Міністерство освіти і науки України
Тернопільський національний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
Кафедра кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломної роботи

на тему: Дослідження вразливостей реалізації криптографічних методів захисту протоколу SSL/TLS

Виконав: студент VI курсу, групи СБм-61
спеціальності 125 “Кібербезпека”
_____ Зимницький О.Г.
Керівник _____ к.т.н., доц. Загородна Н.В.
Нормоконтроль _____ _____
Рецензент _____ _____

м. Тернопіль — 2019

АНОТАЦІЯ

Робота об’ємом 92 сторінок, яка містить 25 рисунків, 7 таблиць, 62 джерела за переліком посилань.

Метою даної кваліфікаційної роботи є аналіз вразливостей реалізації протоколу SSL/TLS та розробка методології виявлення цих вразливостей з використанням існуючих технік та програмного забезпечення.

Об’єктом вивчення є вразливості відкритих реалізацій протоколу SSL/TLS.

Методами дослідження є як загальнонаукові методи пізнання: порівняння, системний аналіз, так і спеціальні, зокрема методи статичного та динамічного аналізу коду.

Наукова новизна полягає у створенні методології аналізу реалізацій протоколу TLS на етапі розробки з використанням методів статичного та динамічного аналізу коду, а також модульного тестування.

Результати роботи можуть бути використані для пошуку вразливостей у реалізаціях протоколу SSL/TLS мовою програмування C на всьому етапі розробки.

Ключові слова: КІБЕРБЕЗПЕКА, ІНФОРМАЦІЙНА БЕЗПЕКА, РОЗРОБКА ЗАХИЩЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, SSL, TLS.

ABSTRACT

Work with 92 pages containing, 25 illustrations, 7 tables, 62 sources by the list of links.

The purpose of this qualification work is to analyze the vulnerabilities of the SSL/ TLS open-source implementations and to build a methodology for their detection during the development process.

The object of the research is the vulnerabilities of the SSL/TLS open-source implementations.

The scientific methods that are used in the study are both general-purpose methods, like comparing or system analysis, and specific methods, such that methods of static and dynamic code analysis.

Scientific innovation is a creation of a methodology for analyzing implementations of TLS protocol at the development stage using static and dynamic code analysis.

Results of the work can be used for detecting vulnerabilities of SSL/TLS protocol implementations that are written in C programming language during the whole development process.

Keywords: CYBERSECURITY, INFORMATION SECURITY, SECURE SOFTWARE DEVELOPMENT, SSL, TLS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SSL — Secure Sockets Layer;

TLS — Transport Layer Security;

ЕЦП — електронно-цифровий підпис;

ПЗ — програмне забезпечення;

WWW — World Wire Web;

URL — Unified Resource Link.

ВСТУП

Плин часу це невід'ємний атрибут нашого світу. Ми можемо спостерігати його кожного дня та кожної миті незалежно від того де ми перебуваємо, у будь якій точці нашої планети чи навіть у космічному просторі. Незалежно від того чим ми займаємося: чи просто рухаємося з точки А у точку Б; чи працюємо над виробництвом матеріальний або ж нематеріальних благ; чи здійснюємо різного роду дослідження, що можуть кардинально вплинути на долю всього людства; чи просто відпочиваємо сидячи перед вікном та дивлячись як часточки снігу падають у вигляді опадів і скупчуються на деревах, змінюючи їх до невпізнаваності; ми відчуваємо плин часу.

Це поняття супроводжує нас все наше життя та існувало, безперечно, ще до появи людства. Воно відіграє роль осі координат, на якій розташовуються події та об'єкти. Останні існують протягом певного інтервалу на цій осі координат у різних, зазвичай, пов'язаних між собою формах. Ми часто зберігаємо різного роду знімки цих об'єктів у вигляді тексту, зображень чи просто спогадів і в подальшому порівнюємо їх та проводимо певний аналіз для формування висновків, які покликані допомогти нам у деталізації нашої картини світу чи вирішення прикладних задач, що складають інтерес для людства чи його частини.

І часто, оглядаючись назад, у нас виникає щире здивування — як швидко плине час. Воно пов'язане з тим як різко відрізняється минуле і теперішнє у деяких аспектах нашого життя. Особливо, це актуально для технологій.

Ще 30 років тому, доступ до комп'ютерних мереж мали лише працівники науково-дослідницьких центрів при університетах та великих корпораціях, а зараз більше половини населення землі, яке постійно збільшується, має доступ до глобальної мережі Інтернет і це число теж постійно збільшується[1, с. 2].

Така кількість користувачів продукує гігантський обсяг трафіку. Вони його використовують для задоволення своїх потреб, а оскільки мережа Інтернет є публічною, то цей трафік легко може потрапити на комп'ютери та пристрої сторонніх осіб, тобто тих, хто не є відправником, адресатом, чи відповідальним за марш-

рутизацію пакету, що становить загрозу конфіденційності інформації користувачів.

Іншою проблемою, з якою можуть зіштовхнутися користувачі та сервіси в мережі, це підміна пакетів та відмова від авторства. що створює загрозу для цілісності інформації та загалом підриває довіру користувачів мережі один до одного чи до самого каналу передачі інформації.

Вирішенням стало використання криптографічних методів захисту інформації під час її передачі через мережу для забезпечення її конфіденційності за рахунок шифрування, цілісності та уникнення відмови від авторства за рахунок електронно-цифрового підпису. Ці засоби і реалізує протокол TLS, який став особливо популярним в останні кілька років з появою сервісу Let’s Encrypt[2], який безкоштовно надає сертифікати для ключів шифрування, а браузери почали помічати сайти, що передають дані без захисту як небезпечні, принаймні на сторінках з елементами для вводу даних, що зробило Інтернет, а саме World Wide Web набагато безпечнішим.

Проте, ці засоби шифрування не є досконалими і кожен має свій запас часу, протягом якого вони можуть забезпечувати безпеку інформації. Ще менш досконалими є реалізації цих методів шифрування та протоколу встановлення з’єднання чи передачі зашифрованих даних, оскільки людині властиво помилятися. Через це щорічно у різних реалізаціях протоколу TLS виявляють десятки вразливостей, які і є об’єктом даного дослідження.

Інші дослідження проводять аналіз виявлених вразливостей та збирають інформацію про них[3], та засуджують розробників, що займаються власною реалізацію протоколу TLS чи його частин чи розглядають процес виявлення вразливостей у скомпільованих застосунках[4] і не дають жодних порад стосовно виявлення цих вразливостей на стадії розробки, залишаючи це питання самим розробникам.

Метою даного дослідження є розробка методології, що дозволить суттєво зменшити ймовірність виникнення вразливостей під час імплементації SSL/TLS за рахунок певних практик у ході розробки та аналіз порядку розробки відкритих

імплементацій протоколу та формування рекомендацій, які можуть використовувати їх розробники, або їх приватні форки, що надає практичне значення даному дослідженню.

Завданнями, які ставляться перед дослідженням є:

— провести всебічний аналіз уже виявлених вразливостей бібліотек OpenSSL, GnuTLS, NSS;

— виявлення причин виникнення цих вразливостей;

— проаналізувати способи, які дозволили виявити ці вразливості;

— сформувати методологію, що дозволить уникати подібних вразливостей під час імплементації протоколу SSL/TLS;

— сформувати алгоритм впровадження даної методології для імплементації протоколу SSL/TLS.

Об’єктом дослідження є відкриті реалізації протоколу SSL/TLS, які часто використовуються розробниками ПЗ і на які припадає основна частина трафіку. До таких імплементацій належать: OpenSSL, GnuTLS, Mozilla Network Security Services (надалі — NSS). А також засоби виявлення вразливостей під час їх розробки.

Предметом дослідження є вразливості програмного коду цих реалізацій.

Методами дослідження є як загальнонаукові методи пізнання: порівняння, системний аналіз, так і спеціальні, зокрема методи статичного та динамічного аналізу коду.

Наукова новизна полягає у створенні методології аналізу реалізацій протоколу TLS на етапі розробки з використанням методів статичного та динамічного аналізу коду, а також модульного тестування.

Апробація результатів. Основні положення дослідження доповідалися й обговорювалися на науково-практичних конференціях: на VII Науково-технічної конференції “Інформаційні моделі, системи та технології” (Тернопіль, 11-12 грудня 2019 року).

Робота складається з семи розділів, у яких розглядаються наступні питання:

1. Загальні відомості про протокол SSL/TLS, що необхідні для повного розуміння вимог до реалізації протоколу та їх вразливостей.

2. Аналіз виявлених протягом 20 років вразливостей відкритих реалізацій SSL/TLS.

3. Методи виявлення вразливостей в ході реалізації SSL/TLS.

4. Налаштування безперервної інтеграції для відкритої імплементації SSL/TLS OpenSSL.

5. Економічна доцільність описаного методу.

6. Охорона праці та безпека в надзвичайних ситуаціях.

7. Екологія.

1 ЗАГАЛЬНІ ПЕРЕДУМОВИ ВИНИКНЕННЯ РЕАЛІЗАЦІЙ ПРОТОКОЛУ TLS

1.1 Виникнення комп’ютерних мереж

Розвиток комп’ютерних мереж почався з проєкту Агентства передових оборонних дослідницьких проєктів при міністерстві оборони США[5]. Метою її створення було забезпечення швидкої передачі інформації між військовими базами та науково-дослідницькими об’єктами в умовах Холодної війни, коли раптовий ядерний удар противника може фізично знищити частину мережі та навіть у таких умовах необхідно забезпечувати стійкий зв’язок між тими вузлами, що залишились неушкодженими.

Вирішенням стала передача інформації малими порціями, так званими пакетами, по динамічно визначеному маршруту, що може змінюватися у випадку втрати з’єднання між деякими вузлами чи втрати самих вузлів. Даний механізм назвали комутацією пакетів.

Першою такою комп’ютерною мережею стала ARPANET, яка об’єднала кілька університетів США. У ній зародилася сучасна електронна пошта, TELNET, FTP, DNS та стек протоколів який згодом переріс у TCP/IP, а ще пізніше у модель OSI, без якої складно уявити комп’ютерні мережі у наш час.

Всі описані вище протоколи передають дані (електронні листи, ASCII-текст, файли, записи DNS тощо) у відкритому вигляді. У зв’язку з малою кількістю вузлів та низьким значенням їх продуктивності, на початкових етапах розвитку мережі це було скоріше перевагою, ніж недоліком. Однак на основі досягнень ARPANET, приватні компанії та урядові організації по всьому світу будували свої мережі, вони об’єднувалися між собою та число вузлів невпинно зростало. Таким чином формувався Інтернет, який складно назвати довіреним каналом передачі даних через наступні причини:

1. Відправник та отримувач пакету часто розділені не лише значною відстанню, але й мережевим обладнанням, яке знаходиться під контролем

провайдера чи інших організацій, які можуть самостійно здійснювати прослуховування трафіку (що вимагається законодавством деяких країн) або можуть бути використані зловмисником для цих цілей без відома працівників підприємства за умови недотримання останніми політики безпеки.

2. Недоліки мережі під час її побудови провайдером або іншою організацією можуть призвести до вразливості, яка дозволить зловмиснику підміняти пакети, або відправляти їх від імені іншого вузла мережі (man-in-the-middle атака), що було особливо актуальним на початку існування мережі Інтернет, коли технології захисту, такі як віртуальні локальні мережі та port security були ще не створені, або не настільки поширені.

Навіть попри постійні зловживання з боку хакерів, ці фактори не заважали розвитку глобальної мережі для обміну документами та іншими даними, що призвело до виникнення WWW.

1.2 World Wide Web

WWW — це інформаційна система, у якій документи та інші ресурси ідентифікуються через URL, можуть бути з’єднані між собою через гіпертекст та є доступними через мережу Інтернет.

Ідея належить Тіму Бернерс-Лі, який таким чином хотів вирішити проблему консолідації інформації, що зберігалась на різних комп’ютерах у датацентрах CERN. Він запропонував ідентифікувати документи та інші ресурси за допомогою URL, який включає протокол передачі, адресу хоста на якому розміщується файл, порт на якому запущено застосунок, та шляхом до файлу, тобто унікальним ідентифікатором у межах цієї машини. Формат URL зараз легко впізнається багатьма користувачами мережі Інтернет, а це protocol:// username:password@hostname:port/path/with/or/without/subdirectories.

Багато інших розробок Тіма, як ось HTTP, що спочатку виглядав лише як надбудова над TELNET чи мова розмітки HTML, зараз є основою сучасної ве-

брозробки, хоча і в дещо змінено вигляді, а на HTTP займає левову частку Інтернет-трафіку.

Перший HTTP-сервер за межами Європи був встановлений у Стенфордському лінійному прискорювальному центрі (SLAC) в Пало-Альто, штат Каліфорнія, для розміщення бази даних SPIERS-HEP. Дані стосовно точної дати цієї події суттєво відрізняються. У часовій шкалі консорціуму World Wide Web йдеться про грудень 1992 року [6], тоді як сам SLAC заявляє про грудень 1991 року [7] [8], як і документ W3C під назвою "Маленька історія всесвітньої павутини" [9]. Основна концепція гіпертексту виникла в попередніх проектах 1960-х років, таких як система редагування гіпертексту (HES) при Браунському університеті, проект Ксанаду Теда Нельсона та система N-Line Дуласа Енгельбарта (NLS). І Нельсон, і Енгельбарт були, в свою чергу, натхнені мемексом Ванневара Буша, який він описав у нарисі 1945 р. "Як ми можемо думати". [10]

Проривом Бернерса-Лі було об’єднання гіпертексту та Інтернету. У своїй книзі "Weaving The Web" він пояснює, що неодноразово підказував членам обох технічних спільнот, що об’єднання цих двох технологіями можливе. Але, коли ніхто не сприйняв його пропозицію серйозно, він взявся за проект самостійно. У процесі роботи він розробив три основні технології:

— система глобально унікальних ідентифікаторів для ресурсів в Інтернеті та інших місцях, універсальний ідентифікатор документа (UDI), пізніше відомий як єдиний локатор ресурсів (URL) та єдиний ідентифікатор ресурсу (URI);

— мова публікації Hypertext Markup Language (HTML);

— протокол передачі гіпертексту (HTTP). [11]

Всесвітня павутина мала декілька відмінностей від інших гіпертекстових систем, наявних у той час. Мережа вимагала лише однонаправлених посилань, а не двонаправлених, що дозволяє комусь посилатися на інший ресурс без дій власника цього ресурсу. Це також значно зменшило труднощі з реалізацією веб-серверів та браузерів (порівняно з попередніми системами), але, в свою чергу,

представило хронічну проблему застарілості посилань з часом. На відміну від попередників, таких як HyperCard, Всесвітня павутина була невласною, що дозволяло самостійно розробляти сервери та клієнтів та додавати розширення без ліцензійних обмежень. 30 квітня 1993 р. CERN оголосив, що всесвітня павутина може бути безкоштовною для всіх[12] Через два місяці після оголошення про те, що серверна реалізація протоколу Gopher більше не вільна у використанні, це призвело до швидкої відмови користувачів від Gopher та переключення на Всесвітню павутину. Ранньою популярною веб-браузером був ViolaWWW для Unix та X Window System.

1.3 Розвиток криптографічних систем

Паралельно відбувався розвиток криптографічних систем. У 1973 році Національне Бюро Стандартів США опублікували запит на відкритий стандарт алгоритму шифрування для потреб уряду. У 1974 році IBM подали і вже через два роки представили свій алгоритм, заснований на мережі Фейстеля[13]. Цей стандарт отримав назву DES і активно використовувався протягом наступних 25 років.

DES є симетричним блоковим шифром, тобто використовує один таємний ключ для шифрування і дешифрування даних, які діляться на окремі блоки по 64 біти (8 байт, чотиримісне машинне слово). Саме повідомлення може мати різну довжину і для шифрування ділиться на блоки відповідної довжини, який доповнюється додатковими бітами у випадку якщо він має меншу довжину, ніж та, що потрібна. Зазвичай, цей процес визначається режимом шифрування (англ. cipher more). Існує кілька режимів шифрування, детально вони розглядаються у пункті 1.4.4.

Симетричне шифрування має один істотний недолік — необхідність домовитися про спільний таємний ключ. Для цього його необхідно попередньо передати через захищений канал.

Цю задачу вирішили Діффі і Хеллман у своїй статті “Нові напрями криптографії”[14]. Вони запропонували метод для узгодження спільного таємного ключа, що формується на основі деяких публічних ключів, якими сторони обмінюються відкрито. Кожна сторона формує таємний ключ за допомогою публічного ключа, отриманого від іншої сторони і свого приватного ключа. Секрет виходить спільним, оскільки публічний ключ кожна сторона (їх може бути кілька) формують на основі приватного. А зловмиснику сформувати таємний ключ не вийде через відсутність приватного ключа бодай однієї сторони, оскільки він не передається по мережі.

Запропонований Діффі і Хеллманом метод дійсно відкрив новий напрям у криптографії — асиметричне шифрування, коли для шифрування і розшифрування даних використовуються різні ключі. Спочатку випадковим чином формується приватний ключ, з нього отримується публічний, для цього використовують важкооборотню функцію, тобто сформувати публічний ключ з приватного просто, а отримати приватний ключ з публічного неможливо за прийнятний час. Перший такий шифр сформували Рон Рівест, Аді Шамір та Леонард Алдеман дещо пізніше, вони використали важкооборотну функцію яку запропонували Діффі і Хеллман, а саме множення великих (1024 біт і більше) простих чисел (обернена операція — факторизація чисел, для якої досі не знайдено рішення за поліноміальний час).

Інша проблема, яку вирішує RSA це перевірка цілісності та авторства повідомлення. Асиметричні криптосистеми дозволяють навпаки використовувати приватний ключ для ЕЦП, а публічний для його перевірки. Таким чином кожен хто знає публічний ключ автора, що має бути унікальним, може перевірити чи дійсно він створив це повідомлення. Цей процес і є електронно-цифровим підписом повідомлення.

Детально процес формування ключів, шифрування/розшифрування даних, а також ЕЦП за допомогою криптосистеми RSA (в честь перших літер прізвищ авторів) можна у оригінальній статті[15].

Множення великих простих чисел не єдина важкооборотня операція. До таких операцій належить множення точки на еліптичній кривій на скаляр, тому це називають криптографією на еліптичних кривих[16]. Перевагою криптографії на еліптичних кривих є забезпечення такої ж стійкості як в RSA при меншій довжині ключа, що робить алгоритм менш вимогливим по ресурсам, що особливо важливо для вбудованих систем[17]. Одним з параметрів криптосистеми є рівняння еліптичної кривої, яка буде використовуватись сторонами. Від цього вибору залежить стійкість всієї криптосистеми, оскільки не кожна крива гарантує важкооборотність операції множення точки на скаляр, тобто дозволяють зловмиснику знайти відповідність між публічним та приватним ключем. Найпопулярніші криві, такі як secp256k1, secp384r1, secp521r1, nistp256, nistp384, nistp521 пройшли ретельний криптоаналіз.

1.4 Обмін публічними ключами

Асиметрична криптографія має одне вразливе місце — обмін публічними ключами. Зловмисник може перехопити і підмінити ці публічні ключі і таким чином здійснити man-in-the-middle атаку. І через відсутність засобів, що допоможуть автентифікувати публічний ключ, а також перевірити чи дійсно інша сторона володіє відповідним приватним ключем для цього публічного ключа конфіденційність та цілісність інформації, що передається ж під загрозою.

Для вирішення цієї проблеми використовують третю сторону, якій довіряють обидві сторони і яка застосовує ЕЦП до їх публічних ключів та різного роду додаткової інформації. Цю третю сторону називають центром сертифікації, а процес накладання ЕЦП на публічний ключ називають сертифікацією. Сертифікат, що містить публічний ключ та іншу інформацію про сторону видається на певний термін, протягом якого він дійсний. Тобто сторона зобов’язана його періодично оновлювати.

Центри сертифікації використовують різні способи для ідентифікації цих сторін перед тим як випустити сертифікати, а також у випадку їх компрометації

вони можуть здійснювати його відкликання, яке, правда, працює не зовсім ефективно.

Також центри сертифікації можуть сертифікувати інші центри сертифікації, при цьому ці центри які сертифіковані самі собою (так звані self-signed сертифікати) називають корінними.

Сертифікати корінних центрів сертифікації часто постачаються з операційною системою чи іншими застосунками.

Разом центри сертифікації, ПЗ та процедури для автентифікації своїх користувачів утворюють інфраструктуру відкритих ключів.

Інфраструктура відкритих ключів (англ. Public PKI) — це набір ролей, політики, апаратного забезпечення, програмного забезпечення та процедур, необхідних для створення, управління, поширення, використання, зберігання та відкликання цифрових сертифікатів та керування шифруванням відкритого ключа. Метою PKI є сприяння безпечній передачі інформації для електронних мереж, таких як електронна комерція, інтернет-банкінг та конфіденційна електронна пошта. Це потрібно для випадків, коли прості паролі є неадекватним методом аутентифікації, а для підтвердження особи сторін, які беруть участь у спілкуванні, та перевірки інформації, що передається, необхідне більш жорстке підтвердження.

У криптографії PKI — це домовленість, яка пов'язує відкриті ключі з відповідними особами суб'єктів (наприклад, людей та організацій). Прив'язка встановлюється через процес реєстрації та видачі сертифікатів у та сертифікаційному органі (CA). Залежно від рівня впевненості зв'язку, це може здійснюватися автоматизованим процесом або під наглядом людини.

Роль PKI, яка забезпечує дійсну та правильну реєстрацію, називається органом реєстрації (RA). RA відповідає за прийняття запитів на цифрові сертифікати та автентифікацію суб'єкта, що подає запит. [18] У Microsoft PKI Microsoft, орган реєстрації, як правило, називається підпорядкованим CA. [19]

Суб'єкт господарювання повинен бути однозначно ідентифікований у кожному домені CA на основі інформації про цю сутність. Сторонній орган з перевірки (VA) може надати цю особу інформацію від імені CA.

Стандарт X.509 визначає найбільш часто використовуваний формат для сертифікатів відкритого ключа. [20]

PKI складається з наступних частин: [21] [22]

— орган сертифікації (CA), який зберігає, видає та підписує цифрові сертифікати;

— реєстраційний орган (PA), який перевіряє особу суб'єктів, які вимагають зберігати їх цифрові сертифікати в ЦЗ;

— центральний каталог - тобто захищене місце, в якому зберігаються та індексуються ключі;

— система управління сертифікатами, яка керує такими речами, як доступ до збережених сертифікатів або доставка сертифікатів, що підлягають видачі;

— політика щодо сертифікатів із зазначенням вимог PKI щодо її процедур, що дозволяє стороннім людям проаналізувати надійність PKI.

1.5 Порівняння протоколів для криптографічного захисту інформації, що передається між вузлами

Найвідомішими альтернативами протоколу TLS є IPSec та OpenVPN.

