МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ С.В.Казмірчук
«_____»_____2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА
ЗА СПЕЦІАЛЬНІСТЮ 125 «КІБЕРБЕЗПЕКА»

Тема: «Засоби та механізми захисту даних користувачів у веб-застосунках»

Виконавець: студентка групи БІ-201МЗ_____Рой А.В.

Керівник: д.т.н., професор_____Пархоменко І.І.

Нормоконтролер: _____ _____
(підпис) (П.І.Б.)

Київ 2020

ВСТУП

На сьогоднішній день кількість інформації, яка циркулює у мережі Інтернет, є незліченною. Технології та вартість утримання сайтів стали більш доступними, що дозволило кількості веб-сайтів та веб-застосунків швидко зрости. За даними ресурсу internet live stats наразі в Інтернеті існує більше ніж 1,7 мілліарда унікальних веб-сайтів [1] та 4,5 мілліарда унікальних користувачів та ця кількість тільки зростає. [2]

Інформація цих користувачів циркулює в мережі кожної секунди і може включати як чисто технічну інформацію, яка не має ніякого сенсу, так і чутливу інформацію, витік якої може бути використаний для завдання шкоди користувачеві. Така чутлива інформація може включати дані кредитних карток, які використовуються для оплати послуг в Інтернеті, адреса проживання, паролі та інше. Навіть така на перший погляд без загрозлива інформація як пошукові запити на інтернет-магазинах може бути використана проти користувача або для збору іншої, більш чутливої інформації. За даними ресурсу have I been pwned в вільному доступі наразі перебуває 135 мільйонів аккаунтів [3], тобто пар логін та пароль, які були отримані через недостатній захист даних веб-застосунками, які обробляли ці дані. В загалі було через такі витоки інформації було зкомпроментовано 9,6 мілліардів аккаунтів і це тільки відомі публічно, безліч залишається прихованою і неопублікованою. [3]

Одним з варіантів, які дозволяють мінімізувати наслідки витоку чутливої інформації є використання сервісів та механізмів безпеки. Звичайно, тільки це не дозволить повністю позбавитися ризику витоку даних, людський фактор та неправильне написання самого коду веб-застосунку, що дозволяє працювати таким сплоїтам як SQL ін’єкція або кросс-сайт скриптінг, є зонами, які не покриваються сервісами та механізмами безпеки. Якщо бути більш точним, сервіси та механізми безпеки дозволяють забезпечити захист даним, які перебувають у стані передачі від клієнта до сервера чи навпаки. В випадку веб-застосунків, як буде визначено далі, клієнтом є

браузер або мобільний застосунок, який надсилає запити до сервера і відповідає за презентацію користувачеві контенту чи для вводу інформації користувачем. Сервер же це застосунок, який оброблює запити клієнта і видає відповідну інформацію. Інакше кажучи, клієнт є інтерфейсом для взаємодії користувача з сервером.

Загальне визначення сервісам та механізмам безпеки надано у специфікації ITU-T X.800 та ISO 7498-2 (Системи обробки інформації - Взаємозв'язок відкритих систем - Основна довідкова модель - Частина 2: Архітектура безпеки) чим ми і будемо користуватися. Проте дані документи представляють собою загальний опис того, які задачі має вирішувати той чи інший сервіс або механізм і не надає його точної реалізації. Впродовж часу, який пройшов з моменту виходу цих документів, була розроблена велика кількість варіантів реалізацій цих сервісів та механізмів, кожен з яких має свої плюси та недоліки.

Виходячи з вищенаведеної інформації, можна сформулювати наступні тезиси.

Актуальність. Питання безпеки стрімко зростаючої кількості веб-застосунків є життєво необхідним для роботи в мережі Інтернет, а сервіси та механізми безпеки частково допомагають у вирішенні цього питання.

Метою роботи є створення експериментальної моделі веб-застосунку з використанням сервісів та механізмів безпеки для захисту даних користувачів.

Задачі, які були поставлені:

- Провести аналіз стандартів, які описують сервіси та механізми безпеки;
- Дослідити стандарти, які описують технології реалізації сервісів та механізмів безпеки для застосування у веб-застосунках;
- Проаналізувати процеси роботи обраних реалізацій механізмів безпеки;
- Дослідити особливості використання обраних механізмів захисту у веб-застосунках;
- Створити експериментальну модель веб-застосунку для демонстрації захисту даних за допомогою сервісів та механізмів безпеки.

Об’єктом дослідження даної роботи є процес захисту інформації користувачів, яка циркулює у веб-застосунках, за допомогою сервісів та механізмів безпеки.

Предметом дослідження є сервіси та механізми безпеки, які використовуються для захисту даних користувачів у веб-застосунках.

Методи дослідження. В роботі використані наступні методи дослідження:

- Порівняння;
- Аналіз;
- Моделювання.

Практична цінність - створення захищеного веб-застосунку з використанням сервісів та механізмів безпеки реалізованих за допомогою стандартизованих технологій.

РОЗДІЛ 1
ПОНЯТТЯ СЕРВІСІВ ТА МЕХАНІЗМІВ БЕЗПЕКИ

1.1 Нормативно-правова база та визначення сервісів безпеки

Служба безпеки - це служба, що надається рівнем, що відповідає за комунікацію, у відкритих системах, який забезпечує безпеку систем або ліній передачі даних, як визначено Рекомендацією ITU-T X.800 [5]. Специфікація X.800 та ISO 7498-2 (Системи обробки інформації - Взаємозв'язок відкритих систем - Основна довідкова модель - Частина 2: Архітектура безпеки) мають схожий технічний зміст.

Більш загальне визначення міститься в Інструкції CNSS № 4009 від 26 квітня 2010 року Комітета з питань національної безпеки Сполучених Штатів Америки: «Можливість, яка підтримує одну або декілька вимог безпеки (конфіденційність, цілісність, доступність). Прикладами служб безпеки є управління ключами, контроль доступу та автентифікація». [7]

Ще одне авторитетне визначення міститься у Глосарії веб-сервісів W3C, прийнятому NIST SP 800-95: «Служба обробки або зв'язку, яка надається системою для надання конкретного виду захисту ресурсам, коли зазначені ресурси можуть перебувати з цією системою або з іншими системами, наприклад, службою автентифікації або назначення атрибутів документам на основі PKI та сервісом автентифікації. Служба безпеки - це набір служб AAA (Authentication, Authorization, Accounting). Служби безпеки, як правило, реалізують частини політики безпеки та реалізуються через механізми безпеки.» [8]

В подальшому будемо використовувати термінологію надану в стандарті X.800. Дана рекомендація:

- надає загальний опис служб безпеки та пов'язаних з ними механізмів, які можуть надаватися відповідною моделлю;

- визначає місця в відповідній моделі, де такі служби та механізми можуть надаватися.

Рекомендація розширює сферу застосування Рекомендації X.200 і охоплює захищені комунікації між відкритими системами.

В даній роботі сервіси та механізми безпеки розшлядаються для використання саме у клієнт-серверних веб-застосунках, тому необхідно дати визначення такому застосунку. Веб-додаток - це концепція, яка була представлена у Serverlet Java Specification версії 2.2. З специфікації (глава 9):

«Веб-додаток - це сукупність сервлетів (програмний компонент Java, який розширює можливості сервера), html-сторінок, класів та інших ресурсів, які можна комплектувати та запускати в декількох контейнерів від багатьох постачальників. Веб-додаток укорінюється певним шляхом у веб-сервері. Наприклад, додаток каталогу може бути розміщений за адресою http: // www.mycorp.com/catalog. Усі запити, які починаються з цього префікса, будуть перенаправлені до ServletContext, який представляє додаток каталогу. Контейнер сервлетів також може встановлювати правила для автоматичного створення веб-додатків. Наприклад, ~ user / mapping може бути використаний для відображення веб-програми на базі / home / user / public_html /». [10]

В більш сучасному та загальному значені веб-застосунки можна визначити як розподілений застосунок, в якому клієнтом виступає браузер, а сервером — веб-сервер. Браузер може бути реалізацією тонких клієнтів — логіка застосунку зосереджується на сервері, а функція браузера полягає переважно у відображенні інформації, завантаженої мережею з сервера, і передачі назад даних користувача.

1.2 Основні сервіси безпеки OSI відповідно до стандарту ITU-T X.800

Для розуміння визначень сервісів необхідно звернутись до рекомендації ITU-T X.200 та стандарту ISO 7498. Згідно з стандартом ISO 7498 існує мережева моделі взаємодії відкритих систем (мережева модель OSI), яка представляє собою абстрактну

мережеву модель для комунікації і розробки мережевих протоколів. Ця модель представляє собою набір рівнів, кожен з яких обробляє свою частину процесу взаємодії. Кожен рівень розташований вертикально один над одним та може взаємодіяти тільки з сусідніми рівнями. [11] В цій моделі виділяють 7 рівнів, назви та загальні функції яких зображено на рисунку 1.

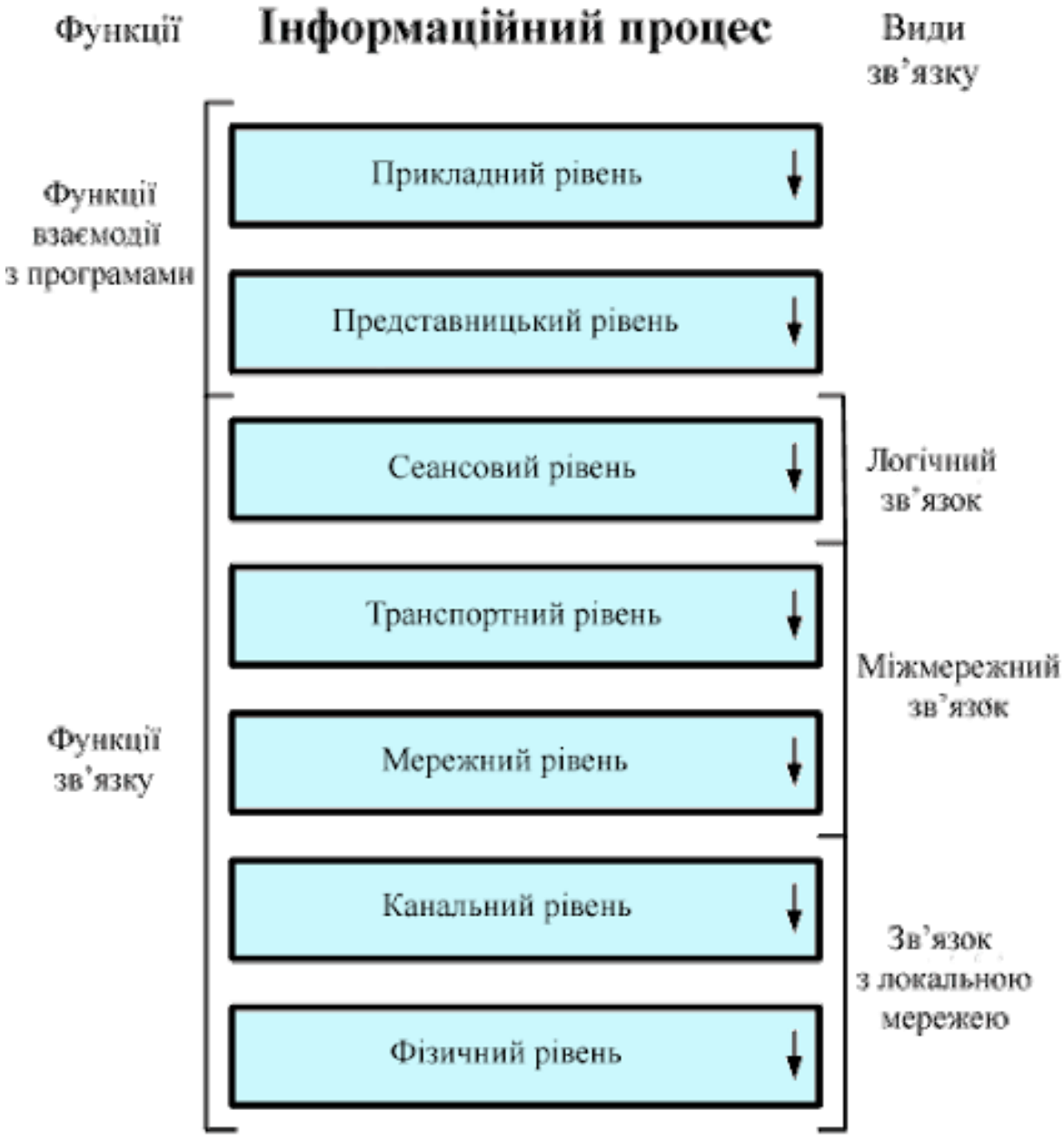


Рисунок 1.1 - Еталонна модель OSI

Згідно з рекомендацією X.200, кожен рівень моделі OSI загалом називається N шаром. Суб'єкт N+1 запитує послуги передачі N об'єкту. На кожному рівні два об'єкти (N-сутність) взаємодіють за допомогою (N) протоколу шляхом передачі блоків даних протоколу (PDU). Блок службових даних (SDU) - це конкретна одиниця даних, яка була передана з рівня OSI на нижчий рівень і ще не була інкапсульована в PDU нижчим рівнем. Це сукупність даних, що надсилається користувачем послуг певного рівня і передається семантично незмінним користувачу однорангової послуги. PDU у будь-якому заданому шарі, шарі 'N', є SDU рівня нижче, шару 'N-1'. Фактично SDU - це «корисне навантаження» даного PDU. Тобто процес зміни SDU на PDU складається з процесу інкапсуляції, який виконується нижчим шаром. Всі дані, що містяться в SDU, інкапсулюються в PDU. Шар N-1 додає заголовки або колонтитули, або обидва, в SDU, перетворюючи його в PDU шару N-1. Додані заголовки або колонтитули є частиною процесу, який використовується для отримання можливості отримати дані з джерела до місця призначення.

Далі надано список служб, які вважаються службами безпеки за Рекомендацією X.800 та можуть надаватися в рамках відповідної моделі OSI.

1.2.1 Автентифікація

Автентифікація - процедура встановлення належності користувачеві інформації в системі за допомогою пред'явленого ним ідентифікатора. З позицій інформаційної безпеки автентифікація є частиною процедури надання доступу для роботи в інформаційній системі, наступною після ідентифікації і передує авторизації.

Дослідження в сфері безпеки визначили, що для позитивної автентифікації слід перевірити елементи щонайменше з двох, а краще всіх трьох, факторів. [13] До цих трьох факторів належать:

- фактор знання: те, що користувач знає. Як приклад, таким знанням може бути пароль, частковий пароль, фраза проходу або персональний ідентифікаційний номер (PIN), відповідь на запитання чи шаблон;
- фактор володіння: те, чим користувач володіє. Наприклад, ідентифікаційна картка, апаратний токен, мобільний телефон із вбудованим апаратним токеном, програмним токеном або мобільним телефоном, що містить програмне забезпечення, що генерує токен;
- фактор невід'ємності: те, яким користувач є. Тобто біометричні параметри такі як відбиток пальців, малюнок сітківки, послідовність ДНК, форма обличчя, голос, унікальні біоелектричні сигнали чи інший біометричний ідентифікатор.

Сервіси автентифікації забезпечують автентифікацію отримувача та джерела даних, як описано нижче:

- Автентифікація отримувача. Ця послуга, коли надається (N) - шар, забезпечує підтвердження (N+1) - отримувача, що отримувач є заявленою (N+1) – сутністю;
- Автентифікація джерела походження даних. Ця послуга, коли надається (N) - шар, забезпечує підтвердження (N+1) - отримувача, що джерелом даних є заявлена (N+1) – сутність;

1.2.2 Управління доступом

Цей сервіс забезпечує захист від несанкціонованого використання ресурсів, доступних через OSI. Це можуть бути як ресурси OSI так і не OSI, до яких можна отримати доступ через протоколи OSI. Ця служба захисту може застосовуватися до різних типів доступу до ресурсу (наприклад, використання ресурсу зв'язку; читання, записування чи видалення інформаційного ресурсу; виконання ресурсу обробки) або до всіх доступів до ресурсу.

1.2.3 Конфіденційність даних

В інформаційній безпеці конфіденційність "є властивістю, що інформація не надається або розголошується стороннім особам, організаціям або процесам". [14] Хоча подібні до "приватності", ці два слова не є взаємозамінними. Швидше конфіденційність - це складова приватності, яка захищає дані від несанкціонованого перегляду. Приклади скомпрометованих електронних даних за фактором конфіденційності включають крадіжку ноутбука, крадіжку пароля або чутливі електронні листи, що надсилаються неправильним особам.

Служби конфіденційності забезпечують властивість конфіденційності за допомогою наступних чотирьох сервісів:

- Конфіденційність з'єднання. Сервіс забезпечує конфіденційність усіх (N) - користувачів даних на (N) - з'єднанні;
- Конфіденційність без встановлення зв’язку. Сервіс забезпечує конфіденційність даних усіх (N) - користувачів у єдиному (N) – SDU без встановлення зв’язку;
- Вибірна конфіденційність поля. Сервіс забезпечує конфіденційність вибраних полів у межах (N) -користувача даних на (N) - з'єднанні або в єдиному (N) - SDU без встановлення зв’язку;
- Конфіденційність потоку трафіку. Сервіс забезпечує захист інформації, яка може бути отримана за допомогою спостереження та аналізу потоків трафіку.

1.2.4 Цілісність даних

В інформаційній безпеці цілісність даних означає збереження та забезпечення точності та повноти даних протягом усього її життєвого циклу. [15] Це означає, що дані не можуть бути змінені несанкціонованим способом або способом, який не можна відслідкувати у системі. Системи захисту інформації зазвичай забезпечують цілісність повідомлення поряд з конфіденційністю.

Служби цілісності протидіють активним загрозам і можуть приймати одну з форм, описаних нижче:

- Цілісність з'єднання з відновленням. Ця послуга забезпечує цілісність даних усіх (N) - користувачів на (N) -з'єднанні та виявляє будь-які зміни, вставлення, видалення чи повтор будь-яких даних у всій послідовності SDU (із спробою відновлення);
- Цілісність з'єднання без відновлення. Ця послуга забезпечує цілісність даних усіх (N) - користувачів на (N) -з'єднанні та виявляє будь-які зміни, вставлення, видалення чи повтор будь-яких даних у всій послідовності SDU (без спроби відновлення);
- Вибіркова цілісність з'єднання поля. Ця послуга забезпечує цілісність обраних полів у даних (N) - користувача (N) - SDU, переданих через з'єднання, і визначає, чи були вибрані поля змінені, вставлені, видалені чи відтворені;
- Цілісність без встановлення зв’язку. Ця послуга, коли надається (N) - шар, забезпечує гарантію цілісності запитуючої (N+1) - сутності. Ця послуга забезпечує цілісність єдиного SDU без встановлення зв’язку і визначає, чи був змінений отриманий SDU. Крім того, може бути передбачена обмежена форма виявлення повтору;
- Вибіркова цілісність поля без встановлення зв’язку. Ця послуга забезпечує цілісність вибраних полів у межах одного SDU без встановлення зв’язку і приймає форму визначення того, чи були вибрані поля змінені.

