": 53,
    "proto": "UDP",
    "tx_id": 0,
    "alert": {
        "action": "allowed",
        "gid": 1,
        "signature_id": 123256,
        "rev": 0,
        "signature": "This domain name was in domain_base64",
        "category": "",
        "severity": 3
    },
    "dns": {
        "query": []
    },
    "app_proto": "dns",
    "flow": {
        "pkts_toserver": 1,
        "pkts_toclient": 0,
        "bytes_toserver": 75,
        "bytes_toclient": 0,
        "start": "2022-11-30T04:31:41.282014-0500"
    }
}
```

Додаток Г

```
from argparse import Namespace, ArgumentParser, FileType
import re
from typing import TextIO
import sys

def __initialize_cmd_args() -> Namespace:
    parser = ArgumentParser()

    parser.add_argument('--category', type=int, default=1,
        help='Specify category id for all ip. Default value is 1.')

    parser.add_argument('--reputation', type=int, default=64,
        help='Specify reputation score for all ip. Default value is 64.')

    parser.add_argument('-o', '--output', type=FileType('w'), help='Specify output file.')

    parser.add_argument('-f', '--file', type=FileType('r'), help='Specify input file.')

    return parser.parse_args()


def __get_input_stream(args : Namespace) -> TextIO:
    if args.file:
        return args.file

    return sys.stdin

def __get_output_stream(args: Namespace) -> TextIO:
    if args.output:
        return args.output

    return sys.stdout

def __is_proper_ip4(ip4 : str) -> str:
    byte_ex = r"(\d|[1-9]\d|1\d\d|2[0-4]\d|25[0-5])"
    ip4_ex = fr"^{byte_ex}\.{byte_ex}\.{byte_ex}\.{byte_ex}(/([1-9]|[1-2]\d|3[0-2]))?$"

    return re.match(ip4_ex, ip4)

def __main():
    args = __initialize_cmd_args()

    reputation = args.reputation
    category = args.category
    line_count = 1
```

```
    with __get_input_stream(args) as input:
        with __get_output_stream(args) as output:
            for line in input:
                if line and not line.isspace():
                    if not __is_proper_ip4(line.strip()):
                        sys.stderr.write(f'Error line {line_count}: "{line.strip()}" is not proper IPv4.')
                        exit(1)

                    output.write(f'{line.strip()},{category},{reputation}')
                    output.write('\n')
            line_count += 1

if __name__ == '__main__':
    __main()
```

Додаток Д

```
function match(args)
    payload = args["payload"]
    if #payload < 16 then
        return 0
    end
    if CheckType(payload) == 0 then
        return 0
    end
    return CheckReserved(payload)
end

function CheckType(payload)
    local packet_type = string.byte(payload)

    if packet_type > 0 and packet_type < 9 then
        return 1
    end

    return 0
end

function CheckReserved(payload)
    for reserved_index = 2, 4, 1 do
        local zero_byte = string.byte(payload, reserved_index)

        if zero_byte ~= 0 then
            return 0
        end
    end

    return 1
end

return 0
```

Додаток Е

```
function init(args)
   local needs = {}
   needs["payload"] = tostring(true)
   return needs
end

function match(args)
   payload = args["payload"]

   if #payload < 44 then
      return 0
   end

   if CheckType(payload) == 0 then
      return 0
   end

   return CheckFirstOpenVpn(payload, 3)
end

function CheckType(payload)
   local packet_type = string.byte(payload, 3)

   if packet_type > 7 and packet_type < 79 then
      return 1
   end

   return 0
end

function CheckFirstOpenVpn(payload, initial_offset)
   local replayId = ReadIntBigEndian(payload, initial_offset + 29)
   if replayId ~= 1 then
      return 0
   end

   local array_lenght = string.byte(payload, initial_offset + 37)

   local packetId_offset = initial_offset + 38

   if array_lenght > 0 then
      packetId_offset = packetId_offset + 8 + array_lenght * 4

      if #payload < packetId_offset + 3 then
         return 0
      end
```

```
    end

    local packetId = ReadIntBigEndian(payload, packetId_offset)

    if packetId ~= 0 then
        return 0
    end

    return 1
end


function ReadIntBigEndian(payload, start_index)
    local result = 0
    for i = 0, 3, 1 do
        result = result * 256