Internet Protocol (IPsec) — це безпечний набір мережевих протоколів, який автентифікує та шифрує пакети даних для забезпечення захищеного зашифрованого зв'язку між двома комп'ютерами через мережу Інтернет. Він використовується у віртуальних приватних мережах (VPN).

IPsec включає протоколи для встановлення взаємної аутентифікації між агентами на початку сеансу та узгодження криптографічних ключів для використання під час сеансу. IPsec може захищати потоки даних між парою хостів (хост-хост), між парою шлюзів безпеки (мережа-мережа) або між шлюзом безпеки та хостом (мережа-хост). IPsec використовує криптографічні методи захисту для захищеної комунікацій через IP-мережі. Він підтримує однорангову аутентифікацію на рівні мережі, автентифікацію відправника даних, цілісність даних, конфіденційність даних (шифрування) та захист від повторного відтворення.

IPv4 був розроблений з невеликими положеннями про безпеку. Як частина вдосконалення IPv4, IPsec є моделлю OSI рівня 3 або схемою захисту від кінця до кінця в Інтернеті, тоді як деякі інші системи безпеки в Інтернеті широко використовуються над рівнем 3, наприклад, протоколи TLS і SSH, обидва вони працюють на транспортному шарі. IPsec може автоматично захищати програми на рівні IP.

Починаючи з початку 1970-х років, Агентство передових дослідницьких проектів спонсорувало серію експериментальних пристроїв шифрування ARPANET, спочатку для нативного шифрування пакетів ARPANET, а згодом для шифрування пакетів TCP/IP; деякі з них були сертифіковані та перевірені на місцях. З 1986 по 1991 рік АНБ спонсорувала розробку протоколів безпеки для Інтернету в рамках програми "Безпечні мережі передачі даних" (SDNS). [23] Це об'єднало різних постачальників, в тому числі Motorola, які виробили мережевий пристрій шифрування в 1988 році. Робота була відкрито опублікована приблизно від 1988 року NIST, і з них Протокол безпеки на 3 рівні (SP3) може врешті-решт перетворитись у стандарт стандарту ISO Network Network Protocol Protocol (NLSP). [24]

З 1992 по 1995 роки різні дослідницькі групи працювали над покращенням SP3 SDNS. У 1992 році Лабораторія досліджень морських досліджень США (NRL) розпочала проект SIPP для дослідження та впровадження шифрування IP. У грудні 1993 р. Експериментальний протокол шифрування програмного забезпечення IP (swIPe) був розроблений на SunOS в Колумбійському університеті та AT&T Bell Labs Джона Іоанідіса та інших. Вей Сю в довірених інформаційних системах (TIS) провів дослідження SWIPe, [25] розширив протоколи безпеки IP та розробив драйвер пристрою стандарту шифрування даних. До грудня 1994 року його команда випустила продукт брандмауера TIS Gauntlet з інтегрованим апаратним шифруванням 3DES та забезпечила комерційну безпеку IP із швидкістю T1, забезпечивши мережі між узбережжям Сходу та Заходу США.

Протягом цього періоду Робоча група з питань захисту інформації з Інтернету (IETF) створила робочу групу [26] для стандартизації цих зусиль як від-

критого, вільно доступного набору розширень безпеки, в подальшому названого IPsec. [27] У 1995 році робоча група опублікувала RFC-1825 через RFC-1827, при цьому NRL отримав першу робочу реалізацію. [28]

OpenVPN — це комерційне програмне забезпечення з відкритим кодом, яке реалізує методи віртуальної приватної мережі (VPN) для створення захищених з'єднань "точка-точка" в маршрутизованих або мостових конфігураціях та об'єктах віддаленого доступу. Для цього використовується користувацький протокол безпеки [29], який використовує SSL/TLS для обміну ключами. Він здатний обходити транслятори мережевих адрес (NAT) та брандмауери. Він був написаний Джеймсом Йонаном і опублікований за загальною ліцензією GNU (GPL). [30]

OpenVPN дозволяє хостам аутентифікувати один одного за допомогою загальнодоступних секретних ключів, сертифікатів або імені користувача та пароля. При використанні в конфігурації мультиклієнт-сервера, він дозволяє серверу випустити сертифікат аутентифікації для кожного клієнта, використовуючи повноваження підписів та сертифікатів. Він широко використовує бібліотеку шифрування OpenSSL, а також протокол TLS і містить безліч функцій безпеки та контролю.

OpenVPN був перенесений та вбудований у декілька систем. Наприклад, DD-WRT має серверну функцію OpenVPN. SoftEther VPN, багатопротокольний VPN-сервер, має реалізацію протоколу OpenVPN.

OpenVPN використовує бібліотеку OpenSSL для забезпечення шифрування як даних, так і каналів управління. Це дозволяє OpenSSL виконувати всі роботи з шифрування та автентифікації, дозволяючи OpenVPN використовувати всі шифри, наявні в пакеті OpenSSL. Він також може використовувати функцію автентифікації пакетів HMAC, щоб додати додатковий рівень безпеки до з'єднання (його розробник називає "брандмауер HMAC"). Він також може використовувати апаратне прискорення для кращої продуктивності шифрування. Підтримка mbed TLS доступна починаючи з версії 2.3.

У OpenVPN є кілька способів аутентифікації однолітків один з одним. OpenVPN пропонує загальнодоступні ключі, на основі сертифікатів, а також автентифікацію на основі імені/пароля. Попередній секретний ключ є найпростішим, а на основі сертифікатів - найміцніший і багатий на функції. У версії 2.0 автентифікацію імені користувача/пароля можна ввімкнути як з сертифікатами, так і без них. Однак, щоб використовувати автентифікацію імені користувача / пароля, OpenVPN залежить від сторонніх модулів.

OpenVPN може виконувати для транспортування UDP або TCP, мультиплексування створених тунелів SSL на одному порті TCP/UDP [31] (RFC 3948 для UDP). [32]

Починаючи з серії 2.3.x, OpenVPN повністю підтримує IPv6 як протокол віртуальної мережі всередині тунелю, а додатки OpenVPN також можуть встановлювати з'єднання через IPv6. [33] Він має можливість працювати через більшість проксі-серверів (включаючи HTTP) і добре працює за допомогою перекладу мережевих адрес (NAT) та виходу через брандмауері. Конфігурація сервера має можливість надсилати та застосовувати певні параметри конфігурації мережі клієнтам. До них відносяться IP-адреси, команди маршрутизації та кілька варіантів підключення. OpenVPN пропонує два типи інтерфейсів для роботи в мережі через драйвер Universal TUN/TAP. Він може створити або IP-тунель на основі рівня 3 (TUN), або Ethernet TAP на основі рівня 2, який може нести трафік будь-якого типу Ethernet. OpenVPN може додатково використовувати бібліотеку стиснення LZO для стиснення потоку даних. Порт 1194 - це офіційний номер порту IANA для OpenVPN. Більш нові версії програми тепер за замовчуванням використовують саме цей порт. Особливість у версії 2.0 дозволяє одним процесом керувати кількома одночасними тунелями, на відміну від вихідного обмеження "один тунель на процес" для серії 1.x.

Використання OpenVPN загальних мережевих протоколів (TCP і UDP) робить його бажаною альтернативою IPsec у ситуаціях, коли Інтернет-провайдер може блокувати конкретні протоколи VPN.

Коли OpenVPN використовує транспортування протоколу управління передачею (TCP) для встановлення тунелю, продуктивність буде прийнятною лише до тих пір, поки на не тунельованій мережевій лінії зв'язку буде достатня надмірна пропускна здатність, щоб гарантувати, що тунельовані таймери TCP не закінчуються. В іншому випадку продуктивність різко падає. Це відоме як "проблема зриву TCP". [34] [35]

OpenVPN працює в просторі користувачів, а не вимагає роботи стека IP (отже, ядра). OpenVPN має можливість скидати кореневі привілеї, використовувати mlockall, щоб запобігти заміні чутливих даних на диску, ввести chroot після ініціалізації та застосувати контекст SELinux після ініціалізації.

OpenVPN запускає користувальницький протокол безпеки, заснований на SSL та TLS [36], а не підтримує IKE, IPsec, L2TP або PPTP. OpenVPN пропонує підтримку смарт-карт за допомогою криптографічних жетонів на основі PKCS # 11.

OpenVPN можна розширити сторонніми плагінами або скриптами, які можна викликати у визначених точках входу. [37] [38] Метою цього часто є розширення OpenVPN за допомогою вдосконаленого ведення журналів, розширеної аутентифікації з іменем користувача та паролями, динамічних оновлень брандмауера, інтеграції RADIUS тощо. Плагіни — це динамічно завантажувані модулі, як правило, написані на C, тоді як інтерфейс сценаріїв може виконувати будь-які скрипти або двійкові файли, доступні OpenVPN. У вихідному коді OpenVPN [39] є кілька прикладів таких плагінів, включаючи плагін автентифікації PAM. Існує також декілька сторонніх плагінів для автентифікації за допомогою баз даних LDAP або SQL, таких як SQLite та MySQL. [40]

Висновки до розділу 1

За результатами проведеного дослідження можна зробити наступні висновки:

1. Кількість користувачів глобальної мережі та обсяг трафіку, що проходить через неї постійно збільшується.

2. Зловмисники можуть здійснювати перехоплення та підміну трафіку у мережі, що становить загрозу конфіденційності та цілісності інформації.

3. Сучасні криптографічні методи захисту інформації мають засоби для вирішення цієї проблеми за допомогою шифрування, ЕЦП та інфраструктури відкритих ключів.

4. Криптографічний захист інформації може відбуватися як на мережевому рівні, так і на прикладному. Для останнього найпопулярнішим протоколом є SSL/TLS.

2 АНАЛІЗ ВРАЗЛИВОСТЕЙ РЕАЛІЗАЦІЇ ПРОТОКОЛУ TLS

2.1 Загальні відомості про протокол TLS

Протокол TLS (Transport Layer Security) покликаний забезпечити шифрування даних на рівні представлення та прикладного рівня моделі OSI між двома кінцевими вузлами (т.з. end-to-end encryption). У цьому протоколі ефективно використовуються описані у першому розділі принципи криптографічні методи для забезпечення конфіденційності та цілісності інформації, що передається.

2.1.1 Історія версій SSL/TLS

Перша спроба описати інтерфейс для захищеної передачі інформації між двома застосунками по мережі здійснена у 1994 році, коли група вчених з Техаського університету виклали працю [41]з високорівневим абстрактним API для захищеного мережевого програмування, що в основному повторював сокети Берклі[42].

Паралельно у компанії Netscape Communications розробляли свій власний протокол для використання при передачі електронних листів та файлів між клієнтом та сервером і навпаки. Перша версія мала багато недоліків і, ніколи не була опублікована.

Друга модернізована версія була представлена в 1995 році[43]. Через рік вийшла оновлена версія[44] SSLv3 з підтримкою DH-RSA, HMAC-SHA1.

У вересні 1999 року SSL було стандартизовано IEFT під новою назвою TLSv1.0[45]. У протокол було додано підтримку ще більшої кількості алгоритмів передачі ключів, як от ECDH-ECDSA, PSK, PSK-RSA, SRP і Kerberos. Серйозним недоліком даної версії була можливість вернутися до більш вразливої на той час версії SSLv3 (так званий downge).

Стандарт TLSv1.1 вийшов у квітні 2006 року[46] і окрім вилучення вразливого вже на той час блокового шифру RC2, було виправлено кілька проблем з режимом CBC блокового шифрування, а саме покращено опрацювання

помилок з вирівнюванням блоку та передачу вектора ініціалізації. Також у протоколі TLSv1.1 було додано підтримку IANA Enterprise Private Numbers[47].

У TLSv1.2 було нарешті вилучено обернену сумісність з SSL та додано нові режими блокового шифрування GCM та CCM, які в подальшому замінили вразливий наразі CBC[48]. Також була здійснена тотальна заміна MD5 та SHA-1 на SHA-2, а саме SHA-256.

Таблиця 2.1 — Криптографічні методи автентифікації та їх підтримка різними версіями стандарту SSL/TLS

Алгоритм	SSLv2	SSLv3	TLSv1.0	TLSv1.1	TLSv1.2	TLSv1.3
RSA	+	+	+	+	+	-
DH-RSA	-	+	+	+	+	-
DHE-RSA	-	+	+	+	+	+
ECDH-RSA	-	-	+	+	+	-
ECDHE-RSA	-	-	+	+	+	+
DH-DSS	-	+	+	+	+	-
DHE-DSS	-	+	+	+	+	-
ECDH-ECDSA	-	-	+	+	+	-
ECDHE-ECDSA	-	-	+	+	+	+
PSK	-	-	+	+	+	-
PSK-RSA	-	-	+	+	+	-
DHE-PSK	-	-	+	+	+	-
ECDHE-PSK	-	-	+	+	+	-
SRP	-	-	+	+	+	-
SRP-DSS	-	-	+	+	+	-
SRP-RSA	-	-	+	+	+	-
Kerberos	-	-	+	+	+	-

TLSv1.3 вийшов відносно недавно у серпні 2018 року[49]. Метою його прийняття було провести глибоку ревізію всього протоколу, в ході якої була вилучена підтримка слабких хеш-функцій MD5 і SHA-224, слабких та маловживаних

еліптичних кривих, PSK, всі алгоритми передачі інформації та всі не ефемерні (англ. ephemeral) алгоритми формування спільного таємного ключа, тобто ті, які формують завжди один і той самий секрет для сторін, оскільки розсекречення цього ключа дасть зловмиснику можливість розшифровувати будь які дані що передаються між сторонами, поки хтось з них не змінить приватний і публічний ключ. В результаті залишено лише три алгоритми.

Загалом, підтримувані різними версіями стандартів SSL/TLS криптографічні методи обміну таємним ключем, симетричного шифрування та забезпечення цілісності наведені у таблицях 2.1, 2.2 та 2.3 відповідно.

Таблиця 2.2 — Симетричні алгоритми шифрування, що підтримуються протоколом SSL/TLS

Алгоритм	SSLv2	SSLv3	TLSv1.0	TLSv1.1	TLSv1.2	TLSv1.3
AES GCM	-	-	-	-	+	+
AES CCM	-	-	-	-	+	+
AES CBC	-	-	+	+	+	-
Camellia GCM	-	-	-	-	+	-
Camellia CBC	-	-	+	+	+	-
ARIA GCM	-	-	-	-	+	-
ARIA CBC	-	-	+	+	+	-
SEED CBC	-	-	+	+	+	-
3DES EDE CBC	+	+	+	+	+	-
IDEA CBC	+	+	+	+	-	-
DES CBC	+	+	+	+	-	-
RC2 CBC	+	+	+	-	-	-
ChaCha20-Poly1305	-	-	-	-	+	+
RC4	+	+	+	+	+	-

Таблиця 2.3 — Алгоритми забезпечення цілісності

Алгоритм	SSLv2	SSLv3	TLSv1.0	TLSv1.1	TLSv1.2	TLSv1.3
HMAC-MD5	+	+	+	+	+	-
HMAC-SHA1	-	+	+	+	+	-
HMAC-SHA2	-	-	-	-	+	-
AEAD	-	-	-	-	+	+

2.1.2 Процес передачі інформації

Наразі найновішою та найзахищенішою версією протоколу є TLSv1.3, що і буде надалі розглянута. Дана версія визначає три підпротоколи, які мають бути реалізовані програмістом для захищеного обміну інформацією між вузлами:

1. Handshake Protocol — протокол обміну публічними ключами та формування спільного секрету.

2. Record Protocol — протокол передачі даних, що зашифровані за допомогою симетричного алгоритму попередньо погодженим таємним ключем.

3. Alert Protocol — протокол обміну подіями чи помилками.

Розглянемо структуру обміну даними по протоколу TLS на реальному прикладі, а саме HTTPS, тобто HTTPv1.1 over TLS.

Для цього було сконфігуровано простий сервер, що передає клієнту стрічку “Hello World!” у відповідь на його запит та здійснено перехоплення сесії за допомогою застосунку Wireshark.

Звичайно, оскільки всі дані прикладного рівня, а саме HTTP-запит клієнта та відповідь сервера є зашифрованими, то перехоплення не досить, щоб порушити конфіденційність чи цілісність інформації. Для відлагодження необхідно ще завантажити у Wireshark keylogfile з ключами шифрування, що використовуються для TLS-сесії (Edit > Preferences > Protocols > SSL > “(Pre)-Master-Secret log filename”). Для формування keylogfile браузер Mozilla Firefox достатньо запустити

з встановленою змінною SSLKEYLOGFILE, яка міститиме шлях, по якому цей файл записувати.

Приклад перехопленого та розшифрованого за допомогою Wireshark HTTPS з'єднання зображено на рис. 2.1.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.2	TCP	74	59428 → 443 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1
2	0.000019824	127.0.0.2	127.0.0.1	TCP	74	443 → 59428 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 S
3	0.000035448	127.0.0.1	127.0.0.2	TCP	66	59428 → 443 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=2700272031
4	0.001962915	127.0.0.1	127.0.0.2	TLSv1.3	711	Client Hello
5	0.001980770	127.0.0.2	127.0.0.1	TCP	66	443 → 59428 [ACK] Seq=1 Ack=646 Win=45056 Len=0 TSval=12935598
6	0.002647457	127.0.0.2	127.0.0.1	TLSv1.3	322	Server Hello, Change Cipher Spec, Encrypted Extensions, Finish
7	0.002658710	127.0.0.1	127.0.0.2	TCP	66	59428 → 443 [ACK] Seq=646 Ack=257 Win=44800 Len=0 TSval=270027
8	0.003474819	127.0.0.1	127.0.0.2	TLSv1.3	146	Change Cipher Spec, Finished
9	0.006345102	127.0.0.1	127.0.0.2	HTTP	435	GET / HTTP/1.1
10	0.006385528	127.0.0.2	127.0.0.1	TCP	66	443 → 59428 [ACK] Seq=257 Ack=1095 Win=46336 Len=0 TSval=12935
11	0.006774134	127.0.0.2	127.0.0.1	HTTP	249	HTTP/1.1 200 OK (text/html)
12	0.049840702	127.0.0.1	127.0.0.2	TCP	66	59428 → 443 [ACK] Seq=1095 Ack=440 Win=45952 Len=0 TSval=27002
17	5.012693976	127.0.0.2	127.0.0.1	TCP	66	443 → 59428 [FIN, ACK] Seq=440 Ack=1095 Win=46336 Len=0 TSval=
18	5.014225121	127.0.0.1	127.0.0.2	TLSv1.3	90	Alert (Level: Warning, Description: Close Notify)
19	5.014317887	127.0.0.2	127.0.0.1	TCP	54	443 → 59428 [RST] Seq=441 Win=0 Len=0

Рисунок 2.1 — Приклад TLS сесії

TLS-з'єднання ініціює клієнт, для цього він передає повідомлення Client Hello, що є частиною Handshake Protocol. Дане повідомлення детальніше зображено на рис. 2.2 та містить у собі наступні частини:

1. Звичайний заголовок TLS-повідомлення, що містить поля Content Type (тип підпротоколу), Version (версія протоколу, задля сумісності Client Hello завжди рівний 0x0301, тобто TLSv1.0) та Length (довжина повідомлення).

2. Повідомлення Handshake-протоколу, що інкапсульовано у TLS-повідомлення.

В залежності від поля Handshake Type, тобто стадії з'єднання, різняться поля, що містить повідомлення Handshake-протоколу. Для Client Hello це розширення (англ. extension) supported_versions, яка містить значення 0x0304, тобто найновішу версію протоколу TLSv1.3, обране клієнтом випадкове число (яке буде використовуватись у подальшому в алгоритмі формування спільного секрету), а також підтримувані клієнтом алгоритми.

Повідомлення Server Hello (рис. 2.3) містить обране сервером випадкове число та обраний сервером шифр. У прикладі це AES-256-GCM з SHA384 для забез-

печення цілісності.