1.2.5 Неспростовність

Неспростовність в даному випадку може бути трактована як намір сторін взаємодії виконувати свої зобов'язання перед деяким договором або контрактом. Це також означає, що одна сторона угоди не може заперечити отримання транзакції, а також не може заперечити відправлення транзакції. [16]

Важливо відзначити, що хоча такі технології, як криптографічні системи, можуть сприяти неспростовності, в основі цієї концепції лежить правова концепція, що виходить за рамки технології. Наприклад, недостатньо показати, що повідомлення відповідає

цифровому підпису, підписаному приватним ключем відправника, і, таким чином, лише відправник міг надіслати повідомлення, і ніхто інший не міг його змінити під час передачі. Передбачуваний відправник може натомість продемонструвати, що алгоритм цифрового підпису є вразливим або несправним, стверджувати чи доводити, що ключ його підпису був порушений. Вина у цих порушеннях може як бути так і не бути на стороні відправника, і такі твердження можуть або не можуть звільнити відправника відповідальності, але твердження призведе до визнання недійсним вимоги про те, що підпис обов'язково підтверджує справжність та цілісність. Таким чином, відправник може відкинути повідомлення, оскільки автентичність та цілісність є необхідними умовами для невідхилення.

Тим не менш, сервіс неспростовності може мати одну або обидві форми:

- Неспростовність джерела. Одержувачу даних надається доказ походження даних. Це захистить від будь-якої спроби відправника помилково спростовувати факт надсилання даних або їх вмісту;
- Неспростовність отримувача. Відправнику даних надається підтвердження доставки даних. Це захистить від будь-якої подальшої спроби одержувача помилково спростовувати факт отримання даних або їх вмісту.

1.3 Механізми безпеки

Механізми безпеки є способами для реалізації вище наданих сервісів безпеки. Фактично, кожен сервіс безпеки будується за допомогою реалізації одного чи більше механізму безпеки для виконання своїх функції. Нижче наведені механізми безпеки надані у стандарті X.200.

1.3.1 Шифрування

Шифрування забезпечує конфіденційність даних або інформації про потік трафіку і може відігравати роль або доповнити ряд інших механізмів безпеки.

Алгоритми шифрування можуть бути оборотні або незворотні. Є дві загальні класифікації оборотних алгоритмів шифрування:

а) симетричне (тобто секретний ключ) шифрування, у якому знання ключа шифрування означає знання ключа розшифровки і навпаки;

б) асиметричне (відкритий ключ) шифрування, у якому знання ключа шифрування не означає знання ключа розшифровки, або навпаки.

Іноді два ключі таких систем називають "відкритий ключ" та "секретнимключ".

Незворотні алгоритми шифрування можуть або використовувати або не використовувати ключ. Якщо вони використовують ключ, цей ключ може бути відкритим або секретним. Наявність механізму шифрування передбачає використання механізму управління ключами, окрім випадків деяких незворотних алгоритмів шифрування.

1.3.2 Цифровий підпис

Цей механізм визначає дві процедури:

а) підписання одиниці даних;

б) перевірка підписаного блоку даних.

Перший процес використовує інформацію, яка є секретною (тобто унікальною та конфіденційною) для підписанта. Другий процес використовує процедури та інформацію, які є загальнодоступними, але з яких не може бути отримана приватна інформація підписувача.

Процес підписання передбачає або шифрування одиниці даних, або виготовлення криптографічного контрольного значення одиниці даних, використовуючи приватну інформацію підписанта як приватний ключ.

Процес верифікації передбачає використання публічних процедур та інформації для визначення того, чи був підпис зроблений з приватною інформацією підписувача.

Основна характеристика механізму підпису полягає в тому, що підпис може бути зроблений лише за допомогою приватної інформації підписувача. Таким чином, коли підпис перевіряється, згодом може бути доведено третій особі (наприклад, судді чи арбітражу) в будь-який час, що тільки унікальний власник приватної інформації міг зробити підпис

1.3.3 Управління доступом

Цей механізм використовує автентифіковану сутність або інформацію про сутність (наприклад, належність сутності до наперед визначеної групи сутностей) або можливості сутності, щоб визначити та забезпечити правами доступу дану сутність. Якщо сутність намагається використати несанкціонований ресурс або авторизований ресурс з неправильним типом доступу, функція контролю доступу відхилить спробу і може додатково повідомити про інцидент для створення сигналу тривоги та / або запису його як частини аудиту безпеки. Будь-яке повідомлення відправника про відмову в доступі для передачі даних без встановлення зв’язку може бути надано лише в результаті контролю доступу, накладеного на джерело.

Механізми контролю доступу можуть базуватися на використанні однієї або декількох з наступних властивостей:

а) Інформаційні бази контролю доступу, де зберігаються права доступу однорангових організацій. Ця інформація може підтримуватися центрами авторизації або суб'єктом, до якого звертаються, і може бути у формі списку контролю доступу або

матриці ієрархічної або розподіленої структури. Це припускає, що автентифікація запитувача була здійснена;

b) Відомості про автентифікацію, такі як паролі, володіння та подальше подання, які є свідченням дозволу сутності, якій надається доступ;

c) Можливості, володіння та подальше подання яких є свідченням права на доступ сутності або ресурсу, визначеного функціональним можливостями.

d) Мітки безпеки, які, пов'язані з організацією, можуть використовуватися для надання або заборони доступу, як правило, відповідно до політики безпеки.

д) Час спроби доступу.

f) Маршрут спроби доступу,

g) Тривалість доступу.

Механізми контролю доступу можуть застосовуватися на будь-якому кінці зв'язку та / або в будь-якій проміжній точці. Контроль доступу, що бере участь у початковому або будь-якому проміжному пункті, використовується для визначення того, чи має право відправник спілкуватися з одержувачем та / або використовувати необхідні комунікаційні ресурси. Вимоги механізмів контролю кінці отримувача при передачі даних без постійного зв’язку повинні бути апріорно відомі на початку та повинні бути записані в інформаційній базі управління безпекою.

1.3.4 Цілісність даних

Два аспекти цілісності даних: цілісність одного блоку даних або поля; і цілісність потоку даних або полів. Загалом, для забезпечення цих двох видів цілісності використовують різні механізми, хоча надання другого без першого не є практичним.

Визначення цілісності одного блоку даних передбачає два процеси: один у об'єкта, що надсилає, і один у приймаючої особи. Суб'єкт, що відправляє, додає до одиниці даних кількість, яка є функцією самих даних. Ця кількість може бути додатковою інформацією, такою як код перевірки блоку або криптографічне значення і саме по собі може бути

зашифрованним. Суб'єкт, що приймає, генерує відповідну кількість і порівнює його з отриманою кількістю, щоб визначити, чи були дані змінені під час транзиту. Сам цей механізм не захистить від повторного використання одного блоку даних. У відповідних шарах архітектури виявлення маніпуляцій може призвести до відновлення (наприклад, за допомогою повторної передачі або виправлення помилок) на тому ж шарі або вище.

Для передачі даних в режимі з'єднання захист цілісності послідовності одиниць даних (тобто захист від неправильного набору, втрати, відтворення та вставки або зміни даних) вимагає додатково певної форми явного впорядкування, такого як нумерація послідовностей, штампування часу або криптографічний ланцюжок. Для безперервної передачі даних, тимчасове штампування може використовуватися для забезпечення обмеженої форми захисту від відтворення окремих одиниць даних.

1.3.5 Обмін автентифікацією

Деякі з методів, які можуть бути застосовані для обміну автентифікацією, є:

а) використання відомостей про автентифікацію, таких як паролі, що надаються особою, що відправляє, і перевіреною особою, що приймає;

б) криптографічні прийоми;

в) використання характеристик та / або володінь сутності.

Механізми можуть бути включені в (N) - шар, щоб забезпечити автентифікацію сутності отримувача. Якщо механізму не вдасться встановити автентифікацію об'єкта, це призведе до відхилення або припинення з'єднання, а також може створити запис у логах аудиту безпеки та / або звіт до центру управління безпекою.

Коли використовуються криптографічні методи, їх можна поєднувати з протоколами "рукостискання" для захисту від повторного відтворення (тобто для забезпечення життєдіяльності).

Вибір методів обміну автентифікацією буде залежати від обставин, в яких вони будуть використовуватися з:

а) штампуванням часу та синхронізацією годинників;

б) дво- і тристоронні рукостискання (для односторонньої та взаємної автентифікації відповідно);

в) послуги, що не відмовляються, досягнуті за допомогою цифрового підпису та / або механізмів нотаріального посвідчення.

1.3.6 Доповнення трафіку

Механізми доповнення трафіку можуть використовуватися для забезпечення різних рівнів захисту від аналізу трафіку. Цей механізм може бути ефективним лише в тому випадку, якщо доповнення трафіку захищено службою конфіденційності.

1.3.7 Контроль маршрутизації

Маршрути можна вибирати як динамічно, так і заздалегідь, щоб використовувати лише фізично захищені підмережі, реле або посилання.

Кінцеві системи можуть, виявляючи постійні атаки, вирішити доручити постачальнику послуг мережі встановити з'єднання за іншим маршрутом.

Дані, що містять певні мітки безпеки, можуть бути заборонені політикою безпеки проходити через певні підмережі, реле або посилання. Також ініціатор з'єднання (або відправник блоку даних без постійного зв’язку) може вказати попередження про маршрутизацію, які вимагають уникати конкретних підмереж, посилань чи ретрансляцій.

1.3.8 Нотаріальне посвідчення

Властивості даних, що передаються між двома або більше об'єктами, такі як їх цілісність, походження, час та місце призначення, можуть бути забезпечені механізмом нотаріального посвідчення. Запевнення надається стороннім нотаріусом, якому довіряють підприємства, які обмінюються даними, і який зберігає необхідну інформацію

для надання необхідної гарантії в доказовому порядку. Кожен екземпляр зв'язку може використовувати механізми цифрового підпису, шифрування та цілісності відповідно до послуги, що надається нотаріусом. Коли такий механізм нотаріального посвідчення викликається, дані передаються між об'єктами, які обмінюються даними, через захищені лінії зв'язку та нотаріуса.

1.4 Зв’язок між сервісами, рівнями та механізмами безпеки

Як було зазначено раніше, сервіси безпеки реалізуються за допомогою механізмів безпеки. Нижче у таблиці 1.1 наведено співвідношення між сервісами та механізмами, за допомогою яких ці сервіси можуть бути реалізовані.

Таблиця 1.1

Зв’язок між механізмами та сервісами безпеки

Сервіс	Механізм							
	Шифрування	Цифровий підпис	Управління доступу	Цілісність даних	Обмін автентифікацією	Доповнення трафіку	Контроль маршрутизації	Нотаріальне посвідчення
Автентифікація отримувача	+	+			+			
Автентифікація джерела походження даних	+	+						
Управління доступом			+					
Конфіденційність з'єднання	+						+	
Конфіденційність без встановлення зв’язку	+						+	
Вибірна конфіденційність поля	+							

Конфіденційність потоку трафіку	+					+	+	
Цілісність з'єднання з відновленням	+			+				
Цілісність з'єднання без відновлення	+			+				
Вибіркова цілісність з'єднання поля	+			+				
Цілісність без встановлення зв’язку	+	+		+				
Вибіркова цілісність поля без встановлення зв’язку	+	+		+				
Неспростовність джерела		+		+				+
Неспростовність отримувача		+		+				+

При організації розподілу сервісів безпеки за рівнями і подальшого розміщення на цих рівнях механізмів безпеки використовуються викладені нижче принципи:

a) число альтернативних способів виконання сервісів слід зводити до мінімуму;

b) допустимо будувати захищену систему, забезпечуючи послуги безпеки на декількох рівнях;

c) додаткова функціональність, необхідна для забезпечення безпеки, не повинна без необхідності дублювати існуючі функції OSI;

d) слід не допускати порушення незалежності рівнів;

e) обсяг довіреної функціональності слід зводити до мінімуму;

f) якщо об'єкт залежить від механізму безпеки, що забезпечується об'єктом нижнього рівня, будь-які проміжні рівні слід створювати таким чином, щоб порушення безпеки було неможливим;

g) по можливості додаткові функції безпеки рівня слід визначати таким чином, щоб не перешкоджати реалізації у вигляді автономного модуля.

Відповідно до вищезазначеного, у стандарті X.800 сервіси безпеки розподілені по рівням згідно таблиці 1.2.

Таблиця 1.2

Зв’язок між сервісами безпеки та рівнями моделі OSI

Сервіс	Рівень						
	Фізичний	Канальний	Мережевий	Транспортний	Сеансовий	Представницький	Прикладний
Автентифікація отримувача			+	+			+
Автентифікація джерела походження даних			+	+			+
Управління доступом			+	+			+
Конфіденційність з'єднання	+	+	+	+		+	+
Конфіденційність без встановлення зв’язку		+	+	+		+	+
Вибірна конфіденційність поля	+					+	+
Конфіденційність потоку трафіку			+				+
Цілісність з'єднання з відновленням				+			+
Цілісність з'єднання без відновлення			+	+			+
Вибіркова цілісність з'єднання поля							+
Цілісність без встановлення зв’язку		+	+	+			+
Вибіркова цілісність поля без встановлення зв’язку		+					+
Неспростовність джерела		+					+
Неспростовність отримувача		+					+

1.5 Клієнт-серверна архітектура

Тепер, коли ми розглянули специфікацію сервісів та механізмів безпеки за рекомендацією X.800, варто розглянути саму модель взаємодії веб-застосунку, яка зветься клієнт-серверною. Клієнт-серверну архітектуру можна визначити, як концепцію інформаційно-комунікаційної мережі в якій основна частина ресурсів зосереджена в

сервері чи серверах, які обслуговують клієнтів. Така архітектура визначає наступні типи компонентів:

- набір серверів, які надають інформацію або інші послуги у відповідь на запити, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами. [17]

На рисунку 1.2 схематично представлено клієнт-серверну архітектуру.

З рисунку можна одразу побачити, що плюсом даної взаємодії є незалежність між платформою клієнта та сервером. Клієнтом може бути як мобільний додаток так і веб-сторінка так і будь яке інше програмне забезпечення, яке може здійснювати запити та бути авторизоване на сервері, наприклад, розумний холодильник.

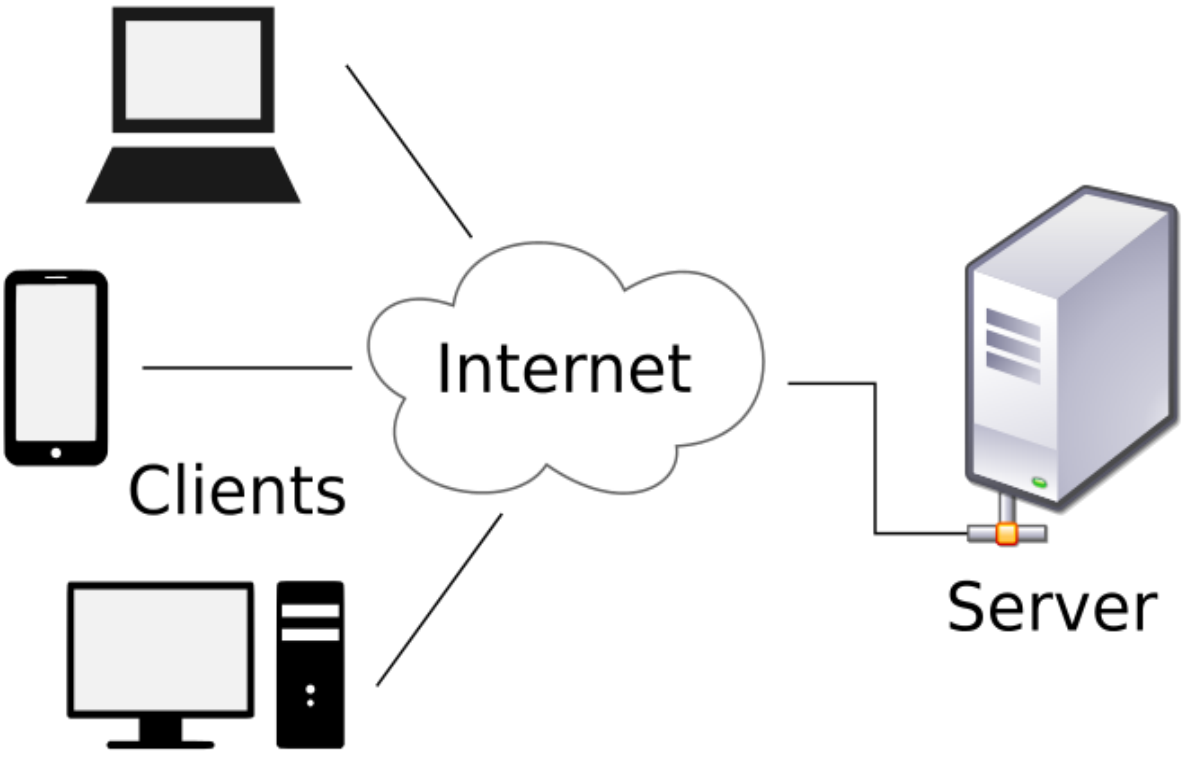


Рисунок 1.2 - Схематичне зображення клієнт-серверної архітектури

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов’язків між клієнтом та сервером. Логічно можна відокремити три основні операції:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд або інших даних, такі як пошукові запити;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Також можна розрізнити два основних типи взаємодії модулів клієнт-серверної архітектури: дво- та триланкова архітектура.

Дволанкова клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп’ютери, мобільні телефони та ін.

Трьохланкова клієнт-серверна архітектура, яка почала розвиватися з середини 90 - х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка застосунку. Програми проміжного рівня можуть функціювати під управлінням спеціальних серверів застосунків, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Управління даними здійснюється сервером даних. [18]

Дволанкова архітектура простіша, так як всі запити обслуговуються одним сервером, але саме через це вона менш надійна і висуває підвищені вимоги до продуктивності сервера.