        result = result + string.byte(payload, start_index + i)
    end
    return result
end

return 0
```

Додаток Ж

```
function init(args)
   local needs = {}
   needs["payload"] = tostring(true)
   return needs
end

function match(args)
   a = args["payload"]

end

function match(args)
   payload = args["payload"]

   if #payload < 16 then
      return 0
   end

   if CheckMagic(payload) == 0 then
      return 0
   end

   return CheckCommand(payload)
end

magic_vals = {
   string.char(0xF9, 0xBE, 0xB4, 0xFE),
   string.char(0x0A, 0x03, 0xCF, 0x40),
   string.char(0x0B, 0x11, 0x09, 0x07),
   string.char(0xF9, 0xBE, 0xB4, 0xD9)
}

function CheckMagic(payload)
   isvalid = 0

   for i, magic in ipairs(magic_vals) do
      isvalid = 1

      for magic_index = 1, 4, 1 do
         if string.byte(magic, magic_index) ~= string.byte(payload, magic_index) then
            isvalid = 0
         end
      end

      if isvalid == 1 then
         return 1
```

```
        end
    end

    return 0
end

command_vals = {
    string.char(0x76, 0x65, 0x72, 0x73, 0x69, 0x6f, 0x6e, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x76, 0x65, 0x72, 0x61, 0x63, 0x6b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x61, 0x64, 0x64, 0x72, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x61, 0x64, 0x64, 0x72, 0x76, 0x32, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x73, 0x65, 0x6e, 0x64, 0x61, 0x64, 0x64, 0x72, 0x76, 0x32, 0x00, 0x00),
    string.char(0x69, 0x6e, 0x76, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x67, 0x65, 0x74, 0x64, 0x61, 0x74, 0x61, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x6d, 0x65, 0x72, 0x6b, 0x6c, 0x65, 0x62, 0x6c, 0x6f, 0x63, 0x6b, 0x00),
    string.char(0x67, 0x65, 0x74, 0x62, 0x6c, 0x6f, 0x63, 0x6b, 0x73, 0x00, 0x00, 0x00),
    string.char(0x67, 0x65, 0x74, 0x68, 0x65, 0x61, 0x64, 0x65, 0x72, 0x73, 0x00, 0x00),
    string.char(0x74, 0x78, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x68, 0x65, 0x61, 0x64, 0x65, 0x72, 0x73, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x62, 0x6c, 0x6f, 0x63, 0x6b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x67, 0x65, 0x74, 0x61, 0x64, 0x64, 0x72, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x6d, 0x65, 0x6d, 0x70, 0x6f, 0x6f, 0x6c, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x70, 0x69, 0x6e, 0x67, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x70, 0x6f, 0x6e, 0x67, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x6e, 0x6f, 0x74, 0x66, 0x6f, 0x75, 0x6e, 0x64, 0x00, 0x00, 0x00, 0x00),
    string.char(0x66, 0x69, 0x6c, 0x74, 0x65, 0x72, 0x6c, 0x6f, 0x61, 0x64, 0x00, 0x00),
    string.char(0x66, 0x69, 0x6c, 0x74, 0x65, 0x72, 0x61, 0x64, 0x64, 0x00, 0x00, 0x00),
    string.char(0x66, 0x69, 0x6c, 0x74, 0x65, 0x72, 0x63, 0x6c, 0x65, 0x61, 0x72, 0x00),
    string.char(0x73, 0x65, 0x6e, 0x64, 0x68, 0x65, 0x61, 0x64, 0x65, 0x72, 0x73, 0x00),
    string.char(0x66, 0x65, 0x65, 0x66, 0x69, 0x6c, 0x74, 0x65, 0x72, 0x00, 0x00, 0x00),
    string.char(0x73, 0x65, 0x6e, 0x64, 0x63, 0x6d, 0x70, 0x63, 0x74, 0x00, 0x00, 0x00),
    string.char(0x63, 0x6d, 0x70, 0x63, 0x74, 0x62, 0x6c, 0x6f, 0x63, 0x6b, 0x00, 0x00),
    string.char(0x67, 0x65, 0x74, 0x62, 0x6c, 0x6f, 0x63, 0x6b, 0x74, 0x78, 0x6e, 0x00),
    string.char(0x62, 0x6c, 0x6f, 0x63, 0x6b, 0x74, 0x78, 0x6e, 0x00, 0x00, 0x00, 0x00),
    string.char(0x67, 0x65, 0x74, 0x63, 0x66, 0x69, 0x6c, 0x74, 0x65, 0x72, 0x73, 0x00),
    string.char(0x63, 0x66, 0x69, 0x6c, 0x74, 0x65, 0x72, 0x00, 0x00, 0x00, 0x00, 0x00),
    string.char(0x67, 0x65, 0x74, 0x63, 0x66, 0x68, 0x65, 0x61, 0x64, 0x65, 0x72, 0x73),
    string.char(0x63, 0x66, 0x68, 0x65, 0x61, 0x64, 0x65, 0x72, 0x73, 0x00, 0x00, 0x00),
    string.char(0x67, 0x65, 0x74, 0x63, 0x66, 0x63, 0x68, 0x65, 0x63, 0x6b, 0x70, 0x74),
    string.char(0x63, 0x66, 0x63, 0x68, 0x65, 0x63, 0x6b, 0x70, 0x74, 0x00, 0x00, 0x00),
    string.char(0x77, 0x74, 0x78, 0x69, 0x64, 0x72, 0x65, 0x6c, 0x61, 0x79, 0x00, 0x00),
    string.char(0x73, 0x65, 0x6e, 0x64, 0x74, 0x78, 0x72, 0x63, 0x6e, 0x63, 0x6c, 0x00)
}