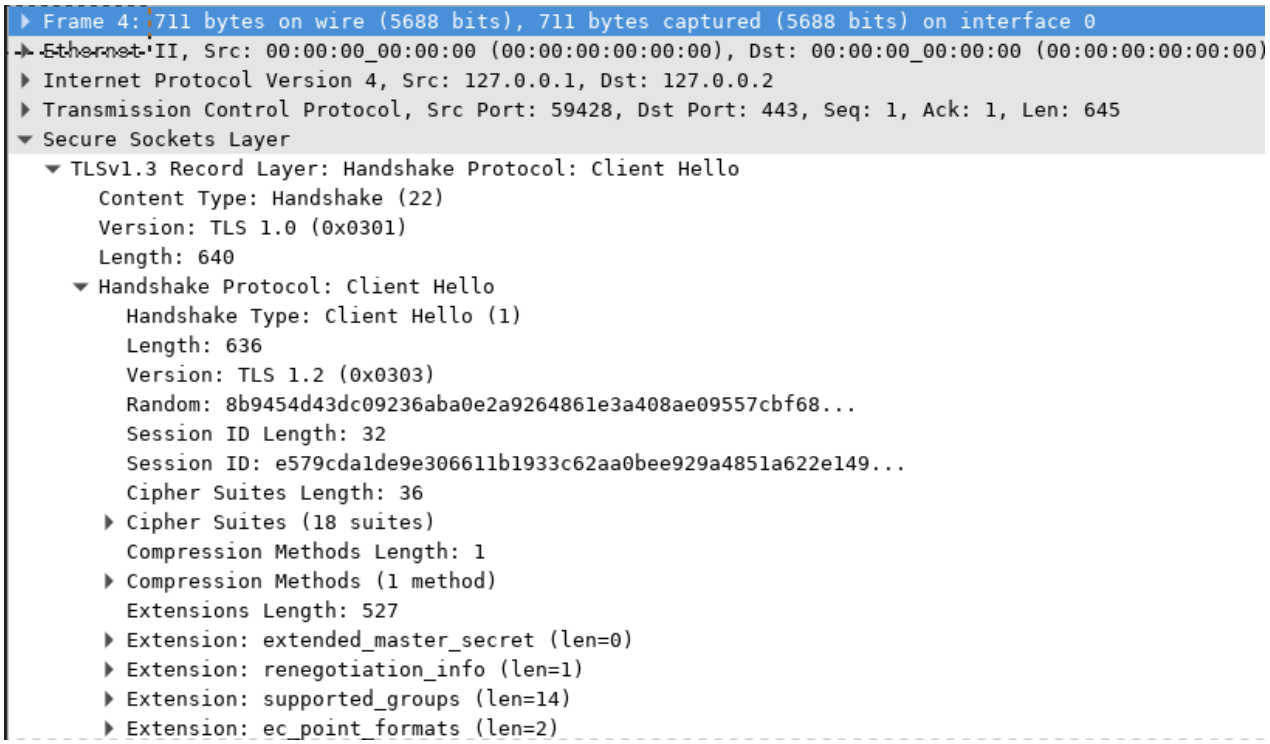


Рисунок 2.2 — Структура Client Hello

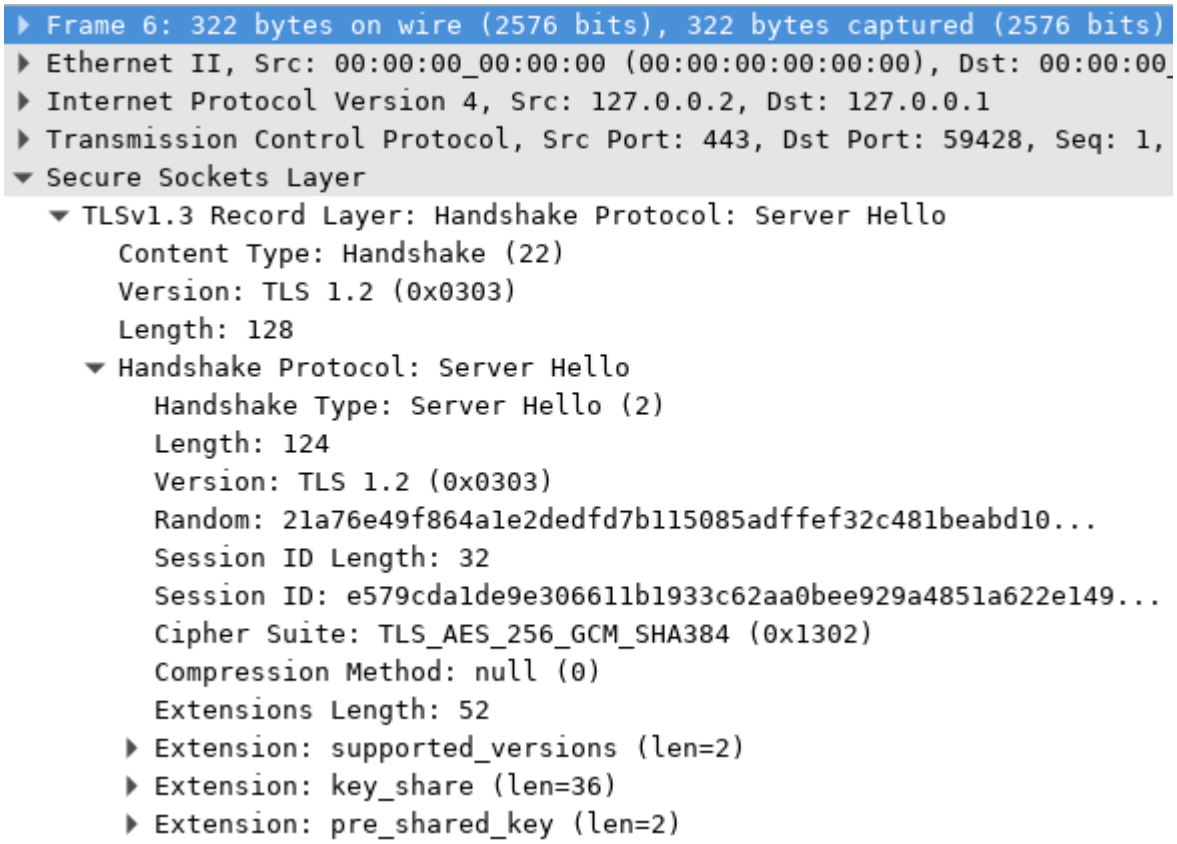


Рисунок 2.3 — Структура Server Hello

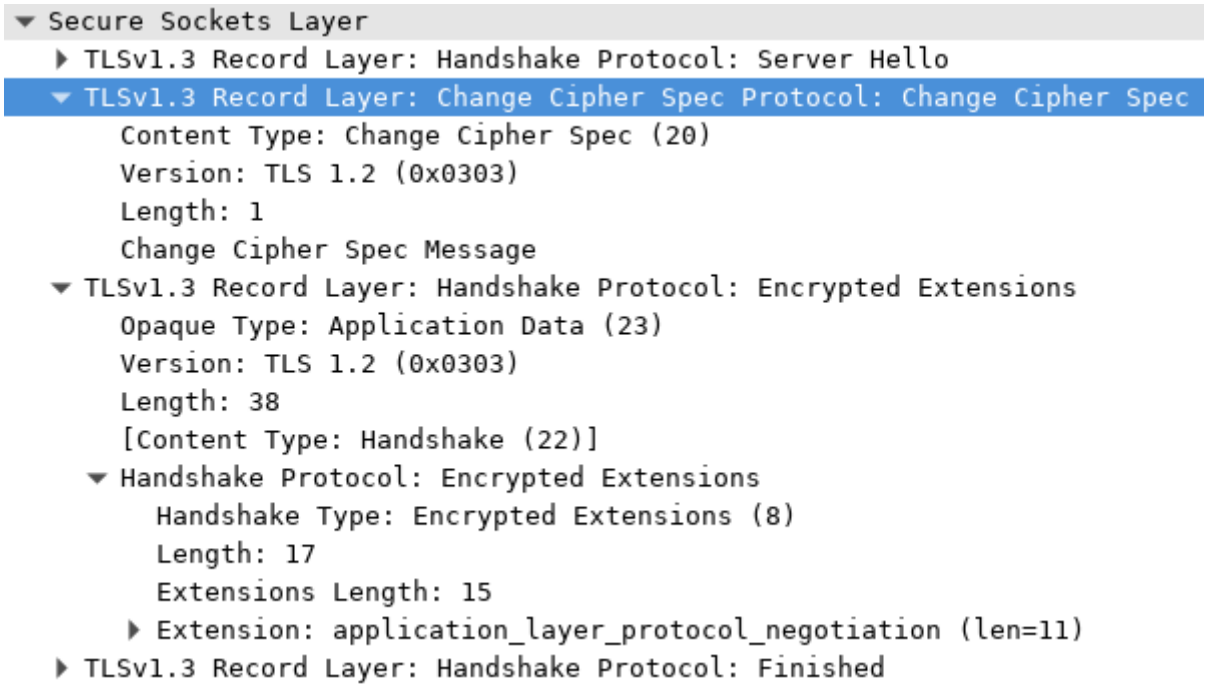


Рисунок 2.4 — Структура Change Cipher Spec, Encrypted Extensions

Одразу з повідомленням Server Hello, сервер надсилає повідомлення Change Cipher Spec Protocol, Encrypted Extensions (рис. 2.4) та Finished (рис. 2.5). Повідомлення Change Cipher Spec Protocol належить до однойменного протоколу, який не використовується у новій версію стандарту і відправляється лише в цілях сумісності. Про бажання сервера домовитися про подальшу передачу інформації через Record Protocol свідчить передане повідомлення Encrypted Extensions.

Повідомлення Finished інформацію для перевірки іншою стороною правильності формування спільного таємного ключа.

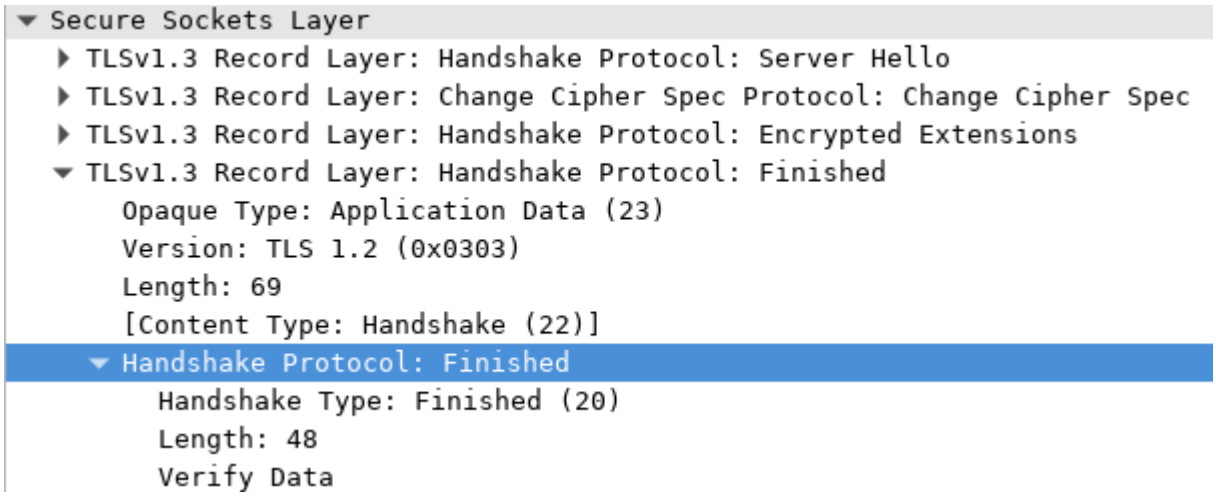


Рисунок 2.5 — Структура Finished

У випадку, якщо клієнт погоджується, він надсилає (див. рис. 2.6) пакунок з Change Cipher Spec, в цілях сумісності, як і сервер, та Finished. Останній теж містить інформацію для сервера, щоб можна було перевірити правильність встановлення TLS-з'єднання з клієнтом.

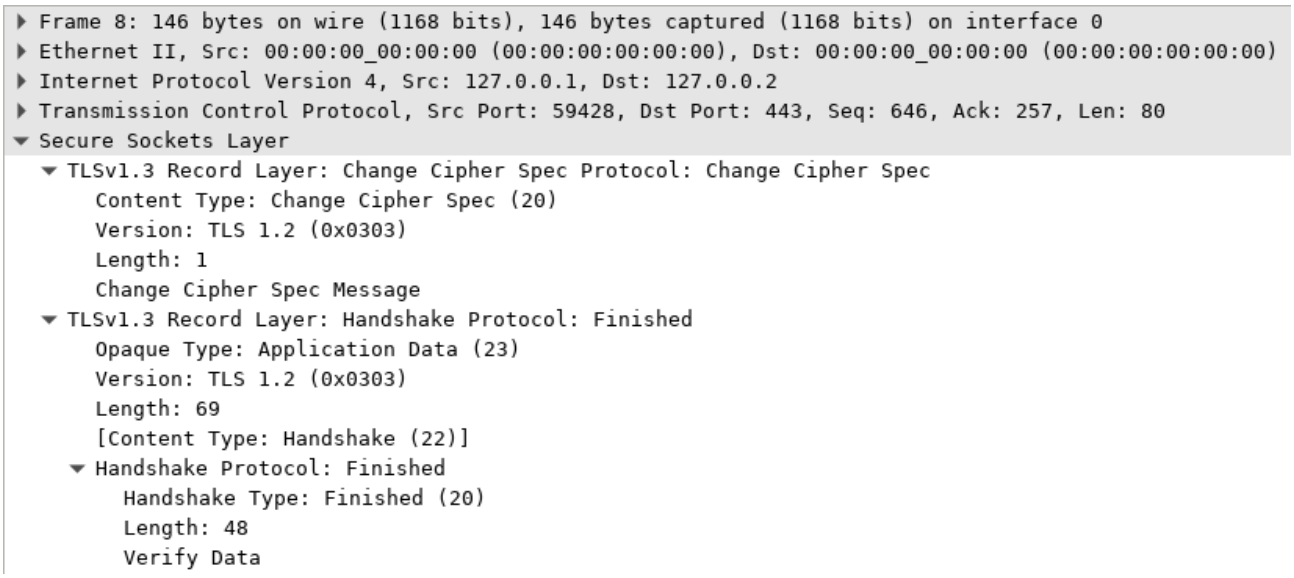


Рисунок 2.6 — Структури Change Cipher Spec, Finished

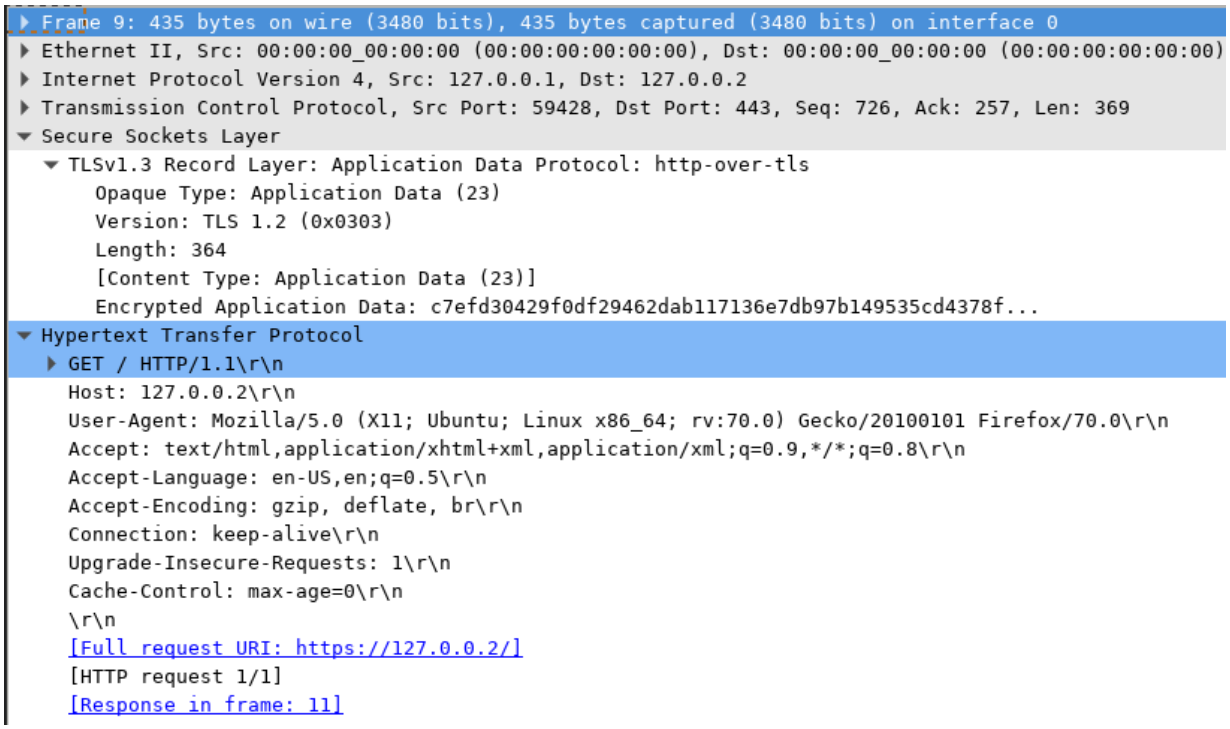


Рисунок 2.7 — Запит клієнта через TLS

На рис. 2.7 та рис. 2.8 відображено запит клієнта HTTP та відповідь сервера через протокол TLS. Дані інкапсулюються у Record Protocol (Application Data Protocol у Wireshark, на відміну від стандарту) у зашифрованому вигляді, які Wireshark зміг розшифрувати завдяки keylogfile.

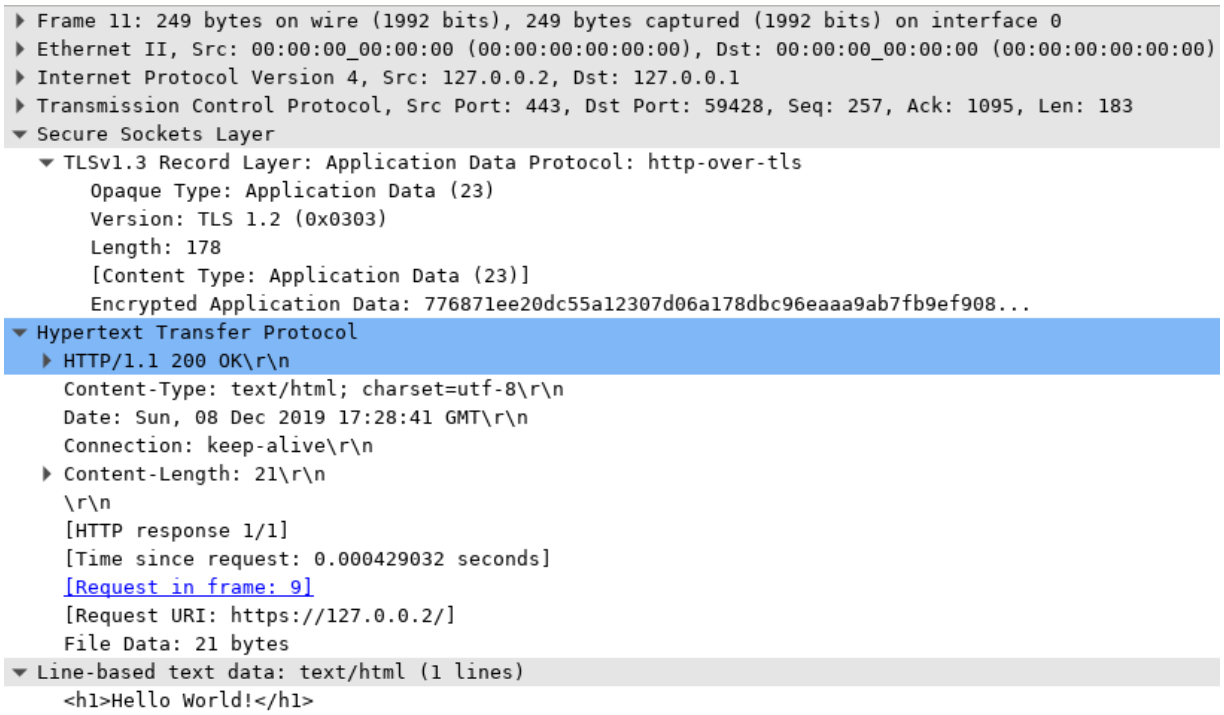


Рисунок 2.8 — Відповідь сервера через TLS

Alert-протокол досить простий, але багатоцільовий протокол, про який детальніше можна прочитати у самому стандарті [49, с. 85]. У даному прикладі (рис. 2.9) повідомленням Close Notify (0x00) сервер інформується про завершення з'єднання після того як протокол верхнього рівня (HTTP) його вже не потребує, наприклад у зв'язку з завершенням передачі.

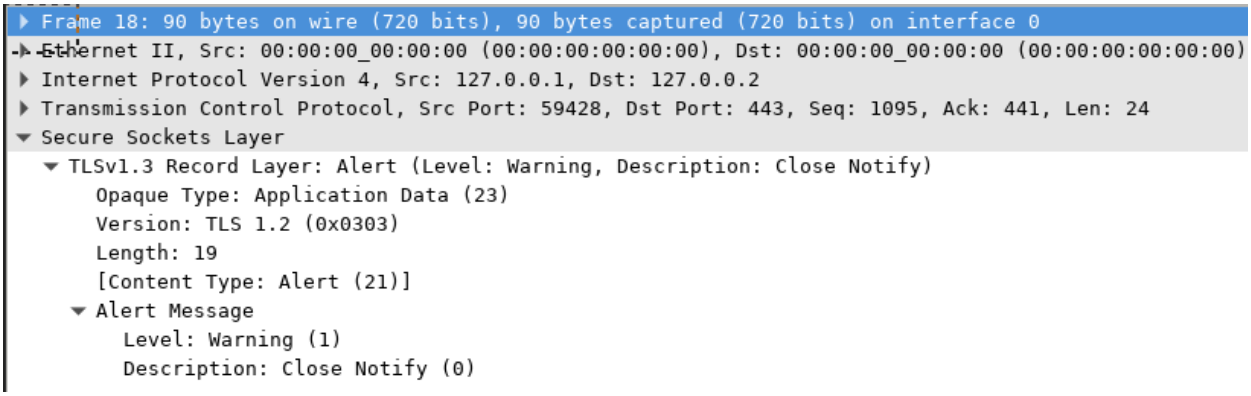


Рисунок 2.9 — Alert стосовно закриття з'єднання

2.2 Відкриті реалізації SSL/TLS

Реалізація SSL/TLS — це застосунок чи бібліотека, яку можуть використовувати застосунки, яка повністю або частково виконує вимоги одного чи кількох стандартів, що стосуються SSL/TLS, таким чином забезпечуючи засоби для захищеної передачі даних між двома застосунками у мережі за допомогою сучасних методів криптографічного захисту інформації.

2.2.1 NSS

Першою реалізацією SSL/TLS є Network Security Services. Вона бере свій початок у компанії Netscape, всередині якої і був розроблений SSL.

Зараз NSS використовується у продуктах Mozilla, таких як Firefox, Firefox OS, SeaMonkey, Thunderbird; клієнтські застосунки з відкритим сирцевим кодом LibreOffice, Evolution, Pidgin; mod_nss модуль для Apache HTTP Server.

2.2.2 OpenSSL

OpenSSL бере свій початок у 1995 році, коли програміст Ерік Янг почав розробку відкритої реалізації недавно випущеного стандарту протоколу SSL з підтримкою RC2 і RC4 під назвою SSLeay. Також проєкт включав одну з перших відкритих реалізації шифру DES.

Загалом, SSLeay популяризував end-to-end шифрування у мережі Інтернет, оскільки дав змогу розробникам ПЗ додавати його підтримку у свої продукти. Наприклад, Тім Гадсон, один з розробників, створив першу імплементацію FTPS (FTP over SSL), тобто захищену версію протоколу для передачі файлів між клієнтом і сервером.

Однак розробка SSLeay фактично припинилася у 1998 році. Попри це, проєкт продовжив розвиватися іншою групою програмістів з новою назвою OpenSSL. При цьому була збережена версійність.

Наразі OpenSSL розвивається спільнотою з 17 розробників, лише двоє з яких є найманими працівниками некомерційної організації The OpenSSL Software Foundation, тоді як інші є волонтерами.

Дана бібліотека є чи не найпопулярнішою і використовується багатьма проєктами — Node.js, Nginx, PostgreSQL, MySQL, Virtualbox та інші. Також, існує багато форків OpenSSL, найпопулярнішими з яких є LibreSSL та BoringSSL. Останній створений та використовується для браузера Google Chrome.

2.2.3 GnuTLS

OpenSSL хоча і є проєктом з відкритим вихідним кодом, але використовує Apache License 2.0, яка не сумісна з ліцензією GPL, під якою поширюється ПЗ GNU. Тому у березні 2003 року Нікос Маврогіаннопоулос створив проєкт GnuTLS, який мав дозволити іншим бібліотекам та застосункам ОС GNU використовувати TLS для своєї роботи.

GnuTLS побудованої на основі низькорівневої криптографічної бібліотеки nettle.

Зараз дану реалізацію використовує значна кількість програмного забезпечення, найвідомішими є стільникове середовище GNOME, поштовий сервер Exim, аналізатор мережевих пакетів Wireshark, браузер Lynx, система для друку CUPS та текстовий редактор Emacs.

2.3 Типи вразливостей

Вразливість ПЗ — це результат допущених помилок чи інших недоліків під його проєктування чи безпосереднього кодування, які “спричиняють нездатність системи протистояти реалізації певної загрози або ж сукупності загроз”[50, ст. 22].