Триланкового архітектура складніша, але завдяки тому, що функції розподілені між серверами другого і третього рівня, ця архітектура проявляє:

- високий ступінь гнучкості і масштабованості;
- високу безпеку (тому що захист можна визначити для кожного сервісу або рівня);
- високу продуктивність (тому що завдання розподілені між серверами).

Сьогодні також стає популярною архітектура, яка пішла далі ніж триланкова клієнт-серверна архітектура – мікросервісна архітектура. Даний архітектурний підхід являє собою велику кількість сервісів, які виконують якусь конкретну частину логіки обробки даних веб-застосунку, комунікують між собою та мають свої незалежні ресурси такі як, наприклад, бази даних. Ці сервіси за філософією даної архітектури мають бути достатньо невеликими, щоб їх можна було швидко додати до множини інших сервісів, а також відповідали за якусь одну єдину функцію. [19] Звичайно це несе у собі і багато проблем, які необхідно вирішувати, одним із самих складних із яких є складність організації комунікацій між сервісами, що призводить до так званого у колі веб-розробників «Пекла мікросервісів».

1.6 Постановка завдання

З вищеописаного можна побачити, що стандарт надає тільки специфікацію тому, як слід будувати сервіси безпеки та з яких механізмів вони мають бути складені і на якому рівні. Стандарт не надає конкретної реалізації цих сервісів чи механізмів безпеки, ця задача стоїть перед розробниками цих сервісів та тими, хто буде їх імплементувати.

Однак, це є гарною стороною даного стандарту. Через те, що він надає тільки специфікацію, а не конкретну реалізацію він є актуальним навіть через 29 років після

його опублікування, так як мережева модель OSI все ще є актуальною. Для реалізації цих сервісів та механізмів на реальних сучасних системах було створено багато різних підходів до вирішення цілей, поставлених перед насамперед механізмами безпеки, які, в свою чергу, реалізують сервіси безпеки. Звичайно, не існує ідеальної реалізації того чи іншого механізму безпеки, всі вони мають свої гарні сторони та недоліки.

Відповідно до твердження, що механізми безпеки реалізують сервіси безпеки, тобто фактично сервіс безпеки це набір взаємодіючих механізмів безпеки, я вважаю доцільним детально розглядати саме механізми безпеки, адже вони є структурними блоками сервісів. Також, як видно з таблиці 1.2, сервіси, а отже і відповідні їм механізми можуть бути всі реалізовані на прикладному рівні еталонної мережевої моделі OSI. Також в рекомендації X.800 є примітка, яка нам каже про те, що на прикладному рівні процеси можуть самі забезпечувати сервіси безпеки.

Так як ми розглядаємо сервіси і механізми безпеки для використання у веб-застосунках, то задача реалізації сервісу безпеки, тобто вибору існуючого механізму захисту, який необхідний даному сервісу, чи, рідше, створення нової реалізації механізму, є задачею покладеною на веб-розробника цього веб-застосунку чи команду таких розробників.

Зачасту в веб-застосунку сервіси безпеки надаються не на однаковому рівні складності і отже гарантії, тому розробникам необхідно визначити:

- Сформулювати набір вимог до сервісів та механізмів безпеки, які застосовуються для забезпечення захисту;
- Визначити технології для реалізації сервісів та механізмів безпеки для застосування у веб-застосунках;
- Провести аналіз процесу роботи обраних реалізацій механізмів безпеки;
- Дослідити сфери використання обраних механізмів захисту у веб-застосунках;
- Створити захищений веб-застосунок.

Виходячи з вище описаного, в другому розділі роботи буде розглянуто різні існуючи способи реалізації того чи іншого механізму безпеки та проаналізовано їх сфера використання, недоліки та складність реалізації.

Висновки за розділом 1

Для реалізації сервісів та механізмів у веб-застосунках необхідно розуміти специфікацію цих сервісів, як відбувається клієнт-серверна взаємодія у веб-застосунках, що і було розглянуто в даному розділі. Власне в даному розділі були розглянуті наступні питання:

- Основні стандарти, які регулюють сервіси та механізми безпеки;
- Еталонна мережева модель OSI;
- Специфікація цих сервісів та механізмів;
- Взаємозв’язок між рівнями OSI, сервісами та механізмами безпеки;
- Визначення та особливості клієнт-серверної архітектури.

Було визначено поняття сервісу та механізму безпеки, веб-застосунку та нормативно-правові акти, а точніше рекомендації та специфікації, які регулюють дане питання. Даними актами є рекомендація ITU-T X.800, ITU-T X.200, ISO 7498-2.

Було описано еталонну модель мережевої взаємодії відкритих систем OSI, її рівні та особливості міжрівневої взаємодії. Можна зробити висновок, що хоча модель і є абстрактною і практично не реалізуємою, вона слугує основним еталоном для поділу мережевого функціоналу та відповідальностей рівнів, а також уніфікує роботу мережевої взаємодії під єдиний формат, що робить розробку протоколів взаємодії простіше.

Були розглянуті основні сервіси безпеки та описано специфікацію їх обов’язки і особливості роботи. Були розглянуті наступні сервіси безпеки:

- Автентифікація отримувача;
- Автентифікація джерела походження даних;
- Управління доступом;
- Конфіденційність з'єднання;
- Конфіденційність без встановлення зв’язку;
- Вибірна конфіденційність поля;
- Конфіденційність потоку трафіку;
- Цілісність з'єднання з відновленням;
- Цілісність з'єднання без відновлення;
- Вибіркова цілісність з'єднання поля;
- Вибіркова цілісність поля без встановлення зв’язку;
- Неспростовність джерела;
- Неспростовність отримувача.

Були розглянуті основні механізми безпеки, їх роль у сервісах безпеки та короткий опис функціям, які вони мають забезпечувати. Серед механізмів безпеки були виділені та описані наступні: шифрування, цифровий підпис, управління доступом, цілісність даних, обмін автентифікацією, доповнення трафіку, контроль маршрутизації, нотаріальне посвідчення.

Після специфікацій був розглянутий взаємозв’язок між сервісами та механізмами безпеки та рівнями моделі OSI, на яких вони можуть бути реалізовані. В цьому ж розділі розглянуто принципи, за якими сервіси були поділені на рівні.

Було розглянуто клієнт-серверну архітектуру, яка використовується у веб-застосунках, їх підвиди, складові, особливості, плюси та недоліки. Також було згадано про мікросервісну архітектуру.

В кінці розділу було визначено основні проблеми с якими стикається веб-розробник при імплементації сервісів безпеки у веб-застосунок та приведено

приблизний план інформації, якою необхідно володіти, щоб визначити реалізацію певного механізму безпеки.

Таким чином, в даному розділі була приведена специфікація сервісів безпеки та механізмів, які вони включають, для подальшого аналізу реалізацій механізмів безпеки для використання у веб застосунках.

РОЗДІЛ 2
АНАЛІЗ ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ МЕХАНІЗМІВ БЕЗПЕКИ

2.1 Аналіз механізмів шифрування

Для взаємодії між клієнтом та сервером використовуються дані, які можуть містити конфіденційні дані, а також для більшості застосунків характерно мати прив’язану базу даних, в якій зберігаються дані користувача. Проте, ми не будемо глибоко занурюватися в питання захисту баз даних в даній роботі, але розглянемо як можна захистити дані ще до збереження в базу даних.

Як було згадано в попередньому розділі, шифрування це, в кратці, таке математичне перетворення, яке перешкоджає осмисленому сприйняттю інформації, яка була отримана в результаті. Такий механізм захисту сприяє тому, що при перехваті або іншому несанкціонованому отриманні даних, зловмисник отримає тільки набір незрозумілих та, в ідеалі, незв’язних символів. Таким чином, тільки санкціоновані користувачі, які мають ключ розшифрування або функцію генерації ключа і саму функцію шифрування зможуть отримати осмислену інформацію з отриманого набору символів.

Це стає можливим за допомогою використання криптографічних алгоритмів, як одно- так і двосторонніми. Серед двосторонніх алгоритмів виділяють симетричні або алгоритми з закритим ключем та асиметричні або алгоритми з відкритим ключем. Обидва типи шифрування мають своє місце в веб-застосунках, проте, забігаючи наперед, в даній главі будуть розглянуті тільки симетричні та односторонні алгоритми шифрування, асиметричні алгоритми будуть розглянуті при аналізі механізму цифрового підпису.

2.1.1 Односторонні алгоритми шифрування

Односторонні алгоритми шифрування це такі алгоритми шифрування, які використовують одностороннюю функцію для шифрування вхідних даних і зашифрований текст таким чином є не придатним для зворотного розшифрування в відкритий текст. Найбільш використовуваною у веб-застосунках односторонньою функцією шифрування є хешування або хеш-функцією.

Хеш-функція - це будь-яка одностороння функція, яка використовується для перетворення даних довільного розміру або довжини у фіксований. Значення, які повертає хеш-функція, називають хеш-значення, хеш-код, дайджестом або просто хешами. Значення використовуються для індексації таблиці фіксованого розміру, яка називається хеш-таблицею.

Хеш-функції та пов’язані з ними хеш-таблиці використовують в веб-застосунках для зберігання та пошуку даних, доступу до даних у малий та майже постійний час для кожного звернення. Мінусом такого швидкого звернення є те, що дані займають більше простору ніж їх просте зберігання, проте таке збільшення не є великим

З хеш-функціями пов'язані контрольні суми, контрольні значення, відбитки пальців, функції рандомізації, коди виправлення помилок. Хоча поняття певною мірою перетинаються, кожна з пов’язаних функцій має власні вимоги та була розроблена та оптимізована по-різному.

В веб-застосунках в розрізі шифрування одним із основних напрямків застосування хеш-функцій є збереження даних для автентифікації користувача, такі як, наприклад, пари логін-пароль. Механізм автентифікації у такому виді наступний: користувач вводить свої дані автентифікації, веб-застосунок отримує їх, обчислює значення хеш функції отриманих даних та порівнює цей хеш з даними у базі і базуючись на цьому результаті порівняння дає або не дає доступ користувачеві. Таким чином, веб-застосунок зберігає зашифровані дані у базі, які неможливо розшифрувати.

Використання хеш-функцій для даних, які не є відображуваними для користувача, є швидким та захищеним спосіб уберегти дані користувача від витоку.

Щодо реалізацій алгоритмів, то зараз широко використовуються два стандартизовані: SHA-2 та SHA-3. Обидва алгоритми є сімейства SHA проте використовують різні криптографічні алгоритми для отримання результату. Перший використовує хеш-функцію Меркла-Демґарда та односторонню функцію Девіса-Мейєра, в той час як другий – хеш-функцію «губки».

Хоча SHA-3 і є новим алгоритмом, в порівнянні з SHA-2 він містить ряд плюсів та недоліків:

- Стійкість до атак. Ранні алгоритми серії SHA-2 мають вразливості до атак знаходження колізій, тобто різних повідомлень з однаковим хешом та знаходження прообразу, тобто невідомого повідомлення за його хешом. [20] Від стійкості геш-функції до знаходження колізій залежить безпека електронного цифрового підпису з використанням цього геш-алгоритму. Від стійкості до знаходження прообразу залежить безпека зберігання гешів паролів для автентифікації;
- Продуктивність. Функції SHA-2, особливо SHA-512, SHA-512/224 і SHA-512/256, як правило, мають більш високу продуктивність, ніж функції SHA-3. Частково це було через ряд проблем в процесі проектування SHA-3. На базі конкурсної версії SHA-3, яка була набагато швидше з такими же вимогами безпеки, було створено такі алгоритми як BLAKE2 та KangarooTwelve, які теж використовуються;
- Квантова стійкість до атак. Було показано, що хеш-функція Меркла-Демґарда, яку використовує SHA-2, руйнується і, як наслідок, має стійкість до квантових атак знаходження колізій [21], але для хеш-функцій «губки», яку використовує SHA-3, автори надають докази лише для випадок, коли функція блоку f не є повністю необоротною. [22]

2.1.2 Симетричні алгоритми шифрування

Варто зауважити, що хоча існує два види симетричних алгоритмів шифрування: потокові та блокові, в даному розділі будуть розглянуті тільки блокові алгоритми через те, що потокові шифри майже не використовуються в сучасних веб-застосунках. Блокові алгоритми є двосторонніми, тобто за допомогою математичних перетворень відкритий текст може бути перетворений у шифротекст, а цей шифротекст, у свою чергу, в відкритий текст. Блокові алгоритми є симетричними, тобто ключ для зашифрування повідомлення і розшифрування є однаковим і через це має бути секретним. Блоковий шифр, як випливає з його назви, це алгоритм, що використовує групи бітів фіксованої довжини, що називаються блоками. Основне використання даних алгоритмів – збереження даних, які потребують захисту, проте можуть бути відтворені до попереднього стану за потреби, на відміну від хешування, де це неможливо.

На сьогодні популярними алгоритмами блокового шифрування є:

- .Потрійний DES. Був розроблений для заміни оригінального алгоритму стандарту шифрування даних (DES), який був достатньо вразливим до атак. Свого часу Triple DES був рекомендованим стандартом і найбільш широко використовуваним симетричним алгоритмом в галузі. Незважаючи на те, що алгоритм поступово виходить з широко використання, Triple DES все-таки зарекомендував себе як надійне рішення для шифрування апаратних засобів для фінансових послуг та інших галузей;
- Blowfish. Ще один алгоритм, призначений замінити DES. Цей симетричний шифр розбиває повідомлення на блоки з 64 біт і шифрує їх окремо. Відомий як своєю величезною швидкістю роботи, так і загальною ефективністю. Також алгоритм є у вільному доступі, що також є плюсом. Blowfish використовується в програмному забезпеченні, починаючи від платформ електронної комерції для забезпечення платежів закінчуючи інструментами управління паролями. Це, безумовно, один із найбільш гнучких доступних методів шифрування;

- Twofish. Експерт із комп'ютерної безпеки Брюс Шнайер є розробником Blowfish та його наступника Twofish. Ключі, які використовуються в цьому алгоритмі, можуть мати довжину до 256 біт з одним ключем. Twofish вважається одним з найшвидших у своєму роді, і ідеально підходить для використання як в апаратному, так і в програмному середовищі. Як і Blowfish, Twofish є у вільному доступі. Він використовується в програмах шифрування, таких як PhotoEncrypt, GPG та популярному програмному забезпеченні з відкритим кодом TrueCrypt;
- AES. Розширений стандарт шифрування (AES) - це алгоритм, який використовується урядом США та численними організаціями. Незважаючи на те, що він є надзвичайно ефективним у 128-бітному вигляді, AES також використовує ключі 192 та 256 біт для шифрування. AES багато в чому вважається непроникним для всіх атак, за винятком грубої сили.

В даній роботі розглянемо алгоритм Blowfish. Алгоритм був розроблений в 1993 Брюсом Шнайдером та на той час був одним із небагатьох стійких шифрів, які не були запатентовані або засекречені. Алгоритм виділявся швидкістю, гнучкістю, компактністю та простотою реалізації.

Серж Воденє вказав на наявність невеликого класу слабких ключів (генеруючих слабкі S-блоки). [23] Імовірність появи слабкого S-блоку дорівнює 2^{-15}. Неможливо заздалегідь визначити, чи є ключ слабким. Перевірити можна тільки після генерації.

Крипостійкість алгоритму можна налаштовувати за рахунок зміни кількості раундів шифрування (збільшуючи довжину масиву ключів P) і кількості використовуваних S-блоків. При зменшенні використовуваних S-блоків зростає ймовірність появи слабких ключів, але зменшується використовувана пам'ять.

Використання в Blowfish 64-бітного блоку (на відміну, наприклад, від 128-бітного блоку AES) робить його вразливим для атаки днів народження, зокрема, в контекстах типу HTTPS. У 2016 році атака SWEET32 продемонструвала, як використовувати атаку днів народження для відновлення відкритого тексту (тобто розшифровки) з 64-бітових

блоків. [24] Проект GnuPG рекомендує не використовувати Blowfish для файлів з розміром, що перевищує 4 ГБ через малий розміру блоку. [25]

Тим не менш, Blowfish зарекомендував себе як надійний алгоритм, тому реалізований у багатьох програмах, де не потрібна часта зміна ключів і необхідна висока швидкість шифрування / розшифрування. Зокрема сфери використання включають:

- хешування паролів;
- захист електронної пошти та файлів (GnuPG);
- в лініях зв'язку: (маршрутизатори Intel Express 8100 з ключем довжиною 144 біти);
- забезпечення безпеки в протоколах мережного і транспортного рівня(SSH та OpenVPN).

2.2 Аналіз механізмів цифрового підпису

Механізм цифрового підпису – це математична схема перевірки автентичності цифрового повідомлення або документу. [26] Дійсний цифровий підпис, з всіма виконаними передумовами, дає одержувачу дуже вагомі підстави вважати, що повідомлення було створене відомим відправником (автентифікація), і що повідомлення не було змінено при передачі (цілісність).

Цифрові підписи є стандартним елементом більшості наборів криптографічних протоколів і зазвичай використовуються для розповсюдження програмного забезпечення, фінансових транзакцій та інших випадках, коли важливо виявити підробку.

Цифрові підписи використовують асиметричну криптографію. У багатьох випадках вони забезпечують рівень перевірки та захист повідомлень, що надсилаються через незахищений канал. Правильно реалізований цифровий підпис дає одержувачу підстави вважати, що повідомлення було надіслано саме заявленим відправником. Цифрові підписи багато в чому еквівалентні традиційним рукописним підписам, але

правильно виконані цифрові підписи важче підробити, ніж рукописні. Цифрові підписи також можуть забезпечити неспростовність, тобто відправник не може успішно стверджувати, що не підписав повідомлення, і в той же час заявляти, що їх приватний ключ залишається таємним. На рисунку 2.2 зображено загальну схему роботи цифрового підпису.