function CheckCommand(payload)
    isvalid = 0

    for i, command in ipairs(command_vals) do
        isvalid = 1
```

```
            for command_index = 1, 12, 1 do
               if string.byte(command, command_index) ~= string.byte(payload, command_index + 4)
then
                  isvalid = 0
               end
            end

            if isvalid == 1 then
               return 1
            end
         end

         return 0
      end

      return 0
```

Додаток 3

#	Timestamp ▾	Src / Dst	Signature
1	2023-04-17 01:29:02 20 minutes ago	S: 10.33.67.40 D: 8.8.8.8	Mining: Request to public known mining node
4	2023-04-17 01:28:30 21 minutes ago	S: 10.33.67.57 D: 192.178.25.174	Mining: Request to public known mining node
5	2023-04-17 01:28:23 21 minutes ago	S: 10.33.67.40 D: 2.21.89.26	Mining: Request to public known mining node
1	2023-04-17 01:26:07 23 minutes ago	S: 10.33.67.40 D: 13.32.110.88	Mining: Request to public known mining node
1	2023-04-17 01:25:01 24 minutes ago	S: 10.33.67.57 D: 172.217.16.46	Mining: Request to public known mining node
1	2023-04-17 01:24:04 25 minutes ago	S: 10.33.75.124 D: 192.178.25.174	Mining: Request to public known mining node
1	2023-04-17 01:23:49 26 minutes ago	S: 10.33.75.12 D: 172.217.16.46	Mining: Request to public known mining node
2	2023-04-17 01:23:18 26 minutes ago	S: 10.33.75.56 D: 47.246.15.235	Mining: Request to public known mining node
1	2023-04-17 01:23:18 26 minutes ago	S: 10.33.75.219 D: 13.32.110.126	Mining: Request to public known mining node
1	2023-04-17 01:22:57 26 minutes ago	S: 10.33.67.91 D: 18.66.15.47	Mining: Request to public known mining node
1	2023-04-17 01:22:43 27 minutes ago	S: 10.33.75.56 D: 47.246.15.232	Mining: Request to public known mining node
2	2023-04-17 01:22:41 27 minutes ago	S: 10.33.75.56 D: 172.217.16.46	Mining: Request to public known mining node
7	2023-04-17 01:21:42 28 minutes ago	S: 10.33.4.232 D: 8.8.4.4	Secure DNS: DoH: Web traffic to Google public DNS, regularly. tls
1	2023-04-17 01:19:55 29 minutes ago	S: 10.33.67.91 D: 192.178.25.174	Mining: Request to public known mining node
1	2023-04-17 01:18:53 30 minutes ago	S: 10.33.67.91 D: 18.66.15.2	Mining: Request to public known mining node