В залежності від того, яку саме загрозу становить вразливість, їх можна поділити на наступні категорії:

— відмова в обслуговуванні (DoS);

— загроза виконання коду;

— переповнення буферу;

— обхід чогось, а саме певного етапу встановлення з’єднання чи передачі інформації каналом, що становить загрозу конфіденційності чи цілісності інформації;

— псування пам’яті.

Кожен з типів вразливостей детально описаний у відповідному пункті цього підрозділу.

2.3.1 Відмова в обслуговуванні

Відмова в обслуговуванні є атакою на доступність інформації, а точніше сервісу, що надає до неї доступ по протоколу TLS.

Зазвичай, відмова в реалізаціях TLS відбувається через зупинку ПЗ чи використанням всіх наявних ресурсів (пам’яті чи процесорного часу), що спричиняють відправлені зловмисником дані протягом однієї чи кількох TLS-сесій через їх неправильну обробку вразливим ПЗ.

Так звану аварійну зупинку ПЗ може спричиняти критична помилка. Це або виклик неправильної інструкції (SIGILL), невірна арифметична операції (SIGFPE), помилка при роботі з пам’яттю (SIGSEGV) чи інші сигнали ОС, або виклик системного виклику exit чи exec, які можуть викликатися програмою за певних обставин.

2.3.2 Переповнення буфера

Переповнення буфера (англ. Buffer Overflow) — це аномалія, в ході якої ПЗ під час запису у буфер переписує сусідню ділянку пам’яті, що слідує до чи після буфера, залежно від того де розташовується буфер та архітектури а також алгоритму запису.

Вразливими до цієї аномалії C. Стандартна бібліотека цієї мови відома невдалими рішеннями своїх розробників в плані безпеки. Наприклад, такі функції для запису стрічок як strcpy, sprintf, vsprintf вразливі і можуть дозволяти переповнення буферу. Проблема, очевидно, була настільки масштабною, що у компіляторах Microsoft (MSVC) з'явились безпечні аналоги strcpy_s, sprintf_s, дійшло навіть до strlen_s. Однак ці функції не прижились і у стандарт потрапили ідентичні їм по сигнатурі strcpy, snprintf, vsnprintf, але не strnlen.

Попри це все, вразливі функції продовжують бути частиною стандарту і навіть використовуватися молодими програмістами у своїх застосунках.

2.3.3 Спонтанне виконання коду

Спонтанне виконання коду (англ. Arbitrary code execution) дозволяє зловмиснику виконувати спонтанні команди чи код на цільовій машині, тобто тій, на якій виконується реалізація TLS.

На певних архітектурах, зокрема i386 та x86_64, дана вразливість може виникати через переповнення буфера стеку. Оскільки адреса повернення з функції міститься у стеку, то вона може бути замінена на адресу у стеку чи глобальному сегменті пам'яті, яка містить користувальницькі дані, які можуть містити виконуваний код. У разі якщо вони не є валідними, найімовірніше, ОС викличе сигнал SIGILL — Illegal instruction, що призведе до відмови в обслуговуванні.

Ця вразливість також може виникати у високорівневих скриптових мовах програмування у яких є конструкції на зразок eval, що дозволяють викликати стрічку як програмний код (скрипт). Це стосується JavaScript, PHP, Python та інших. Тому, використовувати eval є поганою практикою.

2.3.4 Обхід

Обхід — це загроза, яка дозволяє зловмиснику просто обійти певну процедуру передбачену політикою безпеки, що покликана забезпечити певну властивість безпеки інформації.

Наприклад, SQL-ін’єкції у формі входу чи методі авторизації користувача API дозволяють зловмиснику вказати такі дані, що робить можливим просто обійти процес авторизації через модифікацію запиту таким чином, щоб він завжди виконувався успішно, як і у випадку коли існуючий користувач авторизується з відповідним йому паролем.

У випадку з протоколом TLS це обхід певних етапів рукостискання чи процесу передачі інформації. Наприклад, здатність сервера за допомогою певних маніпуляцій змусити клієнта пропустити перевірку серверного сертифікату і таким чином непомітно для нього реалізувати man-in-the-middle атаку.

2.3.5 Псування пам’яті

Вразливість псування пам’яті виникає у комп’ютерній програмі коли вміст області пам’яті модифікується у непередбачений розробниками програми чи мови програмування спосіб.

Деякі мови програмування, зокрема C та C++, надають дуже потужні засоби прямого доступу для читання та запису пам’яті з арифметикою вказівників. Наприклад, у згаданих у попередньому реченні мовах доступ до елемента масиву можна отримати двома тотожними способами (лістинг 2.1).

Лістинг 2.1 - Способи звернення до елементів масиву у C

```c
int main () {
     int array[] = { 0, 1, 2, 3 };
     int i = 2;

     printf("%d\n", array[i]); // спосіб №1
     printf("%d\n", *(array + i)); // спосіб №2

     return 0;
}
```

Але це створює і додаткові ризики. Що якщо значення змінної і береться з даних, що надіслав користувач? Або в ході запису можна попасти за межі буфера

і переписати дані про виділений у кучі буфер і при наступному виділенні пам’яті виникне критична помилка, яка викличе відмову обслуговування. І подібні проблема досить складно відловити.

2.3.6 Отримання інформації

Отримання інформації передбачає загрозу розкриття конфіденційної інформації неуповноваженому політикою безпеки користувачеві.

Оскільки забезпечення конфіденційності - це основне завдання криптографічних методів захисту і протоколу TLS, разом із забезпеченням цілісності, дана вразливість нівелює сам процес їхнього використання для формування захищеного каналу передачі.

Подібні вразливості дають зловмиснику можливість отримувати зачіпки, за які можна провести криптоаналіз сеансу передачі за прийнятний час. Такі вразливості спричинені помилками у реалізації, що дозволяють побачити певну закономірність між шифротекстом та оригінальним повідомленням, або передбачити процес генерації псевдовипадкових чисел.

2.4 Common Vulnerabilities and Exposures

CVE — це база даних загальновідомих вразливостей та впливів на застосунки, операційні системи та апаратне забезпечення, які становлять загрозу інформаційній безпеці.

Концепція CVE була представлена працівниками Mitre Corporation Девідом Манном та Стівеном Крісті[51] на 2-ому семінарі з БД інформаційних вразливостей, що проходив 21-22 січня 1999 року в Університеті Пердью, що в штаті Індіана, США[52].

Перша версія списку CVE була запущена у вересні 1999 року і містила 321 запис, що був зроблений першими 19 членами редакторської ради (англ. CVE Editorial Board).

Кожна вразливість у CVE ідентифікується спеціальним ідентифікатором, у форматі CVE-XXXX-YYYY, де XXXX — це рік, коли вразливість була додана, а YYYY — це номер вразливості.

Частина записів завчасно резервується компаніями, що долучилися до проєкту. Такі записи містять відповідний опис зі словом RESERVED.

2.5 Виявлені у відкритих реалізаціях вразливості

На момент написання цієї роботи, у БД CVE було включено 127491 запис, які містять вразливості багатьох програмних та апаратних продуктів різних виробників. З них 286 стосуються трьох відкритих реалізацій протоколу SSL/TLS, а саме OpenSSL, GnuTLS та NSS.

Як видно на рис. 2.10, найбільше вразливостей виявлено у OpenSSL, на другому місці, з відривом у одну вразливість NSS. У GnuTLS виявлено 46 вразливостей.

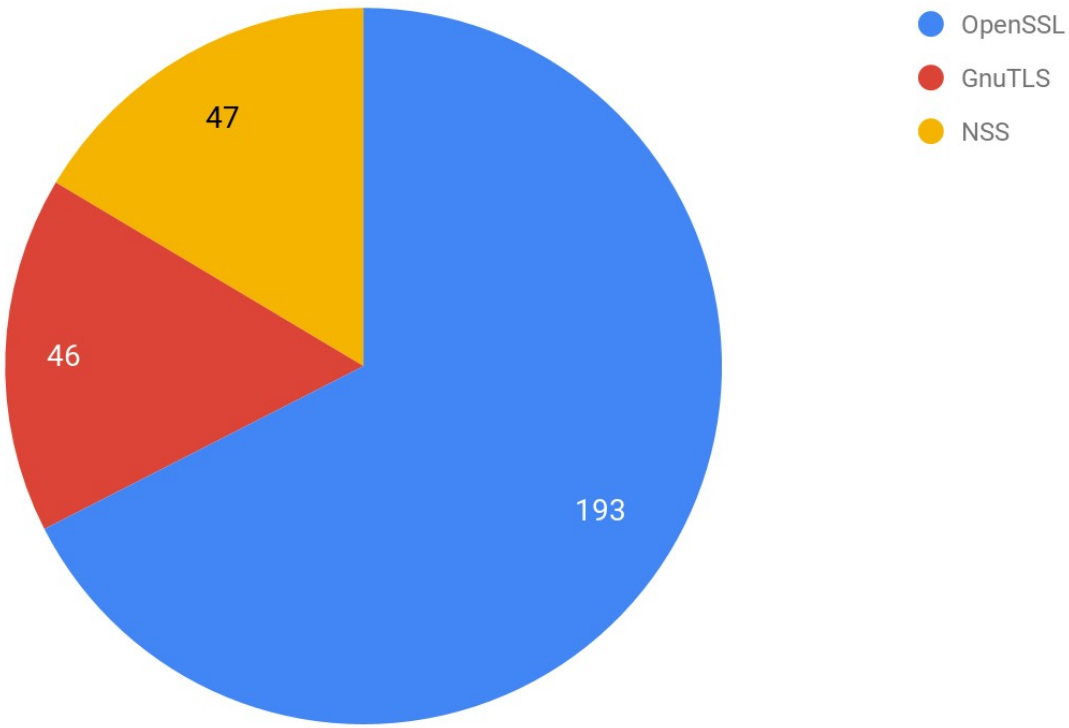


Рисунок 2.10 — Кількість виявлених вразливостей у конкретній реалізації

Складно визначити фактори, через у OpenSSL було виявлено в чотири рази більше вразливостей, ніж в інших реалізаціях.

Ймовірно, що основними причинами цього є два фактори:

1. Більш активна розробка та використання цієї бібліотеки у сторонньому програмному забезпеченні, через що код OpenSSL частіше є об’єктом вивчення та пошуку вразливостей.
2. OpenSSL — це самодостатня бібліотека, яка реалізує весь стек криптографічних алгоритмів, що необхідні для реалізації протоколу TLS, а не лише сам протокол, як це робить GnuTLS, що використовує Nettle.

Якщо дивитися на типи виявлених вразливостей (див. рис. 2.12), то в основному у всіх реалізаціях переважають ті, що призводять до відмови в обслуговуванні, крім NSS, де ця тенденція не так яскраво виражена.

На другому місці проблеми, що пов’язані з переповненням буферу.

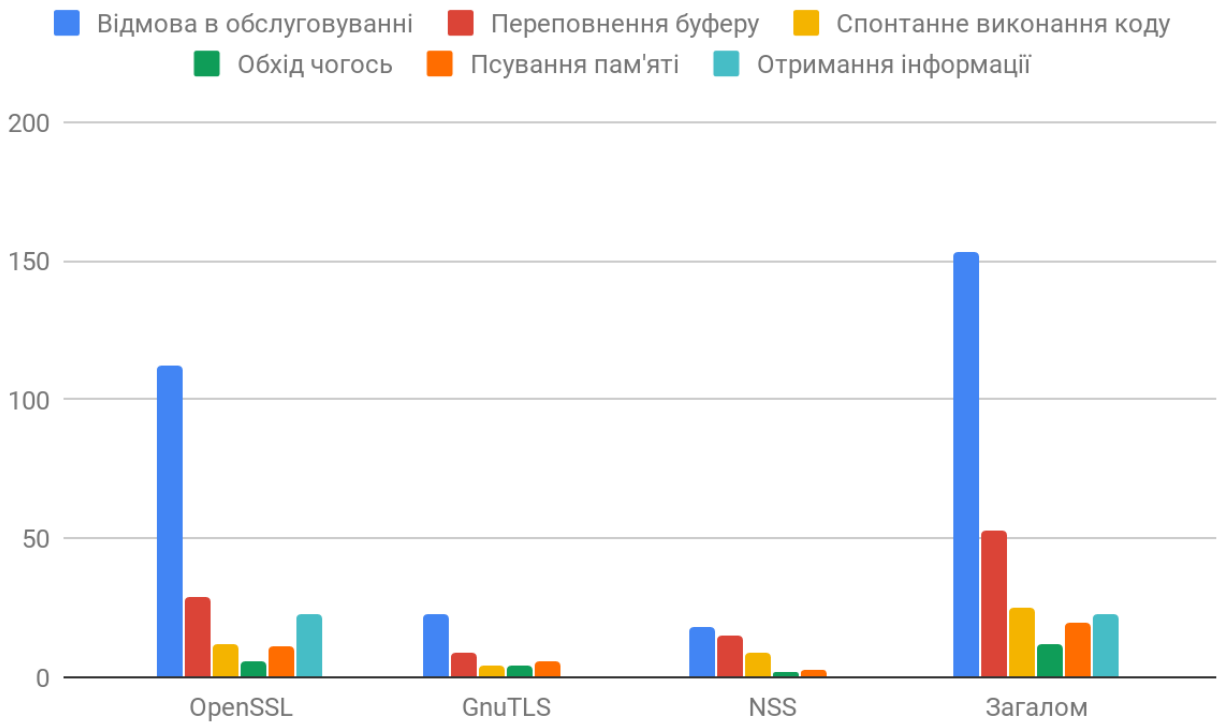


Рисунок 2.11 — Вразливості відкритих реалізацій TLS за типом

У OpenSSL також часто виявляють вразливості, що приводять до отримання зловмисником інформації, у інших реалізаціях подібні вразливості не були виявленими.

Висновки до розділу 2

Найчастіше використовуються три відкриті реалізації протоколу SSL/TLS — OpenSSL, GnuTLS та NSS.

Вразливості можна умовно поділити на ті, що викликають відмову в обслуговуванні, спонтанне виконання коду, переповнення буферу, обхід чогось та псування пам’яті.

Якщо аналізувати відкриті раніше вразливості трьох реалізацій протоколу TLS, то чітко спостерігається, що найбільшою проблемою є відмова в обслуговуванні та переповнення буферу.

Є необхідність у створенні методології розробки, що буде здатна мінімізувати ризики виникнення цих та інших вразливостей.

3 МЕТОДИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ РЕАЛІЗАЦІЇ ПРОТОКОЛУ TLS

3.1 Статичний аналіз коду

Аналіз програмного забезпечення без його реального виконання називають статичним аналізом коду.

Найпопулярнішими статичними аналізаторами для C/C++ є Cppcheck, Clang Static Analyzer та PVS-Studio. Останній є найпотужнішим, але пропрієтарним, на відміну від перших двох, тому у даній роботі не буде розглянутий.

3.1.1 Компілятор як статичний аналізатор

Для проведення статичного аналізу, зазвичай, необхідно так само як компілятор чи транслятор коду здійснити лексичний аналіз (поділ коду на токени), а також синтаксичний аналіз, тобто побудову синтаксичного дерева.

Через це найпростіший статичний аналіз коду, зазвичай, виконує ще компілятор. Так GCC та Clang — два найпопулярніші компілятори для C/C++, вміють виявляти найпростіші помилки у програмному коді і видавати попередження (англ. warning). Для прикладу розглянемо код простої програми в лістингу 3.1.

Лістинг 3.1 - Приклад коду простої програми

```c
#include <stdio.h>

int main() {
     char buffer[10];
     sprintf(buffer, "Hello World!\n");
     return 0;
}
```

Дана програма містить переповнення буферу, а саме спробу записати стрічку ‘Hello World!\n’, довжиною 14 байт (слово ‘Hello’ — 5 байт, пробіл — 1 байт,

слово ‘World’ — 5 байт, знак оклику — 1 байт, знак нового рядка ‘\n’ — 1 байт, нульовий символ завершення рядка — 1 байт) у буфер у стеку розміром 10 байт.

При спробі скомпілювати цей код за допомогою GCC (див. рисунок 3.1), він видасть попередження, а з прапорцем -Werror — помилку, з описом проблеми, а саме намагання записати текст розміром 14 байт у масив розміром 10 байт.

```
[oleh@jupiter ~]$ gcc -g -Wall -Werror -o main main.c
main.c: In function ‘main’:
main.c:10:2: error: ‘__builtin_memcpy’ writing 14 bytes into a region of size 10 overflows the
destination [-Werror=stringop-overflow=]
  sprintf(buffer, "Hello World!\n");
  ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
cc1: all warnings being treated as errors
[oleh@jupiter ~]$
```

Рисунок 3.1 — Компіляція програми з явною помилкою

Проте функцією компілятора є трансляція високорівневого сирцевого коду у нативний байткод і тому його цікавлять ті допущені програмістом помилки, які не дають скомпілювати код, а не ті, що можуть призвести до неочікуваної поведінки програми під час виконання і становити загрозу для безпеки. І дане попередження виникає не при виконанні функції sprintf, а у __builtin_memcpy, оскільки GCC використовує свій вбудований sprintf замість того щоб викликати її зі стандартної бібліотеки libc (яка і використовує __builtin_memcpy) в цілях оптимізації. Якщо вимкнути її використання за допомогою прапорця -fno-builtin-sprintf, то програма успішно скомпілюється і цілком імовірно що буде працювати, оскільки компілятор виділяє масив дещо більшого розміру, знов же в цілях оптимізації.

Отже, таку поведінку компілятора можна пояснити наступними фактами:

— метою компілятора є власне компіляція коду, а не пошук потенційних вразливостей;

— компілятор не може точно знати призначення функцій із зовнішніх бібліотек, наприклад вони можуть приймати адресу масиву і не виконувати запис, або виконувати його лише у частину масиву без переповнення;

— від компілятора очікують виконання цієї задачі за прийнятний час, що обмежує можливість проведення додаткових статичних перевірок коду;

— додавання подібного широкого функціоналу у програму порушує філософію UNIX (програма робить щось одне, але робить це добре).

Саме тому компілятор не може допомогти виявити всі потенційні вразливості у коді і для цього потрібно використовувати стороннє програмне забезпечення.

3.1.2 Cppcheck

Одним з таких сторонніх ПЗ для статичного аналізу коду на мові програмування C, на якій написані найпопулярніші реалізації протоколу SSL/TLS є cppcheck, що розробляється з 2007 року під керівництвом Даніеля Марьямакі для багатьох платформ і поширюється під вільною ліцензією GPL.

Cppcheck дозволяє виявити значну кількість помилок на етапі розробки, які можуть перетворитися у вразливість реалізації протоколу TLS. Основними з них є[53]:

— Мертві вказівники (англ. dead pointers) — вказівники на видалені об’єкти, читання яких можна розглядати як збирання сміття, а запис може призвести до аварійного завершення програми і до відмови в обслуговуванні.

— Ділення на нуль (англ. division by zero) — ситуація коли CPU дають команду виконати цілочисельне ділення на нуль, в результаті чого процесор генерує переривання, яке обробляється операційною системою (у UNIX-подібних системах генерується сигнал SIGFPE), що може спричинити відмову в обслуговуванні.

— Переповнення цілочисельної змінної (англ. integer overflows), що виникає коли результат виконання цілочисельної арифметичної операції не поміщається у змінну. У певних випадках це може призвести до несподіваної поведінки програми. У випадку з реалізаціями SSL/TLS ці несподівані результати можуть призвести до неправильного шифрування чи розшифрування інформації, що в залежності від обставин може становити загрозу цілісності.

— Використання неправильних операндів для операції зсуву (англ. invalid bit shift operands), що теж можуть призвести до неочікуваної поведінки програми. Наприклад, це зсув числа зі знаком (англ. signed), при якому програміст має чітко розуміти який результат він хоче отримати — зсувати його як число без знаку (англ. unsigned), або ж зсувати лише його абсолютне значення. В любому випадку це треба однозначно вказати у коді. Потенційні наслідки для безпеки такі ж як у попередньому пункті.

— Неправильні переведення типів (англ. invalid conversions), що можуть призвести до помилок читання/запису пам’яті або непередбачуваної поведінки програми.

— Неправильне використання стандартної бібліотеки шаблонів (англ. invalid usage of STL) теж може становити загрозу інформації різного роду та степені, що обробляється застосунком. Але реалізації TLS, що ми розглядаємо в основному написані на C, ця проблема, що виявляється cppcheck, не є актуальною.

— Проблеми керування пам’яттю, (англ. memory management) — це проблеми пов’язані з виділенням чи звільненням оперативної пам’яті, що потенційно можуть призвести до витоку пам’яті і рано чи пізно призвести до відмови в обслуговуванні.

— Розіменування пустих вказівників (англ. null pointer dereferences), що фактично є спробою читання пам’яті за неправильною адресою, що може призвести до аварійного завершення програми.

— Перевірка меж (англ. out of bounds checking), це теж помилка пов’язана із роботою з оперативною пам’яттю, а саме проблем, що можуть призвести до переповнення буферу, тобто читання запису сусідніх ділянок пам’яті. Більш детально ця проблема описана у пункті 2.1.2.

— Неініціалізовані змінні (англ. uninitialized variables), що можуть містити будь які дані у мовах C/C++, а отже їх читання до ініціалізації може призвести до неоцікуваної поведінки.

— Запис у змінні лише для читання (англ. writing const data) — це спроба запису у змінну чи масив, що відмічені як константа. Більшість таких помилок не

проходять стадію компіляції, але cppcheck надає ще потужніші засоби для їх виявлення.

Cppcheck це консольна утиліта, якій можна передати на вхід крім опцій шлях до конкретних файлів чи директорій, що містять файли, які потрібно перевірити. Результати cppcheck може записувати у текстовому форматі, або в XML. Останній підходить для іншої утиліти cppcheck-htmlreport, яка є частиною проєкту і може формувати інтерактивний HTML-звіт, у якому крім перегляду виявлених проблем можна одразу переглядати файли та ділянки коду у яких вони були виявлені. Їхнє використання зображено на рисунку 3.2.