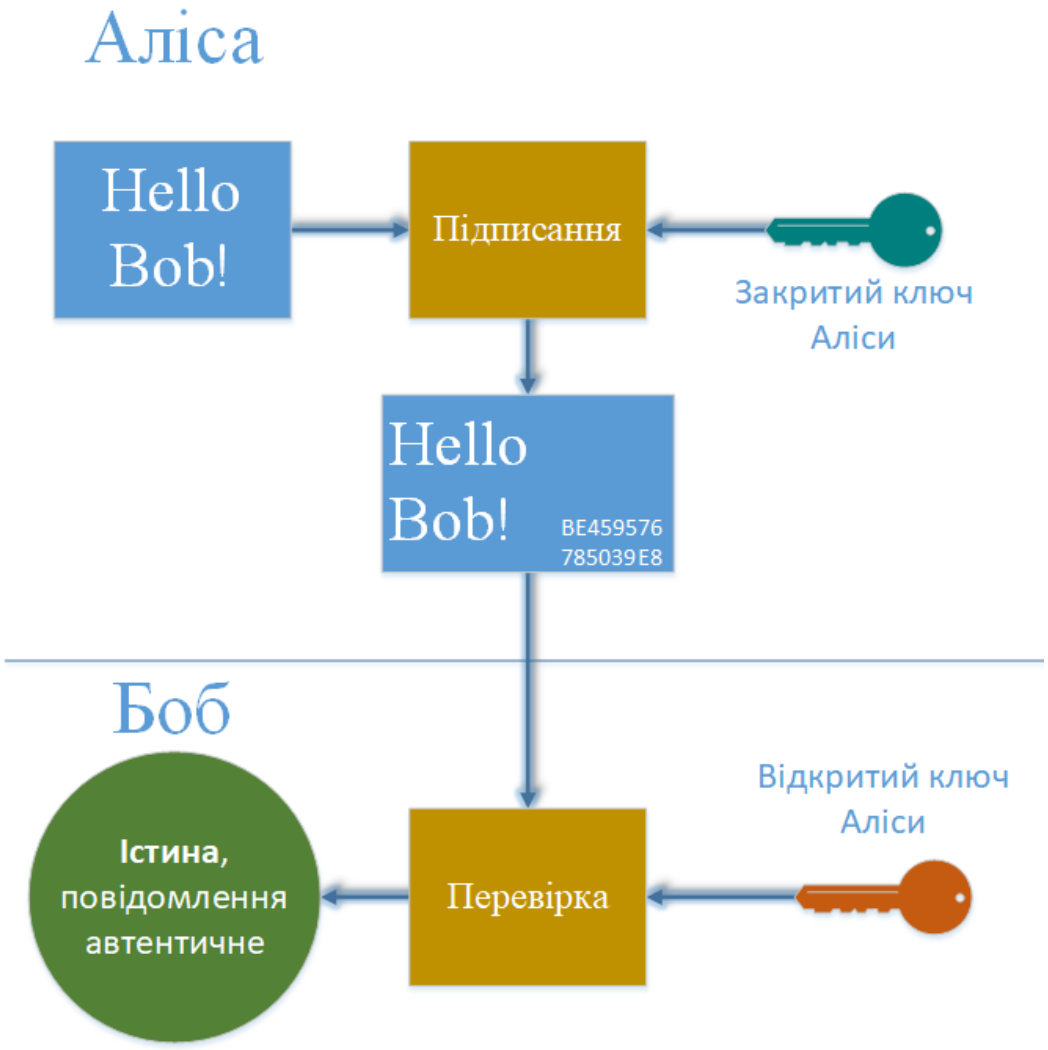


Рисунок 2.1 - Загальна схема роботи механізму цифрового підпису

Прикладами алгоритмів цифрового підпису є:

- Схеми підписів на основі RSA, такі як RSA-PSS;
- DSA та його варіант на еліптичних кривих ECDSA;

- Алгоритм цифрового підпису на кривих Едвардса та його варіант Ed25519;
- Схема підпису Ель-Гамаля, як попередник DSA, а також варіанти підпису Schnorr та алгоритм підпису Pointcheval – Stern;
- Алгоритм підпису Рабіна;
- Схеми на основі пар, такі як BLS.

Найбільш вживаними є алгоритми RSA, DSA та ECDSA. Якщо RSA може бути застосованим як для шифрування так і для підпису, то DSA і його аналог на еліптичних кривих може бути застосований тільки для механізму цифрового підпису через особливості його роботи. Алгоритм DSA для формування підпису використовує хеш-функції, тобто фактично шифрується і перевіряється не саме повідомлення, а його хеш, тобто зворотнє перетворення на відкритий текст за допомогою цього алгоритму – неможливе. Фактично, складність алгоритму визначає хеш-функція, тому основною проблемою є вибір алгоритму з мінімальними колізіями, інакше може виникнути ситуація, коли підпис буде валідним для декількох повідомлень з однаковим хешом. Перша версія стандарту описана у FIPS-186 використовувала хеш-функцію сімейства SHA-1, остання – SHA-2 (FIPS-186-4). Використання SHA-3 поки що не стандартизоване.

Цифрові підписи часто використовуються для впровадження електронних цифрових підписів (ЕЦП), які включають будь-які електронні дані, що зазначають намір підпису, але не всі ЕЦП використовують цифрові підписи. У деяких країнах, включаючи Канаду, Південну Африку, США, Алжир, Туреччину, Індію, Бразилію, Індонезію, Мексику, Саудівську Аравію, Уругвай, Швейцарію , Чилі та країнах Європейського Союзу, ЕЦП мають юридичне значення.

В Україні діяльність ЕЦП визначена Законом України «Про електронні довірчі послуги». [29] Серед недоліків Закону є те, що він не має опису, щодо особливості застосування ЕЦП, до документів, термін дії яких перевищує термін дії ЕЦП. Також не визначено статус підписаних документів, термін дії яких не закінчився, у разі компрометації ЕЦП. Це дозволяє реалізувати два види атак на ЕЦП:

- Використання недійсного ЕЦП (скомпрометованого, або ЕЦП, термін дії якого закінчився) для підпису документів попередньою датою;
- Визнання підписаного документа без позначки часу, сертифікат якого на час перевірки підпису не діє, є недійсним на підставі того, що неможливо встановити чи був документ підписаний дійсним ЕЦП, чи недійсним ЕЦП.

2.3 Аналіз механізмів управління доступом

В даному випадку, управляння доступом можна визначити як механізм перевірки наявності у користувача деякого атрибуту або належності користувача до певної групи, що дозволяє або забороняє йому отримати доступ до ресурсу. Фактично доступ до об’єкту може приймати один із таких дозволів як читання, запис або модифікація, видалення або їх комбінацію. Також, будь-яка система управління доступом має суперкористувача, який може встановлювати права доступу до об’єктів для користувачів або групи користувачів.

Серед часто використовуваних способів реалізації механізму управління доступом у веб-застосунках можна виділити два:

- Модель управління доступом на базі ролей (RBAC);
- Модель управління доступом на базі атрибутів (ABAC).

Для доступу до файлових сховищ також може бути використана модель управління доступом на базі списків контролю доступу (ACL) проте даний механізм є застарілим і не є дуже зручним для користування крім випадків доступу до файлів з обмеженим доступом.

Як RBAC так і ABAC користуються поняттям дозволу як фундаментального будівельного блоку в моделях. Проте те як організована агрегація цих дозволів має різний вигляд у цих моделях. Для початку розглянемо RBAC.

Модель управління на основі ролей, яку використовують і по сьогодні, була розроблена у 1992 році Ферайоло і Куном, допрацьована у 2000 році та стандартизована у 2004 році Американським національним інститутом стандартів та Міжнародним

комітетом по стандартам інформаційних технологій (ANSI/INCITS) в стандарті ANSI/INCITS 359-2004. RBAC намагається вирішити проблему кількох облікових записів користувачів з динамічним рівнем доступу, які можуть призвести до нестабільності ключа шифрування, що дозволяє несанкціонованому користувачеві використовувати слабкість для отримання доступу. [30]

Для RBAC визначено три основні правила:

- Присвоєння ролі: Суб'єкт може використовувати дозвіл лише у тому випадку, якщо суб'єкт був обраний або йому призначено роль;
- Авторизація ролей: Активна роль суб'єкта повинна бути авторизована. Беручи до уваги перше правило, це гарантує, що користувачі можуть виконувати лише ті ролі, на які вони мають право;
- Дозвіл авторизацію: Суб'єкт може використовувати дозвіл, лише якщо дозвіл авторизований для активної ролі суб'єкта. Об’єднуючи з правилом 1 та 2 це правило гарантує, що користувачі можуть здійснювати лише ті дозволи, які були авторизовані.

Також можуть застосовуватися додаткові обмеження, і ролі можуть поєднуватися в ієрархії, коли ролі вищого рівня містять дозволи, що належать підролям. Ролі призначаються суб'єктам, внаслідок чого суб'єкти отримують ті чи інші дозволи через ролі. RBAC вимагає саме такого призначення, а не прямого - призначення дозволів суб'єктам, інакше це призводить до складно контрольованих відносин між суб'єктами і дозволами На можливість успадкування дозволів від протилежних ролей накладається обмежувальна норма, яка дозволяє досягти належного поділу ролей. Наприклад, одній і тій же особі може бути не дозволено створити обліковий запис для когось, а потім авторизуватися під цим обліковим записом.

RBAC де факто являється стандартним механізмом управління доступом через його простоту реалізації та гнучкість, хоча керування такою моделлю в застосунку з величезною кількістю користувачів є важкою задачею. RBAC широко використовується для управління призначеними для користувача привілеями в межах єдиної системи або

додатку. Список таких систем включає в себе Microsoft Active Directory, SELinux, FreeBSD, Solaris, СУБД Oracle, PostgreSQL 8.1, SAP R/3, Lotus Notes і безліч інших.

На противагу RBAC в деяких веб-застосунках, де потрібна більш точна агрегація дозволів ніж надана ролі, наприклад GitHub або Google Cloud використовується модель доступу на основі атрибутів (ABAC). [31] У попередній моделі основним структурним блоком є «роль», проте в даному випадку таким блоком є «політика». Політика - це деяке твердження, яке об'єднує атрибути, щоб виразити дозволи, які дозволені, а які заборонені. Політики в ABAC можуть бути політиками дозволу або заборони. Політика також може бути локальною або глобальною і може бути створена таким чином, що вона перевизначає інші політики Ця модель має наступну рекомендовану архітектуру:

- Пункт забезпечення виконання політики: несе відповідальність за захист програм і даних, до яких потрібно застосувати ABAC. Він перевіряє запит та формує з нього запит на авторизацію, який він надсилає пункту вирішення політики;
- Пункт вирішення політики - мозок архітектури: оцінює вхідні запити щодо політик, з якими він був налаштований. Він повертає рішення щодо дозволу/заборони. Також може використовувати пункт надання інформації про політику для отримання відсутніх метаданих;
- Пункт надання інформації про політику зв'язує пункт вирішення політики і зовнішні джерела атрибутів, наприклад LDAP або базу даних.

Як було зазначено вище, політики використовують атрибути. Такі атрибути можуть бути про що завгодно і будь-кого. Вони, як правило, поділяються на 4 різні категорії:

- Предметні атрибути: атрибути, які описують користувача, який намагається отримати доступ, наприклад вік, дозвіл, відділ, роль, посада;
- Атрибути дії: атрибути, які описують спробу, що робиться, наприклад читати, видаляти, переглядати, схвалювати;

- Атрибути об'єкта: атрибути, які описують об'єкт (або ресурс), до якого отримують доступ, наприклад тип об'єкта (медична карта, банківський рахунок ...), відділення, класифікація чи чутливість, місцезнаходження;
- Контекстуальні (оточуючі) атрибути: атрибути, які стосуються часу, місця

ABAC застосовується при проектуванні архітектури систем безпеки для компаній, в яких чисельність користувачів вимірюється тисячами, відповідно кількість привілеїв в кілька разів більше у порівнянні з RBAC, багато динамічно вирішуваних запитів до системи і потужна інтеграція частин загальної інфраструктури. Це можуть бути різні комерційні організації, банки, фінансові ринки і інші фірми і державні підприємства, що вимагають оперативну підтримку в контролі доступу і мають численну базу співробітників, партнерів і клієнтів, які поділяють доступ до даних. Хоча доопрацювання рольової моделі в атрибутивну передбачає великі трудовитрати, гнучкість управління доступом в разі переходу підвищується, і в подальшому при хорошому проектуванні для зміни політик не доводиться залучати розробників. [32]

У багатьох хмарні обчислення на рівні інфраструктури як послуги (IaaS) використовується модель на основі атрибутів. Вона забезпечує безпеку сховищ даних, мереж, обчислювальних засобів і інших компонент схеми IaaS. Прикладами відомих провайдерів можуть бути Amazon Web Service, OpenStack і інші. [33]

2.4 Аналіз механізмів цілісності даних

Цілісність даних фактично означає надійність та достовірність даних протягом усього життєвого циклу. Цей механізм може описати стан даних (наприклад, валідний чи ні) або процес забезпечення та збереження валідності та точності даних. Перевірка на помилки та валідація є загальними методами забезпечення цілісності даних як частини процесу. Цей механізм в першу чергу сприяє коректності роботи системи, завдяки ньому користувач або сам веб-застосунок може використовувати точні дані, які не містять

помилок, для прийняття рішень або підтверджувати, що дані не були загублені чи пошкоджені під час передачі.

Цілісність даних може бути порушена через помилки людини або, що ще гірше, через зловмисні дії. До загальних загроз, які можуть змінити стан цілісності даних, належать:

- Людська помилка;
- Ненавмисні помилки при передачі;
- Неправильні налаштування та помилки безпеки;
- Зловмисне програмне забезпечення, інсайдерські погрози та кібератаки;
- Компрометоване обладнання.

Механізм цілісності даних може бути забезпечений такими підмеханізмами як валідація вводимих даних, валідація отриманих даних на наявність помилок та створення бекапу даних.

Валідація вводимих даних може бути представлена в вигляді валідації полів інтерфейсу веб-застосунку. Наприклад, в поле, де користувач має ввести ім’я необхідно поставити обмеження на допустимі символи, логічно, що в імені не може бути символів окрім букв (хоча з останніми новинами про ім’я сина Ілона Маска дане твердження має виняток). Також гарною ідеєю для полів, в яких користувач може вводити великий текст, проводити перевірку правопису, що також гарно впливає на точність даних.

Створення бекапу даних або інакше резервної копії даних є одним із спсобів мінімізувати втрати даних в разі їх видалення через помилку в роботі системи чи через зловмисні дії. Рекомендовано роботи резервні копії доволі часто так, щоби не перешкоджати всій роботі застосунка, проте неможливо однозначно сказати через який період часу потрібно робити бекап, це залежить від застосунку. Самі резервні копії варто зберігати не поруч з основним сховищем даних, а в якомусь іншому сховище через те, що в разі компроментації основного сховища і порушення цілісності його, адміністратори системи не втратили ще й резервні копії. Операцію створення резервної копії даних можна робити як вручну, тобто через установлений період часу оператор

вручну створює копію, або автоматично, наприклад, деякий програмний застосунок по таймеру або розкладу автоматично створює копію та завантажує її в сховище резервних копій. Це дужа гнучка система, а її впровадження мінімізує ризики втрати всіх даних.

Незважаючи на попередні два підмеханізми, мабуть найкращим прикладом механізму цілісності даних є валідація даних на наявність помилок, які могли виникнути під час передачі чи за рахунок дій зловмисника, а також спроба виправити дані помилки.

Одним із найпростіших способів перевірки цілісності даних є згадані в першій главі хеш-функції. Процес перевірки повідомлення може бути описаний наступним чином: відправник на своїй стороні обчислює хеш корисних даних та прикріплює його до повідомлення; приймач отримує дані, розділяє корисні дані та значення хешу, заново обчислює хеш корисних даних повідомлення та порівнює отриманий хеш та хеш, який був йому переданий. Якщо обидва хешу рівні, то цілісність повідомлення не була порушена, інакше – виникли помилки під час передачі чи через дії зловмисника.

Для обчислення хешу або іншими словами дайджесту повідомлення використовуються безключові хеш-функції, які в реальних системах крім особливостей звичайної хеш-функції (стиснення даних та простота обчислення дайджесту) має наступні додаткові:

- Необоротність;
- Стійкість до колізій першого роду;
- Стійкість до колізій другого роду.

Відповідно до вимог, для цього механізму використовувані хеш-функцій можна поділити за способом їх побудови:

- На блокових шифрах (алгоритм «Matyas-Meyer-Oseas», алгоритм «Davies-Meyer», алгоритм «Miyaguchi-Preneel»);
- Спеціальні. Такі алгоритми хешування, в яких акцентується швидкість, і які не мають залежності від інших компонент . (MD4, MD5, SHA-1, SHA-2);
- На модульній арифметиці (MASH-1, MASH-2).

В разі виявлення помилок система відхиляє повідомлення і відправляє повідомлення клієнту. В разі, якщо використовуються коди виправлення помилок, то система може намагається відтворити втрачені дані за допомогою таких кодів. Основна ідея полягає в тому, що відправник кодує повідомлення із зайвою інформацією у вигляді кодів виправлення помилок (ECC). Надлишок дозволяє одержувачу виявляти обмежену кількість помилок, які можуть виникнути в будь-якому місці повідомлення, і часто виправляти ці помилки без повторної передачі. Американський математик Річард Хеммінг став першим в цій галузі в 40-х роках і в 1950 році винайшов перший код для виправлення помилок: код Хеммінга (7,4). [34] Дві основні категорії кодів ECC - це блокові та згорткові коди. [35] Блокові коди працюють з блоками бітів фіксованого розміру або символів заздалегідь визначеного розміру.

Згорткові коди працюють на бітових або символьних потоках довільної довжини. Вони найчастіше декодуються за допомогою алгоритму Вітербі, хоча іноді використовуються й інші алгоритми. Декодування Вітербі дозволяє оптимізувати ефективність декодування зі збільшенням довжини обмежувального коду, але за рахунок зростаючої складності.

Існує багато типів блокових кодів, кодування Ріда-Соломона широко використовується в компакт-дисках, DVD-дисках і на жорстких дисках. Інші приклади класичних блокових кодів включають коди Golay та коди Хеммінга. Варто зауважити, що такі коди зазвичай виправляють лише помилки біт-фліпи, але не біт-вставки або бітові видалення.

2.5 Аналіз механізмів обміну автентифікацією

Мабуть першим що приходить у голову під час розмов про безпеку в веб-застосунках є автентифікація користувача. Цей механізм дозволяю перевірити чи має користувач доступ до системи за допомогою представленого їм ідентифікатора. Дане

поняття часто плутають з авторизацією та ідентифікацією. Для порівняння, нижче наведено список визначень даних термінів:

- Ідентифікація – процес надання наперед визначеного ідентифікатора користувачем (наприклад, логін);
- Автентифікація – процес перевірки наданого в процесі ідентифікації ідентифікатора на належність до системи;
- Авторизація – процес перевірки чи має користувач доступ або певні права на модифікацію деякого ресурсу.