За результатами виконання статичного аналізу коду бібліотеки OpenSSL, що імплементує протокол TLS було виявлено 129 проблем. Багато з них це слабкості, які належать до певної категорії CWE (див. рисунок 3.3), що будуть описані надалі.

```
[oleh@jupiter openssl]$ cppcheck --force --xml . 2> ../err.xml
[oleh@jupiter openssl]$ cppcheck-htmlreport --file ../err.xml --report-dir ../cppcheck
```

Рисунок 3.2 — Запуск cppcheck та формування інтерактивного HTML-звіту

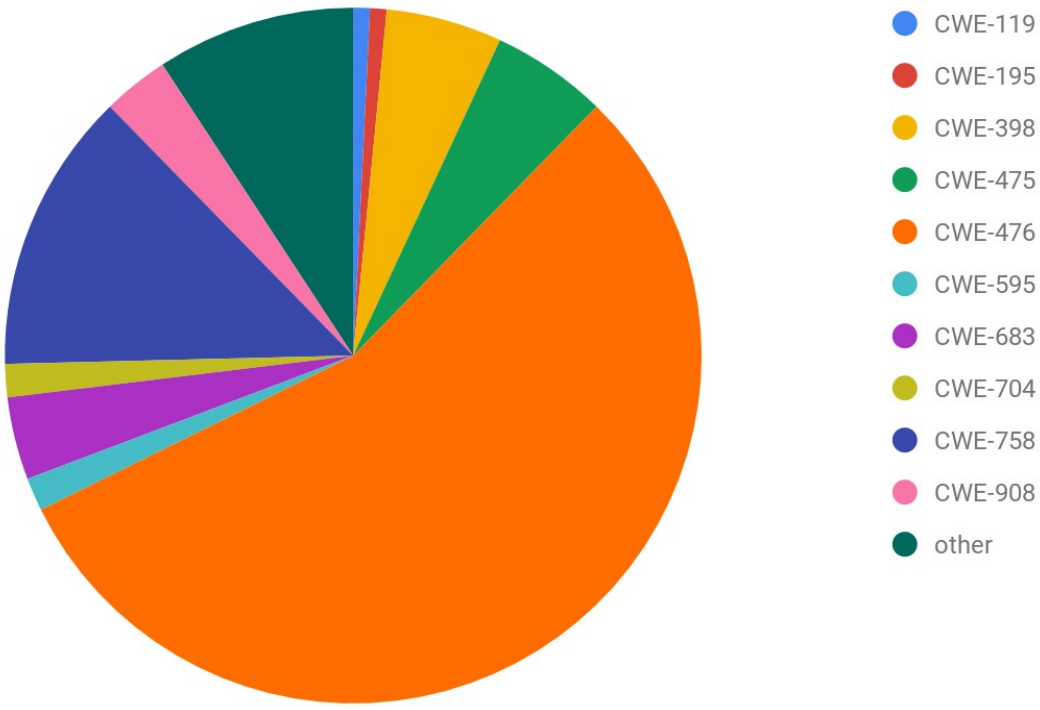


Рисунок 3.3 — Виявлені в OpenSSL за допомогою cppcheck проблеми за CWE

Найбільше виявлено вразливостей, що належать до CWE-476, тобто розіменування нульового вказівника. Однак більшість з них це попередження, які знайдені у стрічках на зразок if (!ossl_assert(ctx != NULL)). Тобто у місцях де відбувається перевірка вказівника на те чи він не NULL, тобто це не становить загрозу. Тому для більшої достовірності пропоную розглядати лише помилки, які виявив Cppcheck.

На рисунку 3.4 видно, що CWE-476 лише на третьому місці. І ці три виявлені помилки пов’язані з відсутністю перевірки на NULL значення, поверненого попередньо викликаною функцією, їх необхідно додати для виправлення, але судячи з контексту вони не становлять загрози безпеці.

Серед помилок на першому місці категорія CWE-758, тобто несподівана поведінка програми, які спричинені в основному приведенням числа до типу int перед побітовим зсувом, що може призвести до несподіваної поведінки, проте це не є загрозою, оскільки результат виконання зсуву записується і надалі використовується як ціле число без знаку.

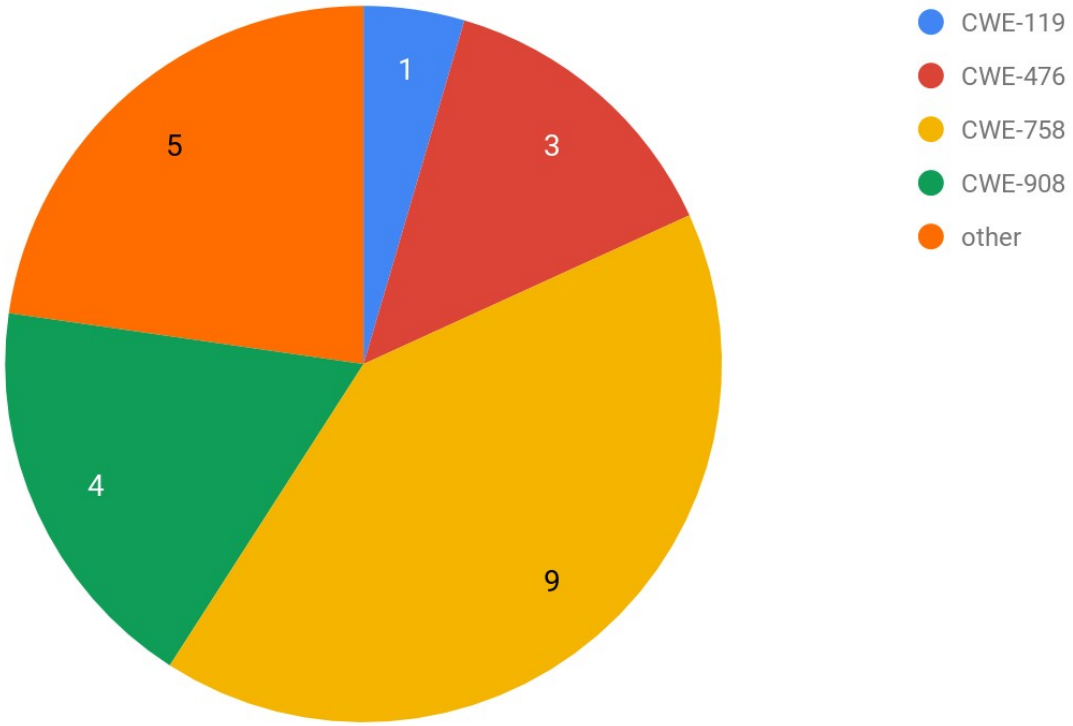


Рисунок 3.4 — Помилки в OpenSSL, що виявив Cppcheck

CWE-908 це використання неініціалізованої змінної, що трапляється чотири рази:

1. Ініціалізація змінної файлі crypto/ec/ec_asn1.c відбувається всередині #ifdef блоку, тобто в залежності від змінних, що були передані компілятора змінна може бути або ініціалізована, або ні. Для виправлення варто додати ще один #ifdef щоб не використовувати її в другому випадку, або ж ініціалізувати її завжди і передбачити правильне опрацювання значення за замовчуванням.

2. Неініціалізована змінна у crypto/modes/cts128.c передається у функцію, бо має проводити у неї запис, у такому випадку варто всерівно проводити її ініціалізацію, про всяк випадок.

3. Ідентична помилка у тому ж файлі трохи нижче.

4. Використання неіціалізованої змінною у тесті.

Інші помилки пов’язані з #ifdef блоками та поєднанням макросів, які можливо і не призначені поєднуватися.

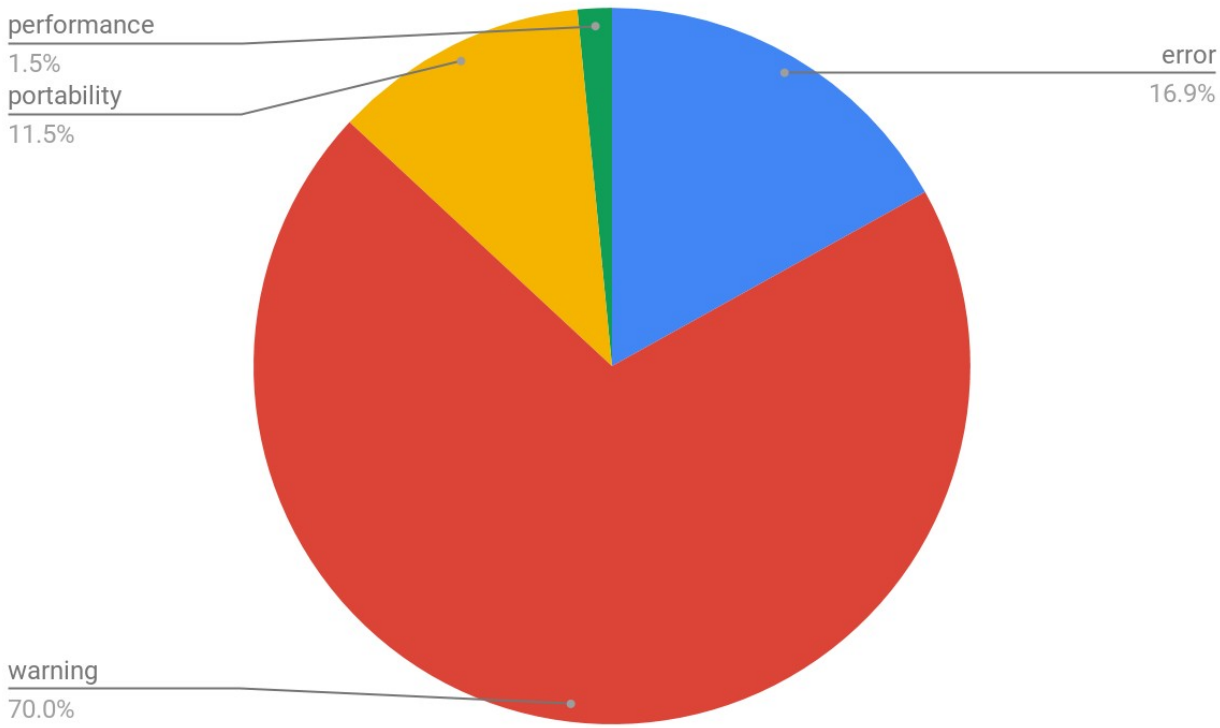


Рисунок 3.5 — Відношення помилок за серйозністю

Загалом, як видно на рисунку 3.5, з виявлених 130 проблем, лише біля 17% є помилками, тоді як всі інші є попередженнями, зокрема більшість з них через процес перевірки змінної на рівність NULL. Тому можна вважати, що OpenSSL успішно пройшов перевірку.

3.1.3 Clang Static Analyzer

Іншим вільним статичним аналізатором є Clang Static Analyzer, що розробляється в рамках проєкту вільного компілятора Clang. Іншим статичним аналізатором, що постачається разом з Clang є clang-tidy, який окрім виявлення проблем може перевіряти відповідність коду певним стандартам кодування мовами C/C++, а також дозволяє розробникам додавати свої власні перевірки.

Даний аналізатор працює протягом збирання проекту, замінюючи справжній компілятор деяким фейковим, що і проводить статичний аналіз коду. Для цього він передає шлях до нього через змінну середовища CC та CXX, які часто використовуються у UNIX-подібних операційних системах. Тому для використання цього аналізатора необхідно використовувати систему збірки проекту, що підтримує ці змінні середовища.

OpenSSL використовує make, тому запуск Clang Static Analyzer виглядає так, як зображено на рисунку 3.6.

Результати перевірки можна переглянути за допомогою інтерактивної вебсторінки, щоб запускається утилітою scan-view (в кінці сканування scan-build підкаже точну команду, їй передається тимчасова директорія з результатами). У OpenSSL було виявлено 255 проблем.

Як видно на рисунку 3.7, 70% виявлених проблем належать до групи dead store, тобто запис у змінну, значення якої потім не використовується. До них належать два типи проблем — dead assignment та dead increment. Назва пов’язана зі значенням dead code, тобто старий код, що більше не використовується. Проте частина виявлених проблем здається хибними і значення потім дійсно використовується після присвоєння.

```
[oleh@jupiter openssl]$ ./config
Operating system: x86_64-whatever-linux2
Configuring OpenSSL version 3.0.0-dev for target linux-x86_64
Using os-specific seed configuration
Creating configdata.pm
Running configdata.pm
Creating Makefile

**********************************************************************
***                                                                ***
***   OpenSSL has been successfully configured                     ***
***                                                                ***
***   If you encounter a problem while building, please open an    ***
***   issue on GitHub <https://github.com/openssl/openssl/issues>  ***
***   and include the output from the following command:            ***
***                                                                ***
***       perl configdata.pm --dump                                ***
***                                                                ***
***   (If you are new to OpenSSL, you might want to consult the    ***
***   'Troubleshooting' section in the INSTALL file first)         ***
***                                                                ***
**********************************************************************
[oleh@jupiter openssl]$ scan-build-8 make
```

Рисунок 3.6 — Запуск Clang Static Analyzer для OpenSSL

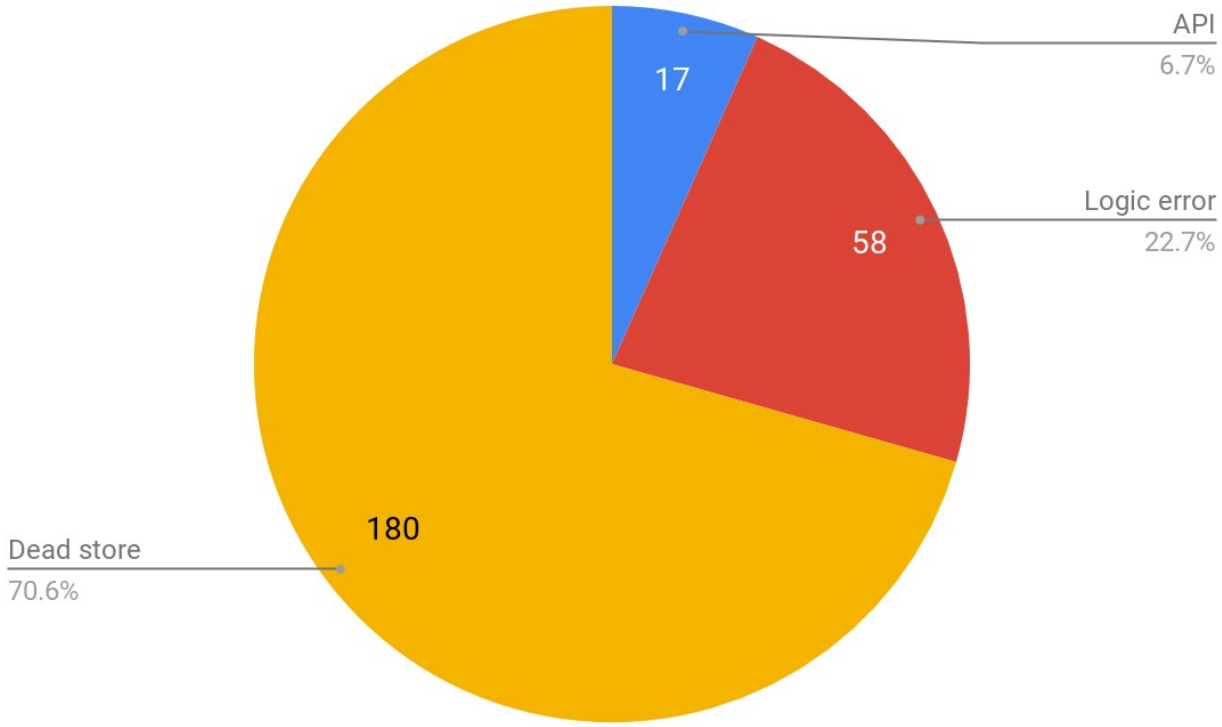


Рисунок 3.7 — Кількість виявлених проблем за групою

Інша група проблем — це логічні помилки, їх знайдено 58, і це в основному наступні проблеми:

— ймовірне розіменування нульового вказівника;

— цілочисельне ділення на нуль;

— виконання побітових операцій з операндами різного типу, що може призвести до непередбачуваної поведінки програми;

— використання неініціалізованих змінних, в основному в тестах або ж змінні передаються для запису, варто проаналізувати детальніше та все ж ініціалізувати ці змінні, задля безпеки.

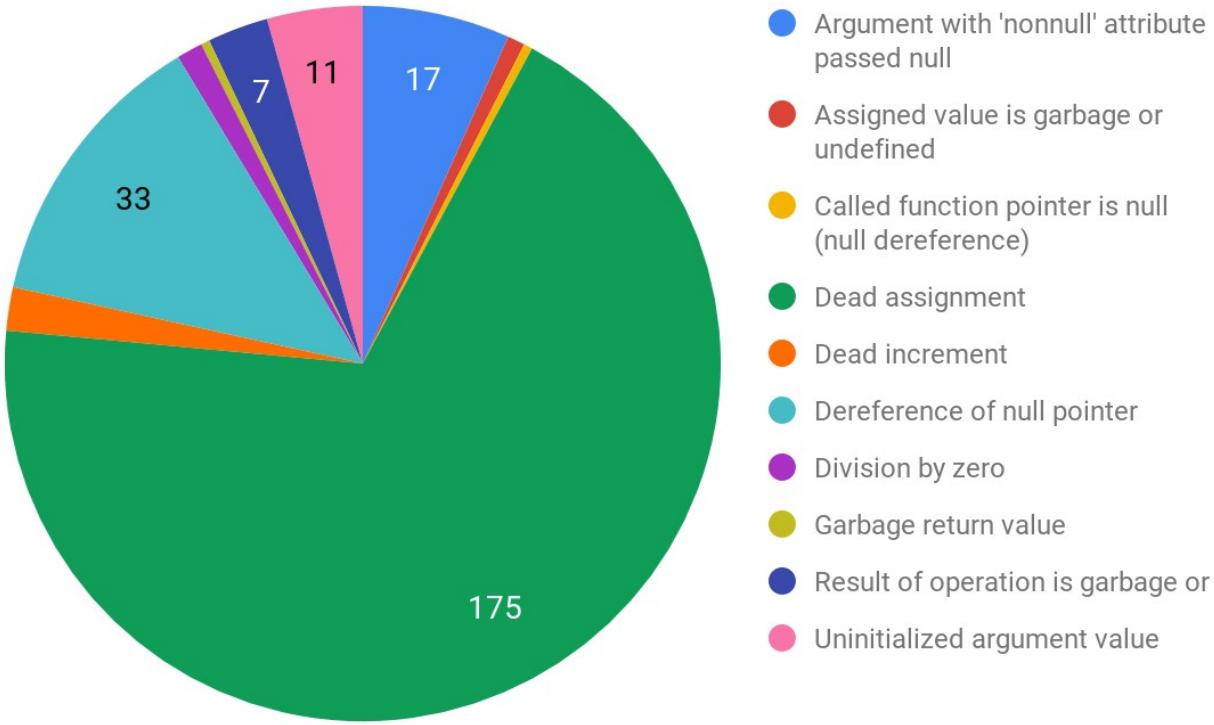


Рисунок 3.8 — Кількість виявлених проблем за типом

Також, було виявлено кілька спрацьовувань, які мають ознаки хибності. Звідси можна зробити висновок, що Clang Static Analyzer проводить дуже ретельний, часом параноїдальний, аналіз. І через велику кількість виявлених проблем у великих проектах його використання може бути ускладненим. Найкраще проводити статичний аналіз коду з ним з першого дня розробки реалізації протоколу SSL/TLS і намагатися писати код таким чином, щоб Clang Static Analyzer

був точно впевненим у відсутності потенційної вразливості. Приємним бонусом є те, зрозумілий для статичного аналізатора код має бути вкрай читабельним для інших програмістів.

3.2 Модульне тестування коду

Модульні тести (англ. unit tests) — це, зазвичай, автоматизовані тести, що розробляються програмістами до, після чи під час написання коду, які призначені для тестування найменшої одиниці програми, що може бути протестована. Це може бути окрема функція, клас чи його метод, або цілий модуль.

Загалом, цінність модульного тестування складно переоцінити, оскільки воно може використовуватися для вирішення наступних прикладних задач:

— тести можуть виконувати роль специфікації для програмного коду, яка дозволяє краще оцінити його призначення і сформувати інтерфейс, що найкраще підходить для вирішення поставлених перед модулем, що тестується, задач (практика, коли код розробляється після написання тестів називають Test-Driven Development) і загалом робить код більш передбачуваним;

— модульні тести дозволяють легко проводити регресивне тестування при додаванні нового коду чи внесенні змін у вже існуючий;

— після виявлення помилок за допомогою статичного чи динамічного аналізу, можна писати модульні, або інші види тестів (інтеграційні, системні тощо), що будуть відтворювати помилку і таким чином впевнитися у її виправленні та подальшій відсутності регресії (див. попередній пункт списку).

Але якість модульного тестування, і автоматизованого загалом, залежить від якості виконуваних тестів. Чим більше різних ситуацій вони покривають, тим повнішими вони є і тим нижчий ризик виникнення помилок, а, відповідно, і вразливостей у програмному коді. Щоб покращити якість тестів використовують ті ж засоби, що й для покращення якості коду — статичне та динамічне тестування. Окремо варто відзначити аналіз покриття коду та мутаційне тестування, що детальніше описані у пунктах 3.2.1 та 3.2.2 відповідно.

Для написання модульних тестів часто використовують окремі фреймворки — бібліотеки, що полегшують процес написання модульних тестів. Першим таким фреймворком був SUnit для мови програмування SmallTalk. З часом у нього з’явилося багато клонів для інших мов програмування, наприклад CppUnit для C++, RUnit для R, JUnit для Java.

Проте їх використання не є обов’язковим і OpenSSL реалізує власні допоміжні функції (англ. helpers) для полегшення розробки та запуску модульних тестів, що містяться у папці test/testutil/.

Самі модульні тести бібліотеки OpenSSL містяться у папці test/ і для їх запуску використовують ціль test з файлу Makefile, що генерується скріптом Configure. Наразі, OpenSSL містить 1733 тестів в 186 файлах для реалізацій окремих алгоритмів криптографічного захисту інформації та частин протоколу SSL/TLS.

3.2.1 Тестування покриття коду

Покриття коду — це міра, яка визначає якість тестів як відсоткове відношення покритих тестами функцій, стрічок чи розгалужень до їх загального обсягу.

Все досить просто — чим більший відсоток покритості, тим краще тестується код, і відповідно більш передбачуваним є код, тим менше у ньому вад та вразливостей, що вкрай актуально для реалізацій протоколу TLS.

Для визначення покритості коду існує досить багато засобів, але є вбудований у GCC gcov. Для його використання достатньо скомпілювати програму з прапорцем --coverage. В результаті компілятор згенерує ще файли з розширенням .gcno та вбудує певний код у бінарні файли, що будуть відловлювати виклик елементів коду і записуватимуть інформацію про їх виклики у файли з розширенням .gcda, на основі яких застосунок gcov зможе сформувати звіт про покриття коду.

Зокрема, для проведення даного аналізу та формування HTML-звіту та його перегляду для бібліотеки OpenSSL треба виконати команди на рис. 3.9.

```
[oleh@jupiter openssl]$ make CFLAGS="--coverage" -j4
[oleh@jupiter openssl]$ make test
[oleh@jupiter openssl]$ lcov -c --directory . --output-file main_coverage.info
[oleh@jupiter openssl]$ genhtml main_coverage.info --output-directory out
[oleh@jupiter openssl]$ google-chrome out/index.html
[oleh@jupiter openssl]$
```

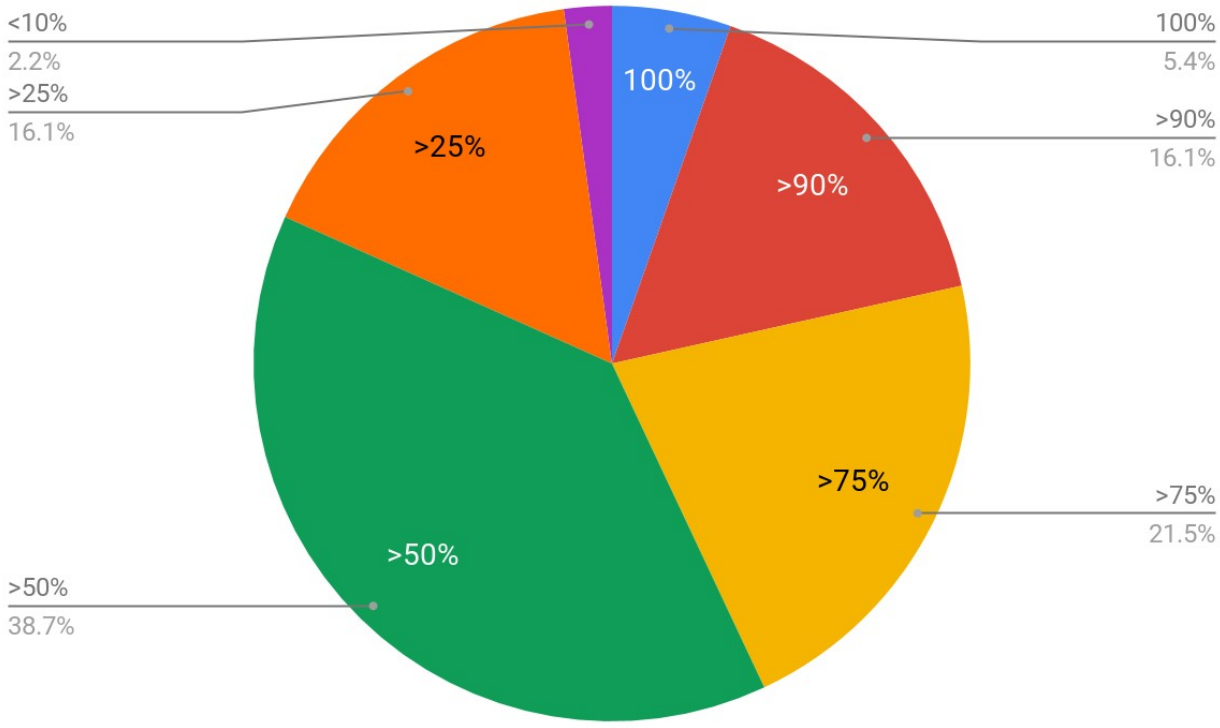
Рисунок 3.9 — Формування звіту покритості тестами коду OpenSSL

В результаті аналізу покриття тестами коду відкритої бібліотеки OpenSSL було встановлено наступні факти (див. рис. 3.10):

1. Більше 5% файлів покриті тестами повністю, тобто на 100%.

2. Більше 75% файлів покриті тестами хоча б на половину.

3. Лише 2.2% файлів покриті тестами на менш ніж 10%, це в основному нові файли, що були додані до двох місяців тому і які, ймовірно, ще не потрапили у жоден реліз.



4. Рисунок 3.10 — Покриття тестами файлів коду OpenSSL

3.2.2 Мутаційне тестування

Мутаційне тестування — це вид автоматизованого тестування модульних тестів через невеликі зміни коду, що імітують типові помилки при його написанні, наприклад неправильне використання операторів.

Для цього аналізатор визначає набір змін, які можна застосувати до коду і по черзі їх застосовує. Такі зміни називають мутаціями, звідси і назва терміну.

Над таким мутованим кодом виконуються модульні тести і якщо воно проходять успішно, то мутант вважається вижившим, а тести недостатніми.

Даний вид тестування є надзвичайно ефективним, оскільки дозволяє на порядок покращити якість юніттестів. Але їх виконання це дуже довготривалий процес, що є критичним для великих проектів. Тому в рамках даного дослідження їх не було застосовано до проекту OpenSSL.

Однак даний вид тестування включається в методологію, оскільки його можна сміливо застосовувати у нових проектах та при створенні нових тестів для вже існуючих проектів і для коду на мовах C/C++ можна використовувати застосунок Mull.

3.3 Динамічний аналіз коду

Динамічний аналіз коду передбачає його безпосереднє виконання на реальному або ж віртуальному процесорі та перевірку певних характеристик.

В більшості випадків для виконання динамічного аналізу необхідно надати аналізатору певну додаткову інформацію про сирцевий код програма, яка буде використовуватися для аналізу та формування звіту. Зазвичай, для цього в програму під час компіляції вбудовують деякі додаткові дані для відладки (англ. debug symbols). Компілятори GCC та Clang для цього використовують прапорець -g. Однак, деякі динамічні аналізатори можуть потребувати більш специфічної інформації і вимагати перекомпіляцію з додатковими опціями.

Варто зазначити, що дані для відладки фактично включають сирцевий код програми, що полегшує реверс-інжиніринг програми і суттєво збільшують розмір

програми, тобто об’єм пам’яті необхідний для її збереження. Для виправлення цього перед релізом на UNIX-подібних системах використовують утиліту strip, яка є частиною проєкту binutils, яка просто вилучає дані символи з бінарного файлу.

3.3.1 Виявлення помилок при роботі з пам’яттю

Найбільш практичне застосування динамічного аналізу коду на мові C/C++ — це пошук помилок пов’язаних з керуванням пам’яттю та загалом при роботі з нею, тобто читання/запис.

Динамічні аналізатори для пошуку подібних проблем виконують код у захищеному середовищі, здійснюють контроль над процесом виділення пам’яті через стандартні засоби мови C/C++, а саме функції malloc/free та оператори new/delete, а потім здійснюють спостереження за процесом читання та запису.

Таким чином подібні аналізатори здатні виявляти такі аномалії як читання/запис за межами буфера (для цього їх виділяють з великим запасом, тобто відстанями один між одним) та витіки пам’яті, коли пам’ять виділяється, але не вивільняється після використання.

Отже, використання подібних інструментів дозволяє різко знизити ризики виникнення загрози переповнення буфера та відмови в обслуговуванні під час імплементації протоколу SSL/TLS.

Одним з найкращих подібних аналізаторів є Valgrind, що поширюється під вільною ліцензією GPL. На рис. 3.11 зображено його використання для аналізу одного з модульних тестів бібліотеки OpenSSL, а саме тесту криптосистеми RSA.

Видно, що у даному тестові і, відповідно, у частині бібліотеки, яку він тестує не виявлено жодних помилок при роботі з пам’яттю.

Насправді, Valgrind і подібні аналізатори призначені не лише для тестів, а для любого застосунку у якого включені символи для відлагодження програми. У цьому його застосування схоже із застосуванням програми-відлагоджувача, на зразок gdb.

```
[oleh@jupiter openssl]$ LD_LIBRARY_PATH=. valgrind --tool=memcheck ./test/rsa_test > /dev/null
==16202== Memcheck, a memory error detector
==16202== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16202== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==16202== Command: ./test/rsa_test
==16202==
==16202==
==16202== HEAP SUMMARY:
==16202==     in use at exit: 100 bytes in 2 blocks
==16202==   total heap usage: 29,034 allocs, 29,032 frees, 27,673,681 bytes allocated
==16202==
==16202== LEAK SUMMARY:
==16202==    definitely lost: 0 bytes in 0 blocks
==16202==    indirectly lost: 0 bytes in 0 blocks
==16202==      possibly lost: 0 bytes in 0 blocks
==16202==    still reachable: 100 bytes in 2 blocks
==16202==         suppressed: 0 bytes in 0 blocks
==16202== Rerun with --leak-check=full to see details of leaked memory
==16202==
==16202== For counts of detected and suppressed errors, rerun with: -v
==16202== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[oleh@jupiter openssl]$
```

Рисунок 3.11 — Приклад звіту Valgrind при перевірці одного з модульних тестів OpenSSL

Висновки до розділу 3

В результаті дослідження вразливостей відкритих реалізацій протоколу SSL/TLS, а саме OpenSSL, GnuTLS та NSS було обрано набір сучасних інструментів для аналізу програмного коду на мові C, якою написані дані реалізації та сформовано методологію, що має на меті знизити ймовірність появи вразливостей у реалізації протоколу TLS мовами програмування C/C++.

Дана методологія складається з наступних пунктів:

1. Використовувати актуальні версії компіляторів GCC чи Clang.

2. При компіляції розглядати навіть попередження як помилки та не приймати код, який містить попередження, а тим паче помилки.

3. Для пошуку потенційних вразливостей проводити статичний аналіз коду за допомогою інструментів Cppcheck, Clang Static Analyzer, Clang-Tidy.

4. Поєднувати кілька статичних аналізаторів для більш широкого вивчення коду та пошуку потенційних вразливостей.

5. Покривати код модульними тестами криптографічних алгоритмів і мережевих модулів, за бажанням використовувати якийсь фреймворк для тестування ПЗ.

6. Для забезпечення якості модульних тестів стежити за покриттям коду тестами з утилітами gcov та lcov, а також, за можливості, проводити мутаційне тестування модульних тестів за допомогою застосунку Mull.

7. Проводити періодичний динамічний аналіз коду, зокрема пошук проблем при роботі з пам’яттю з аналізатором Valgrind.

В результаті застосування даної методології до бібліотеки OpenSSL було сформовано наступні рекомендації:

— варто ініціалізувати всі змінні при їх оголошенні і коректно обробляти значення за замовчуванням;

— додати у виявлених статичним аналізатором місцях більш очевидну програмних засобів перевірку вказівників на рівність NULL;

— покрити тестами ту частину коду, чиє покриття менше 50%;

— продовжувати використовувати Valgrind для пошуку помилок та вразливостей, що пов’язані з використанням пам’яті.

4 ОРГАНІЗАЦІЯ НЕПЕРЕРВНОЇ ІНТЕГРАЦІЇ РЕАЛІЗАЦІЇ TLS

В інженерії програмного забезпечення безперервна інтеграція (англ. Continuous Integration, далі — CI) — це практика об'єднання робочих копій всіх розробників до спільної вітки кілька разів на день [54]. Грейді Бух вперше запропонував термін CI у своєму методі 1991 року [55], хоча він не виступав за це, інтегруючи кілька разів на день. Екстремальне програмування (XP) прийняло концепцію CI і виступало за інтеграцію більше одного разу на день , навіть десятки разів на день.

Найбільш ранньою відомою роботою з безперервної інтеграції було середовище Infuse, що розробили Г.Е. Кайзер, Д.Є. Перрі і В.М. Шелл. [56]

У 1994 році Grady Booch використовував фразу безперервної інтеграції в об'єктно-орієнтованому аналізі та дизайні з додатками (2-е видання) [57], щоб пояснити, як при розробці за допомогою мікропроцесів внутрішні релізи являють собою своєрідну безперервну інтеграцію системи; існують для примусового закриття мікропроцесу.

У 1997 році Кент Бек і Рон Джеффріс винайшли екстремальне програмування (XP), перебуваючи в проекті комплексної компенсації Chrysler, включаючи безперервну інтеграцію. [53] Бек опублікував інформацію про постійну інтеграцію в 1998 році, наголосивши на важливості спілкування віч-на-віч при наданні технологічної підтримки. [58] У 1999 році Бек детальніше розробив свою першу повну книгу про екстремальне програмування. CruiseControl, один з перших інструментів CI з відкритим кодом, був випущений у 2001 році.

Безперервна інтеграція складається з наступних етапів:

1. Виконання тестів локально. CI призначений для використання в поєднанні з автоматизованими модульними тестами, написаними з використанням практики Test-Driven Development. Це робиться шляхом запуску та проходження всіх модульних тестів у локальному середовищі розробника, перш ніж перейти на основну вітку. Це допомагає уникнути випадків коли код двох розробників не є сумісним.

2. Збирання проекту на виділеному сервері. Сервер збирання періодично збирає код або навіть після публікації кожної зміни і повідомляє про результати розробникам. У більшості випадків сервер збірки також виконує одиничні тести. Використання серверів збирання було запроваджено спільнотою XP, але в наш час багато організацій прийняли CI, не прийнявши XP повністю.

3. Контроль якості. Крім автоматизованих тестових одиниць, організації, що використовують CI, зазвичай використовують сервер збірки для здійснення безперервних процесів застосування контролю якості в цілому. Окрім запуску тестування інтеграції та інтеграції, такі процеси виконують додаткові статичні аналізи, вимірювання та характеристики профілю, витягують та форматують документацію з вихідного коду та полегшують ручні процеси забезпечення якості, що ідеально підходить для застосування створеної методології. Постійне застосування контролю якості має на меті покращити якість програмного забезпечення та скоротити час, необхідний для його доставки, замінивши традиційну практику застосування контролю якості після завершення всієї розробки. Це дуже схоже на початкову ідею інтеграції частіше, щоб полегшити інтеграцію, застосовувану лише до процесів забезпечення якості.

4. CI/CD. Зараз CI часто переплітається з постійною доставкою в так званому процесі CI/CD. CI гарантує, що версія програмного забезпечення, що знаходиться у головній лінії проєкту, завжди знаходиться в стані, який можна розгорнути для користувачів, а CD робить процес розгортання повністю автоматизованим.

Для прикладу розглянемо організацію безперервної інтеграції відкритої імплементації протоколу TLS OpenSSL за допомогою відкритого сервісу Gitlab CI. Для цього в корені проекту необхідно створити файл під назвою .gitlab-ci.yml.

При будь якому оновленні віддаленого Git-репозиторія GitLab шукатиме цей файл і запускатиме описані у ньому завдання за допомогою застосунку Gitlab Runner.

Оскільки .gitlab-ci.yml знаходиться у Git-репозиторії та контролюється версіями, старі версії теж можуть успішно запускатися, відгалуження проєкту можуть легко використовувати CI, а гілки мати різні завдання.

Приклад налаштування файлу для OpenSSL зображено на рисунку 4.1.

```
image: ubuntu:18.04
before_script:
  - apt-get update && apt-get install -y build-essential clang-tools-8 cppcheck lcov
  - ./config

cppcheck:
  script:
    - cppcheck --force --xml . 2> report.xml
    - cppcheck-htmlreport --file report.xml --report-dir cppcheck_report/
  artifacts:
    paths:
      - cppcheck_report/

clang-analyzer:
  script:
    - scan-build-8 make

test:
  script:
    - make test CFLAGS="--coverage"
    - "gcov $(find . -name '*.c')"
    - "lcov --coverage --directory . --output-file coverage.info"
  artifacts:
    paths:
      - coverage.info
  coverage: '/Test Coverage: ([0-9]{1,3}%)/'
```

Рисунок 4.1 — Приклад конфігурації Gitlab CI

Оскільки Gitlab Runner побудований на основі Docker, необхідно вказати образ (англ. image), який буде використовуватись. У нашому випадку було обрано образ Ubuntu 18.04.

Також було налаштовано before_script — набір команд, що буде викликатися до основного набору команд. Оскільки він налаштований у глобальному контексті поза задачею, то цей набір команд буде викликано для кожного завдання. Це чудовий момент для налаштування середовища у якому будуть запускатися тести, наприклад, встановити засоби розробки — компілятор, компоновщик, систему для збірки make та необхідні для методології аналізатори, а саме scan-build, cppcheck та lcov.

Кожне завдання містить набір команд, що стосується запуску перевірок та формування звіту, які зберігаються у якості артефактів у системі Gitlab.

Висновки до розділу 4

Неперервна інтеграція — це потужний інструмент при розробці програмного забезпечення загалом та імплементацій протоколу TLS зокрема. Дана техніка дозволяє автоматизовано запускати тести та інші метрики імплементації TLS на різних подіях, таких як публікація комітів чи нового релізу, чи створенні запиту на злиття коду.

Прикладом ПЗ для проведення неперервної інтеграції є Gitlab CI. У даному розділі було налаштовано файл конфігурації .gitlab-ci.yml для автоматизованого запуску тестів, що передбачені розробленою методологією.

5 ОБҐРУНТУВАННЯ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

Для техніко-економічного обґрунтування вразливостей реалізації криптографічних методів захисту протоколу SSL/TLS пропонується побудувати Business Model Canvas стартапу з надання послуг аналізу коду імплементацій протоколу TLS на наявність вразливостей.

Business Model Canvas — це шаблон для розробки нових або документування бізнес-моделей, що вже існують, один з методів стратегічного управління та LEAN startup[59]. Це візуальна діаграма з елементами, що описують ціннісну пропозицію фірми чи товару, інфраструктуру, клієнтів та фінанси, що допомагає фірмам вирівнювати свою діяльність, демонструючи потенційні плюси і мінуси.

Даний шаблон спочатку був запропонований Олександром Остервальдером на основі його попередньої роботи з онтології бізнес-моделі. З часу виходу творіння Остервальдера близько 2008 року з'явилися нові шаблони для конкретних ніш.

Формальні описи бізнесу стають складовими елементами його діяльності. Існує багато різних бізнес-концепцій — у книзі Остервальдера за 2010 рік [60] та дипломній роботі 2004 року [61] пропонується єдина модель, що базується на подібності широкого спектру бізнес-моделей. Завдяки цьому шаблону підприємство може легко описати свою бізнес-модель. Шаблон Остервальдера має дев'ять полів; Кожне поле детально описано нижче.

Основні види діяльності. Найважливіші заходи щодо виконання вартісної пропозиції компанії. Прикладом для виробника ручок Віс буде створення ефективного ланцюга постачання для зменшення витрат.

Основні ресурси — це ресурси, що необхідні для створення цінності для замовника. Вони вважаються активами компанії, які необхідні для підтримки бізнесу. Ці ресурси можуть бути людськими, фінансовими, фізичними та інтелектуальними.

Ключові партнери. Для оптимізації операцій та зменшення ризиків бізнес-моделі організації зазвичай розвивають відносини покупець-постачальник, щоб вони могли зосередити свою основну діяльність. Додаткові бізнес-альянси також можуть розглядатися через спільні підприємства або стратегічні альянси між конкурентами або не конкурентами.

Ціннісні пропозиції — колекція продуктів та послуг, які пропонує бізнес, щоб задовольнити потреби своїх клієнтів. За даними Osterwalder (2004), ціннісна пропозиція компанії — це те, що відрізняє її від своїх конкурентів. Пропозиція про цінність забезпечує цінність за допомогою різних елементів, таких як новизна, продуктивність, налаштування, "виконання роботи", дизайн, бренд, статус, зниження ціни, зниження ризику, доступність та зручність використання.

Цінні пропозиції можуть бути:

— кількісні - ціна та ефективність;

— якісні - загальний досвід та результат клієнта.

Сегменти клієнтів. Для побудови ефективної бізнес-моделі компанія повинна визначити, яких клієнтів вона намагається обслуговувати. Різні набори клієнтів можуть бути сегментовані, виходячи з різних їх потреб та ознак, щоб забезпечити належну реалізацію корпоративної стратегії для задоволення характеристик вибраних груп клієнтів. До різних типів сегментів клієнтів належать:

— масовий ринок: немає специфічної сегментації для компанії, яка націлена на масовий ринок, оскільки організація демонструє широкий перелік потенційних клієнтів, наприклад автомобільні компанії;

— ринок ніші: сегментація клієнтів на основі спеціалізованих потреб та характеристик своїх клієнтів, прикладом є Rolex;

— сегментований, коли компанія застосовує додаткову сегментацію в межах сегментів клієнтів, що існують, в такій ситуації бізнес може надалі розрізняти своїх клієнтів за ознакою статі, віку та/або доходу;

— диверсифікований, коли бізнес обслуговує декілька сегментів клієнтів з різними потребами та характеристиками;

— багатостороння платформа/ринок, коли для безперебійного щоденного ведення бізнесу деякі компанії обслуговуватимуть взаємозалежні сегменти клієнтів, наприклад компанія може надавати послуги власникам кредитних карт, одночасно допомагаючи торговцям, які приймають ці кредитні картки.

Канали. Компанія може надати цінні пропозиції своїм цільовим клієнтам через різні канали. Ефективні канали поширюватимуть ціннісну пропозицію компанії способами, які є швидкими, ефективними та рентабельними. Організація може охопити своїх клієнтів через власні канали (фірмові магазин), партнерські канали (дистриб'ютори) або комбінацію обох.

Взаємовідносини з клієнтами. Щоб забезпечити виживання та успіх будь-якого бізнесу, компанії повинні визначити тип відносин, які вони хочуть створити з різними сегментами своїх клієнтів. Різні форми взаємовідносин із клієнтами включають:

— особиста допомога — допомога у формі взаємодії працівник-клієнт. Така допомога здійснюється під час продажу та/або після продажу;

— виділена особиста допомога — практична особиста допомога, в якій торговий представник призначений для вирішення всіх потреб та питань спеціального набору клієнтів;

— самообслуговування — тип відносин, що складається з непрямої взаємодії між компанією та клієнтами, коли організація пропонує інструменти, необхідні клієнтам, щоб легко та ефективно обслуговувати себе;

— автоматизовані послуги — система, схожа на самообслуговування, але більш персоналізована, оскільки має можливість ідентифікувати окремих клієнтів та їх переваги, прикладом цього може бути Amazon.com, який робить пропозиції щодо книг, виходячи з особливостей попередніх закупівель книг;

— спільноти — створення спільноти дозволяє здійснювати пряму взаємодію між різними клієнтами та компанією; платформа спільноти створює сценарій, коли можна обмінюватися знаннями та вирішувати проблеми між різними клієнтами.

Структура витрат. Тут описані найважливіші грошові наслідки під час роботи за різними бізнес-моделями. DOC компанії.

Класи бізнес-структур:

— на основі витрат — ця бізнес-модель орієнтована на мінімізацію всіх витрат та відсутність надмірностей, наприклад авіакомпанії — лоукостери;

— ціннісна модель — ця бізнес-модель орієнтована на створення вартості продуктів та послуг, наприклад Louis Vuitton, Rolex.

— Характеристика структур витрат:

— фіксовані витрати — витрати не змінюються в різних програмах. наприклад зарплата, орендна плата;

— змінні витрати — витрати варіюються залежно від обсягу виробництва товарів чи послуг, наприклад музичні фестивалі;

— економія з масштабом — витрати зменшуються, коли більша кількість товарів замовляється або виробляється;

— економія сфери застосування — витрати зменшуються завдяки включенню інших підприємств, які мають пряме відношення до оригінального продукту.