Як видно, всі ці поняття пов’язані між собою, і, фактично, слідують один за одним в процесі встановлення дозволів користувача. Проте розглянемо більш детально процес автентифікації.

Цей процес або механізм, як було зазначено в першому розділі, використовує ідентифікатори, які можуть бути поділені на 3 типи за факторами: фактор знання, фактор володіння та фактор наслідування або біометричний.

Найпростішим та найпопулярнішим способом автентифікації в веб-застосунках є парольна автентифікація. Така схема авторизації передбачає наявність у користувача ідентифікатора за фактором знання, наприклад пароль, частина паролю або ПІН. Користувач надсилає запит на авторизацію який містить пару логін – пароль. Паролем можуть виступати будь-яке відоме лише санкціонованому користувачу значення. Логіном має бути щось, що унікально ідентифікує користувача у системі, тобто в системі не може бути два однакових логіна. Прикладами логіна можуть виступати номер мобільного телефону, електронна пошта, унікальний набір символів, порядковий номер та інше.

Для забезпечення більшого захисту варто використовувати двуфакторну автентифікацію. Найпростіше поєднати парольну автентифікацію за допомогою мобільного пристрою або електронної пошти. В такому випадку тільки знання паролю не є достатнім для автентифікації користувача. Після проходження парольної автентифікації користувачу на прив’язаний мобільний пристрій або електронну пошту

надсилається деякий, часто шестизначний код, який необхідно ввести, щоб завершити процедуру автентифікації. Також можливе використання додатків на мобільних пристроях для такого типу автентифікації. В такому випадку в додатку генерується код, який є активним тільки деякий дуже короткий проміжок часу (до 1 хвилини) і користувач має ввести такий код. Цей підхід дозволяє звільнитися від передачі даних іншими каналами, де можливий перехват коду автентифікації, так як запит на перевірку коду надсилається вже після введення коду, а не до як у випадку електронної пошти.

Сам процес обміну ідентифікатором та надання доступу визначений у стандарті RFC 7235, який визначає фреймворк автентифікації для HTTP для того щоб сервер зміг

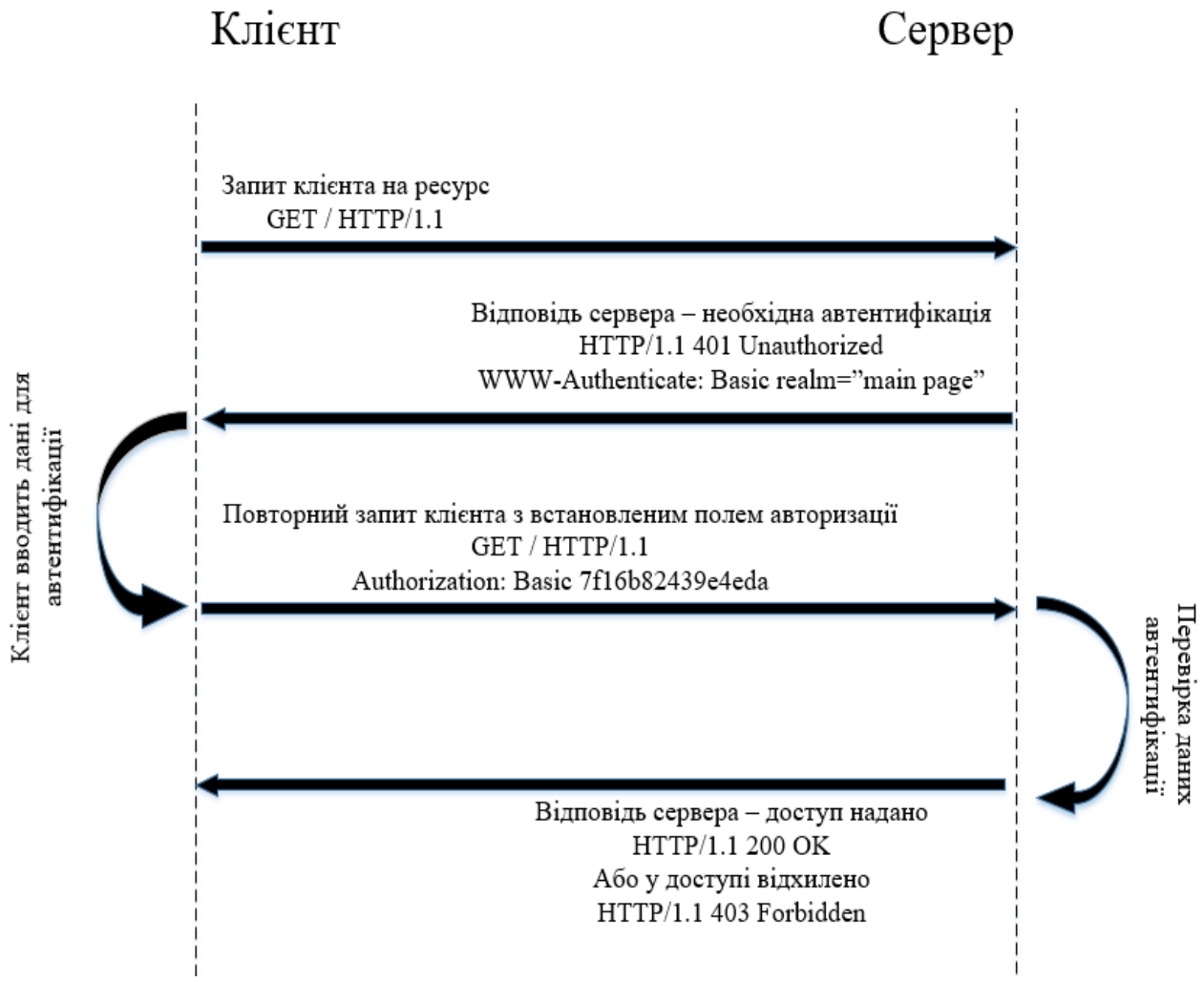


Рисунок 2.2 - Фреймворк автентифікації для HTTP

здійснювати запит автентифікації до клієнта, а клієнт – надавати таку інформацію. [36] Цей процес схематично зображено на рисунку 2.3.

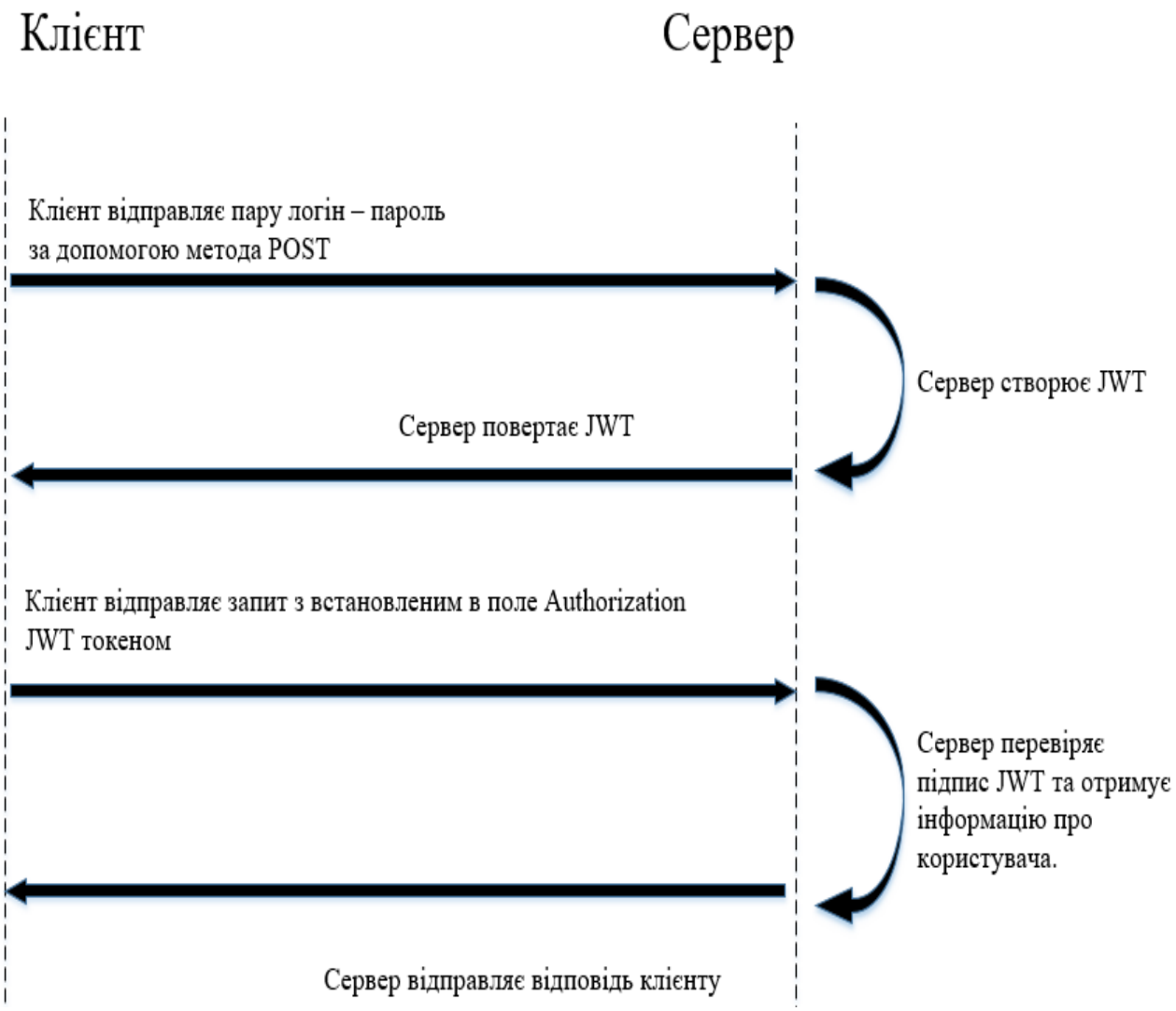


Рисунок 2.3 - Схема автентифікації на основі JWT токену

На рисунку можна побачити, що неавторизований клієнт намагається звернутися до головної сторінки веб-застосунку. Сервер, обробляючи запит, визначає, що для доступу до цієї сторінки користувач має надати дані авторизації, тому відповідає статус кодом 401 і передає інформацію про те, як необхідно авторизуватися в застосунку за допомогою поля заголовку WWW-Authenticate. Якщо клієнт хоче продовжити роботи то

він надає дані для автентифікації і надсилає новий запит до сервера з встановленим полем Authorization. Сервер перевіряє значення поля і порівнює його з списком існуючих ідентифікаторів. Якщо дані рівні, то сервер повертає необхідну сторінку з статус кодом 200, інакше – повертає статус код 403.

Проте кожен раз запитувати користувача його дані для автентифікації було б незручно у користувача не був би від цього в захваті. В веб-застосунках зі станом для цього досягали за рахунок зберігання стану користувача та так званих «липких» сесій або централізованого сховища станів. Проте сучасні веб-застосунки відійшли від концепції збереження стану взагалі і не зберігають його, що означає, що будь-який запит буде виковуватися на сервері так само як і перший отриманий без прив’язки до користувача. Для вирішення цієї проблеми було запропоновано використання токенів.

Як приклад токену можна використати Json Web Token (JWT). Взагалі токен – це зашифрована в певному вигляді інформація про користувача. JWT це відкритий стандарт RFC 7519, який визначає формат токену, його вміст, шифрування та програмні реалізації для різних мов програмування. [37] Токен в цьому стандарті є дуже простим і складається з 3 частин:

- Заголовок. Вміщує тип токену, що в даному випадку є сталою «JWT» та хеш-алгоритм;
- Дані. Вміщує всю корисну інформацію користувача таку як ідентифікатор користувача, дату спливу дії, а також ролі, дозволи чи будь-яку інші інформацію, яку хочуть додати розробники;
- Підпис. Фактична хеш-сума повідомлення для визначення цілісності та підтвердження джерела, як було описано в попередніх главах.

Ці три частини об’єднуються в один рядок, кодуються алгоритмом Base64, розділяються точкою та повертаються до користувача, де вони зберігаються в браузерному локальному або сесійному сховищі або так званих «cookies». Цей токен потім прикріплюється до кожного запиту до серверу в полі заголовка протоколу HTTP

Authorization або в іншому полі чи самому тілі пакету, проте це достатньо незручно. Процес отримання токену зображено на рисунку 2.4.

Для веб-застосунків з дуже широким колом користувачів та і взагалі для полегшення горизонтального масштабування на базі автентифікації на основі токенів був розроблений стандарт OAuth, який передбачає інтеграцію власного серверу автентифікації з сервісами автентифікації третіх довірених сторін. [38] Тобто користувач може авторизуватися в застосунку за допомогою аккаунту іншої організації, визначити доступи до інформації, які він надає вашому веб-застосунку, та спокійно продовжувати повноцінно працювати з веб-застосунком. В такому випадку, третя сторона сформує необхідний токен автентифікації та в той же час авторизації, який буде валідним у веб-застосунку і буде представляти користувача.

Це є дуже гнучким рішенням, дозволяє позбавитися від необхідності зберігати паролі користувачів у самій системі та покращує простоту роботи користувача з веб-застосунком. Такими третім сторонами або провайдерами аккаунтів можуть виступати такі IT гіганти як Google, [39] Facebook, [40] Twitter, Amazon і так далі.

2.6 Аналіз механізмів доповнення трафіку

Механізм доповнення трафіку відповідає за додавання до трафіку інформації, яка не має корисного навантаження, простіше, є шумом, для покращення криптостійкості. Офіційні повідомлення часто починаються і закінчуються передбачувано, наприклад, «З повагою, ...». Основна мета застосування доповнення до класичними шифрами - позбавити криптоаналитика можливості використовувати таку передбачуваність при криптоанализі відомого тексту. Випадкова довжина доповнення також не дозволяє криптоаналітику дізнатися точну довжину повідомлення.

Навіть якщо використовувати досконалі криптографічні алгоритми, зловмисник може отримати інформацію про кількість генерованого трафіку. Зловмисник може не знати, про що говорили Аліса і Боб, але може знати факт і час розмови. За деяких обставин цей витік може бути компрометуючим. Наприклад, якщо в ході війни одна зі сторін організовує таємний напад на іншу, то може бути достатньо попередити сторону захисту, що відбувається багато таємної діяльності.

Як більш прикладний приклад, при шифруванні потоків технології Voice Over IP, використовується кодування змінної швидкості передачі бітів, проте кількість бітів за одиницю часу не приховано, і це можна використовувати для визначення переданих фраз. [41] Подібним чином, шаблони вибуху, які створюють звичайні кодери відео, часто є достатніми для однозначної ідентифікації потокового відео, яке користувач переглядає. [42] Навіть загальний розмір одного об'єкта, наприклад веб-сайту, файлу, завантаження програмного пакету або відео в Інтернеті, може однозначно ідентифікувати об'єкт, якщо зловмисник знає або може здогадатися про відомий набір, з якого походить об'єкт. Побічний канал зашифрованої довжини вмісту використовувався для вилучення паролів з HTTPS-зв’язку у відомих атаках CRIME і BREACH. [43]

Додання шуму до зашифрованого повідомлення може ускладнити аналіз трафіку, маскуючи справжню довжину його корисного навантаження. Вибір довжини для доповнення повідомлення може бути або чітко визначеним, або випадковим чином; кожен підхід має сильні та слабкі сторони, які застосовуються в різних контекстах.

Варто зауважити, що самостійно використаний цей механізм не має жодного сенсу, він використовується тільки тоді, коли забезпечується механізм шифрування. Для асиметричного шифрування доповнення - це процес підготовки повідомлення для шифрування або підписання з використанням специфікації або схеми. Сучасною формою доповнення для таких алгоритмів є OAEP, застосований до алгоритму RSA, коли він використовується для шифрування обмеженої кількості байтів. Спочатку випадковий матеріал просто додавали до повідомлення, щоб зробити його досить довгим для шифрування. Ця форма не є надійною і тому більше не застосовується. Сучасна

схема доповнення спрямована на те, щоб нападник не міг маніпулювати відкритих текстом і зазвичай супроводжується доказом, часто у моделі випадкового оракула.

Для симетричного шифрування використовуються наступні схеми доповнення:

- Хеш-функції. Хеш-функції завжди видають результат однакової довжини;
- Блокові режими шифрування;
- ANSI X/923. Доповнюється нулями і останній байт визначає кількість доданих байтів;
- ISO 10126. Доповнюється випадковими байтами і останній байт визначає кількість доданих байтів;
- PKCS7. Доповнюється кількістю доданих байтів;
- ISO/IEC 7816-4. Перший доповнений байт є «80», решта доповнених є «00»;
- Доповнення нулями.

В більшості веб-застосунків такий механізм не застосовується, що ставить під вразливість дані користувачів, які передаються. Тому для захисту інформації від вищезазначених атак варто проводити роботи по імплементації такого механізму у веб-застосунку, хоча це і ускладнює сам механізм.

2.7 Аналіз механізмів контролю маршрутизації

Так як веб-застосунки представляють собою модуль клієнт-серверної взаємодії, то звісно існує деякий сервер, який приймає запити від клієнта і обробляє їх певним чином. Проте немає гарантії, що сервер завжди отримує запити саме від клієнта. Якщо зловмисник захоче відправити запит на порт, наприклад, бази даних, а він буде відкритий на сервері, то це може призвести до порушення безпеки веб-застосунку. Механізм контролю маршрутизації намагається вирішити цю проблему.

Для початку повинні бути ідентифіковані всі критичні сервери та програми, які спілкуються з іншими компонентами через мережеві порти та ізолювати їх. Найпоширеніший метод ізоляції - це використання брандмауерів для захисту

відповідних серверів і додатків. Також повинно гарантуватися, що всі зміни дозволені, а набори правил періодично переглядаються.

Крім того, жоден брандмауер не повинен дозволяти використовувати будь-які відомі ризиковані сервіси чи протоколи, так як такі сервіси чи протоколи надають зловмисникам простий шлях всередину веб-застосунку. До таких ризикових протоколів належать, але не обмежуються ними, FTP (порт 21 / протокол tcp), Telnet (23 / tcp), Rlogin (513 / tcp), Rsh (514 / tcp), Netbios (137-139 / tcp, udp) та інші. Будь-які ризиковані протоколи або послуги повинні бути негайно видалені з політики брандмауера.