Джерела доходу — спосіб отримання прибутку від кожного сегменту клієнтів. Є кілька способів отримання потоку доходу:

— продаж активів — це найпоширеніший тип, що передбачає продаж прав власності на фізичний товар, наприклад роздрібні корпорації;

— плата за користування — гроші, отримані за рахунок використання певної послуги, наприклад Нова пошта;

— плата за підписку — дохід, отриманий від продажу доступу до безперервної послуги, наприклад Netflix чи Youtube Premium;

— кредитування/лізинг/оренда — надання ексклюзивного права на актив протягом певного періоду часу, наприклад оренда автомобіля;

— ліцензування — дохід, отриманий від стягнення плати за використання захищеної інтелектуальної власності;

— плата за посередництво — дохід, отриманий від проміжного обслуговування між двома сторонами, наприклад брокер, що продає будинок і отримує комісію;

— реклама - дохід, отриманий від стягнення плати за рекламу товару.

Складена бізнес-модель для стартапу з надання послуг аналізу коду імплементацій протоколу TLS на наявність вразливостей представлена у таблиці 5.1.

Висновки до розділу 5

У сучасному світі для побудови бізнес-моделей часто використовують уніфіковані шаблони, такі як Business Model Canvas. Побудована бізнес-модель може бути легко застосована для організації підприємства з надання послуг з аналізу імплементацій протоколу TLS на вразливості.

Таблиця 5.1 — Business Model Canvas бізнесу з надання послуг виявлення вразливостей реалізації TLS

<table>
<tr><td>Проблема</td><td>Рішення</td><td>Унікальна ціннісна пропозиція</td><td>Перевага перед конкурентами</td><td>Сегмент користувачів</td></tr>
<tr><td>Під час реалізації протоколу TLS, що найчастіше робиться мовою програмування C, через помилки програмістів виникають вразливості, які можуть привести до порушення конфіденційності даних користувачів ПЗ, що розробляється</td><td>Запропонована методологія дозволяє виявляти потенційно вразливі ділянки коду під час розробки</td><td>Шляхом проведення аналізу реалізації протоколу TLS виявити потенційні вразливості на стадії розробки, до того як вони можуть завдати шкоди
Аналіз використовуваної версії реалізації протоколу TLS на вразливості</td><td>Методологія є відкритою та використовує лише вільне програмне забезпечення, що суттєво зменшує ціну її використання, що, відповідно, робить її доступнішою для замовника</td><td rowspan="3">Розробники програмного забезпечення, що займаються реалізацією протоколу TLS
Розробники програмного забезпечення, що використовують відкриті реалізації протоколу TLS</td></tr>
<tr><td>Існуючі альтернативи</td><td>Ключові метрики</td><td>Високорівнева концепція</td><td>Канали комунікації</td></tr>
<tr><td>Використовувати існуючі реалізації TLS, що можуть не відповідати потребам розробників
Інші методології пошуку вразливостей</td><td>Кількість проаналізованих реалізацій
Кількість проаналізованих стрічок коду
Кількість виявлених вразливостей</td><td>Дана методологія потенційно може бути використана для будь якого програмного запезпечення, а не лише реалізацій протоколу TLS</td><td>Сервіси, що використовуються розробниками для кооперації — Github, Gitlab, Gitter</td></tr>
<tr><td colspan="3">Структура ціни</td><td colspan="2">Джерела доходу</td></tr>
<tr><td colspan="3">Робочий час аналітика
Обчислювальні ресурси для проведення аналізу</td><td colspan="2">Здійснення комплексного аудиту реалізацій протоколу TLS на основі даної методології</td></tr>
</table>

6 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

6.1 Охорона праці

Робоче приміщення, у якому проходило дослідження та аналіз вразливостей реалізацій протоколу SSL/TLS, має відповідати вимогам щодо охорони праці при організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ВДТ).

Дане приміщення має 3 робочих місця. Розглянемо відповідність характеристик робочого місця нормативним. Для цього зведемо основні вимоги до організації робочого місця з і відповідні фактичні значення для робочого місця, за яким виконується робота, у табл. 6.1.

Робоче приміщення та місце відповідає вимогам щодо охорони праці при організації роботи з ВДТ електронно-обчислювальних машин.

Таблиця 6.1 — Характеристики робочого місця

Параметр	Позначення	Величина
Довжина, м	A	5
Ширина, м	B	4
Висота, м	H	3
Кількість робочих місць	N	3
Площа, $м^2$	S	20
Об’єм, $м^3$	V	60

Відповідно до ДСН 3.3.6.042-99 роботи, що виконуються користувачами ЕОМ, відносяться до легких фізичних робіт – категорії Іа. У виробничих приміщеннях на робочих місцях з ВДТ мають забезпечуватись оптимальні значення параметрів мікроклімату.

Згідно ДБН В.2.5-28:2018 приміщення, що розглядається, повинне мати природне і штучне освітлення.

Денне (природне) освітлення приміщення відбувається за системою однобічного бічного освітлення. Природне світло проникає у приміщення через три світлові прорізи (віконні отвори), які мають регулювальні пристрої для відкривання. Також наявні штори (жалюзі) з можливістю захисту працюючих від прямого попадання сонячних променів і регулювання рівня освітленості в приміщенні. Вікна приміщення орієнтовані на північний схід. Оскільки будинок розташований у відносній віддаленості від прилеглих будівель, то які небудь перешкоди природному освітленню розглянутого приміщення відсутні.

Всередині приміщення стіни обклеєні світлими шпалерами, стеля побілена (переважає білий колір), у якості підлогового покриття використаний лінолеум світло-жовтого кольору.

Наявність постійного шуму в робочій зоні призводить до розладу центральної нервової системи і до таких захворювань як неврози, однак фактичний обмірюваний рівень шуму в робочій зоні склав 43 дБА, що задовольняє нормативному рівню шуму (не повинен перевищувати 50 дБА), тому додаткових заходів по поліпшенню цього фактору не потрібно.

Проаналізуємо стан електробезпеки в робочому приміщенні:

— всі прилади в кабінеті використовують напругу 220 В;

— електропроводка захована і ізольована від працівників спеціальним коробом;

— всі робочі місця з ПЕОМ використовують спільні розетки по 220 В;

— споживачі електроенергії — 3 ПЕОМ у вигляді ноутбуків;

— відносна вологість повітря – 60%, температура повітря +22 °С — +24 °С, струмопровідний пил і хімічно активні речовини в повітрі відсутні;

— підлога: ізолююча – лінолеум.

Проаналізувавши наведене вище, можемо сказати, що кабінет відноситься до приміщень без підвищеної електронебезпеки.

ПЕОМ, що використовуються в даному кабінеті підключаються до трифазної мережі і мають захисне занулення (за допомогою окремого захисного нульового провідника). Корпуси ВДТ та принтера виготовлені з пластику і не являються струмопровідними. Щодо корпусів самих ПЕОМ, вони виготовлені зі струмопровідного матеріалу, крім передньої панелі, що виготовлена з пластику.

При виконанні робіт по ремонту і обслуговуванню ПЕОМ обслуговуючий персонал зобов'язаний керуватися “Правилами техніки безпеки при експлуатації електроустановок споживачами". До роботи не допускаються особи, які не пройшли навчання з техніки безпеки.

Джерелом електромагнітного випромінювання в сучасному офісі є візуальні дисплейні термінали. Нормування електромагнітного випромінювання ВДТ а здійснюється згідно НПАОП 0.00-7.15-18 “Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями”.

6.2 Фактори, що впливають на функціональний стан користувачів комп’ютера

Трудова діяльність користувачів комп'ютерів (ВДТ) відбувається у певному виробничому середовищі, яке впливає на їх функціональний стан. Найбільш значимі — фізичні фактори виробничого середовища, до яких належать електромагнітні хвилі різних частотних діапазонів, електростатичні поля, шум, параметри мікроклімату та ціла низка світлотехнічних показників. Вплив хімічних та, особливо, біологічних факторів виробничого середовища на користувачів комп'ютерів — значно менший.

Трудовий процес суттєво впливає на психофізіологічні можливості користувачів комп'ютерів, оскільки їх діяльність характеризується значними статичними фізичними навантаженнями; недостатньою руховою активністю; напруженнями сенсорного апарату, вищих нервових центрів, які забезпечують функції уваги, мислення, регуляції рухів. Окрім того, трудовий процес користувачів комп'ютерів відзначається значними інформаційними навантаженнями.

Професійні якості та виробничий досвід, які визначають внутрішні засоби діяльності, обумовлюють надійну та безпомилкову діяльність користувачів комп'ютерів, дозволяють знаходити безпечні методи розв'язання виробничих завдань навіть у нестандартних ситуаціях.

Зовнішні засоби діяльності, які в основному визначаються ергономічними показниками щодо організації робочого місця, форми та параметрів його елементів, просторового розташування основного і допоміжного устаткування, можуть суттєво знизити фізичні та психофізіологічні навантаження, що діють на користувачів комп'ютерів.

6.2.1 Особливості роботи користувачів комп'ютерів

У професійних операторів частіше зустрічаються порушення органів зору, опорно-рухового апарату, центральної нервової, серцево-судинної, імунної та статевої систем, захворювання шкіри. Зафіксована значна кількість скарг операторського персоналу на загальне недомагання, передчасне стомлювання, головний біль, порушення функцій органів зору, які здійснювали несприятливий психофізіологічний вплив на самопочуття та працездатність операторів.

Сучасна професія користувача ВДТ належить до розумової праці, яка характеризується: високою напруженістю зорових функцій; одноманітною позою; великою кількістю стереотипних висококоординованих рухів, що виконуються лише м'язами кистей рук на фоні малої загальної рухової активності; значним нервово-емоційним компонентом, особливо в умовах дефіциту часу; роботою з великими масивами інформації, що викликає активізацію уваги та інших вищих психічних функцій. Крім того, при роботі з дисплеями на електронно-променевих трубках виникає вплив на користувача цілої низки факторів фізичної природи — електростатичні поля, радіочастотне та рентгенівське випромінювання тощо.

Діяльність професіоналів можна поділити на три групи:

1. Діяльність, яка пов'язана з виконанням нескладних багаторазово повторюваних операцій, що не вимагають великого розумового напруження. Наприклад, робота операторів комп'ютерного набору, працівників довідкових служб.

2. Діяльність, яка пов'язана із здійсненням логічних операцій, що постійно повторюються. Це робота інженера-економіста, проектувальника, оператора автоматизованого виробництва.

3. Діяльність, коли в процесі роботи необхідно приймати рішення за відсутності заздалегідь відомого алгоритму. Наприклад, робота інженера-програміста, диспетчерів руху залізничного транспорту, аеропортів тощо.

У користувачів, які інтенсивно використовують комп'ютер в умовах значних розумових напружень досить часто (40—70%) виникають психологічні та поведінкові порушення (нервозність, роздратування, тривога, нерішучість, замкнутість тощо). Серед користувачів ВДТ в США і Європі значного поширення набуло специфічне захворювання, яке отримало назву синдром комп'ютерного стресу (СКС). СКС супроводжується головним болем, запаленням очей, алергією, роздратованістю, млявістю і депресією. Інформаційне перевантаження користувачів ВДТ супроводжується низкою специфічних захворювань, які називають інформаційними. Першим симптомом їх є головний біль. Дослідження, проведені в США, Німеччині, Швейцарії та інших країнах, показали, що робота з обслуговування ВДТ супроводжується підвищеним напруженням зору, інтенсивністю і монотонністю праці, збільшенням статичних навантажень, нервово-психічним напруженням, впливом різного виду випромінювань та ін. Внаслідок цього серед операторів ВДТ, як зазначають фахівці Всесвітньої організації охорони здоров'я, частіше, ніж в інших групах працюючих, трапляються такі професійні захворювання, як передчасна стомлюваність, погіршення зору, м'язові і головні болі, психічні й нервові розлади, хвороби серцево-судинної системи, онкологічні захворювання та ін. Вважається, що стан організму операторів ВДТ визначається комплексним впливом факторів трудового процесу і середовища, значення яких є неоднаковим. На операторів з малим стажем роботи на ВДТ домінуючий вплив

чинять фактори середовища, а на операторів зі стажем понад 5 років - фактори трудового процесу.

6.2.2 Зоровий дискомфорт

Комп'ютерний зоровий синдром (КЗС) – комплекс порушень здоров'я, який може виникати у користувачів персональних комп'ютерів (ПК). Діагноз ставлять, якщо людина, що працює за ПК протягом двох годин, висловлює хоча б дві з десяти скарг:

— головний біль;

— сльозотеча;

— різь;

— туман;

— двоїння;

— свербіж;

— важкість в очах;

— фотофобія;

— миготіння знаків на екрані;

— нудота.

У користувачів ПК дуже поширені кон'юнктивіти і блефарити, патогенетично пов'язані з КЗС.

Синдром розвивається при умові, що робоче місце організовано неправильно – у користувача незручне крісло, відсутні пюпітри для паперів, підставки для ніг та кистей рук, не встановлена висота і нахил монітора відносно очей, відстань від очей до екрана. За таких умов тіло людини при роботі займає вимушене положення: спина статично напружена, шия витягнута, плечі жорстко фіксовані. Напружені м'язи погіршують кровотік у сонних артеріях, а недостатнє кровозабезпечення головного мозку веде до очманіння, появи головного болю. На фоні шийного остеохондрозу з'являється відчуття випирання очних яблук, туману в очах, мушок та райдужних кіл у полі зору. Розвитку КЗС сприяє поганий мі-

кроклімат приміщення, значна загальна іонізація та мікробне забруднення, а також куріння.

Національною радою з наукових досліджень США для стану зорового дискомфорту був введений термін "астенопія", який означає "будь-які суб'єктивні зорові симптоми чи емоційний дискомфорт, що є результатом зорової діяльності". Симптоми астенопії були класифіковані на "очні" (біль, печія та різь в очах, почервоніння повік та очних яблук, ломота у надбрівній частині тощо) та "зорові" (пелена перед очима, мерехтіння, швидка втома під час зорової роботи та ін.).

У операторів ВДТ "очні" симптоми трапляються частіше, ніж "зорові", причому частота проявів астенопії вища у жінок, ніж у чоловіків і більше виражена в осіб середнього і старшого віку. Причиною вважається електромагнітне випромінювання від ВДТ.

При роботі з ВДТ основне навантаження припадає на всі елементи зорового аналізатора.

Робота з ВДТ може призвести до розвитку короткозорості, так як у користувачів комп'ютерів, в основному, "працює" ближній зір.

При аналізі зорової роботи операторів ВДТ, встановлено, що через дві години частота флуктуацій акомодації зменшується, а внесок низькочастотної компоненти підвищується. Це може бути причиною скарг на втому зорового аналізатора. Тривала робота на ВДТ може призвести до розвитку короткозорості, оскільки у користувачів ВДТ головним чином "працює" ближній зір.

У 100 пацієнтів із 150, які працювали на ВДТ по шість годин на день протягом чотирьох років, були виявлені проблеми з фокусуванням зору.

Робота за комп'ютером характеризується також тим, що постійний напружений погляд на екран монітора зменшує частоту моргання. При цьому погіршується зволоження поверхні очного яблука сльозовою рідиною, яка захищає рогівку ока від висихання, пилу та інших забруднень. Це може призвести до виникнення так званого синдрому Сікка: рогівка висихає і мутніє, і як наслідок розвивається сліпота.

Також при напруженій зоровій роботі за ЕОМ можуть бути не лише порушення функції зору, а й виникнення головного болю, посилення нервово-психічного напруження, зниження працездатності.

Виникнення та розвиток патології зорової функції зумовлені:

Умовами зорової роботи на ВДТ (зменшення вільного руху очей, зменшення функціонального поля сітківки та ін.). В природних умовах людина розглядає предмети, які знаходяться поблизу неї і на різних відстанях включно до горизонту (розслабляючи при цьому м'язи ока). Крім того, має місце вільний рух очей у всі боки. Відтак функціонує все поле сітківки ока. Різноманітні м'язи ока і різноманітні ділянки поля сітківки функціонують поперемінно, отримуючи можливість відновлювати свій функціональний потенціал. Умови зорової роботи при використанні ВДТ набагато жорсткіші, оскільки у користувача комп'ютера "працює" лише ближній зір, тому елементи ока, що його забезпечують знаходяться у постійному напруженні.

Змінами умов, характерних для традиційного зорового процесу читання (темні знаки на світлому фоні при падаючому світловому потоці), а також демонстрування зображення на майже вертикальній поверхні, що випромінює світловий потік, а отже, потребує пониженого загального освітлення на робочому місці. В деяких випадках ВДТ відтворює яскраві знаки на темному фоні (зворотнє зображення затруднює адаптацію).

Світлотехнічною різнорідністю об'єктів зорової роботи що пов'язана з наявністю трьох об'єктів (екран, клавіатура, документація), розташованих у різних зонах спостереження, що вимагає багаторазового переведення лінії зору від одного до іншого. Умови роботи з ВДТ ускладнюються необхідністю постійної перебудови апаратів акомодації та конвергенції, не кажучи вже про постійну необхідність переадаптації від яскравих об'єктів з позитивним контрастом на темні — з негативним. Разом узяті всі ці особливості створюють багато незручностей, а також напруження м'язового та світловідчуваючого апарату очей.

Робота з пульсуючим самосвітним об'єктом, який постійно перебуває у центрі поля зору, що не відповідає нормативним вимогам щодо обмеження пу-

льсації та засліпленості. Наявність пульсації яскравості знаків викликає дискомфорт і втому, загальну й здорову.

Несприятливим розподілом яскравості у полі зору (стеля, стіни, меблі тощо можуть виявитися світлішими, ніж центр поля зору - темний, обмежено освітлений та іноді малозаповнений знаками екран монітора);

Засліплююча дія світильників, які освітлюють приміщення на робочому місці з ВДТ більша, ніж на інших, бо лінія зору користувача при роботі з екраном майже горизонтальна, що призводить до зменшення кута дії різних засліплюючих джерел (світильники, вікна і т. п.) і, відповідно, до зростання засліпленості.

Отже, порушення зорових функцій користувачів ВДТ пов'язані, головним чином, з чотирма групами факторів:

— параметрами освітлення робочого місця;

— характеристиками дисплея;

— специфікою роботи на ВДТ;

— неправильною організацією робочого місця.

Висновки до розділу 6

В даному розділі було проаналізовано основні проблеми охорони праці, що можуть виникнути під час роботи працівника. Було виділено основні вимоги до приміщення, мікроклімату в приміщенні, освітлення та основних ергономічних характеристик.

У приміщенні застосовується бокове природне освітлення та штучне (два ряди світильників Л201Б 4х40-0.3, у кожному з яких знаходиться по чотири лампи типу ЛБ-40). Встановлено, що температура повітря у приміщенні становить 24 С. Зазначено, що приміщення за групою електронебезпечності відноситься до приміщень без підвищеної небезпеки ураження струмом.

Також окремо було розглянуто фактори, що впливають на стан користувачів комп’ютера. Зокрема, було детально розглянуто зоровий дискомфорт, його прояви та причини.

7 ЕКОЛОГІЯ

7.1 Роль матеріало- та ресурсозбереження у вирішенні екологічних проблем

У наші дні багато хто з нас шукають способи заощадити енергію, як допомогти навколишньому середовищу, так і заощадити гроші: придбати енергоефективні лампочки та прилади, налаштувати терморегулятор і спробувати «зменшити, повторно використовувати і переробити» там, де це можливо.

Але ми можемо не помітити того, що може бути величезною витратою енергії (вироблення якої завдає значної шкоди навколишньому середовищі на всіх етапах), збільшуючи щомісячні рахунки за комунальні послуги та без потреби заповнюючи сміттєзвалища і забруднюючи нашу планету — це наші комп’ютери.

За даними Агентства з охорони навколишнього природного середовища (EPA) та Західної мережі стійкості та запобігання забрудненню (WSPPN) лише американці створіть понад п'ять мільярдів фунтів комп’ютерних відходів; викидають 82% комп’ютерів на сміттєзвалищах замість того, щоб переробляти їх, лише в 2007 році було скинуто понад 40 мільйонів комп'ютерів та в середньому відмовляються від свого ПК після лише 30 місяців використання.

Задля забезпечення енергоефективності ПК рекомендується дотримуватися наступних правил:

1. Вимикати монітор коли його не використовують. Монітор витрачає багато енергії — на нього припадає приблизно третина електричного споживання всього комп'ютера. Варто налаштувати функцію сну монітора для автоматичного вимкнення живлення, коли ви перебуваєте поза комп'ютером (шукайте "Параметри живлення" на панелі керування). Цей режим сну все ще використовує деяку кількість енергію, тому наприкінці дня або якщо ви деякий час будете далеко від комп'ютера — просто вимкніть монітор. Також можна розглянути можливість зниження яскравості монітора; чим яскравіший дисплей, тим більше енергії він використовує. І може бути додатковий бонус: деякі люди повідомили, що затемнення дисплея зменшує напругу очей.

2. Вимикати заставку (англ. screensaver). Вона спочатку була розроблена для захисту старих монохроматичних моніторів, типу моніторів, які вже рідко зустрічаються. Нові монітори, такі як ті, що використовують рідкокристалічну та світлодіодну технологію, взагалі не потрібні заставки. Анімовані заставки можуть бути цікавими, але вони споживають стільки ж енергії, як монітор, який використовується, не кажучи вже про те, щоб витрачаються ресурси процесора та пам’яті, що спричиняє також зайве нагрівання ПК, а це, в свою чергу, потребує більшої потужності вентилятора, щоб зберегти його в прохолоді: це ефект доміно, що всі додає до непотрібного зливу енергії. За даними Агенції з охорони навколишнього середовища, вимкнення заставки може заощадити від 25 до 75 доларів на рік на енерговитратах.

3. Використовувати режими енергозбереження. Багато людей залишають свої комп’ютери, які працюють цілодобово — часто це робиться для того, щоб уникнути тривалого процесу завантаження, але іноді ми хочемо залишити свої ПК, щоб ми могли отримати доступ до них віддалено або щоб автоматичні процеси обслуговування могли працювати. Один із способів поєднати енергозбереження із зручністю ПК, що постійно працює, — це включити режим очікування або сплячий режим (також один із "Параметри живлення" в Windows). За допомогою цих режимів енергозбереження можна налаштувати комп’ютер на режим "сну" після його простою протягом певного часу, а комп'ютер буде "прокидатися" лише після повернення або під час виконання завдань з технічного обслуговування. У режимі очікування ПК використовує дуже мало енергії; деякі з них все ще використовуються, але живлення таких предметів, як монітор та жорсткий диск, зменшено. Коли користувач повернеться до свого ПК, ви будете швидко працювати, то комп’ютер буде використовувати більше енергії, ніж у сплячому режимі. У режимі сну ПК взагалі не використовує енергію. З двох, цей режим, безумовно, економить найбільше енергії, але часу, необхідного для ввімкнення всього, буде трохи довше.

4. Слідкувати за налаштуваннями ПК. Для будь-якого з ваших приладів правильне обслуговування зменшує відходи, і те саме стосується комп’ютерів.

Правильно налаштований комп'ютер, як і автомобіль, може працювати ефективніше: худорлявий ПК споживає менше електроенергії та працює набагато краще. З часом ПК стає роздутим і неефективним: налаштування застарівають, коли додаються та видаляються програми, а також через кеш програм, який залишається зберігається на комп’ютері.

7.2 Статистика екології об’єктів природного середовища

Збором та обробкою даних про стан та охорону об’єктів природного середовища в Україні займається Державна служба статистики (далі — Держстат) під керівництвом міністерства економічного розвитку і торгівлі України.

На їхньому офіційному сайті можна ознайомитися з актуальною статистикою наступних типів:

— демографічна та соціальна;

— економічна;

— багатогалузева статистична інформація.

Статистику про навколишнє природне середовище Держстат визначає як економічну інформацію. Представлено наступні категорії:

— відходи;

— викиди забруднюючих речовин в атмосферне повітря;

— витрати на охорону навколишнього природного середовища;

— використання та охорона водних ресурсів;

— екологічні показники, рекомендовані ЄЕК ООН, що виробляються органами державної статистики;

— екологічні рахунки.

Наприклад, на рис. 7.1 зображено статистику викидів забруднюючих речовин в атмосферне повітря з 1991 по 2018 рік включно. На графіку видно позити-

вну тенденцію. Також, на рис. 7.2 зображено кінцеве використання електроенергії по галузях.

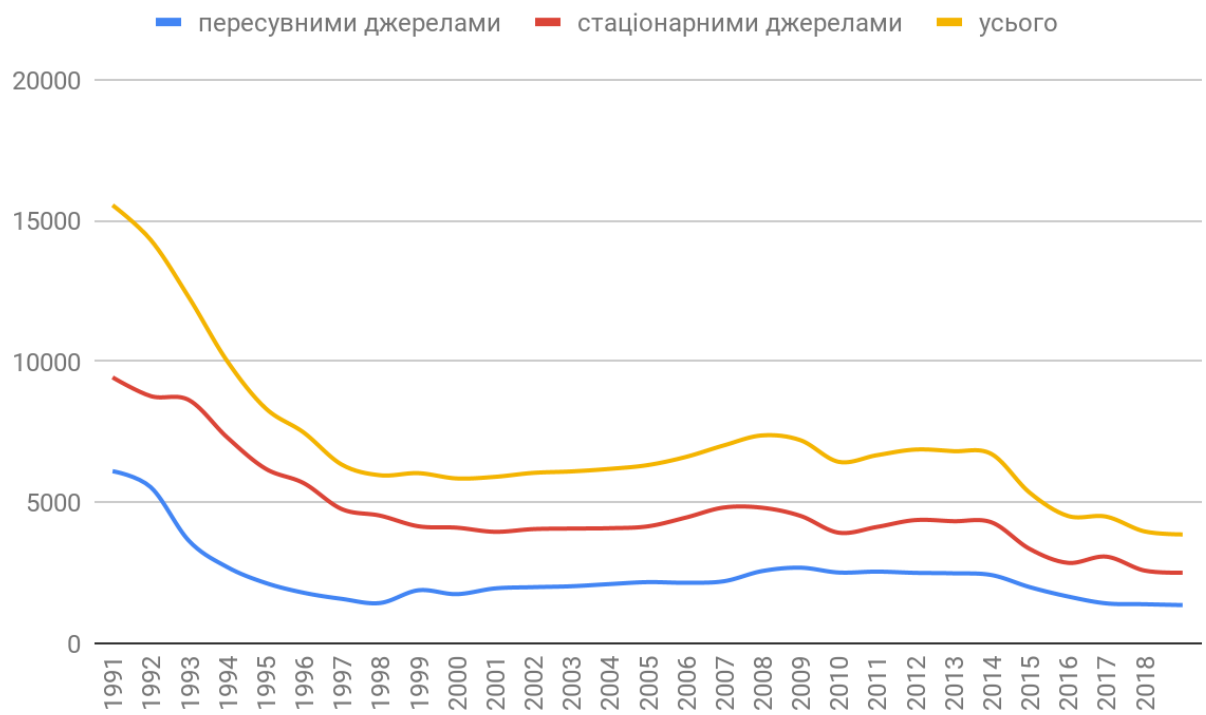


Рис 7.1 — Викиди забруднюючих речовин в атмосферне повітря

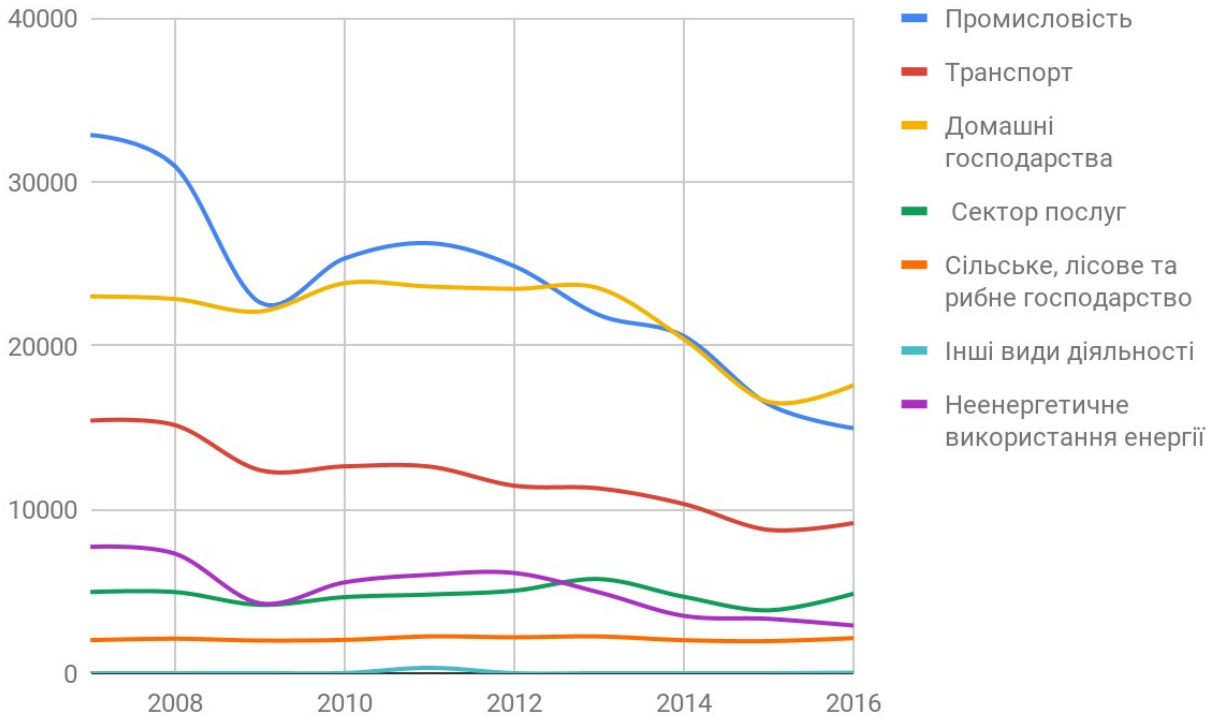


Рисунок 7.2 — Використання електроенергії за галузями

Висновки до розділу 7

Екологія є важливою наукою про навколишнє середовище і один з її напрямів вивчає вплив людини, прогресу і технологій на довкілля та організми. Саме завдяки цій науці ми знаємо про те наскільки згубною є діяльність людського виду і можемо шукати способи послабити цей негативний вплив.

Зокрема, у даному розділі розглянуто методи ефективного використання ресурсів користувачами ПК та статистику екології об’єктів природного середовища, що є лише малою частиною питань, що розглядає екологія.

ВИСНОВКИ

Проблема захисту інформації при її передачі по мережі залишатиметься актуальною доки ці мережі будуть активно використовуватись підприємствами, установами, організаціями та окремими людьми по всьому світу.

А зараз складно уявити життя інакшим, тобто без миттєвої передачі даних у цифровому форматі на великі відстані. Це пов'язано зі зручністю, яку дарує така можливість при спілкування з друзями чи веденні бізнесу.

Одним із засобів захисту інформації при її передачі по мережі є end-to-end шифрування трафіку, завдяки чому інформація залишається конфіденційною, а за допомогою ЕЦП, сторони можуть бути впевнені у цілісності повідомлень та їх авторстві. Процес криптографічного захисту регламентується відповідним стандартом протоколу передачі. Ці протоколи можуть працювати на мережевому або на сеансовому рівні моделі OSI. Найпопулярнішими такими протоколами є IPSec і OpenVPN, що працюють на мережевому рівні, TLS, що працює на сеансовому рівні. Останній і розглядається у даній роботі.

Протокол TLS активно використовується у всесвітній павутині з 1995 року. За цей час вийшло 6 версій, найновішою на момент створення цієї роботи була версія TLSv1.3.

Оскільки стандарт TLS є відкритим, то існує значна кількість реалізацій протоколу, найпопулярнішими з яких є OpenSSL, GnuTLS та Network Security Services (NSS). Всі вони написані мовою програмування C. При розробці любого ПЗ виникають помилки та інші вади, що можуть призвести до виникнення критичних для інформаційної безпеки вразливостей, що ставлять під сумнів захищеність даних, що передаються протоколом TLS. Тому пошук вразливостей на стадії розробки імплементації протоколу TLS є надзвичайно важливим процесом.

У базі даних вразливостей CVE міститься 286 вразливостей трьох згаданих раніше реалізацій протоколу TLS, що належать до наступних категорій:

— відмова в обслуговуванні (DoS);

— загроза виконання коду;

— переповнення буферу;

— обхід чогось, а саме певного етапу встановлення з’єднання чи передачі інформації каналом, що становить загрозу конфіденційності чи цілісності інформації;

— псування пам’яті.

Найбільше вразливостей виявлено у бібліотеці OpenSSL.

У даній роботі пропонується методологія, яка дозволяє суттєво зменшити кількість вразливостей у коді імплементації протоколу TLS. Вона складається з наступних пунктів:

1. Статичний аналіз коду за допомогою двох відкритих аналізаторів коду — Cppcheck та Clang Static Analyzer, які розглядаються у підрозділі 3.1.

2. Написання модульних тестів паралельно з кодом протоколу та визначення покриття коду тестами за допомогою вбудованої у компілятор GCC функції та застосунків gcov та lcov.

3. Проведення динамічного аналізу коду реалізації протоколу TLS за допомогою аналізатора Valgrind для пошуку вразливостей, що пов’язані з неправильною роботою з оперативною пам’яттю програми.

Розроблену методологію було застосовано до відкритої реалізації протоколу TLS під назвою OpenSSL, в результаті чого було сформовано рекомендації щодо вдосконалення коду цієї бібліотеки. Їх виконання дозволить своєчасно знаходити вразливості, що виникають під час розробки нових версії бібліотеки.

Дана методологія може легко розширюватися та проводитися автоматизовано. Також, вона може бути легко застосовано до ПЗ інших типів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. International Telecommunication Union Measuring the Information Society Report Volume 1 URL: https://www.itu.int/en/ITU-D/Statistics/Documents/publications/misr2018/MISR-2018-Vol-1-E.pdf (дата звернення 15.12.2019).

2. Let's Encrypt - Free SSL/TLS Certificates URL: https://letsencrypt.org/ (дата звернення 15.12.2019).

3. Y. Sheffer, R. Holz, P. Saint-Andre Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS).

4. David Sounthiraraj Justin Sahs Garret Greenwood Zhiqiang Lin Latifur Khan SMV-HUNTER: Large Scale, Automated Detection of SSL/TLS Man-in-the-Middle Vulnerabilities in Android Apps.

5. Janet Ellen Abbate. From ARPANET to Internet: A history of ARPA-sponsored computer networks, 1966-1988.

6. "W3C timeline". Archived from the original on 31 March 2010. Retrieved 30 March 2010.

7. "The Early World Wide Web at SLAC". Archived from the original on 24 November 2005.

8. "About SPIRES". Archived from the original on 12 February 2010. Retrieved 30 March 2010.

9. "A Little History of the World Wide Web". Archived from the original on 6 May 2013.

10. Conklin, Jeff (1987), IEEE Computer, 20, pp. 17–41

11. "Inventor of the Week Archive: The World Wide Web". Massachusetts Institute of Technology: MIT School of Engineering. Archived from the original on 8 June 2010. Retrieved 23 July 2009.

12. "Ten Years Public Domain for the Original Web Software". Tenyears-www.web.cern.ch. 30 April 2003. Archived from the original on 13 August 2009. Retrieved 27 July 2009.

13. Menezes, Alfred J.; Oorschot, Paul C. van; Vanstone, Scott A. (2001). Handbook of Applied Cryptography (Fifth ed.). p. 251. ISBN 978-0849385230.

14. W. Diffie and M. E. Hellman New Directions in Cryptography.

15. R.L. Rivest, A. Shamir, and L. Adleman A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. URL: http://people.csail.mit.edu/rivest/Rsapaper.pdf

16. Certicom Research SEC 1: Elliptic Curve Cryptography. URL: http://www.secg.org/sec1-v2.pdf

17. John Catsoulis Designing Embedded Hardware, 2nd Edition. O'Reilly. ISBN 0-596-00755-8.

18. "An Overview of Public Key Infrastructures (PKI)". Techotopia.

19. "Public Key Infrastructure". MSDN.

20. "Using Client-Certificate based authentication with NGINX on Ubuntu - SSLTrust". SSLTrust.

21. Vacca, Jhn R. (2004). Public key infrastructure: building trusted applications and Web services. CRC Press. p. 8. ISBN 978-0-8493-0822-2.

22. McKinley, Barton (January 17, 2001). "The ABCs of PKI: Decrypting the complex task of setting up a public key infrastructure". Network World. Archived from the original on May 29, 2012.

23. "Implementation of IPSec Protocol - IEEE Conference Publication". ieeexplore.ieee.org. Retrieved 2018-05-12.

24. Network Encryption - history and patents
URL: http://www.toad.com/gnu/netcrypt.html

25. The history of VPN creation
URL: https://www.trustedcoupon.com/the-history-of-vpn-creation/

26. IETF IP Security Protocol (ipsec) Working group History
URL: https://datatracker.ietf.org/wg/ipsec/history/

27. "RFC4301: Security Architecture for the Internet Protocol". Network Working Group of the IETF. December 2005. p. 4. The spelling "IPsec" is preferred and used

throughout this and all related IPsec standards. All other capitalizations of IPsec [...] are deprecated.

28. "NRL ITD Accomplishments - IPSec and IPv6" (PDF). US Naval Research Laboratories.

29. The Internet Key Exchange (IKE), RFC 2409, §1 Abstract

30. Harkins, D.; Carrel, D. (November 1998). The Internet Key Exchange (IKE). IETF. doi:10.17487/RFC2409. RFC 2409.

31. OpenVPN man page, section "TLS Mode Options"

32. Petros Daras; Oscar Mayora (2013). User Centric Media: First International Conference, UCMedia 2009, Venice, Italy, December 9-11, 2009, Revised Selected Papers. Springer Science & Business Media. p. 239. ISBN 978-3-642-12629-1.

33. OpenVPN community wiki, IPv6 in OpenVPN

34. Titz, Olaf Why TCP Over TCP Is A Bad Idea

35. Honda, Osamu; Ohsaki, Hiroyuki; Imase, Makoto; Ishizuka, Mika; Murayama, Junichi "Understanding TCP over TCP: effects of TCP tunneling on end-to-end throughput and latency".

36. "OpenVPN Security Overview".

37. "OpenVPN script entry points". Openvpn.net.

38. OpenVPN plug-in entry points for C based modules.

39. "OpenVPN example plug-ins". Openvpn.git.sourceforge.net.

40. OpenVPN Community Wiki - Related Projects

41. Thomas Y.C. Woo, Raghuram Bindignavle, Shaowen Su and Simon S. Lam. Department of Computer Sciences The University of Texas at Austin, Austin, Texas 78712-118

42. Stuart Sechrest An Introductory 4.4BSD Interprocess Communication Tutorial URL: https://docs.freebsd.org/44doc/psd/20.ipctut/paper.pdf

43. draft-hickman-netscape-ssl-00

URL: https://tools.ietf.org/html/draft-hickman-netscape-ssl-00

44. RFC 6101 - The Secure Sockets Layer (SSL) Protocol Version 3.0

URL: https://tools.ietf.org/html/rfc6101

45. RFC 2246 - The TLS Protocol Version 1.0
URL: https://tools.ietf.org/html/rfc2246
46. RFC 4346 - The Transport Layer Security (TLS) Protocol Version 1.1
URL: https://tools.ietf.org/html/rfc4346
47. PRIVATE ENTERPRISE NUMBERS
URL: https://www.iana.org/assignments/enterprise-numbers/enterprise-numbers
48. RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2
URL: https://tools.ietf.org/html/rfc5246
49. RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3
URL: https://tools.ietf.org/html/rfc8446
50. Гайворонський М. В., Новіков О. М. Безпека інформаційно-комунікаційних систем. — К.: Видавнича група BHV, 2009. — 608с.
51. CVE - Towards a Common Enumeration of Vulnerabilities
URL: https://cve.mitre.org/docs/docs-2000/cerias.html
52. CVE — History URL: https://cve.mitre.org/about/history.html
53. Cppcheck - A tool for static C/C++ code analysis
URL: http://cppcheck.sourceforge.net/#features
54. Fowler, Martin (1 May 2006). "Continuous Integration". martinfowler.com. Retrieved 9 January 2014.
55. Booch, Grady (1991). Object Oriented Design: With Applications. Benjamin Cummings. p. 209. ISBN 9780805300918. Retrieved 18 August 2014.
56. G. E. Kaiser, D. E. Perry and W. M. Schell, "Infuse: fusing integration test management with change management," [1989] Proceedings of the Thirteenth Annual International Computer Software & Applications Conference, Orlando, FL, USA, 1989, pp. 552-558. doi:10.1109/CMPSAC.1989.65147
57. Booch, Grady (December 1998). "Object-Oriented Analysis and Design with applications (2nd edition, 15th printing)" (PDF). www.cvauni.edu. Retrieved 2 December 2014.
58. Beck, Kent (28 March 1998). "Extreme Programming: A Humanistic Discipline of Software Development". Fundamental Approaches to Software Engineering: First

International Conference, FASE'98, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, 28 March – 4 April 1998, Proceedings, Volume 1. Lisbon: Springer. p. 4. ISBN 9783540643036.

59. Barquet, Ana Paula B., et al. "Business model elements for product-service system". Functional Thinking for Value Creation. Springer Berlin Heidelberg, 2011. 332–337: They stated that "The Canvas business model was applied and tested in many organizations (eg IBM and Ericsson), being successfully used to easily describe and manipulate business models to create new strategic alternatives."

60. Osterwalder, Alexander; Pigneur, Yves; Clark, Tim (2010). Business Model Generation: A Handbook For Visionaries, Game Changers, and Challengers. Strategyzer series. Hoboken, NJ: John Wiley & Sons. ISBN 9780470876411. OCLC 648031756. With contributions from 470 practitioners from 45 countries.

61. Osterwalder, Alexander (2004). The Business Model Ontology: A Proposition In A Design Science Approach (PDF) (Ph.D. thesis). Lausanne: University of Lausanne. OCLC 717647749. See also: Osterwalder, Alexander; Pigneur, Yves; Tucci, Christopher L. (2005). "Clarifying business models: origins, present, and future of the concept". Communications of the Association for Information Systems. 16 (1): 1. doi:10.17705/1CAIS.01601.

62. Державна служба статистики України Економічна статистика / Навколишнє природне середовище URL: http://www.ukrstat.gov.ua/operativ/menu/menu_u/ns.htm

Додатки

Додаток А

УДК 004.056.5

О.Г. Зимницький

Тернопільський національний технічний університет імені Івана Пулюя

ВРАЗЛИВОСТІ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ МЕТОДІВ ЗАХИСТУ ПРОТОКОЛУ SSL/TLS

Сучасні методи криптографічного захисту інформації забезпечують достатній рівень захисту оскільки вони ґрунтуються на складних математичних принципах, а за рахунок відкритості алгоритму їх ретельний криптоаналіз проведений і продовжує проводитися значною кількістю спеціалістів у сфері захисту інформації по всьому світу.

У зв'язку з цим вразливості у методах шифрування досить рідкісне явище. Частіше вони стають застарілими через довжину ключа чи інші фактори і з розвитком апаратного забезпечення та технологіями розподілених обчислень стають вразливими до атак перебору чи різного роду колізійних атак.

Проте у мережі ці методи криптографічного захисту використовуються у рамках певного протоколу, найпопулярнішим з яких на сьогодні є TLS. І вразливості можуть бути закладеними у саму архітектуру протоколу, як це трапилося з SSLv3 і вразливістю CVE-2014-3566 (так звана POODLE атака). Невідомо скільки зловмисників користувалися цією атакою до моменту її виявлення у 2014 році. Чи CVE-2011-3389 (BEAST) до якої були вразливі всі блокові шифри, що використовуються у SSLv3 і TLSv1.0

Однак найчастіше виникають вразливості через помилки робробників бібліотек, що реалізують TLS протокол. Лише за поточний рік у CVE зареєстровано 9 вразливостей OpenSSL, дві у GnuTLS. І чим більшою є спільнота розробників тим важче попереджати чи виявляти навмисне чи ненавмисне внесення вразливостей у вихідний код бібліотеки чи програми, що реалізує криптографічний захист інформації і виникає потреба в автоматизації цього процесу.

Для попередження виникнення таких вразливостей необхідно проводити ретельний аналіз змін, що роблять розробники та всього коду в цілому. Це можна робити за допомогою статичного та динамічного аналізу коду, зокрема модульного та інтеграційного тестування коду.

Сучасні статичні аналізатори здатні виявляти вразливі ділянки коду, це неправильне використання пам'яті, блокування потоку чи виклик вразливих чи застарілих функцій з зовнішніх бібліотек.

Динамічний аналіз передбачає запуск програми або її частин. Деякі аналізатори дозволяють виявити приховані вади, переважно пов'язані з неправильним доступом до оперативної пам'яті, що можуть становити загрозу.

Модульне та інтеграційне тестування робить код криптографічної підсистеми більш передбачуваним. Наприклад, це дозволяє впевнитися у правильності реалізації (тобто відповідності стандартам) окремих алгоритмів шифрування, хешування та ЕЦП, а також у їх сумісності під час передачі інформації по мережі. Можна перевіряти різні сценарії атак та заносити існуючі у попередніх версіях вразливості для попередження їх появи знов. Проте, все залежить від якості подібних тестів, тобто наскільки добре вони покривають код. Це можна визначати по відсотку покритих тестами стрічок (англ. code coverage), що викликаються під час тестування, а також за допомогою мутаційного тестування. Останнє суттєво покращує якість автоматизованих тестів.