2.8 Аналіз механізмів нотаріального посвідчення

Для цілей кібербезпеки, термін нотаріального посвідчення з підтримкою інфраструктури відкритих ключів (PKI) визначається як синонім "сертифікації даних". Механізм нотаріального посвідчення використовується для забезпечення цілісності, джерела чи отримувача і часу отримання чи відправки переданих даних. Такий механізм підтвердження може бути частиною використовуваних мережевих протоколів, таких як HTTPS, та/або довірчої третьої сторони, яка забезпечує постійність з’єднання та неспростовність. Механізм може підтримуватися іншими механізмами, такими як механізм цифрового підпису, шифрування чи цілісності даних.

Найчастіше у веб-застосунках для реалізації цього механізму використовують рівень захищених сокетів (SSL) протоколу HTTPS. SSL – це криптографічний протокол розроблений компанією Netscape Communications у 1995, сертифікований в 1999 стандартом RFC 2246 як протокол безпеки транспортного рівня (TLS). [44] Цей протокол забезпечує захист даних між клієнтом та сервером, які використовують стек протоколів TCP/IP, використовуючи асиметричний алгоритм шифрування. В HTTPS цей протокол застосовується як певний «конверт» для даних, тобто пакет HTTP передається за

допомогою SSL або TLS. Майже всі основні елементи протоколу HTTP шифруються: URL-запити, включаючи шлях та назву ресурсу, параметри запиту, заголовки, які часто містять ідентифікаційні дані про користувача. Не шифруються: назва або адреса хоста (веб-сайту) та порт, оскільки вони використовуються транспортним протоколом TCP/IP для встановлення з'єднання.

Для приймання сервером пакетів протоколу HTTPS необхідно мати на сервері сертифікат SSL, який містить відкритий ключ та був підписаний уповноваженим на видачу сертифікатів, що дає гарантію того, що отримувач сертифікату саме той, за кого він себе видає. Браузери включають в себе сертифікати, які були отримані центрами сертифікації верхнього рівня і це дозволяє браузерам перевіряти сертифікати, які були підписані такими центрами.

Протокол SSL включає в собі 2 протоколи: протокол запису та протокол рукостискання. Перший – визначає формат передачі даних. Проте з точки зору механізму нотаріального посвідчення, нам цікавий саме протокол рукостискання. Цей протокол визначає процес встановлення першого зв’язку між клієнтом та сервером в ході якого буде визначений набір параметрів, які будуть використані для забезпечення безпеки з’єднання. Процес рукостискання можна описати наступним чином:

- Рукостискання починається тоді, коли клієнт підключається до серверу. Такий запит представляє собою список підтримуваних асиметричних шифрів та хеш-функцій. Часто використовуваними є такі асиметричні алгоритми як RSA, Діффі-Хеллман. Хеш-функції: SHA, MD5, MD4, MD2;
- Отримавши список, сервер вибирає найкриптозахищений алгоритм з цього списку, який він також підтримує. Сервер надсилає це рішення у вигляді цифрового сертифікату, який містить ім’я серверу, довірений Центр Сертифікації і відкритий ключ шифрування серверу. Клієнт може зв’язатися з центром, який видав сертифікат і переконатися, що він є справжнім;
- Для генерації ключів сеансу клієнт шифрує випадкове число за допомогою отриманого від серверу ключа і відправляє результат на сервер. Так як тільки сервер

може розшифрувати його використовуючи свій закритий ключ, то вже на даному етапі можна сказати, що у третьої сторони немає доступу до ключів сеансу;

- З отриманого випадкового числа обидві сторони створюють ключові дані для шифрування, як диктується алгоритмом, який був обумовлений на першому кроці.

На цьому рукостискання завершується і далі з'єднання є захищеним. Якщо якись з етапів не вдалося пройти, то з'єднання не встановлюється.

Висновки за розділом 2

В даному розділі були розглянуті визначені в рекомендації X.800 механізми безпеки та їх варіанти реалізації у веб-застосунках. Для кожного прикладу рекомендованих до використання реалізацій було надано як мінімум загальний опис та схему роботи. Були надані приклади, де саме в веб-застосунку ці реалізації використовуються, проведено аналіз плюсів і недоліків даних реалізацій.

Було розглянуто одностороннє, симетричне і асиметричне шифрування та їх відмінність. Як приклад одностороннього шифрування було надано хешування, як приклад реалізації – хеш-функції сімейства SHA: SHA-2 та SHA-3. У симетричному шифруванні були розглянуті блочні шифри, а саме Blowfish.

Був розглянутий механізм цифрового підпису, схема його роботи та описано алгоритм цифрового підпису DSA.

Було розглянуто рольове управління доступом та управління доступом на основі атрибутів. Було надано порівняльну характеристику цих двох моделей та надано рекомендації випадків застосування.

Були представлені підмеханізми цілісності даних: створення резервних копій, валідація вводу, валідація даних та коди виправлення помилок.

Було визначено відмінності між ідентифікацією, автентифікацією та авторизацією. Було представлено парольну та багатофакторну парольні аутентифікацію. Були описані

схеми авторизації на базі фреймворку протоколу HTTP, на базі JWT токенів та протоколу OAuth.

Для механізму доповнення трафіку був описаний процес доповнення, вимоги механізму, а також варіанти реалізацій для симетричних та асиметричних шифрів. Брандмауери були описані як приклад механізмів контролю трафіку.

Протокол HTTPS та SSL/TLS як і процес встановлення захищеного зв’язку були описані в главі про механізм нотаріального посвідчення.

Таким чином, якщо звернутися до задачі, яка була поставлена в першому розділі, то цей розділ описує можливі реалізацій механізмів безпеки для різних сценаріїв впровадження веб-застосунку, серед яких веб-розробник має вибрати одну або декілька таких реалізацій.

РОЗДІЛ 3
СТВОРЕННЯ ЕКСПЕРИМЕНТАЛЬНОЇ МОДЕЛІ ВЕБ-ЗАСТОСУНКУ З ВИКОРИСТАННЯМ СЕРВІСІВ ТА МЕХАНІЗМІВ БЕЗПЕКИ

3.1 Загальний опис експериментальної моделі

Експериментальна модель, що розробляється, є клієнт-серверним веб-застосунком без збереження сесії користувача, який складається серверу, що оброблює дані, та веб-сторінки, яка є інтерфейсом для користувача та слугує для отримання та надсилання даних до серверу. Фактично, ваб-застосунок э прикладом реалізації RESTful сервісу. [45] Базовий протокол для клієнт-серверної взаємодії – HTTP.

Веб-застосунок забезпечує наступні сервіси безпеки:

- Автентифікація отримувача;
- Управління доступом;
- Конфіденційність з’єднання;
- Вибірна конфіденційність поля;
- Цілісність даних без відновлення;
- Неспростовність джерела та отримувача.

Розроблена модель може бути поділена на умовні три шари: інтерфейс, API та рівень управління даними. Загальний процес обробки запиту у моделі можна схематично зобразити на рисунку 3.1.

Інтерфейс представляє собою Angular 8 застосунок, який є веб-сторінкою, яка написана за допомогою HTML, CSS та TypeScript (при виконанні код мови TypeScript інтерпретується в JavaScript). Застосунок хоститься на локальному сервері Node.js та відповідає за отримання введених даних користувача та відображення даних, які були отримані сервером у зручному форматі.

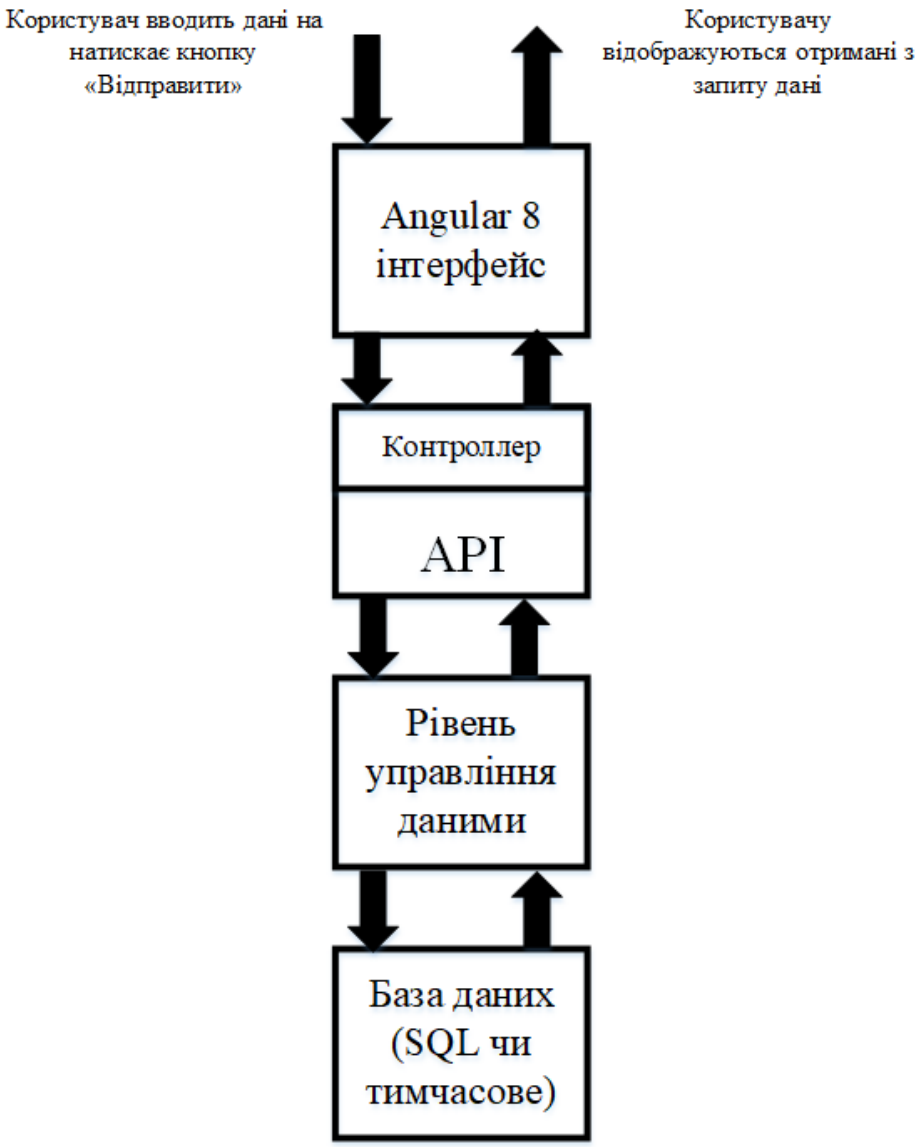


Рисунок 3.1 - Процес обробки запиту в експериментальній моделі

API є програмним застосунок, який написано на фреймворку Microsoft ASP.NET Core на мові програмування C#. Даний шар веб-застосунку відповідає за отримання запитів, їх валідацію, валідацію відправника даних, його авторизацію та неспростовність. Фактично, цей шар є основним шаром веб-застосунку, який і реалізує сервіси та механізми безпеки. Цей шар хоститься на IIS, через вбудоване розширення Microsoft Visual Studio.

Клієнти спілкуються з API сервером за допомогою ендпоїнтів, логіка обробки даних яких описується в даному фреймворку методами контроллерів. Проте перед тим як дані потраплять до ендпоїнту відбувається виконання так званих ПЗ проміжного рівня, яке надано самим фреймворком, [48] яке включає: обробку виняткових ситуацій, строге захищене з'єднання через HTTP (HSTS), перенаправлення HTTPS, спільне використання ресурсів з різних джерел (CORS) та інші. Повний шлях запиту та відповіді описаний у документації Microsoft та зображено на малюнку 3.2.

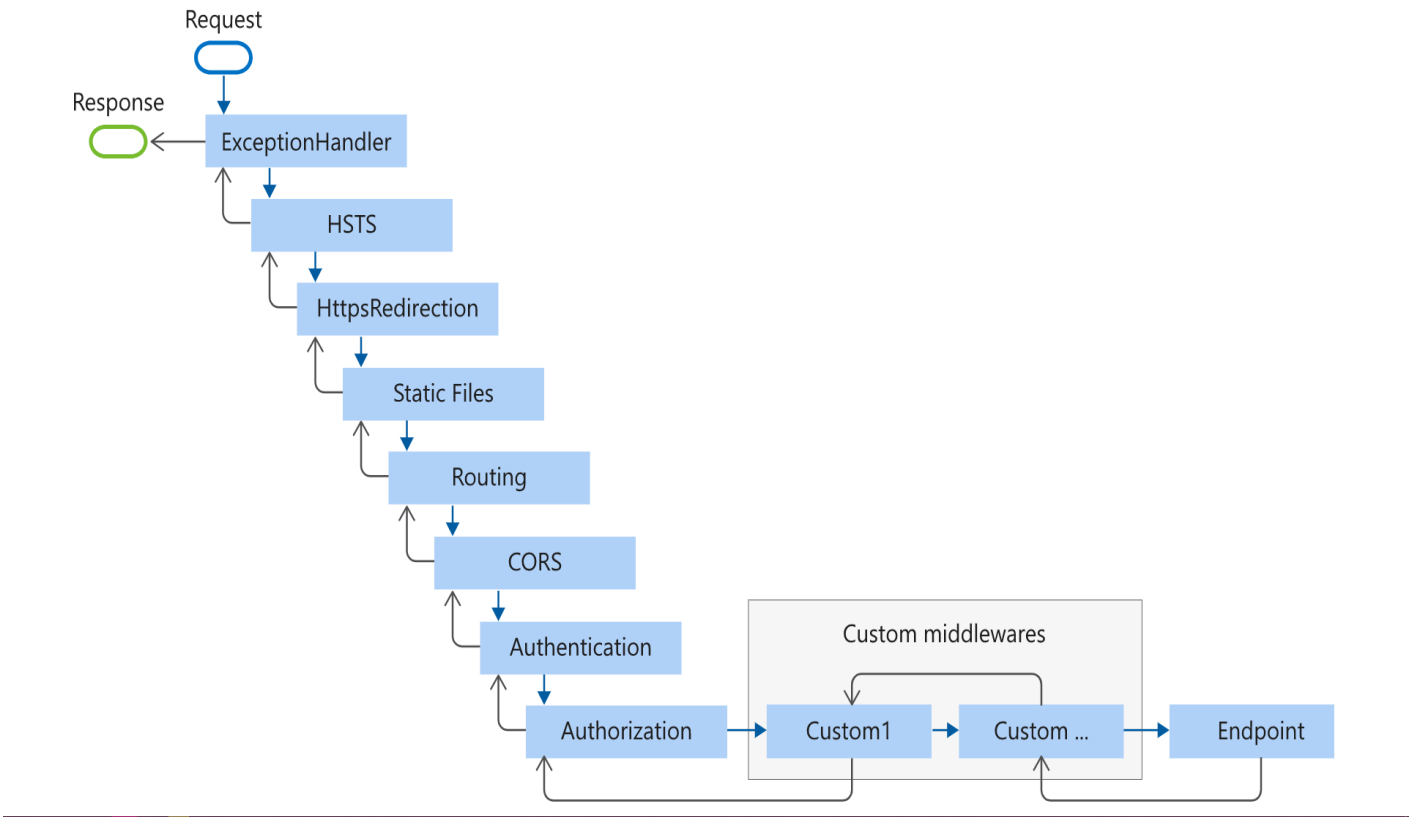


Рисунок 3.2 - Схематичне зображення обробки запиту у фреймворку ASP.NET Core

Рівень управління даними відповідає за базові операції з даними CRUD (Create – створення, Read – читання, Update – оновлення, Delete – видалення). Також відповідає за створення зв'язку до бази даних, яка знаходиться на сервері Microsoft SQL, якщо встановлено відповідний прапор у налаштуваннях. Для перетворення об'єктів та методів мови C# використовується об'єктно-реляційне відображення, яке надається

фреймворком Entity Framework Core. В коді не має жодного явного SQL запиту – вся комунікація з базою даних SQL відбувається за допомогою вищеназваного фреймворку. Якщо прапор не встановлено використовується сховище даних, яке існує тільки на час виконання програми та повністю видаляється після завершення.

3.2 Забезпечення сервісу автентифікації

В розробленій моделі цей сервіс забезпечується механізмами цифрового підпису та обміну автентифікації. Обидва механізми в обраній реалізації взаємопов’язані між собою.

Механізм обміну автентифікацією реалізований за допомогою JWT токенів, які були описані в відповідній главі другого розділу. Як схема HTTP автентифікації була обрана схема автентифікації носія токену (bearer authentication). [50] Така схема включає токени, що називаються токенами носія. Назву схеми можна розуміти як схему надання доступу для носія цього токену. Токен носія - це зашифрований рядок, який генерується сервером у відповідь на запит на вхід до системи. Клієнт повинен надіслати цей токен у заголовку HTTP Authorization під час спроби отримання доступу до захищених ресурсів. Формат заголовку має бути наступним: Authorization: Bearer <значення токену>.

Схема автентифікації Bearer була спочатку створена як частина OAuth 2.0 в RFC 6750, але іноді також використовується самостійно. Аналогічно базовій автентифікації, автентифікацію носія слід використовувати лише через HTTPS (SSL). Дана схема може працювати не тільки з JWT токенами, проте і іншими специфікаціями токенів, для цього встановлюється відповідне значення у поле bearerFormat.

Послуги схеми автентифікації Bearer та JWT токенів надаються бібліотеками Microsoft Microsoft.AspNetCore.Authentication, нам залишається тільки зконфігурувати схему автентифікації та параметри валідації токену. В секції налаштування серверу ми

визначаємо схему автентифікації, а також тип токену, та параметри його валідації. Якщо згадати структуру JWT токену, то він має таку секцію як підпис, яку ми і перевіряємо в налаштуваннях валідації токену. Ми зазначаємо, що перевірятися буде орган, який видав сертифікат для підпису токену, кому був виданий цей сертифікат, ключ, а також час дії токену після чого він стає невалідним. Процес підпису токену є реалізацією механізму цифрового підпису, який також використовує механізм шифрування для генерації підпису та кодування токену. Як алгоритм створення цифрового підпису використовується HMACSHA256.

Використаним типом автентифікації є парольна автентифікація. Користувач вводить логін та пароль у відповідні поля веб-сторінки, клієнт формує HTTP запит до серверу. На стороні серверу, дані запиту валідуються: необхідно, щоб був наданий логін та пароль, інакше повертається статус код 400. Потім перевіряється наявність логіну в базі даних та чи відповідає пароль даному логіну, в разі успішної перевірки формується токен. В токен записується логін користувача та його роль. Після успішної автентифікації користувачу у відповідь надсилаються його дані такі як логін, ім’я, роль та статус аккаунту, а також токен у окремому полі тіла відповіді. На стороні клієнта цей токен зберігається у локальному сховищі браузера. Після його отримання і до моменту, коли він стає невалідним клієнт за допомогою так званого перехоплювача, який перед відправкою повідомлення додає необхідне поле у заголовку HTTP, додає токен до всіх запитів. Валідація токену після його отримання сервером реалізується вбудованим ПЗ проміжного рівня, як було описано у попередній главі.

Програмна реалізація створення токену та процесу входу користувача до системи надана в додатку А, а на малюнку 3.3 зображено інтерфейс сторінки логіну та реєстрації.

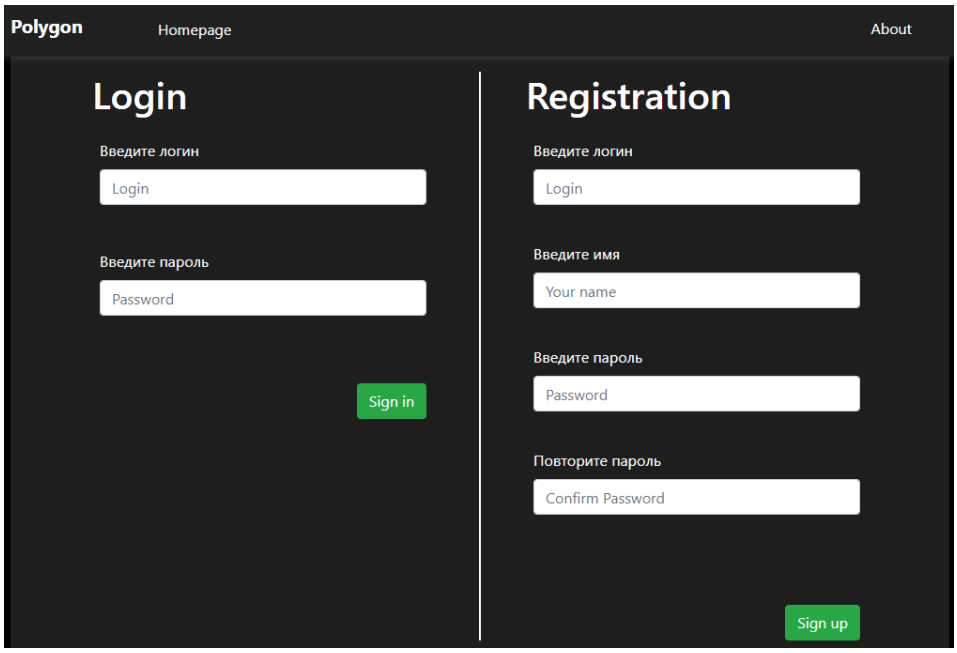


Рисунок 3.3 - Інтерфейс входу до веб-застосунку

3.3 Забезпечення сервісу управління доступом

У розробленій моделі даний сервіс забезпечується одноіменним механізмом безпеки та реалізований моделлю управління доступом, яка базується на ролях. Для такого невеликого веб-застосунку як дана експериментальна модель такий підхід є простішим у керуванні через те, що дозволів доволі невелика кількість. Якщо точніше, то дозволи включають:

- Проведення входу;
- Проведення реєстрації;
- Перегляд даних користувача;
- Зміна статусу аккаунту користувача;
- Зміна ролі користувача;
- Зміна паролю користувача.

В системі існують такі ролі: гість (неавторизований, тому дана роль не задана явно), користувач та адміністратор. Відповідно до ролі, користувачу відображається різний інтерфейс, а також він може виконувати різний набір функцій. Ролі мають ієрархічну структуру та розширюють одна одну: «користувач» має дозволи «гостя» та власні дозволи, «адміністратор» має всі дозволи «користувача», тобто і дозволи успадковані від «гостя», та власні дозволи. Як було сказано при розгляді моделі RBAC, дана модель передбачає наявність ролі суперкористувача, яка має всі доступи, і дана система не виняток, тут це роль «адміністратор».

Роль «гість» має дозвіл переглядати інтерфейс входу, посилати запити на вхід та реєстрацію. Роль «користувач» додає до дозволів ролі «гостя» можливість переглядати сторінку з своїми даними, змінювати пароль та виходити з системи. Роль «адміністратор» розширює роль «користувач» і додає можливість переглядати список існуючих аккаунтів, змінювати статус аккаунтів інших користувачів та змінювати ролі користувачів. Порівняння вигляду сторінки профілю для ролі «користувача» і «адміністратора» показана у додатку Б.

На стороні серверу управління доступом реалізується за допомогою вбудованого ПЗ проміжного рівня фреймворку ASP.NET Core. Дане ПЗ виконується після перевірки автентифікації також за допомогою ПЗ проміжного рівня та перевіряє наявність певного маркеру у запиті. В даній реалізації це поле «Роль» в частині токену, яка відповідає за зберігання інформації користувача. На ендпоїнти можуть додаватися атрибути, які визначають, яка роль необхідна для обробки запиту цим ендпоїнтом. Наприклад, в даній реалізації на ендпоїнт, який відповідає за зміну ролі, додано атрибут, який позначає що тільки користувачі ролі «адміністратор» можуть використовувати цей функціонал. Якщо ж запит буде здійснено на цей ендпоїнт без токену або з токеном користувача, який не знаходиться у ролі «адміністратор», то такий запит не буде оброблено і буде повернено статус код 403.

На стороні клієнту перевіряється роль користувача для відображення деяких елементів інтерфейсу. Так, можливо у браузері інвертувати логіку відображення і

відображати все, проте це не дозволить отримати конфіденційну інформацію через те, що такі запити просто не будуть оброблені сервером і будуть відхилені.

3.4 Забезпечення сервісу конфіденційності з'єднання та неспростовності отримувача та відправника

Для обох цих сервісів використовується механізм захищеного зв'язку SSL, який дозволяє організувати як захищений канал зв'язку, так і неспростовності джерела та отримувача за допомогою SSL-сертифікатів.

В фреймворку ASP.NET Core використання протоколу HTTPS увімкнуто за замовчанням, тому всі дані, які передаються між клієнтом та сервером передаються в захищеному вигляді. SSL-сертифікат, який необхідний для встановлення захищеного зв'язку, для цілей даної експериментальної моделі є самопідписаним та згенерований тільки для даного серверу, проте це не заважає встановлювати захищений зв'язок. Звичайно, для використання у мережі Інтернет для даного веб-застосунку має бути отриманий валідний сертифікат, який є виданим довіреним центром сертифікації. На рисунку 3.4 можна побачити що дані передаються захищеним каналом зв'язку, де localhost:44399 це адреса серверу.

#	Result	Protocol	Host	URL	Body
108	400	HTTPS	localhost:44399	/api/identities/	32
153	204	HTTPS	localhost:44399	/api/identities/	0
154	400	HTTPS	localhost:44399	/api/identities/	32
172	200	HTTPS	localhost:44399	/api/identities/	415
174	204	HTTPS	localhost:44399	/api/identities	0
176	200	HTTPS	localhost:44399	/api/identities	391
194	204	HTTPS	localhost:44399	/api/identities/Banned	0
195	500	HTTPS	localhost:44399	/api/identities/Banned	19,180

Рисунок 3.4 - Знімок трафіку між клієнтом та сервером

Додатково для сервісу конфіденційності з’єднання використовується механізм використання ресурсів з різних джерел CORS. [51] Цей механізм, за допомогою HTTP-заголовків дає джерелу дозвіл завантажувати ресурси з певного ресурсу, отриманого з відмінного джерела. Веб-застосунок виконує перехресний HTTP-запит коли потребує ресурсів з ресурсу (домен, протокол чи порт), відмінного від його власного.

Наприклад, реалізований сервер за замовчанням буде відхиляти запити від клієнта, який розташований за адресою «https://localhost:4200», через те, що його власна адреса відрізняється, так як він розташований за адресою «https://localhost:44399». Механізм CORS є доволі гнучким та дозволяє блокувати запити до серверу залежно від значення адреси, методу HTTP, заголовку HTTP, часу зберігання кешу та чи може відповідь бути отримана авторизованою особою.

В моделі встановлено обмеження на виконання запитів. Він виконується тільки якщо запит має у полі відправника значення адреси серверу, тобто «https://localhost:4200».

3.5 Забезпечення сервісу вибірної конфіденційності поля

Цей сервіс забезпечує зберігання деяких даних користувачів у захищеному вигляді та реалізується за допомогою механізму шифрування. В моделі використане як хешування так і блокове симетричне шифрування для безпечного збереження даних користувачів у веб-застосунку.

Для хешування використовується алгоритм хешування SHA-512, реалізація якого надана бібліотекою від Microsoft. Хешування використовується для паролей. Перед збереженням паролів до бази даних або порівнянням двох паролів до них застосовується вищезазначена хеш-функція. Хеш-функція працює з байтовим масивом, тому рядок з паролем у вигляді відкритого тексту перетворюється у байтовий масив з використанням

кодування UTF-8. В результаті хешування отримується набір з 64 байтів, які записуються як рядок та зберігаються у базі даних.

Для шифрування інших даних, таких як логін та ім’я користувача був реалізований алгоритм Blowfish Розглянемо схему його роботи. Схематично алгоритм зображено на рисунку 3.5.

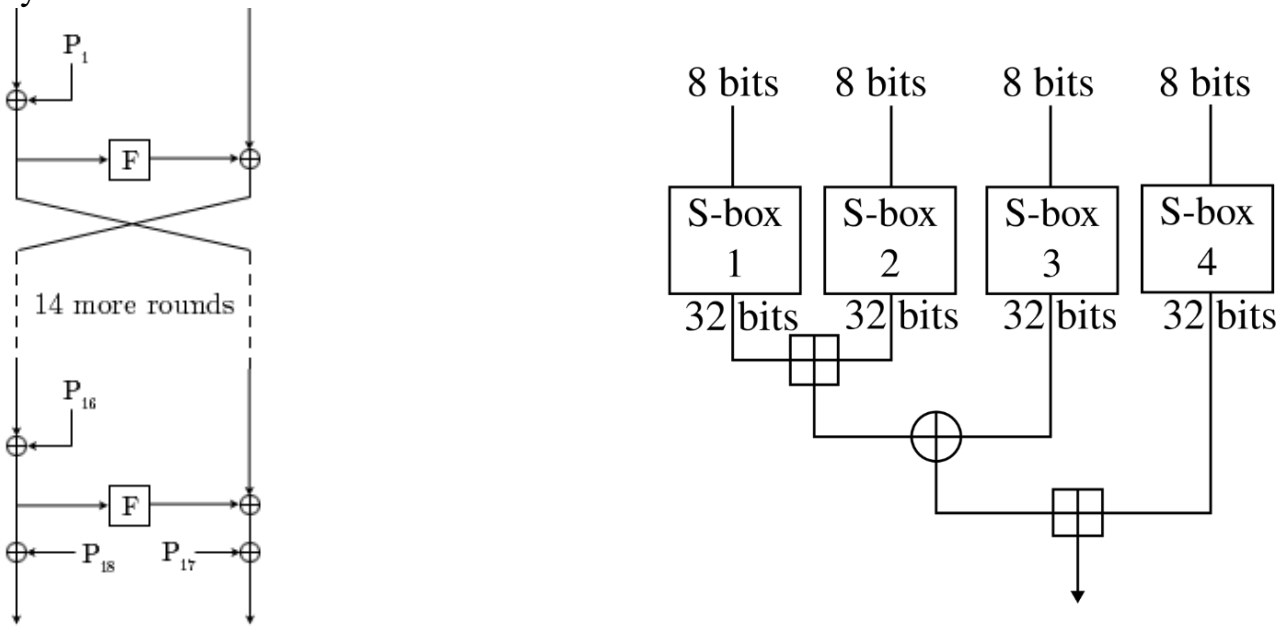


Рисунок 3.5 - Схематичне зображення мережі Фейстеля (зліва) та функції шифрування (праворуч) для алгоритму Blowfish

Алгоритм включає секретний ключ K довжиною від 32 до 448 біт, 18 32 бітних раундових ключів шифрування $P_1 – P_{18}$ та 4 32 бітових таблиць заміни $S_1 – S_4$. Загальний алгоритм шифрування покроково виглядає наступним чином:

- Вхідний текст розбивається на блок даних розміром 64 біти. Якщо розбити неможливо, то використовується один з режимів шифрування;
- Отриманий блок розбивається на два блоки по 64 біти – лівий L_0 та правий R_0;
- Для перших 16 раундів (i = 1…16) виконуються перетворення $L_i = L_{i-1} \oplus P_i$; $R_i = R_{i-1} \oplus F(L_i)$;

○ Сама функція шифрування F(X) надана у формулі 3.1. Фактично, блок даних X розміром 32 біти розбивається на 4 блоки X_1 - X_4 по 8 біт, індекси яких відповідають індексам блоків замін S_1 - S_4. Значення $S_1[X_1]$ і $S_2[X_2]$ складаються за модулем 2^{32}, потім складаються по модулю 2 з $S_3[X_3]$ і в кінці складаються за модулем 2^{32} з $S_4[X_4]$;

$$F(X) = F(X_1, X_2, X_3, X_4) = (\big((S_1[X_1] + S_2[X_2]\ mod 2^{32} \oplus S_3[X_3]) + S_4[X_4]\big) mod 2^{32}, \quad (3.1)$$

де $X_1 – X_4$ – 8 бітні блоки вхідного тексту;

$S_1 – S_4$ – 32 бітні таблиці заміни.

- Після 16-го раунду шифрування блоки L_{16} та R_{16} змінюються місцями і має місце наступне перетворення: $L_{17} = L_{16} \oplus P_{18}$; $R_{17} = R_{16} \oplus P_{17}$;
- Результат конкатенації блоків L_{17} та R_{17} є результатом шифрування.

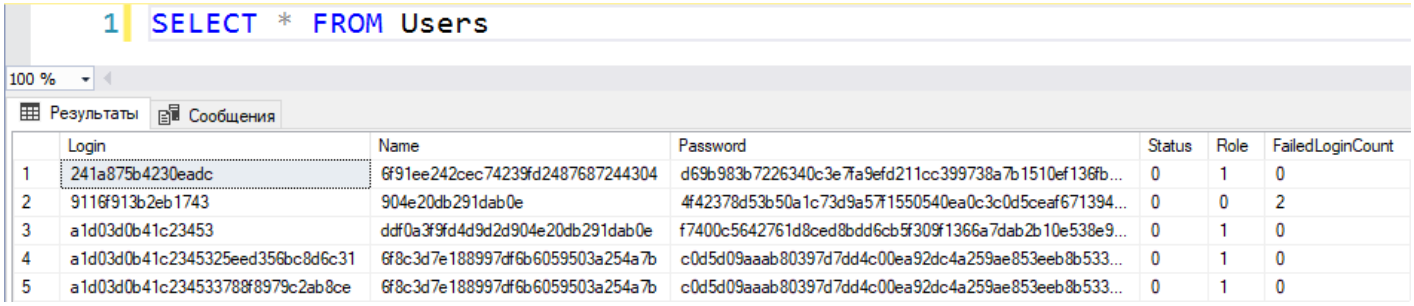
Для генерації раундових ключів шифрування $P_1 – P_{18}$ та таблиць заміни $S_1 – S_4$ також визначено алгоритм:

- Ініціалізуються масиви раундових ключів $P_1 – P_{18}$ та таблиць замін $S_1 – S_4$, які є сталим рядком, що складається з мантиси числа пі у шістнадцятковому представлені. Дані масиви є у відкритому доступі;
- P_1 додається по модулю 2 з першими 32 бітами ключа K, P_2 з другими 32-бітами і так далі. Якщо ключ K виявляється коротким, то він накладається циклічно.
- Використовуючи початкові раундові ключі $P_1 – P_{18}$ і таблиці замін $S_1 – S_4$, які були отримані в попередньому кроці, шифрується 64 бітний нульовий рядок. Результат записується в P_1 та P_2.
- P_1 і P_2 шифрується зміненими значеннями ключів і таблиць замін. Результат записується в P_3 і P_4. Шифрування триває до зміни всіх ключів $P_1 – P_{18}$ і таблиць замін $S_1 – S_4$.

Так як алгоритм працює з блоками довжини 64 біта або 8 байт, а ми не можемо гарантувати, що до алгоритму завжди буде надходити рядок саме такої довжини, то було

реалізовано доповнення блоку шифрування ANSI X9.23. Згідно реалізації, якщо довжина байтового масиву, який отриманий після конвертації рядка до масиву байтів за допомогою кодування UTF-8, не є кратна 8 байтам, то до масиву додається необхідна кількість байтів, які мають значення 0, а останнє значення масиву є кількістю доданих байтів. Також, для отримання такого самого значення після дешифрування, доповнення прибирається з масиву байтів.

В результаті виконання цього сервісу, всі чутливі дані користувачів, які збережені в базі даних веб-застосунка, є зашифрованими або за допомогою хешування або за допомогою блокового шифру. На рисунку 3.6 зображено вигляд даних у базі даних веб-застосунка. Код реалізації як шифрування так і хешування надано в додатку В.



	Login	Name	Password	Status	Role	FailedLoginCount
1	241a875b4230eadc	6f91ee242cec74239fd2487687244304	d69b983b7226340c3e7fa9efd211cc399738a7b1510ef136fb...	0	1	0
2	9116f913b2eb1743	904e20db291dab0e	4f42378d53b50a1c73d9a57f1550540ea0c3c0d5ceaf671394...	0	0	2
3	a1d03d0b41c23453	ddf0a3f9fd4d9d2d904e20db291dab0e	f7400c5642761d8ced8bdd6cb5f309f1366a7dab2b10e538e9...	0	1	0
4	a1d03d0b41c2345325eed356bc8d6c31	6f8c3d7e188997df6b6059503a254a7b	c0d5d09aaab80397d7dd4c00ea92dc4a259ae853eeb8b533...	0	1	0
5	a1d03d0b41c234533788f8979c2ab8ce	6f8c3d7e188997df6b6059503a254a7b	c0d5d09aaab80397d7dd4c00ea92dc4a259ae853eeb8b533...	0	1	0

Рисунок 3.6 - Результат спроби отримати дані користувачів з бази даних

Як видно з рисунку, дані неможливо прочитати без дешифрування і інформація є захищеною.

3.6 Забезпечення сервісу цілісності даних без відновлення

Веб-застосунок реалізує цей сервіс як на стороні сервера так і на стороні клієнта. Сервіс реалізований за допомогою механізму цілісності даних, а саме перевірки дайджесту повідомлення перед його обробкою на сервері. Для експериментальної моделі я застосовую перевірку тільки HTTP повідомлень методу POST.

Реалізований механізм за допомогою алгоритму хешування SHA-512. Для серверної сторони використовується та ж реалізація хешування, яка надана сервісом

вибіркової конфіденційності поля та надана у додатку В. Для клієнта хешування реалізоване в інтерсепторі, який перехоплює всі виходящі HTTP пакети та додає поле заголовку HashSum з результатом хешування тіла пакету. Якщо тіло відсутнє в поле ставиться пустий рядок.

На стороні сервера реалізовано власне ПЗ проміжного рівня та включено до процесу обробки запитів фреймворку ASP.NET Core. Дане ПЗ проміжного рівня зчитує значення хешу з заголовку, виділює тіло запиту та обраховує кеш. Якщо хеш не співпадає, то пакет не оброблюється ендпоїнтом і повертається статус код 500.

В результаті вираховування хеша HTTP пакет має такий вигляд як на рисунку 3.7.

Як видно, у запиті наявний хеш тіла HTTP пакету та сервер надіслав у відповідь статус код 200 та деякий набір даних, отже валідація хешкоду пройшла успішно.

Висновки за розділом 3

Отже в даному розділі був розроблений веб-застосунок, який для захисту даних користувачів використовує такі сервіси безпеки:

- Автентифікація отримувача;
- Управління доступом;
- Конфіденційність з’єднання;
- Вибірна конфіденційність поля;
- Цілісність даних без відновлення;
- Неспростовність джерела та отримувача.

Веб-застосунок реалізує тришарову модель та містить в собі шар інтерфейсу, API та рівень управління даними, кожен з яких має свої розподілені обов’язки та функціонал.

Сервіс автентифікації отримувача був реалізований за допомогою механізмів цифрового підпису та обміну автентифікації. Для реалізації механізму обміну автентифікації було обрано парольну автентифікацію, схему автентифікації носія та

JWT токени. Для механізму цифрового підпису вибрано реалізацію за допомогою HMACSHA256.

Сервісу управління доступом використовує однойменний механізм. Через невелику кількість користувачів і ролей було обрано модель управління доступом на основі ролей.

Сервіси конфіденційності з'єднання та неспростовності джерела та отримувача реалізовані механізмами нотаріального посвідчення, а саме SSL. Конфіденційність зв'язку забезпечує захищений зв'язок протоколу HTTPS, а встановлений на сервері SSL-сертифікат забезпечує неспростовність. Додатково сервіс конфіденційності з'єднання реалізується за допомогою механізму контролю маршрутизації, а саме його реалізацією механізму використання ресурсів з різних джерел.

Сервіс виборчої конфіденційності поля використовує механізм шифрування. Для реалізації хешування було обрано алгоритм SHA512, а для блокового шифрування – алгоритм Blowfish. Було детально розглянуто процес шифрування та генерації ключів так як це було необхідно для програмної реалізації даного функціоналу. Блоковий шифр застосовувався для шифрування імені та логіна користувача, так як ці значення мають бути отримані у вигляді відкритого тексту під час взаємодії, хешування – для паролів, так як достатньо порівнювати лише хеш.

Сервіс цілісності без відновлення використовує механізм забезпечення цілісності даних, а саме перевірку цілісності даних. Реалізовано за допомогою порівняння хеша, який отриманий на стороні клієнта, та хеша, який отриманий на стороні серверу.

Для виводу експериментальної моделі в реальне робоче середовище, модель має бути допрацьована, наприклад, використанням не самопідписаних SSL-сертифікатів для забезпечення захищеного зв'язку, але можна побачити, що дана модель показує як легко можна підключити сервіси безпеки до веб-застосунку та те, що вони покращують рівень захисту даних користувачів, які обробляються сервером.

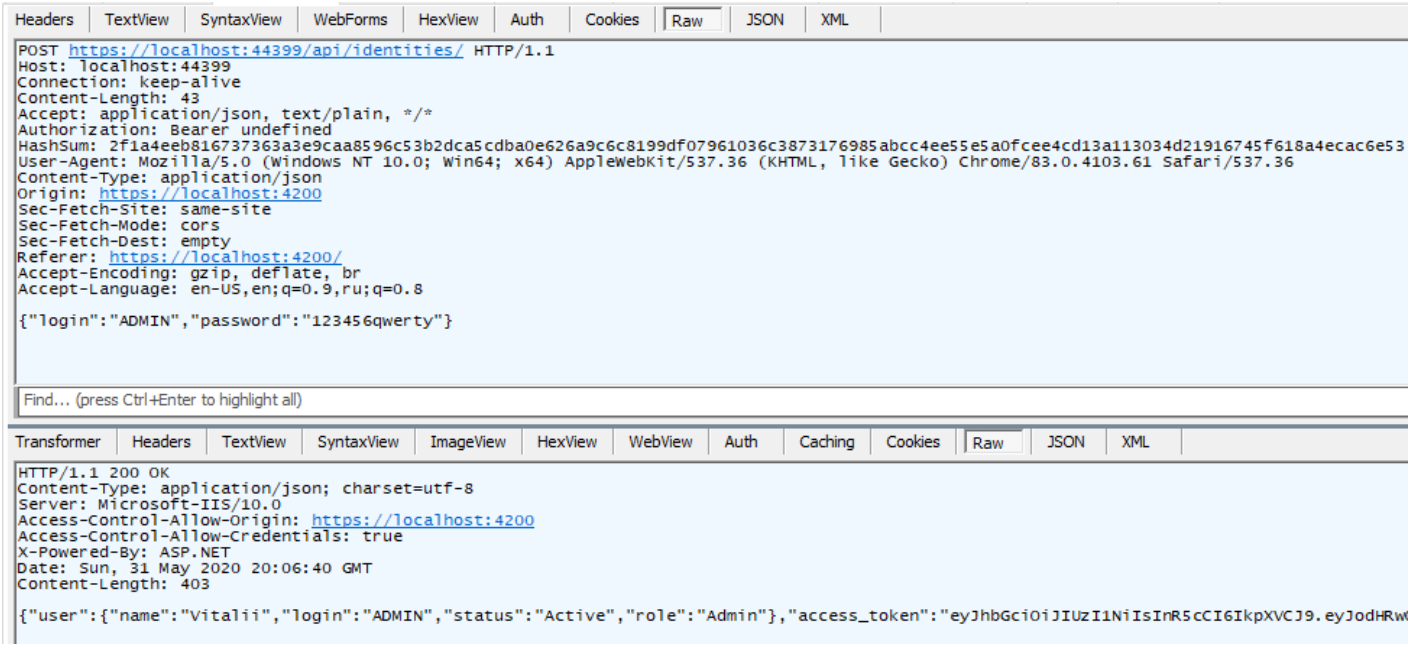


Рисунок 3.7 - HTTP запит з встановленим хешом повідомлення та відповідь сервера

ВИСНОВКИ

Основною метою даної роботи було створення експериментальної моделі веб-застосунку з використанням сервісів та механізмів безпеки для захисту даних користувачів. Створення такої моделі потребує, як було описано в першому розділі, формулювання набору вимог до сервісів та механізмів безпеки, визначення технологій реалізацій сервісів та механізмів безпеки, аналізу процесу роботи обраних реалізацій механізмів безпеки, дослідження сфери використання обраних механізмів захисту у веб-застосунках та власне створення захищеного веб-застосунка.

Результатами даної роботи є:

- Досліджено стандарти, які визначають специфікацію сервісів та механізмів безпеки, а також клієнт-серверної взаємодії;
- Сформовано набір вимог до сервісів та механізмів безпеки;
- Визначені основні технології, які реалізують механізми безпеки;
- Проаналізований процес роботи обраних реалізацій сервісів та механізмів безпеки на прикладі веб-застосунку;
- Досліджено сферу використання механізмів безпеки у веб-застосунку;
- Створено робочу експериментальну модель веб-застосунку, який використовує сервіси безпеки для забезпечення захисту даних користувачів за допомогою механізмів безпеки;

Реалізована експериментальна модель веб-застосунку для захисту користувачів використовує такі сервіси як:

- Автентифікація отримувача;
- Управління доступом;
- Конфіденційність з’єднання;
- Вибірна конфіденційність поля;
- Цілісність даних без відновлення;

- Неспростовність джерела та отримувача

Створена модель використовує сучасний підхід до створення застосунку, а сам розділення інтерфейсу та API, тобто RESTful підхід до створення сервісів та застосунків, а сам сервер є сервером без збереження статусу.

Всі сервіси та реалізації їх механізмів були описані в третьому розділі з показаними результатами використання обраних сервісів безпеки. Для підсумку, я вважаю, що дана модель веб-застосунку показує як легко можна підключити сервіси безпеки до веб-застосунку та те, як саме вони покращують рівень захисту даних користувачів, які обробляються сервером.

Дана модель може бути покращена використанням SSL-сертифікату, який видано довіреним органом сертифікації, замість самопідписаного, зміна типу автентифікації на двухфакторну, а також додаванням інших сервісів безпеки окрім реалізованих. Проте перед модифікацією веб-застосунку, будь то експериментальна модель чи ні, варто пам’ятати що сервіси безпеки є дуже гнучкими і можливо, що використання одного набору сервісів буде мати різний вплив на різних веб-застосунках.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Total number of Websites [Електронний ресурс] – Режим доступу до ресурсу: https://www.internetlivestats.com/total-number-of-websites/.

2. Internet Users [Електронний ресурс] – Режим доступу до ресурсу: https://www.internetlivestats.com/internet-users/.

3. ';--have i been pwned? [Електронний ресурс] – Режим доступу до ресурсу: https://haveibeenpwned.com/.

4. В. П. П'ятигор ПРОГРАМНО-ТЕХНІЧНІ ЗАСОБИ ЗАХИСТУ КОМУНІКАЦІЙНОЇ ВЗАЄМОДІЇ ПРИСТРОЇВ ІНТЕРНЕТУ РЕЧЕЙ З ХАБ-СЕРВЕРОМ / Б. І. Середа, В. П. П'ятигор, І. І. Пархоменко, 2020. – (3 Міжнародна науково-практична конференція “Проблеми кібербезпеки інформаційно-телекомунікаційних систем” (PCSITS)). – С. 26–29.

5. Recommendation X.800. INTERNATIONAL TELECOMMUNICATION UNION. SECURITY ARCHITECTURE FOR OPEN SYSTEM INTERCONNECTION FOR CCIT APPLICATIONS, 1991. – 46 с.

6. ISO 7498-2:1989. Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 2: Security Architecture, 1989. – 32 с.

7. Committee on National Security Systems. Committee on National Security Systems (CNSS) Glossary, 2015. – С. 112.

8. Anoop Singhal (NIST). Guide to Secure Web Services. SP 800-95 / Anoop Singhal (NIST), Theodore Winograd (BAH), Karen Scarfone (NIST), 2007. – С. 111.

9. Hugo Haas. Web Services Glossary [Електронний ресурс] / Hugo Haas, Allen Brown. – 2004. – Режим доступу до ресурсу: https://www.w3.org/TR/ws-gloss/.

10. Davidson, James Duncan. Java Servlet Specification ("Specification") Version: 2.2 Final Release. / Davidson, James Duncan, Coward, Danny., 2001. – С. 43–46.

11. Recommendation X.200. Information technology - Open Systems Interconnection - Basic Reference Model: The basic model, 1994. – 63 с.

12. Federal Financial Institutions Examination Council. Authentication in an Internet Banking Environment" / Federal Financial Institutions Examination Council., 2008. – 14 с.

13. James Ohwofasa Akpeninor. Modern Concepts of Security / James Ohwofasa Akpeninor., 2013. – С. 135.

14. Kristian Beckers. Pattern and Security Requirements: Engineering-Based Establishment of Security Standards / Kristian Beckers., 2015. – С. 100.

15. Boritz, J. Efrim. IS Practitioners' Views on Core Concepts of Information Integrity / Boritz, J. Efrim // International Journal of Accounting Information Systems, 2005. – С. 260–279.

16. McCarthy, C. Digital Libraries: Security and Preservation Considerations / McCarthy, C // Handbook of Information Security, Threats, Vulnerabilities, Prevention, Detection, and Management, 2006. – С. 49–76.

17. George Reese. Distributed Application Architecture / George Reese // Database Programming with JDBC and Java, Second Edition / George Reese., 2000. – С. 126–145.

18. B. Benatallah. Web service conversation modeling: a cornerstone for e-business automation / B. Benatallah, F. Casati, F. Toumani // IEEE Internet Computing / B. Benatallah, F. Casati, F. Toumani., 2004. – С. 46 – 54.

19. What are microservices? [Електронний ресурс] – Режим доступу до ресурсу: https://microservices.io/.

20. Somitra Kumar Sanadhya. Deterministic Constructions of 21-Step Collisions for the SHA-2 Hash Family / Somitra Kumar Sanadhya, Palash Sarkar // Lecture Notes in Computer Science. – Berlin: Springer, 2008. – С. 244–259.

21. Dominique Unruh. Collapse-binding quantum commitments without random oracles / Dominique Unruh., 2016. – 47 с.

22. Dominique Unruh. Collapsing sponges: Post-quantum security of the sponge construction / Dominique Unruh., 2017. – 24 с.

23. Serge Vaudenay. On the Weak Keys of Blowfish / Serge Vaudenay // Lecture Notes in Computer Science. – Berlin: Springer Berlin Heidelberg, 1996. – С. 27–32.

24. Karthikeyan Bhargavan. On the Practical (In-)Security of 64-bit Block Ciphers / Karthikeyan Bhargavan, Gaëtan Leurent // 23rd ACM Conference on Computer and Communications Security. – Vienna, Austria, 2016. – С. 456–467.

25. GnuPG Frequently Asked Questions [Електронний ресурс] – Режим доступу до ресурсу: https://gnupg.org/faq/gnupg-faq.html#define_fish.

26. What is Digital Signature- How it works, Benefits, Objectives, Concept [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.emptrust.com/blog/benefits-of-using-digital-signatures.

27. Ashiq JA. Recommendations for Providing Digital Signature Services [Електронний ресурс] / Ashiq JA. – 2016. – Режим доступу до ресурсу: https://www.cryptomathic.com/news-events/blog/recommendations-for-providing-digital-signature-services.

28. Dawn M. Turner. Major standards and compliance of digital signatures - a world-wide consideration [Електронний ресурс] / Dawn M. Turner. – 2016. – Режим доступу до ресурсу: https://www.cryptomathic.com/news-events/blog/major-standards-and-compliance-of-digital-signatures-a-world-wide-consideration.

29. ЗАКОН УКРАЇНИ Про електронні довірчі послуги [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2155-19.

30. D.F. Ferraiolo. Role-Based Access Controls / D.F. Ferraiolo, D.R. Kuhn // 15th National Computer Security Conference / D.F. Ferraiolo, D.R. Kuhn., 1992.

31. NIST SP 800-162. Guide to Attribute Based Access Control (ABAC) Definition and Considerations / Vincent C. Hu, David Ferraiolo, Rick Kuhn та ін.]., 2014. – 47 с.

32. Петр Марголис. Петр Марголис (CUSTIS): Управление правами доступа должно быть централизованным и гибким [Електронний ресурс] / Петр Марголис. – 2016. – Режим доступу до ресурсу: https://bosfera.ru/bo/petr-margolis-custis-upravlenie-pravami-dostupa-dolzhno-byt-centralizovannym-i-gibkim.

33. Xin Jin. ATTRIBUTE-BASED ACCESS CONTROL MODELS AND IMPLEMENTATION IN CLOUD INFRASTRUCTURE AS A SERVICE / Xin Jin., 2014. – 144 с.

34. Thompson, Thomas M. From Error-Correcting Codes through Sphere Packings to Simple Groups / Thompson, Thomas M.., 1983. – 228 с. – (The Carus Mathematical Monographs).

35. Gupta, Vikas. Error Detection and Correction: An Introduction / Gupta, Vikas, Verma, Chanderkant., 2012. – (nternational Journal of Advanced Research in Computer Science and Software Engineering).

36. R. Fielding, Ed. RFC 7235 Hypertext Transfer Protocol (HTTP/1.1): Authentication [Електронний ресурс] / R. Fielding, Ed., J. Reschke, Ed.. – 2014. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc7235.

37. M. Jones. RFC 7519. JSON Web Token (JWT) [Електронний ресурс] / M. Jones, J. Bradley, N. Sakimura. – 2015. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc7519.

38. D. Hardt, Ed. RFC 6749. The OAuth 2.0 Authorization Framework [Електронний ресурс] / D. Hardt, Ed.. – 2012. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6749.

39. Google Sign-In [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/identity.

40. Facebook Login [Електронний ресурс] – Режим доступу до ресурсу: https://developers.facebook.com/docs/facebook-login.

41. Uncovering Spoken Phrases in Encrypted Voice over IP Conversations / CHARLES V. WRIGHT, LUCAS BALLARD, SCOTT E. COULL та ін.]., 2010. – 30 с.

42. Roei Schuster. Beauty and the Burst: Remote Identification of Encrypted Video Streams / Roei Schuster, Vitaly Shmatikov, Eran Tromer // Proceedings of the 26th USENIX Security Symposium. – Vancouver, BC, Canada, 2017. – С. 1357–1374.

43. Y. Sheffer. RFC 7457. Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS) [Електронний ресурс] / Y. Sheffer, R. Holz, P. Saint-Andre. – 2015. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc7457.

44. T. Dierks. RFC 2246. The TLS Protocol Version 1.0 [Електронний ресурс] / T. Dierks, C. Allen. – 1999. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc2246.

45. Roy Thomas Fielding. Representational State Transfer (REST) [Електронний ресурс] / Roy Thomas Fielding. – 2000. – Режим доступу до ресурсу: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

46. Introduction to the Angular Docs [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs.

47. ASP.NET documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1.

48. Rick Anderson. ASP.NET Core Middleware [Електронний ресурс] / Rick Anderson, Steve Smith. – 2020. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-3.1.

49. Entity Framework Core [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/ef/core/.

50. M. Jones. The OAuth 2.0 Authorization Framework: Bearer Token Usage [Електронний ресурс] / M. Jones, D. Hardt – 2012 – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6750.

51. Cross-Origin Resource Sharing (CORS) [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/HTTP/CORS.